A Memory Hierarchy-Aware Metadata Management Technique for Solid State Disks

Kwanhu Bang, Sang-Hoon Park, Minje Jun and Eui-Young Chung
School of Electrical and
Electronic Engineering
Yonsei University
Seoul, Korea 120-749
Email: {khbang, soskhong}@dtl.yonsei.ac.kr, {jjuninho, eychung}@yonsei.ac.kr

***Abstract*— Solid State Disk (SSD) drives are rapidly replacing conventional hard disk drives (HDDs) due to their remarkable performance gains. For emulating HDDs, SSDs require a flash translation layer (FTL) which hides the out-of-place-update feature of NAND flash memories. In the latest large-capacity SSDs, FTLs must manage huge metadata such as a logical-to-physical address mapping table, a pool of free blocks, or a list of garbage blocks with their erase counts. The total metadata cannot reside on a small on-chip SRAM so that it must be hierarchically distributed in DRAM or NAND flash memories. This paper presents an efficient metadata management technique for SSDs which fully exploits memory hierarchy of an SSD. By the proposed technique, the distributed metadata can be efficiently searched or updated with small overheads. Experimental results show that overheads of metadata management become considerably large in the latest SSDs and they are minimized efficiently by the proposed technique.**

I. Introduction

NAND flash memory is being widely adopted in new storage devices, replacing conventional magnetic disks. Flash-based storage devices have many advantages over magnetic disks, such as small size, lightweight, low power consumption, and shock resistance. Due to these advantages, NAND flash memory became the most appropriate storage element for many portable devices such as digital cameras, portable media players, smart phones, and tablets as well as personal computers and servers. With explosive growth of consumer electronics market, demand for NAND flash memory also continues to grow exponentially even now.

However, NAND flash memory has some weak points, e.g. asymmetric read / write access speed, much larger erase unit than read / write unit, etc. The most critical factor that causes performance degradation of SSDs is that NAND flash memory does not support in-place update, unlike conventional hard disk drives (HDDs). This characteristic not only causes performance penalties, but also forces SSDs to require different access methods from conventional HDDs.

To amortize aforementioned characteristics and to emulate the same access interface as that of HDDs, most of SSDs are equipped with an internal software layer called *flash translation layer (FTL)*. It enables SSDs to simply replace conventional block devices and minimizes modification in disk I/O management of the operating systems. Major functions of FTL are 1) logical-to-physical address translation, 2) garbage collection which reclaims used blocks, and 3) wear-leveling to increase lifespan of NAND flash. In order to execute these functions efficiently, many data structures for FTL, such as address mapping table, pool of free blocks, and list of garbage blocks with their erase counts, are needed, and these data are typically called FTL *metadata*.

The FTL metadata have two conflicting characteristics which complicates their proper management. First, they need to be stored on non-volatile memory, otherwise the user data will be permanently unaccessible when the system experiences power-failure. Second, the metadata experience much more frequent access than user data since all the accesses to the user data in NAND flash require an access to the metadata (e.g. reference to the address mapping table), while its size is much smaller than that of the whole user data. These two characteristics imply that frequently accessed metadata should be stored and updated in slow NAND flash memory, which obviously has adverse effect on SSD's performance and lifespan. This problem can be resolved if the whole metadata can be loaded onto a faster memory such as SRAM or DRAM during the runtime and stored back into NAND flash at system shutdown. However, unfortunately, the size of metadata is proportional to the total capacity of the SSD, and it is unavoidable to manage the metadata in cooperation with NAND flash for these days' large-scale SSDs.

To tackle this problem, this paper introduces an efficient FTL metadata management technique for large-scale SSDs, which fully exploits internal memory hierarchy of the SSD. With the proposed technique, the FTL software can read or update its metadata quickly and all the overheads caused by metadata transfers in the memory hierarchy can be minimized. The rest of the paper consists of the followings. In Section II, the related works about FTL metadata management are listed. The proposed metadata management technique is described in Section III, and experimental results for evaluating the technique is showed in Section IV. Finally, the conclusion is given in Section V.

II. Related Works

As explained before, FTL must have three major functions, i.e. address translation, garbage collection, and wear-leveling. Among them, address translation scheme is enthusiastically

978-1-61284-857-0/11/$26.00

researched than the others [1]–[3]. NAND flash is accessed by the unit of a page or a block, hence, basically, block-level mapping and page-level mapping are possible. Since a block consists of many pages, block-level mapping requires only a small mapping table, but it is very inefficient for small and frequent random updates due to its coarser mapping granularity. On the other extreme, page-level mapping can cope with random updates more efficiently than block-level mapping, but it requires huge memory space to store the mapping table. To resolve this problem, many block-page hybrid address mapping schemes have been proposed which trade-off between two corners. Block-page hybrid address mapping schemes have tackled associativity between *a data block* which takes block-level mapping and *a log block* which has page-level mapping. In [1], *a data block* is associated with only one *log block*, or they are fully associated in [2].

There are some works consider very large FTL metadata which cannot reside on on-chip SRAM [4]. Under this constraint, several FTL metadata management schemes are also proposed [5], [6]. In [5], the authors firstly proposed on-demand loading of FTL metadata stored in NAND flash into a small SRAM, called DFTL. However, they did not fully consider different characteristics of the on-chip SRAM, the external DRAM cache buffer, and NAND flash. The authors of [6] tried to reduce the overhead incurred by DFTL - the frequent updates of metadata in NAND flash, by reducing the size of the mapping table using block-level mapping. They additionally introduced a two-level caching of FTL metadata in order to overcome weak points of the block-level mapping, but they did not focus mainly on internal memory hierarchy.

In summary, there have been no works proposing efficient management algorithm of FTL metadata with consideration of important hardware constraints such as characteristics of different types of memories equipped in SSDs. Our technique pays attention to the proper distribution of FTL metadata to fully utilize advantages of each memory element's characteristics. Moreover, our work also considers many newest hardware schemes such as large-scale battery-backed DRAM and multi-channel / multi-way composition of NAND flash, which the previous works did not take into account.

III. A Memory Hierarchy-Aware Metadata Management

A. Internal Hardware and Software of an SSD

Fig. 1 shows the internal architecture of an SSD we use. It is equipped with an on-chip SRAM, which is located nearby the microprocessor inside the SSD controller, an external DRAM as cache buffer, and multi-channel/multi-way NAND flash memory chips. The on-chip SRAM is used as scratchpad for FTL execution, thus it requires to keep a small portion of metadata needed for FTL operation. The external DRAM acts as a read cache as well as a write buffer not only for user data, but also for FTL metadata. NAND flash, which is the only non-volatile memory in the SSD, is the main storage medium of user data and the lowest level memory for FTL metadata. Even though it is not recommended to update data stored in NAND flash frequently due to slower speed and limited lifetime of NAND flash, every write request from host incurs updates in FTL metadata which must be propagated to the lowest level memory to prevent data loss caused by sudden power cut-offs.

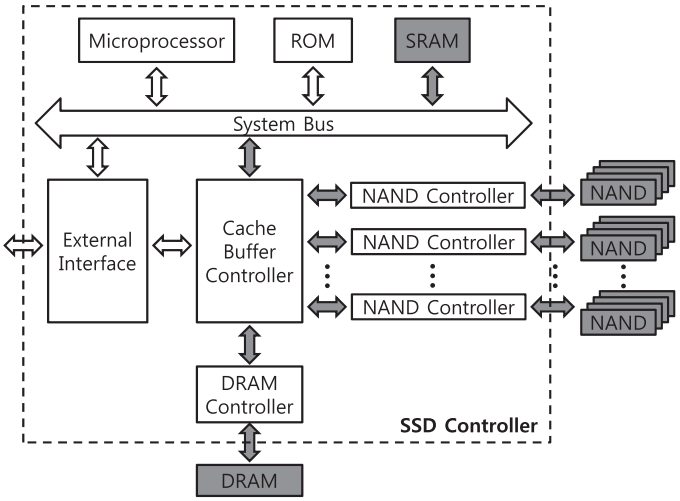

Fig. 1. Typical Internal Architecture of an SSD

An FTL used by the SSD is mainly implemented as a software for flexibility and scalability. It manages its own metadata transparent to the host file system. The most important part is the mapping table. The mapping table provides address translation and has larger size than the other elements of FTL metadata. Like other FTLs, our FTL also utilizes over-provisioning blocks, which is the additional blocks to mitigate out-of-place feature of NAND flash. The number of over-provisioning blocks affects not only performance of SSDs, but also the size of mapping table of the FTL. In addition, a list of free blocks and statistics for wear-leveling (e.g. erase count of each block) must be managed. Any other data structures can be added for different FTLs.

B. The Proposed FTL Metadata Management Scheme

In this subsection, we describe details of the proposed scheme in terms of four essential points of it. The main goal of the design is the minimization of overheads caused by search or update of FTL metadata. The overall management algorithm and distributed data structures are shown in Fig. 2.

1) Address Mapping for Multi-channel/Multi-Way Architecture: In the multi-channel/multi-way architecture, selection of proper channel and way determines the efficient utilization of NAND flash.To be optimal, the selection needs to trace and search FTL metadata globally all over the channels and ways, which results in the inter-channel/way scanning for every request from the host. In order to accelerate this process, we allocate channels and way statically to each request based on its logical addresses. More specifically, by masking lower-order bits of the logical address, the specific channel and way are easily determined. As a result, requests with adjacent addresses are allocated to different channels and ways. In fact, the efficiency thanks to maximally exploited parallelism for

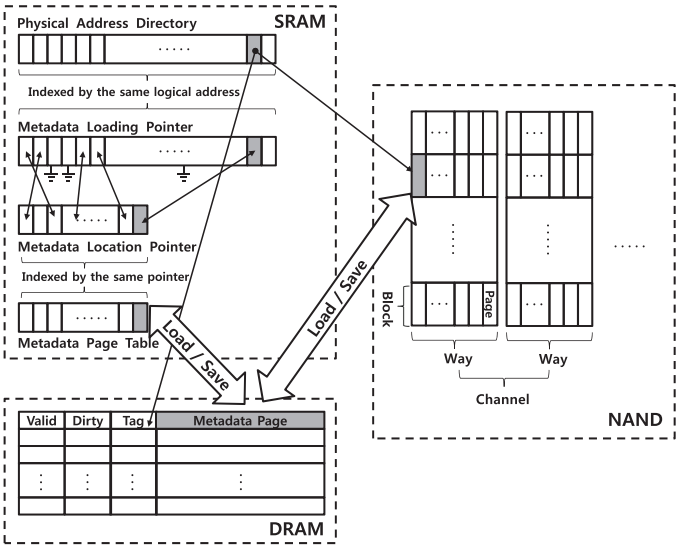

Fig. 2. The Proposed Memory Hierarchy-Aware Metadata Management Structure

user data has been already reported in [7]. By the same mechanism, with FTL metadata statically distributed to channels and ways according to their corresponding logical addresses, the resource conflicts of channels and ways are minimized and the utilization of multi-channel/multi-way is maximized.

2) Directory Structures in On-chip SRAM: The on-chip SRAM is the fastest and most frequently accessed memory for an FTL, therefore, all the top-level directory structures are stored in this memory. For fast searching, all these data structures are directly indexed by logical addresses or pointers derived from logical addresses. Fig. 2 presents the four main data structures residing in SRAM. The first one is *the physical address directory*, whose elements point out the physical addresses of NAND flash pages storing metadata. When metadata not loaded on SRAM are requested or propagation of updates is required, the corresponding physical page is easily determined by this directory, which is indexed by their logical addresses. The second and the third tables are *the metadata loading pointer* and *the metadata location pointer*, whose elements are linked bi-directionally. *The metadata loading pointer* shares the indexing rule with *the physical address directory*, thus the FTL can be easily informed whether the requested metadata are loaded on SRAM or not, only using logical addresses. If they are determined to reside on SRAM, *the metadata location pointer* additionally gives the index of the requested metadata pages using the bi-directional pointer, which also directs *the metadata page table*. After SRAM is fully filled by metadata, eviction of metadata pages should be occurred. *the metadata location pointer* and *the metadata page table* are managed by the least-recently-used (LRU) policy since the most important property determining efficiency is the temporal locality of metadata pages.

3) Cache Buffer Using Large-capacity DRAM: When the requested metadata are not on SRAM, the FTL requests them to the cache buffer controller. At that time, the cache buffer is transparent to the FTL so that metadata to be loaded are requested using *the physical address directory* on SRAM. The cache buffer loads the requested metadata according to the request from SRAM. Also, the updated metadata on SRAM must be propagated to the cache buffer and NAND flash for synchronization with the host system. First, both SRAM and the cache buffer are byte-accessible memories and support in-place update, hance, the propagation between them is easily done. In addition, to reduce the number of write to NAND flash, write-back cache buffer can be used with the battery-backed DRAM. Fig. 2 shows the direct-mapped cache buffer table with write-back policy in DRAM. Each cache line has flags and tags for write-back cache, followed by FTL metadata in units of a page. When using the recent giga-scale DRAM, all the metadata can be loaded on DRAM, which can drastically reduce NAND flash writes with write-back policy. The cache buffer can store not only metadata but also user data to reduce the latency of read request.

4) Metadata Block in NAND Flash: All the metadata must be saved in NAND flash eventually because of the normal power shutdown. It is often performed on demand, periodically, or during idle periods. Therefore, the reserved blocks for metadata should be offered in NAND flash. As shown in Fig. 2, metadata pages on SRAM or on the cache buffer are loaded or saved in the unit of a page to prevent the partial page programming which may incur performance penalties.

IV. Experimental Results

A. Experimental Setup

In order to evaluate the efficiency of the proposed memory hierarchy-aware FTL metadata management technique, we implemented a cycle-accurate level SSD platform using SystemC language, as shown in Fig. 1. The SSD platform mainly consists of a ARM 7 microprocessor, a SATA interface, a bank-interleaved DDR SDRAM controller, and 8-channel 4-way NAND flash memory chips. The total capacity of the SSD is configured as 64GB and the additional 8GB NAND flash are provided for the advantages of the over-provisioning. The size of the external DRAM is 64MB, and that of on-chip SRAM is 32KB. We implemented an FTL which employs on-demand garbage collection by merge operation and wear-leveling using free-block queue on top of BAST [1]. In the FTL, about 24KB of SRAM are used for top-level directory structures, and the remaining 8KB are used for the metadata page table. Also, 8MB in DRAM are reserved for FTL metadata and the

TABLE I
The Traces Used in the Experiments

Name	# of Req.	Description
Crystal Sequential	1000	Crystal sequential benchmark [8]
Crystal Random	1000	Crystal random benchmark [8]
Photoshop	1000	Edit large-size pictures
FTL Download	1000	Downloading with an FTP client
WinXP General	1000	Web browsing, word processing, etc.

remaining 56MB are used for user data. Finally, the I/O traces used in the experiments are summarized in Table I.

B. The Performance of FTL Metadata Management

In order to measure the performance of our method, we explored four memory configurations.

- *IDEAL_SRAM* is a memory configuration with ideally large SRAM keeping the entire metadata not to incur any FTL metadata loading overheads.
- *NO_CACHE* is a memory configuration which includes 32KB SRAM and 64GB NAND flash without DRAM cache buffer.
- *CACHE* is a memory configuration which adds a 64MB DRAM cache to the *NO_CACHE* configuration. We assume write-through policy in this configuration.
- *CACHE+BATTERY* is identical to *CACHE* except that it adopts write-back cache buffer applying a battery-backed DRAM.

Fig. 3 shows the execution cycles of the internal operations with respect to these memory configurations. To appreciate the proposed technique, we analyzed the execution cycles of internal operations of an SSD - SATA transfer, FTL metadata loading, FTL execution, and internal user data transfer. Note that user data are also cached with the cache buffer in all the cases, but there is no interference by user data in measuring the performance of the proposed method.

IDEAL_SRAM and *NO_CACHE* show the oracle case and the worst case execution times. These inform that overheads of FTL metadata management are considerably large in the large-size SSDs, from 5.5% for Photoshop trace to 64.6% for Crystal Sequential trace. In Crystal Sequential case, large I/O accesses are efficiently parallelized to each channel / way so that the execution cycle of internal user data transfer is rather small. As a result, the portion of FTL metadata loading becomes quite large. On the contrary, the execution cycle of FTL metadata loading becomes large in Crystal Random case because of the randomness, though the portion of that is decreased compared to Crystal Sequential case because of non-parallelized small random accesses. By allocating DRAM cache buffer for FTL metadata, the execution cycle caused by FTL metadata loading are decreased by more than half in all cases, except 43.1% down in Photoshop case. Finally, the remaining execution cycle of FTL metadata loading are almost removed again by the battery-backed DRAM with write-back policy. Quantitatively speaking, the portion of FTL metadata loading in the execution cycle is only from 0.8% to 3.8% when *CACHE+BATTERY* configuration is used. Especially, for traces with random characteristics such as FTP Download, WinXP General, and Crystal Random, the battery-backed DRAM is very effective. However, Photoshop trace shows relatively marginal performance gain due to its high data reusability.

V. Conclusion

This paper presents an efficient FTL metadata management technique for SSDs by efficiently exploiting their internal memory hierarchy. In this hierarchy, SRAM is mainly used as a top-level directory for fast address translation and mapping with the consideration of temporal locality. In addition, the DRAM cache buffer reduces the execution cycle of FTL metadata loading by more than half using efficient caching policy. Finally, write-back cache buffer combined with the battery-backed DRAM makes the overheads negligible and reduces NAND flash accesses to further increase performance and lifetime. In the future, the capacity of SSD will be increased steadily, and huge metadata can be searched or updated with small overheads with the proposed technique.

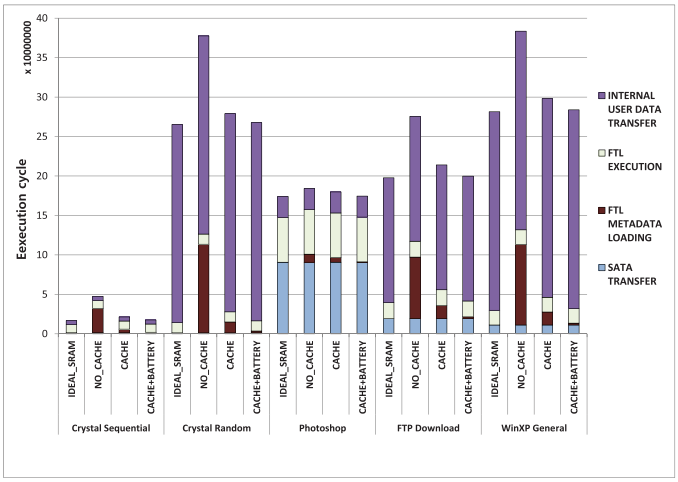

Fig. 3. SSD Internal Execution Cycles Measured from an SSD Platform

Acknowledgment

This work was supported in part by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(2010-0025423), by Samsung Electronics Company, and by IDEC (IC Design Education Center).

References

[1] J. Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. Cho, "A Space-efficient Flash Translation Layer for CompactFlash Systems," Consumer Electronics, IEEE Transactions on, vol.48, no.2, pp.366-375, May 2002.

[2] S.-W. Lee, W.-K. Choi, and D.-J. Park, "FAST : An Efficient Flash Translation Layer for Flash Memory," EUC Workshops 2006, pp.879-887, 2006.

[3] J.-U. Kang, H. Jo, J.-S. Kim, and J. Lee, "A Superblock-based Flash Translation Layer for NAND Flash Memory," EMSOFT06, Oct. 2006.

[4] J.-W. Park, S.-H. Park, G.-H. Park, and S.-D. Kim, "An Integrated Mapping Table for Hybrid FTL with Fault-tolerant Address Cache," Electronics Express, IEICE Transactions on, vol.6, no.7, pp.368-374, Apr. 2009.

[5] A. Gupta Y. Kim and B. Urgaonkar, "DFTL: a Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings," ASPLOS09, Mar. 2009.

[6] Z. Qin, Y. Wang, D. Liu, and Z. Shao, "Demand-Based Block-Level Address Mapping in Large-Scale NAND Flash Storage Systems," CODES+ISSS10, Oct. 2010.

[7] S.-H. Park, S.-H. Ha, K. Bang, and E.-Y. Chung, "Design and Analysis of Flash Translation Layers for Multi-Channel NAND Flash-based Storage Devices," Consumer Electronics, IEEE Transactions on, vol.55, no.3, pp.1392-1400, Aug. 2009.

[8] CrystalDiskMark, http://crystalmark.info/software/CrystalDiskMark/index-e.html