# Object-based SCM: An Efficient Interface for Storage Class Memories

Yangwook Kang Jingpei Yang Ethan L. Miller
Storage Systems Research Center, University of California, Santa Cruz
{ywkang, yangjp, elm}@cs.ucsc.edu

## Abstract

*Storage Class Memory (SCM) has become increasingly popular in storage systems. However, replacing hard drives with SCMs often forces either major changes in file systems or suboptimal performance, because the current block-based interface does not deliver enough information to the device to allow it to optimize data management for specific device characteristics such as the out-of-place update. To alleviate this problem and fully utilize different characteristics of SCMs, we propose the use of an object-based model that provides the hardware and firmware the ability to optimize performance for the underlying implementation, and allows drop-in replacement for devices based on new types of SCM. We implement an object-based flash memory prototype.*

## 1. Introduction and Motivation

Storage class memories (SCMs) are playing an increasingly important role in the storage hierarchy. The combination of low power consumption, relatively large capacity, fast random I/O performance and shock resistance make SCMs attractive for use in desktops and servers as well as in embedded systems. Recently, deployment of Solid State Drives (SSDs) using NAND flash has rapidly accelerated. However, there are many other SCM technologies beyond NAND flash, including FeRAM, Phase Change RAM (PCM), and carbon nanotube, that may see dramatically increased use in the near future. It is critical to design systems that can both fully utilize flash memory and easily accept drop-in replacements using future technologies.

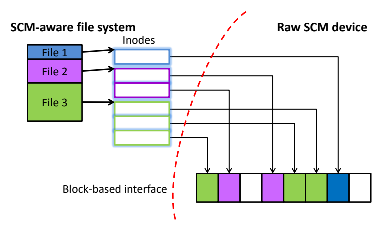
Although SCMs generally provide better performance than hard drives, they require more intelligent algorithms to efficiently handle unique requirements such as out-of-place update. As SCM technologies differ in many characteristics, the design of file systems optimized for each technology also varies significantly, creating issues of portability and compatibility. Efforts to exploit these new characteristics in file systems have driven a great deal of research, primarily using one of two approaches; the direct-access model and the FTL-based model. The first model (Figure 1(a)), either places SCM on the main memory path [1], or uses a specific file system that allows SCMs to work properly in the system [4]. This model provides optimal performance for a specific hardware configuration, but suffers from a potential requirement to redesign the file system to optimally utilize (or simply function properly with) different SCMs.

The second model (Figure 1(b)), interposes firmware (Flash Translation Layer, or FTL) between the raw device and a standard block-based file system, hiding the complexities of managing hardware from the file system and allowing devices to be accessed directly by an unmodified disk file system. However, this approach achieves suboptimal performance due to the lack of file system semantics delivered to the hardware and the duplication of block mapping in both the file system and the device.

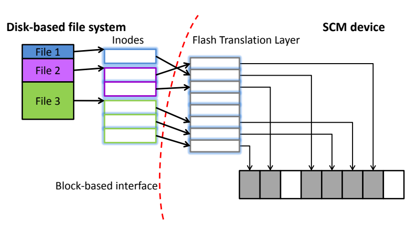
To alleviate the problems of current approaches to integrating SCMs into the file system and exploit the characteristics of various SCM devices without either limiting the design flexibility or introducing additional overhead, we explore the use of an object-based storage model [2] for SCMs. This model offloads the storage management layer from the file system to the underlying hardware, enabling device manufacturers to optimize the mapping layer based on the hardware configuration (Figure 1(c)). The POSIX-level requests are encapsulated in an object with their metadata information and sent to the device through an object-based interface. By doing so, the object-based storage model provides an easy transition between different SCM devices. Moreover, this model can be implemented on any type of SCM device more flexibly, while having one generic object-based file system on the host.

## 2. Object-based SCMs

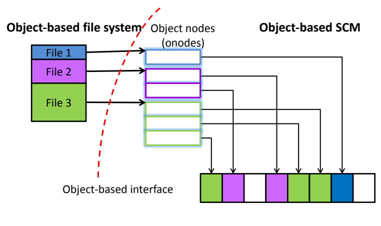
In systems built on the object-based storage model, a file system does name resolution on the host side, of-



(a) Direct access model.



(b) FTL-based model.



(c) Object-based model.

Figure 1. Three approaches to access SCMs from file system

floading the storage management layer to the OSD. By isolating device-specific technology behind an object-based interface, this model allows the file system to be independent of the particulars of the storage medium while the characteristics of the storage medium are efficiently handled within the device. Thus, a single file system can be efficiently used with different types of SCM devices, in contrast to the current approaches that either require significant changes in the system or sacrifice I/O performance.

The object-based interface delivers objects (which contain both data and associated metadata) and recognizes all types of requests that the file system does. This allows devices to provide features such as hot/cold separation to reduce cleaning overhead. For small objects, OSDs can achieve better space efficiency than block-based devices due to the lack of a minimum allocation size. OSDs can reduce index overheads by using extent-based allocation for large, infrequently updated objects. In addition, by encapsulating data in objects, OSDs can provide more advanced features, such as object-level reliability and compression. Moreover, adding an object interface will not significantly complicate make existing FTL firmware since SCM devices already need a translation layer for data placement and segment management. For example, when hybrid phase-change/flash memory is used, the file system can store data efficiently by simply sending a write-object request to the OSD with no need to know about the two types of memory in the device.

## 3. Current Status

We built an object-based model prototype as a file system module in the 2.6 Linux kernel to investigate the design issues in the object-based model for SCMs. We explore three data placement policies, which separates data and metadata, and further extracts access time from metadata, to reduce the overall cleaning overhead. We use a wandering tree combined with extent-based allocation to show the effects of extent-based allocation. In addition, we implemented object-

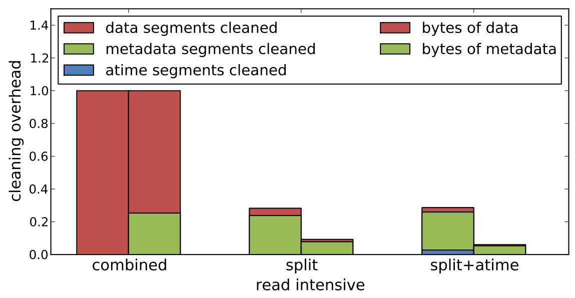


Figure 2. Cleaning overhead of three data placement policies under read-intensive workload. The overhead is normalized to the combined policy

level reliability in our prototype to investigate the impact of advanced features.

Our experiments on Postmark benchmark for both read- and write- intensive wokload shows that the cleaning overhead is significanly reduced by sepearating data, metadata, and *atime*, as shown in Figure 2. When extent-based allocation is enabled, we can get much benefit by reducing the number of page I/Os issued by the index structure.

In summary, to avoid the limitations of standard block-based interfaces, we propose the use of object-based SCMs. This approach allows systems to immediately utilize new SCM technologies with no change to host systems, and enables the device to optimize the performance flexibly.

## References

[1] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee. Better I/O through byte-addressable, persistent memory. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles*, Oct. 2009.

[2] G. A. Gibson and R. Van Meter. Network attached storage architecture. *Commun. ACM*, 43(11):37–45, 2000.

[3] Y. Kang, J. Yang, and E. L. Miller. Efficient storage management for object-based flash memory. In *MASCOTS'10*, Aug. 2010.

[4] A. O. Ltd. YAFFS: Yet another flash file system. http://www.yaffs.net.