Journal of Systems Architecture 60 (2014) 329–344

Contents lists available at ScienceDirect

Journal of Systems Architecture

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

Integrated write buffer management for solid state drives

Sungmin Park, Jaehyuk Cha, Sooyong Kang *

Division of Computer Science and Engineering, Hanyang University, Seoul 133-791, Republic of Korea

ARTICLE INFO

Article history:
Received 20 November 2012
Received in revised form 14 January 2014
Accepted 20 January 2014
Available online 29 January 2014

Keywords:
Flash memory
Write buffer
Log block
Flash translation layer
Solid-state disk
Storage device

ABSTRACT

NAND flash memory-based Solid State Drives (SSD) have many merits, in comparison to the traditional hard disk drives (HDD). However, random write within SSD is still far slower than sequential read/write and random read. There are two independent approaches for resolving this problem as follows: (1) using overprovisioning so that reserved portion of the physical memory space can be used as, for example, log blocks, for performance enhancement, and (2) using internal write buffer (DRAM or Non-Volatile RAM) within SSD. While log blocks are managed by the Flash Translation Layer (FTL), write buffer management has been treated separately from the FTL. Write buffer management schemes did not use the exact status of log blocks, and log block management schemes in FTL did not consider the behavior of the write buffer management scheme. This paper first demonstrates that log blocks and write buffers maintain a tight relationship, which necessitates integrated management to both of them. Since log blocks can also be viewed as another type of write buffer, we can manage both of them as an integrated write buffer. Then we propose an Integrated Write buffer Management scheme (IWM), which collectively manages both the write buffer and log blocks. The proposed scheme greatly outperforms previous schemes in terms of write amplification, block erase count, and execution time.

1. Introduction

Recent advancements within the NAND-FLASH technology have supported the growing usage of a Solid State Drive (SSD)[1] as a commodity storage media. Although an SSD has still a higher cost per byte, as compared to an HDD, it is expected in the near future that SSD will replace most HDDs [10]. SSD, unlike HDD, has no mechanical movement when accessing data, and this distinguishing feature guarantees the excellent performance of random reads (i.e. 20–50 times faster than HDDs [12]). In contrast, flash memory's lack of an in-place update enforces it to erase data blocks prior to the data update operation. Within flash memory, read/write operations are done in a page unit, while an erase operation is done in a block unit. In order to update particular data within a specific block, flash memory first erases the entire target block storing the data and then writes newly updated data in company with the pre-existing non-updated data stored in the same block. Due to the fact that an SSD includes such a drawback of flash memory, heavy random write requests generate numerous copy and erase operations, even for valid (i.e. non-updated) pages internally. This is the reason for both the degraded performance and shortened lifespan.

In recent years, many studies from different system layers have been performed to overcome such a write anomaly within flash memory. In order to reduce the number of write operations in flash memories, flash-aware buffer management schemes [24,14,13, 26,29] and write buffer management schemes in SSD [17,16] have been proposed. As an approach that aims to directly manage flash memory by developing flash file systems, YAFFS [5], JFFS2 [8], and UBIFS [21], have been developed. Meanwhile, there are attempts to research on file systems for HDDs, which also nicely accommodate the characteristics of SSDs, such as NILFS [22], BtrFS [23], and ZFS [28]. At the host interface layer, new commands (for example, TRIM command) that are specifically used for an SSD are added in order to reduce the number of copies and erases for valid pages by passing the information for erased data blocks to the SSD. Nevertheless, if these approaches are used in a system that has both SSDs and HDDs, there are certain limitations that prevent users from adopting these approaches: (1) buffer replacement policies might cause performance degradation when HDD is used together with SSD, (2) it require users to install an additional flash file system, besides a file system for HDD, and (3) a flash-aware general purpose file system might not simultaneously optimize the performance of both SSD and HDD. Hence, for the widespread use of an SSD, it is needed to resolve the SSD's internal problems *inside* an SSD.

* Corresponding author. Address: Division of Computer Science and Engineering, Hanyang University, 222 Wangsimni-ro, Seongdong-gu, Seoul 133-791, Republic of Korea. Tel.: +82 2 2220 1725; fax: +82-2-2220-4244.
E-mail address: sykang@hanyang.ac.kr (S. Kang).

[1] Throughout this paper, flash memory means NAND flash memory and SSD means NAND flash memory-based SSD.

http://dx.doi.org/10.1016/j.sysarc.2014.01.005
1383-7621/

There are two research directions that can be used for solving the random write problem inside an SSD. The first approach is to use certain data blocks as log blocks or replacement blocks so that the updated data is written on log blocks instead of original data blocks. The Flash Translation Layer manages these log blocks, and the random write performance and the number of operations differ largely according to the number of log blocks and the management policy of an FTL. In this FTL category, we have seen NFTL [20], AFTL [7], Superblock Scheme [15], BAST [18], and FAST [19] in literature. The other approach is to exploit a write buffer inside an SSD in order to reduce the number of write and erase operations, and we have seen BPLRU [17], CLC [16] and PUD-LRU [30].[2]

As the two aforementioned approaches have been researched independently, there is no such thing as interdisciplinary study considering both the write buffer management policy and the log block management policy in union. In our previous work [16], we proposed an Optimistic FTL, a log block management policy that is designed by observing destaged data patterns from a write buffer to flash memory. Optimistic FTL manages log blocks while considering the characteristics of normal write buffer management policies. In contrast, REF [26] assumed a normal log block management policy and proposed an OS-level write buffer replacement policy that works according to the assumed log block management policy. Although these two studies suggest the necessity of mutual recognition between a log block management policy and a write buffer management policy, they remained as a reactive policy, which considers only the result of the other policy, not a proactive policy that operates closely with the other. However, since both log block management and write buffer management policies are under the control of an SSD controller, the information sharing and the possible ties between the two policies are feasible. In addition, from a broad perspective, a log block also can be regarded as a kind of write buffer. Hence, it is necessary to research an interdisciplinary study between these two policies.

The main contribution of this paper is to introduce the necessity of integrated management for both log blocks and write buffers to the SSD community. In detail, we first delve further into the correlation between the two management policies by means of extensive interrelated experiments using a variety of workloads, and we then devise an example scheme, Integrated Write Buffer Management Policy (IWM), in order to validate our proposed necessity. Extensive experiments, under a variety of experimental configurations, show that IWM, in most cases, shows a better performance than any other policy. It successfully reveals the potential of the integrated buffer management, in terms of both the performance and lifetime of SSD, and we think that it can be a stepping stone for developing a more effective integrated buffer management scheme, in the future.

The rest of the paper is organized as follows: Section 2 explains background materials and motivations, Section 3 discusses IWM in details, Section 4 shows performance studies. In Section 5, we discuss the similarities and differences between our work and prior works, and in Section 5 we conclude the paper.

2. Background and motivation

2.1. Background

Log block management: Hybrid mapping schemes use three kinds of merge operations – switch merge, partial merge, and full merge. Among them, only full merge is implemented differently according to the log block management policy in each Hybrid mapping scheme. Switch Merge is used when all pages in a log block are sequentially written. Partial Merge is performed when some pages (including the first page) in a log block are sequentially written and the remaining pages are free. Full Merge is executed when pages in a log block are randomly written, and is the most expensive merge operation. The two most representative hybrid mapping schemes are BAST and FAST. While FAST copes with random writes more efficiently than BAST, it shows very large fluctuations in the full merge overhead.

The page mapping policy usually manages the entire address space in a page unit, eliminating the necessity of log blocks. It has the advantage of efficiently handling random writes, but has the drawback of maintaining a large page table that maps the entire pages within SSD. For example, 1 TB SSD needs at least 1 GB DRAM for mapping table. In order to remedy this, DFTL [11] proposed a novel method that implements the page mapping strategy with a small amount of memory, which stores only a certain part of the entire page table. This is achieved by viewing the locality information of given workloads. However, this approach cannot help showing poor performance for low locality workloads. Especially, as the SSD capacity becomes larger and larger while its performance becomes better and better, an SSD can be used in multi-purpose systems which provide multiple services such as web, mail, file and ftp services, simultaneously. In those systems, the aggregated IO accesses can have low overall locality, which can make the caching-based FTL poor. In this regard, more recent works, such as Janus-FTL [31] and WAFTL [32], try to use both block and page mappings, simultaneously, in a single device. These works exploit their own operations which correspond to merge operations in Hybrid mapping schemes (i.e., *Buffer Zone Migration* in WAFTL and *Fusion* and *Defusion* operations in Janus-FTL).

By effectively exploiting both block and page mappings, they could not only decrease the amount of mapping information but also improve the performance. For example, in WAFTL, they showed that by workload-adaptively using multiple mapping granularities and exploiting appropriate caching strategy, they could achieve up to 34% performance improvement over DFTL. It means that if the caching scheme is combined with the hybrid granularity mapping scheme, it is possible to achieve an improved performance than the pure caching-based FTL. In Janus-FTL, they showed that their scheme, in some cases, could outperform pure page-mapped FTL which has no outstanding optimization technique for garbage collection. Hence, assuming the SSD capacity will ever increase, we believe that the basic mechanism of the Hybrid mapping schemes will still be valuable in terms of both the theoretic and practical aspects. Since our contribution is providing a framework for an integrated management between write buffer and log block, provided that a given FTL exploits a log block-like scheme, our framework can be effectively used to further increase the performance of the FTL.

Write buffer management: For many decades, we have seen numerous studies on buffer management policies for an HDD. Considering both hit ratio in write buffers and mechanical movements inside an HDD, write buffer management schemes, such as Stack Model [6] and WOW [9], have regarded temporal and spatial locality as important factors. In opposition to this, write buffer management policies for flash memory do not need to consider mechanical movements, rather, there should be an effort to reduce the number of extra operations occurring inside an FTL. In traditional write buffer management policies, for the efficient merge operation of an FTL, the method of clustering pages in the same block of flash memory and writing these clustered pages together is widely used within research work [13,17,26,16,30]. FAB [13] proposed a DRAM buffer management policy for a Portable Media Player using flash memory, and it was the first attempt to cluster pages aligned by the erase boundary in flash memory. When replacing buffers, FAB selects the biggest cluster as a victim cluster

[2] Though FAB and REF were initially devised for the page replacement scheme within Operating Systems, they also can be used as a write buffer management scheme in SSD. Therefore, in this paper, we categorize them as write buffer management policies.