CMFS: Compressed Metadata File System For Hybrid Storage

Taizhong Quan, Dylan Yeo, Youjip Won

{coolquan| valkun| yjwon}@ece.hanyang.ac.kr

Abstract

Storage CLASS Memory (SCM) is a next generation of solid-state, nonvolatile memory. It combines the benefits of a DRAM, such as high performance and robustness, with the low cost of conventional hard-disk. The biggest difference between SCM and NAND FLASH is that SCM can be accessed in byte-granularity and access speed is much faster. PRAM (Phase Change RAM), FRAM(Ferro-electric RAM), MRAM (Magnetic RAM), RRAM (Resistive RAM), and Solid Electrolyte Freitas et al are all SCM. The advent of SCM may possibly to make big improvement in storage landscape. In this work, we develop a file system - CMFS (compress metadata file system), which is used on SCM and NAND flash hierarchical storage environment. We make two contributions in this work. First, we store in-memory data in the SCM to overcome the drawback of the current log-structured file system that the mount time dramatically increases as the scale of NAND flash increases. Compared to YAFFS, we show that this file system requires only minimal time for mounting. Second, considering the physical characteristics of SCM, its scale is an order of magnitude smaller and the cost is much higher than the current storage device, we just store some part of the in-memory data to reduce the requirement size of the SCM. The capacity of SCM which is required by CMFS is much smaller than the current file system for hybrid storage. Compare with the MiNVFS [8], the capacity requirement by CMFS is reduced to 14%.

Keywords Non-Volatile RAM, Flash memory, File system, Metadata,

1 Introduction

The SCM technologies such as FRAM (Ferroelectric RAM), MRAM (Magnetic RAM) and PRAM (Phase change RAM) are now concerned as the next generation memory technology. These SCM technologies provide better characteristics that are low power consumption, byte-addressability, non-volatility, and in-place update. These characteristic of SCM will bring various changes to operating systems. In particular, it is expected that SCM will substitute NAND flash memory in area of embedded systems. Although NAND flash memory can be read or written in page size, a unit of erase operation is a block that consists of many pages. This inconsistency of operation unit makes it hard to implement effective storage system. SCM technologies promise effective storage system that has high performance and reliability. Nevertheless, SCM is not used because of low CPM (Cost per MB). Instead of substituting NAND flash memory, combination of NAND flash and SCM promises cost-effectiveness and higher performance.

In this paper, we propose an efficient file system CMFS (compress metadata file system) using on SCM-NAND flash hybrid structure storage. We store metadata in SCM for higher performance. In order to make our approach cost-effective, we reduce the required SCM space for metadata extremely by excluding unnecessary part in the metadata and compressing the necessary part. The CMFS realized efficient real-time update and it also can reduce the mount time sharply when an unclean shutdown happened.

2 Related work

Many researches to improve performance by using SCM have focused on disk-based file systems. However late researches are exploiting file system for SCM and NAND flash memory.

Conquest [4] is a disk/persistent-RAM hybrid file system. It improves a performance by storing metadata and frequently accessed small file in persistent RAM. HeRMES [6] realizes that more than 50% of the file systems accesses are metadata accesses. They use SCM as metadata storage and write buffer. FRASH [7] uses SCM as storage of metadata in flash file system. FRASH [101 develops a “Copy-On-Mount” technique to speed up the mount procedure. It stores in-memory data and on-disk data in SCM and copy the in-memory data into main memory during mount phase. It regularly synchronizes in-memory data to SCM. But when system crash, FRASH has to read the on-disk

structure region of SCM scans NAND Flash storage and reconstructs the in-memory data structure region in the SCM. MiNVFS [8] maintains all the metadata in SCM, while storing all file data in Flash memory. The paper presents experimental results that show how much performance gains can be possible by exploiting SCM. Compared to YAFFS, a typical Flash file system, MiNVFS requires only minimal time for mounting. MiNVFS outperforms YAFFS by an average of around 400% in terms of the total execution time for the realistic workloads. PFFS [9] stores metadata of file systems in PRAM and construct double indirect index similar to ext2. They improve file system performance and consider wear-out of PRAM. MRAMFS [5] write a compressed version to SCM due to the low capacity and relatively high cost of SCM. It compared several compress algorithms. And it shows that LZO is suitable for large size of files.

These file systems store all the metadata in the SCM to improve performance. However they have the same problem - a huge size of SCM is requested and the SCM is very expensive. We can't add enough SCM for a big size NAND flash. For a 1G NAND flash, the spare area is 32MB. A 1000G NAND flash needs 32G SCM. It is too huge to use. And in order to enhance the performance some file systems need much more than that. For example, MiNVFS [8] shows that MiNVFS needs 315KB of SCM when a single 32MB file fills up the 32MB Flash memory. In the worst case, however, the SCM requirement is around 18MB when MiNVFS fills Flash memory with 65,536 files that are 512B in size. In the worse case, an 8G NAND flash needs more than 4G SCM. It is too expensive to add a 4G SCM to an 8G NAND flash.

3 Motivation

SCM technologies provide better characteristics that are low power consumption, byte-addressability, non-volatility, and in-place update. The advent of SCM may possibly bring about drastic changes to the system software landscape. When SCM is efficiently exploited in the system software layer, we expect that the system performance can be significantly improved. NAND flash is erased and programmed in large blocks. The random access time is long but compare to SCM, the cost per bit is low. NAND flash is used as main storage in many places.

In this research, we want to use SCM-NAND flash hybrid structure to improve the performance of the total file system. According to the character of SCM We stored some run time states and object information which are stored in the page spare areas of NAND flash and used in the remounting procedure. It can accelerate the remounting speed.

Some hybrid file systems store in-memory data in the SCM, but they need huge size of SCM and they don't supply an efficient crash recovery method when the system shut down unexpectedly. Some of them should scan all the area of NAND flash to remount the file system when the system crash unexpectedly. It seems not very efficient.

	In-memory data	On-disk data	Compress	synchronization
MiNVFS	O			
FRASH	O	O		O
MRAMFS		O	O	
CMFS	O		O	O

Table1: Characters of file systems

Table 1 shows the different characters between CMFS with other 3 file systems. MiNVFS, FRASH [10] and CMFS all store the in-memory data in the SCM to speed up the mount time. But compare to MiNVFS and FRASH [10], CMFS only stores some part of the in-memory metadata. MRAMFS stores on-disk metadata and small size Files in the SCM. Although it uses compress algorithm to reduce the requirement capacity of SCM, it still requires huge size of SCM. And compare with the 3 file systems, the size of SCM required by CMFS is the smallest. And in CMFS, we develop an efficient synchronization algorithm.

4 Data in SCM

In CMFS, 3 different data structures are used to store the runtime states and object information which are needed when we mount CMFS on the NAND flash. We call them SCM metadata in this paper. They are CMFS_validity, CMFS_device, and CMFS_object. In order to reduce the size of SCM, we minimize the data structures. We only select some part of in-memory data which is used on the remounting and stored in the data pages' spare area. CMFS_validity: stores the mark information which shows whether the SCM is valid. CMFS_device: stores statistical information used in mount time such as the number of erased blocks, the number of files awaiting deletion, the block number which is being allocating and so on. The block number which is being allocating will be used when the SCM metadata is invalid to use. We describe it in section 5. It also stores the block Information about the NAND flash. Such as the number of pages is used in the block, block states etc. We compressed the CMFS_device with LZO [12] algorithm to reduce the requirement size of the SCM. CMFS_object: contains the information about files and directories in the NAND flash. Such as the relationship between others files or directories, file type and the states of the file. If it is a

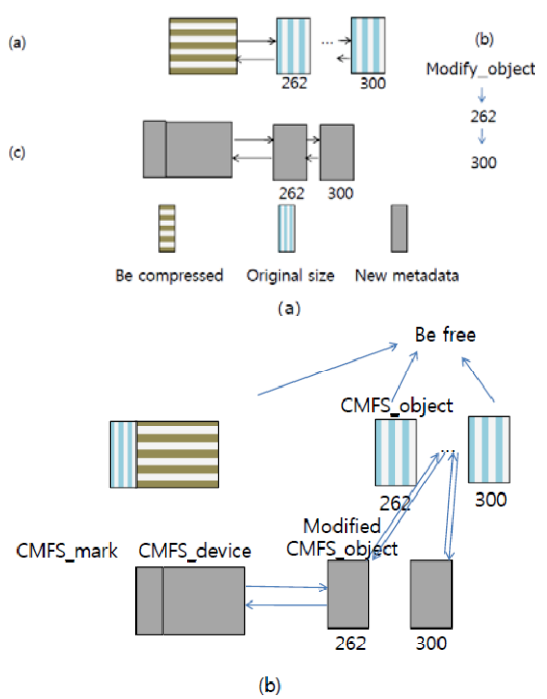


Figure 3:Real time update

file type object, it also contains the information about chunk offset.

We partition the SCM metadata into two parts: fixed-size metadata and variable-size metadata. CMFS_validity, CMFS_device have the fixed size. We store them in a continuous area and they are compressed by LZO [12] algorithm. There is variable number of CMFS_object. And the size of CMFS_object is not fixed. We store them separately and connected them with pointer. The structures of SCM metadata are showed in Figure 1. Figure 2(a) shows a logical hierarchy of directories and files. In Figure 2(b) shows how this would be represented in CMFS. As depicted in Figure 2(a), there are the root directory, 2 subdirectories, and a file-a.txt and the dotted line link the file (b.txt) which will be added to the directory named usr. Each directory or file is represented by a CMFS_object. Every file or subdirectory belongs to a directory will be linked after the directory's CMFS_object and the one which is most recently added in will be just after the directory's CMFS_object. In Figure 2(a), the var, usr and a.txt all belongs to the root directory. So they are all linked after the root directory. And the a.txt file is belongs to the usr so it is linked by the usr directory's CMFS_object. If the b.txt file is added to the usr directory, the position is showed with the dotted line. All the in memory objects have a pointer to their SCM CMFS_object, so we can find them easily.

This kind of data structure has some advantages. First they are very similar with the on-memory data structure. So it is easy to translate the SCM metadata to in-memory data structure to make mounting efficiently. Second using link list is flexible to update the metadata. Adding and deleting an object is easy. When we add a new object we just find the SCM position of its directory and linked it just after the parent directory

5 SCM metadata update

In CMFS, we want to use the SCM metadata to speed up the remounting procedure. But there is a problem. The SSD are written or updated frequently. When a new chuck is written to the SSD, the SCM metadata will become invalid. In order to keep the SCM metadata valid to use, the SCM metadata needs to be updated regularly. Considering the update overhead, the SCM metadata can't be updated every time when every chunk is written in the SSD. In our system, we flush the cache buffer in to the SSD every 5 seconds, the metadata in the SCM also be refresh at that time. Figure 3 shows the procedure of update

When a SSD device is mounted, we will check whether there is any SCM metadata in the SCM. If there is none, we will build it after the device is mounted. After that it will be regularly updated. In Figure 3(a) there are 3 main parts. Part (a) is the outdate SCM metadata. Part b shows the object information need to be updated. Part c is the new SCM metadata. In CMFS, in order to reduce the updating overhead, only the changed part of SCM metadata will be refreshed. Part b shows a link list data structure. It is an in-memory data structure. When an object's information is refreshed in the SSD, the objectId, object type and the SCM metadata position will be stored in the modify_object data structure. According to the modify object list we rewrite the CMFS_object information into the SCM. Part (c) shows the refreshed SCM metadata. The CMFS_device are refreshed every time. Sometimes there are different SCM metadata in the SCM we use the CMFS_validity to verify which one is valid to use. In Figure 3(a) the object 262,300 are written to the SSD. So in part c we can see the new SCM metadata are written in to the SCM. And then as the Figure 3(b) shows, the new SCM metadata will replace the old one and the old metadata space will be released. In order to prevent the SCM metadata being destroyed when an unexpected system crash we should keep all the data in part (a) and part (c) in the SCM until the update is finished. It will be described in section 5.

6 Experiment

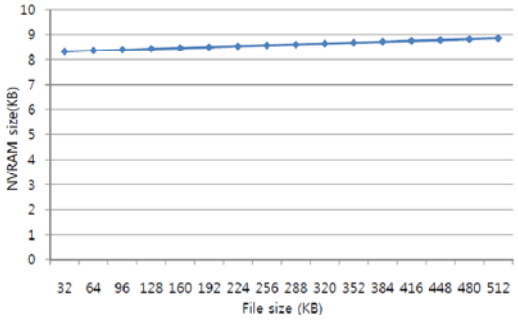
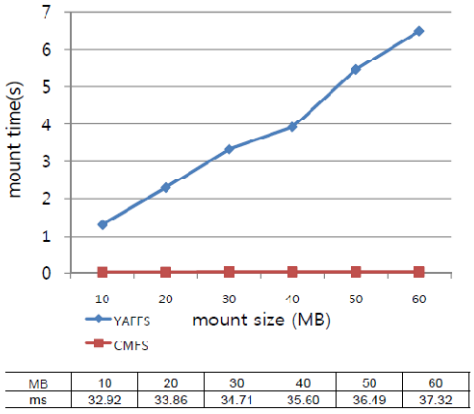


Figure 5: Capacity of SCM



MB	10	20	30	40	50	60
ms	32.92	33.86	34.71	35.60	36.49	37.32

Figure 6: mount time

In our paper, we suggest CMFS (compress metadata file system). We implemented CMFS on embedded system with Marvell PXA320 core, 128 SDRAM, and 128MB large block NAND flash memory. We divided NAND flash into 2 MTD partition and we use one partition of them for experiment.

6.1 SCM Space requirement for CMFS

CMFS uses SCM as an extension of storage to maintain certain metadata to speed up the mount procedure. In this section we analyze the amount of SCM needed by CMFS in the real world. In CMFS, the data structure for the metadata in SCM consists of the CMFS_validity, CMFS_device and CMFS_object. ; Figure 5 shows the variety of SCM space requirement by different file sizes. We assume that there is only one file with different file size in the NAND flash. We can find that the demanded capacity of SCM increased as the file size increasing. The capacity of SCM increases 36byte when the file increases 32KB. For a 32MB file it needs 44KB SCM. The capacity of SCM needed by CMFS is nearly 14% of MiNVFS.

6.2 Mount time

In this subsection, we compare the mount time for YAFFS and CMFS. In YAFFS, the metadata is scattered in Flash memory spare area. On mount time, all the spare area is scanned to build in-memory data structure. With the capacity of NAND flash grows, mount time will proportionally increase. In CMFS, however, we store the metadata in the SCM as described before; hence, it greatly reduces the mount time. There are two different mount procedures. When the metadata stored in the SCM is valid, we use the metadata to rebuild the in-memory data structure on mount time. On the other hand, we should recovery the SCM metadata first, and then remount the device.

Figure 6 shows the mount time for YAFFS and CMFS with different percentage of Flash memory utilization. Compare with YAFFS, CMFS significantly reduces the mount time. And with the utilization of the NAND flash increases, it increases slowly. For YAFFS, mount time increases linearly. In our device YAFFS' mount time increase nearly 1 second when the utilization increase 10MB.

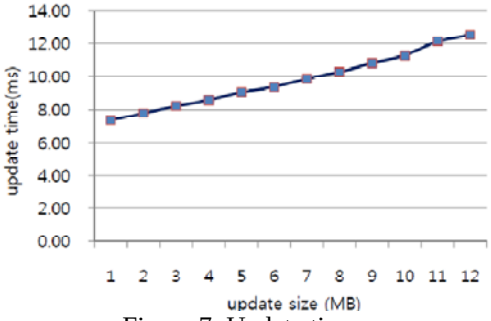


Figure 7: Update time

Figure 7 shows the time which is consumed by the update procedure. It is not related with the utilization of NAND flash. In figure 7, we shows the update time with different NAND flash size need to be updated. We show the update size from 1MB to 12MB. In our device we find that 12MB is almost the biggest update size in 5 second. So the maximum update time in our system is about 13ms. It means that in CMFS we should sacrifice 2.6‰ performance to update the SCM metadata.

7 Conclusion

Due to SCM has the non-volatility of storage and fast access speed of DRAM, SCM is likely to be used in various purpose. However, relatively low capacity and expensive cost is an obstacle for use of SCM. In this work, we handled these disadvantages of SCM with compaction and compression. We efficiently solve the slow speed of mount operation in log structured file system. Compare to MINFS, the SCM space requirement is reduced to 14%.

Acknowledgement

This work was supported by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MEST) (R0A-2009-0083128).

Reference

[1] Aleph One Company, “The Yet Another Flash Filing System (YAFFS)”, http://www.yaffs.net

[2] D. Woodhouse, “JFFS: The Journaling Flash File System”, Proceeding of the Ottawa Linux Symposium, RedHat inc. 2001.

[3] R. Card, T. Ts'o, and S. Tweedie, “Design and Implementation of the Second Extended Filesystem,” The HyperNews Linux KHG Discussion,http:/www.linuxdoc.org, 1999.

[4] A.-I.A. Wang et al., “Conquest: Better Performance through a Disk/Persistent-RAM Hybrid File System,” Proceedings 2002 USENIX Ann. Technical Conf., June 2002.

[5] Nathan K. Edel , Deepa Tuteja , Ethan L. Miller , Scott A. Brandt, “MRAMFS: A Compressing File System for Non-Volatile RAM”, Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04), p.596-603, October 04-08, 2004.

[6] Ethan L. Miller, Scott A. Brandt, Darrell D. E. Long, “HeRMES: High-Performance Reliable MRAM-Enabled Storage”, Proceedings of the Eighth Workshop on Hot Topics in Operating Systems, p.95, May 20-22, 2001.

[7] Eun-ki Kim, Hyungjong Shin, Byung-gil Jeon, Seohhee Han, Jaemin Jung, Youjip Won, “FRASH: Hierarchical File System for FRAM and Flash”, ICCSA 2007, Malaysia, pp. 238-251, 2007

[8] In Hwan Doh, Jongmoo Choi, Donghee Lee, Sam H. Noh, “Exploiting non-volatile RAM to enhance flash file system performance”, Proceedings of the 7th ACM & IEEE international conference on Embedded software, 2007.

[9] Youngwoo Park, Seung-Ho Lim, Chul Lee, Kyu Ho Park, “PFFS: a scalable flash memory file system for the hybrid architecture of phase-change RAM and NAND flash”, Proceedings of the 2008 ACM symposium on Applied computing, 2008.

[10] Jung, J. and Won, Y. and Kim, E. and Shin, H. and Jeon, B, ” FRASH: Exploiting Storage Class Memory in Hybrid File System for Hierarchical Storage”

[11] RAMTRON Datasheet

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