Microprocessors and Microsystems 50 (2017) 54–65

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Virtual duplication and mapping prefetching for emerging storage primitives in NAND flash memory storage systems

Yi Wang[a,b,*], Lisha Dong[b], Zhong Ming[b], Yong Guan[c], Zili Shao[d]

[a] Beijing Advanced Innovation Center for Imaging Technology, China
[b] College of Computer Science and Software Engineering, Shenzhen University, China
[c] College of Information Engineering, Capital Normal University, China
[d] Embedded Systems and CPS Laboratory, Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China

ARTICLE INFO

Article history:
Received 2 February 2016
Revised 19 December 2016
Accepted 28 February 2017
Available online 7 March 2017

Keywords:
Storage primitives
NAND flash memory
On-demand
Mapping prefetching
Virtualization

ABSTRACT

NAND flash memory has become the mainstream storage medium for both enterprise high performance computers and embedded systems. However, over the past several decades, the storage primitives that access secondary storage have remained unchanged, forcing NAND flash memory to serve merely as a block device like hard disk drive. Recently, several emerging storage primitives have been presented to explore the potential value of non-volatile memory devices. Although these primitives can significantly boost the access performance by providing virtual to logical address mappings, they still suffer from large RAM footprint to maintain the address mapping table and require further support for update operations.

This paper presents *ESP* to optimize *E* merging *S* torage *P* rimitives with virtualization for flash memory storage systems. We propose two optimization strategies, virtual duplication and mapping prefetching to solve the critical issues in existing emerging storage primitives. The objective is to reduce unnecessary flash memory accesses and keep RAM footprint of address mapping table well under control. We have evaluated ESP on an embedded development platform. Experimental results show that ESP can significantly improve the write/read performance and reduce over 30% of garbage collection operations.

1. Introduction

During past decades, the capacity of NAND flash memory has been increasing dramatically, leading to the use of nonvolatile flash in the system's memory hierarchy [1]. However, storage interfaces for accessing secondary storage devices have remained largely unchanged since the invention of magnetic disk. Three storage primitives (i.e., seek, read, and write) are still being used as fundamental interfaces to persistent storage. These legacy storage primitives force secondary storage devices to behave as a fast block device like magnetic disk and ignore the possibility to boost the performance of storage devices with other types of non-volatile memory, especially NAND flash memory. As existing storage primitives are designed to cater for the traditional hard disk, considerable efforts at architectural and operating system levels are required to facilitate smooth access to NAND flash memory as secondary storage.

In order to utilize the unique feature of NAND flash memory, storage virtualization has been adopted to introduce abstraction between physical storage device and the logical device presented to the host application. Some recent virtualization techniques adopt a new class of emerging storage primitives beyond traditional block I/O. By providing address remapping at the application-level [2], it effectively extends current storage primitives and provides more flexibility to users. Despite its promising benefits, this technique still suffers from unnecessary write and update operations for NAND flash memory devices, which will significantly affect the endurance and access latency of storage systems. The large RAM cost to maintain the address mappings in these virtualization techniques will also degrade the performance of storage systems. These observations motivate us to design an optimization strategy to enhance the emerging storage primitives and to reduce RAM footprint for flash memory storage devices.

In previous studies, advanced storage virtualization techniques, such as deduplication and thin-provisioning, have shown greater flexibility to manipulate address remapping and re-direct I/O [3–5]. Flash translation layer (FTL) is also a type of virtualization that manages the address translation between logical and physical

* Corresponding author.
E-mail address: yiwang@szu.edu.cn (Y. Wang).

http://dx.doi.org/10.1016/j.micpro.2017.02.008
0141-9331/

addresses for flash memory devices. However, these optimization techniques are still working behind the block device interface, resulting in sub-optimal access performance across various storage layers. FTL also restricts the range of the logical addresses to be the same as that of the physical addresses [6]. This will prevent the application users from directly controlling the I/O accesses in flash memory devices.

In this paper, we present *ESP* to optimize *E* merging *S* torage *P* rimitives with virtualization for flash memory storage systems. ESP adopts emerging storage primitives and handles the virtual addresses to achieve reduced updates to the flash memory storage systems, with the resulting advantage that *file operations from application users can directly manipulate the I/O accesses without incurring extra RAM cost.* With ESP, file system users can continue to enjoy advanced file system features, such as scalability and performance enhancements for large files. By manipulating the address translation between virtual address and logical address, ESP can significantly reduce unnecessary I/O operations. Therefore, it can boost the access performance and extend the life time of NAND flash memory devices.

ESP adopts two strategies to handle write/read requests from the file system and transparently issues the requests to flash translation layer. ESP uses *virtual duplication* strategy to maintain the address mappings where multiple virtual addresses with the same content could be pointed to the same logical address. Emerging storage primitives are utilized to facilitate the address translation. ESP also uses *mapping prefetching* strategy to reduce the RAM cost for maintaining virtual to logical address mappings. The address mapping table is stored in flash memory, and only limited mapping items are prefetched to working RAM based on the demand fashion. By using these two strategies, ESP can take advantage of emerging storage primitives and keep the RAM cost well under control.

We applied ESP to the ext3 file system and tested the performance using a variety of I/O traces. We use response time and garbage collection overhead as metrics to evaluate the performance of ESP. Experimental results show that our approach can improve the response time by 56.71% and reduce an average of 34.10% for garbage collection operations in comparison with a representative scheme that uses emerging storage primitives.

This paper makes the following contributions:

- We present reliability enhancement strategies that can transparently protect the data integrity of metadata in NAND flash memory.
- We demonstrate the effectiveness of our technique by comparing with representative works using a set of realistic I/O traces.

The rest of this paper is organized as follows. Section 2 discusses the motivation and analyzes the problem. Section 3 presents our proposed ESP in detail. Section 4 presents experimental results. Finally, Section 5 concludes this paper and discusses future work.

2. Background and motivation

Modern file systems (e.g., ext3 file system) normally adopt journaling file systems. For a journaling file system, it will first write information about pending updates to a write-ahead log [7], and then commit the updates to disk. The design of journaling file systems is to guarantee the consistency of the file system upon power failure or other system crashes. This process works well when the secondary storage is a hard disk drive, as hard disk drives accept in-place update that will not incur time and space overhead. However, when a journaling file system is built on a flash memory device, the same file system changes in the main memory have to be written to storage twice [8].

A NAND flash memory chip contains multiple blocks, and each physical block consists of a fixed number of physical pages. A block is the basic unit for erase operations, while a page is the minimum unit for read/program operations. NAND flash memory has several distinct characteristics (i.e., endurance and “out-of-place” update) [9,10]. To conceal these unfavorable characteristics, an intermediate software module called flash translation layer (FTL) is employed to emulate NAND flash as a block device [11–14]. Solid-state drives (SSDs) use NAND flash memory as storage media and also adopt similar FTLs to improve the performance [15–17].

Fig. 1 illustrates the system architecture for NAND flash memory storage system. Basically, there are two types of flash memory storage systems, file system based flash memory storage system, and flash translation layer (FTL) based flash memory storage system. File system based flash memory storage system provides a flash-aware interface that can directly handle the flash device. FTL-based system hides the properties of flash and emulates the flash memory as the block device. The inputs of file system based and FTL-based system are both virtual addresses. Therefore, it is possible to manipulate the virtual-logical-physical address translations for both types of flash memory storage systems.

A typical flash-based NAND flash storage system normally consists of two layers, FTL layer and Memory Technology Device (MTD) layer. MTD layer implements primitive functions over flash memory, such as read, write, and erase operations. The main role of FTL is to redirect logical addresses from the file system of a host into physical addresses in NAND flash, and maintains a mapping table to keep track of the mapping information. FTL also provides useful components, such as garbage collector and wear-leveler, to optimize the space utilization and maintain the same level of wear for each block in NAND flash memory [18].

As FTL uses logical address to identify read/write accesses, the aforementioned process in journaling file system will hold two separate logical addresses for the journal and the home locations. Therefore, the duplication of the identical data will be issued to two different physical locations. In order to reduce unnecessary write accesses to flash devices for the same content, emerging storage primitives (i.e., *clone, move*, and *delete*) have been proposed [2,19]. Fig. 1(b) shows the system architecture that adopts emerging storage primitives. In this architecture, these storage primitives provide virtual address spaces that are much larger than those of the logical or the physical address spaces in the storage system. The host system can directly manipulate the address map and only issue one single access to storage devices for identical data.

In order to fully explore the unique aspects of the flash memory and take advantage of emerging storage primitives, several issues have to be considered. First, current storage systems (e.g., ANViL [2]) that use emerging storage primitives did not consider the update operations to the journal or the host location. When an update operation comes, they will treat the updated data as new logical addresses and allocate new physical address to store the updated data. Then the journal and the host location will be pointed by two separate logical addresses. Writing the updated data from the scratch will incur extra time and space overhead, as the journal and the host location may still share a large portion of identical data. Second, these storage systems normally maintain the virtual-to-logical address mappings with multiple virtual addresses point to the same logical address. This address mapping table will be much larger than the address mapping table for the logical-to-physical address mappings handled by FTL. How to effectively manage this virtual-to-logical address mapping table becomes a critical issue to ensure that emerging storage primitives can be successfully applied to flash memory devices. These observations motive us to propose new address mapping strategies to supplement emerging storage primitives and to provide a more efficient flash memory.