A Hybrid Approach to NAND-Flash-Based Solid-State Disks

Li-Pin Chang

Abstract—Replacing power-hungry disks with NAND-flash-based solid-state disks (SSDs) is a recently emerging trend in flash-memory applications. One important SSD design issue is achieving a good balance between cost, performance, and lifetime. This study introduces a hybrid approach to large SSDs that combines MLC NAND flash and SLC NAND flash. Each of these flash architectures has its own drawbacks and benefits, and this study proposes that the two can complement each other. However, there are technical challenges pertaining to data placement, data migration, and wear leveling in heterogeneous NAND flash. The experimental results of our study show that combining 256 MB SLC flash with 20 GB MLC flash produces a hybrid SSD. This hybrid SSD is 1.8 times faster than a purely MLC-flash-based SSD in terms of average response time and improves energy consumption by 46 percent. The proposed hybrid SSD costs only four percent more than a purely MLC-flash-based SSD. The extra cost of a hybrid SSD is very limited and rewarding.

Index Terms—Flash memory, storage systems, file systems, solid-state disks.

1 Introduction

One of the design challenges of mobile computers is that disk-based mass storage systems are power hungry and fragile to shock. Because flash memory is nonvolatile, power economic, shock resistant, and free from positioning overheads, venders of mobile computers have recently started replacing hard drives with NAND-flash-based solid-state disks (SSDs). An SSD interacts with the host computer (e.g., a laptop) via a standard interface, such as PATA or SATA, and behaves much like a standard hard drive. An SSD comprises a controller and a flash-memory array (usually including 4-16 chips) [3], [19], and the SSD controller firmware performs disk emulation.

Even though SSDs are much better suited to embedded computers, the cost of flash memory still greatly decreases the popularity of SSDs. Ordinarily, each flash-memory cell represents a binary value. This design is referred to as single-level-cell flash (i.e., SLC flash). If multiple voltage thresholds are defined and a memory cell can be reliably charged, then more than one bit can be represented. The design is referred to as multilevel-cell flash (i.e., MLC flash). MLC flash is cheaper and larger sized than SLC flash. However, SLC flash still has an edge over MLC flash in terms of speed, durability, and power consumption. Table 1 compares typical SLC flash and MLC flash specifications [17], [18]: SLC flash is much faster than MLC flash in terms of read and write. As their operating currents are the same, SLC flash requires less energy to carry out reads and writes. The geometry of SLC flash is also finer than that of MLC flash, which means that garbage collection in SLC flash is much easier than in MLC flash. Further, an SLC-flash block is one order of magnitude more durable than an MLC-flash block. Although SLC flash seems better than MLC flash in many aspects, MLC flash is about one third the cost of SLC flash.

This paper proposes a hybrid design for large SSDs. Unlike conventional SSDs, a hybrid SSD comprises a single small SLC-flash chip and a number of large MLC-flash chips. These two types of NAND flash have their own advantages and disadvantages, and combining them allows both types of flash to complement each other. The goal of this hybrid-SSD design is to have a response time as fast as a purely SLC-based SSD, with a hardware cost as low as a purely MLC-based SSD. However, the management of data over heterogeneous NAND flash introduces several new challenges. First, data placement in SLC flash and MLC flash must consider not only the differences between these two types of NAND flash, but also the characteristics of disk access patterns. SLC flash better handles small, hot data, while MLC flash is suitable for storing large, cold data. Second, as SLC flash is relatively expensive, the SLC flash in a hybrid SSD must be small. A strategy is also needed to phase out data from the SLC flash in a timely manner to maximize its accommodation of hot data. Lastly, wear leveling is conducted in flash memory of different endurance limits, so a new strategy is needed to balance the lifespan of heterogeneous flash memory.

The rest of this paper is organized as follows: Section 2 reviews previous study on storage systems that employ heterogeneous memory architectures. Section 3 proposes the idea of hybrid SSDs. Section 4 analyzes hybrid SSDs in terms of performance and overheads. Section 5 presents experimental results, and Section 6 concludes this paper.

2 Related Work

In consideration of cost, power consumption, and form factor, embedded computers are usually equipped with heterogeneous memory such as SRAM, DRAM, or nonvolatile RAM. As a result, the management of heterogeneous memory has

• *The author is with the Department of Computer Science, National Chiao-Tung University, 1001 University Road, Hsinchu, Taiwan 300, R.O.C. E-mail: lpchang@cs.nctu.edu.tw.*

Manuscript received 29 Apr. 2008; revised 22 June 2009; accepted 9 Nov. 2009; published online 4 Jan. 2010.
Recommended for acceptance by J. Antonio.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2008-04-0185.
Digital Object Identifier no. 10.1109/TC.2010.14.

TABLE 1
Comparison between the Specifications of Typical SLC Flash and MLC Flash [17], [18]

Type	SLC	MLC
Product No	Samsung K9F2G08U0A	Samsung K9GAG08B0M
Capacity	256M*8 bits	2G*8 bits
Page Size / Block Size	2K+64 / 128K+4K bytes	4K+128 / 512K+16K bytes
Page Read	25 us	60 us
Page Write	200 us	800 us
Block Erase	1.5 ms	1.5 ms
Block Endurance	100K cycles	5K cycles
Operating Characteristics of Read, Write, and Erase	3.3V / 15mA	3.3V / 15mA
Standby Power Dissipation	3.3V / 10uA	3.3V / 10uA
Cost per GB (early 2008)*	$11.05	$3.22

* From DRAMeXchange®

become an increasingly important issue, and helps address challenges related to data placement and data migration.

Avissar et al. [16] used a compile-time technique to partition data among scratch-pad SRAM, local DRAM, shared DRAM, and ROM. They showed that this technique can significantly improve program runtime. Lee and Chang [9] investigated energy-aware memory allocation and migration over different types of nonvolatile RAM, including battery-backed SDRAM, NOR flash, and NAND flash. Unlike static data placement, upon the arrival of a write, a hybrid SSD must direct the data to either the SLC flash or the MLC flash. Exclusive data allocation is another difference between the SLC flash and a standard cache of the MLC flash. As reported by Wong and Wilkes [29], this exclusivity improves hierarchal storage in terms of space utilization.

The rationale behind this study is closely related to (but fundamentally different from) the disk-caching disks (DCDs) proposed by Hu and Yang [25] and the hybrid log (HyLog) proposed by Wang et al. [22]. Both of these methods attempt to optimize random writes by logging. The idea of a DCD is to separate logging activities from normal data accesses using an extra disk. The HyLog, logs hot data updates out of place, and overwrites cold data in place. However, neither DCD nor HyLog method have to deal with flash-memory management issues, including garbage collection and wear leveling.

Heterogeneous memory management for storage systems has become a popular research topic. In industry, Samsung and Segate exhibited a hybrid-disk prototype combining a hard drive and NAND flash in 2006. Hybrid disks provide the host OS with new commands to help "pin" data in NAND flash. The design accelerates the system boot process because NAND flash reads much faster than hard drives. Related technologies include Microsoft's ReadyBoost and Intel's Turbo Memory. In academia, Kim et al. [24] proposed a file-placement method using NAND flash and a hard drive to reduce energy consumption. Bisson and Brandt [20] and Chen et al. [7] proposed using a USB flash drive as a block-level buffer cache to save energy.

Marsh et al. [2] proposed using NOR flash as a disk-system read cache. However, modern computers have a large DRAM disk cache, a large device-side read cache is of little benefit. Kgil et al. [27] proposed replacing DRAM in the host-side disk buffer cache with NAND flash. This approach utilizes multimode operations, as a memory cell of advanced NAND flash can switch between SLC mode and MLC mode. This study introduces a density-control strategy that services hot data in the SLC mode and cold data in the MLC mode. Comparing to density control, a hybrid SSD can separate hot data from cold and manage them in the SLC and MLC flash, respectively, effectively reducing the management costs of NAND flash.

Researchers generally agree that disk-storage performance is subject to the handling of small files and file-system metadata. Wang et al. [28] proposed a hybrid file system. This type of file system puts small files and metadata in battery-backed RAM, while large files are left to disks. Unlike traditional disk storage, flash memory has no seek penalty, but is subject to garbage collection and limited endurance. Also, beyond the storage-device boundary, an SSD has no explicit intelligence to distinguish metadata and user data. Other studies consider the use of SDRAM in the SSD write buffer. Kim and Ahn [33] and Jo et al. [34] showed that the write buffer prefers committing large, cold write bursts to NAND flash. However, a standard 64-MB SDRAM is nearly five times more expensive than a 1 GB SLC flash as of mid 2009.

Recently, Yoon et al. [10] proposed Chameleon, which is a solid-state disk that combines ferroelectric RAM (FeRAM) with NAND flash. Park et al. [26] introduced PFFS, which is a native flash-memory file system based on a hybrid architecture of phase-change RAM (PcRAM) and NAND flash. Both FeRAM and PcRAM are nonvolatile, byte addressable, and capable of in-place updates. Chameleon maintains mapping tables for disk emulation in FeRAM. PFFS writes file-system metadata and mapping tables to PcRAM. The approach not only reduces the frequency of NAND-flash small writes, but also eliminates the need to rebuild the mapping information at start-up. However, FeRAM and PcRAM are very expensive. A standard 128 K-word FeRAM ($128\ \mathrm{K} \times 16\ \mathrm{bits}$) is about 21 USD as of mid 2009. FeRAM and PcRAM do not suffer from the endurance problem, which is a major concern of using NAND flash. Of course, these different hybrid architectures need not be exclusive. Various memory technologies can be combined if performance is the first priority.

3 Hybrid Solid-State Disks

3.1 Overview

An SSD consists of a microcontroller, a NAND-flash array, and an optional external SDRAM. In typical SSDs, the NAND-flash chips are all of the same type (i.e., either SLC flash or MLC flash). In the rest of this paper, unless explicitly specified, a purely MLC-flash-based SSD is called a *conventional SSD*. The SSD firmware implements a NAND-Flash Translation Layer (i.e., NFTL) to emulate disk geometry. An SSD accepts reads and writes of disk sectors, translates disk geometry into flash-memory addresses, handles garbage collection and wear leveling, and responds to the host upon completing the data transfer. Note that an SSD never buffers data in volatile memory. This guarantees data integrity with the write-through semantic.

Fig. 1 shows the architecture of a hybrid SSD. The design is very similar to that of a conventional SSD, but includes an

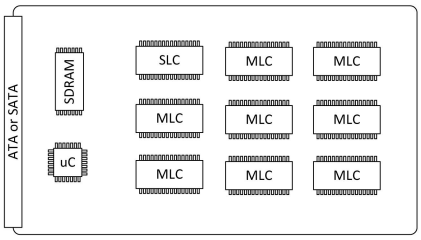

Fig. 1. The architecture of a hybrid SSD, which combines an SLC-flash chip with an MLC-flash array.

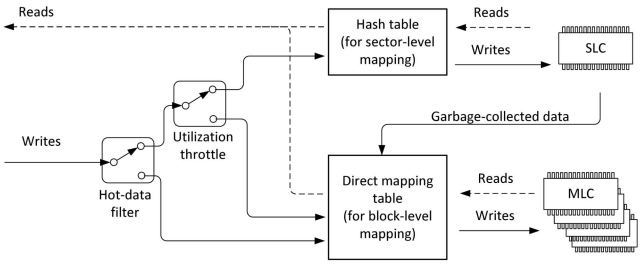

Fig. 2. Data/control flows of the handling of reads and writes inside a hybrid SSD.

extra SLC-flash chip. It is quite easy to create a hybrid SSD from a conventional SSD, as many typical SLC-flash chips and MLC-flash chips share the same pin definition and package dimension (e.g., both [17] and [18] are available in 48-pin TSOP I).

Let a piece of data be referred to as hot if it is frequently updated. Otherwise, it is cold/nonhot. Fig. 2 shows the data flow and control transfer in a hybrid SSD. Upon the arrival of a write request, the write is accepted by the *hot-data filter* if it goes to hot data. Unaccepted writes are dispatched to the MLC flash. A write that successfully passes through the hot-data filter arrives at the *utilization throttle*. As Table 1 shows, since SLC flash and MLC flash have different levels of endurance, the utilization throttle balances the lifespan of SLC flash and MLC flash. Specifically, if the SLC flash is being worn much faster than the MLC flash, then the utilization throttle will reduce the write traffic to the SLC flash. Because the SLC flash is small, cold data should be phased out of the SLC flash to maximize its accommodation of hot data. Hot-data filter and utilization throttle are parts of the firmware of a hybrid SSD.

3.2 The Hot-Data Filter

The hot-data filter determines whether or not the SLC flash should accept a newly arriving write. This decision is made based on not only the characteristics of realistic disk workloads, but also on the differences between the characteristics of SLC flash and MLC flash.

SLC flash has some advantages over MLC flash: 1) SLC flash is much faster than MLC flash in terms of writes, 2) SLC-flash blocks are more durable than MLC-flash blocks, and 3) SLC flash has a finer geometry than MLC flash. The disadvantage of SLC flash is that it is more expensive than MLC flash. A through inspection of real-life disk workloads in this study shows that small writes arrive at very high frequencies, and they repeatedly go to a small amount of hot data. Nonhot data are touched by bulk writes, but at very low frequencies. An even larger amount of data is immutable. The above observations suggest that SLC flash can better handle hot-data writes. Servicing the frequently arriving writes of hot data with fast SLC flash can greatly improve overall performance. In addition, the SLC flash does not need to be large, because hot data have very short life cycles.

This then raises the question of how writes of hot data can be identified. Previous studies on hot-data identification have proposed various solutions, including least recently used (LRU), LRU-k [8], and hash-table-based approaches [11]. However, these methods are complicated and may require large RAM space. Our inspection of real-life workloads shows that the distribution of write-request counts with respect to different request sizes is bimodal. In other words, most of the writes are either very large or very small, and are rarely medium sized. Therefore, this clear-cut distribution suggests a simple method of hot-data identification.

This study proposes a threshold algorithm in Algorithm 1. The algorithm finds the two peak frequencies in the distribution of the write-request counts with respect to different request sizes. The algorithm is based on a k-means clustering algorithm [6]. In Algorithm 1, $c[]$ is an array of counters and $c[i]$ is the accumulated number of writes of 2^i sectors. The largest acceptable write is 1,024 sectors, so the algorithm maintains only 11 counters (i.e., 2^0-2^{10}). For any write whose size is not a power of two, its size is rounded to the nearest power of two to meet algorithm requirements. The algorithm iteratively finds a partition of the counters (i.e., p in Step 4) and then the centroid of each counter group (i.e., i and j in Step 3), subject to minimizing the value function f.[1]

Algorithm 1. The threshold (2-means clustering) algorithm

Require: c[0...n]: n counters that accumulate the total numbers of writes of sizes 2^0-2^n.

Define $f(i,j,p) = \sum_{k=0}^{p} c[i] \cdot |i-p| + \sum_{k=p+1}^{n} c[j] \cdot |j-p|$

1: $i = 0$; $j = 0$; $p = 0$;
2: **repeat**
3: Fix p. Find i and j to minimize $c = f(i,j,p)$;
4: Fix i and j. Find p to minimize $c' = f(i,j,p)$;
5: **until** $(c' == c)$
6: return i; /* i is the threshold of small writes */

The output i is the threshold of small writes. The proposed SSD design periodically invokes the algorithm (e.g., every 1,000 writes) to redefine the threshold. The SLC flash accepts any arriving write that is not larger than the threshold. In our experiments, the threshold remains at 4 KB (i.e., eight sectors) most of the time.

3.3 SLC-Flash Management

Not surprisingly, the management of SLC flash involves address translation, garbage collection, and wear leveling. Address translation maps disk sectors to SLC-flash pages. Garbage collection reclaims SLC-flash pages storing invalid data by erasing blocks. Wear leveling tries to evenly erase

1. K-means clustering is inherently intractable. However, in our case, the search space is very limited. Interested readers are referred to [32] for the complexity analysis of Algorithm 1.

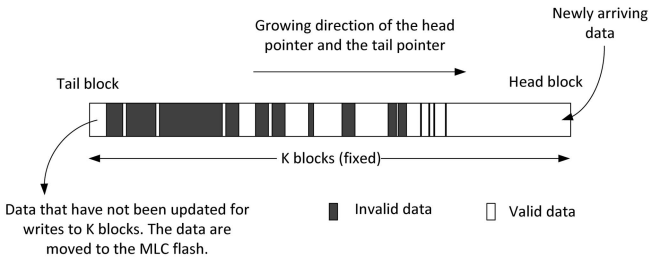

Fig. 3. Newly arriving data are written to the head block, and nonhot data in the tail block are phased out from the SLC flash.

each SLC-flash block. In addition, because the SLC flash is small, any nonhot data should be removed from the SLC flash to maximize its accommodation of hot data.

This study proposes that SLC flash can be treated as a circular log space, as shown in Fig. 3. In the design, blocks in the log space are ordered by their physical-block addresses (PBAs), and the first physical block succeeds the last physical block. A head pointer refers to the head block where newly arriving data are written, and a tail pointer indicates the tail block from which free space is reclaimed. The number of blocks between the two pointers is no greater than a predefined parameter k. Initially, the two pointers refer to the same block. Blocks that are not between the two pointers must be free space. The two pointers move toward large physical-block addresses. If a pointer moves beyond the last physical block, it goes back to the first physical block.

Upon the arrival of a write, the data are written to free space in the head block. The head pointer advances to the next block if there is no free space left. A piece of newly arriving data invalidates its old copies on the SLC flash. In this way, the most recent data always appear close to the head pointer. Whenever the head pointer is ahead of the tail pointer by more than k blocks, the tail block is erased for garbage collection, and then the tail pointer advances. The closer a piece of valid data is to the tail block, the lower is its update frequency. Therefore, before the tail block can be erased, any valid data found in the tail block are not hot and can be phased out from the SLC flash (i.e., copied to the MLC flash). This method has two major benefits: First, because most of the SLC-flash data are hot, valid data are rarely found in the tail block. As a result, garbage collection involves relatively few copy operations. Second, wear leveling across SLC-flash blocks is perfect because blocks are erased sequentially.

In real-life disk workloads, hot data occupy only a small amount of disk sectors. If a direct mapping table is used to translate disk sectors to SLC-flash pages, RAM-space utilization would be very low because the mapping is sparse. Instead, this study proposes a small hash table for address translation. Let the hash table have H_n entries (i.e., buckets). Whenever a write of sector number s arrives, the bucket number is computed by $H(s) = s$ **mod** H_d, in which H_d is a prime number immediately smaller than H_n. Probes are used to resolve hash collision. As soon as a free bucket is located, a pair of disk-sector number and the corresponding SLC-flash block/page number is written to the bucket. Section 5 evaluates different probing schemes and hash-table sizes.

3.4 The Utilization Throttle

In the proposed design, wear leveling in SLC-flash blocks and MLC-flash blocks is separately enforced. In a hybrid SSD, however, not only does the volume of traffic to SLC flash and to MLC flash dynamically changes, but the SLC-flash blocks and MLC-flash blocks have different endurance levels as well (see also Table 1). As a result, it is very possible that the SLC flash may wear out before the MLC flash. This raises the question of how to balance the lifespan of heterogeneous flash memory.

To address this problem, this study proposes a mechanism called the *utilization throttle*. Whenever the SLC flash wears much faster than the MLC flash, the utilization throttle reduces write traffic to the SLC flash. The idea here is to reject "not-so-hot" data, while still accepting very hot data. Let the total numbers of SLC-flash blocks and MLC-flash blocks be n_s and n_m, respectively. Let d_s and d_m be the endurance of an SLC-flash block and an MLC-flash block, respectively. The erasure-cycle count of an SLC-flash block b_i^s is $e^{b_i^s}$, and $e^{b_j^m}$ is accordingly defined for an MLC-flash block b_j^m. The relative wearing rw_s of the SLC flash and rw_m of the MLC flash are defined as

$$rw_s = \frac{\sum_{i=0}^{n_s} e^{b_i^s}}{n_s} \Big/ \frac{d_s}{d_m}, \text{ and } rw_m = \frac{\sum_{j=0}^{n_m} e^{b_j^m}}{n_m} \Big/ 1,$$

respectively.

Whenever $rw_s \geq rw_m$ is true, the utilization throttle is activated. Two strategies are adopted. First, it starts rejecting writes of data that are not already in the SLC flash. If a piece of data is very hot, then it repeatedly appears close to the head block and will not be phased out. Thus, the SLC flash still accepts writes to very hot data. If $rw_s \geq rw_m$ remains true for a period of time, the utilization throttle starts decreasing parameter k. As k decreases, the tail pointer quickly catches up with a piece of data. In this case, garbage collection phase out all data that is not updated at high frequencies, and thus the rejection of writes becomes aggressive. Still, writes of very hot data are not affected because they are rarely phased out. Once the wearing of the SLC flash is properly controlled (i.e., $rw_s < rw_m$), the utilization throttle becomes inactive and parameter k is gradually enlarged.

When the SLC flash starts throttling, it only accepts updates to existing data. However, the host working set may change from time to time. Therefore, the throttling algorithm must be improved so that it accepts new hot data: After rejecting a nonupdate write request, the written sectors are inserted in the hash table of the SLC flash as *virtual sectors*. The rejected request is actually serviced by the MLC flash, but it leaves dummy mapping information in the hash table. Fig. 4 illustrates the use of virtual sectors. Dummy mapping information associates a virtual sector with the physical location of the regular sector that is written immediately before the virtual sector. Virtual-sector information is removed from the hash table when the block it is associated with is erased during garbage collection. If a virtual sector is written again before it retires from the hash table, it is officially accepted as a regular sector.

Algorithm 2 shows the procedure of the proposed throttling algorithm. In Step 2, if the written sectors are

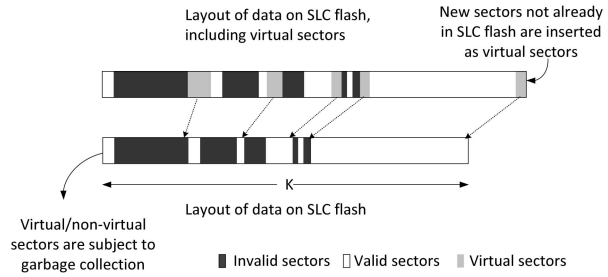

Fig. 4. Using virtual sectors in the throttling mode. A virtual sector appears only in the hash table of the SLC flash.

found in the hash table, then the incoming write request is accepted. If the sectors are absent from the hash table, they are inserted to the hash table as virtual sectors in Step 5, and are then rejected in Step 11. Steps 7-10 decrease parameter k at a frequency of every 1,000 requests (Step 7) and a step of 100 (Step 8). Whenever the utilization throttle becomes inactive, parameter k is enlarged at the same frequency and step.

Algorithm 2. The algorithm for utilization throttling
Require: W, D: write request W goes to data D;
c: a static counter, initially 0.
k: the number of blocks between the head block and the tail block.
Ensure: $rw_s \geq rw_m$ {Throttling is now activated}
1: $c \leftarrow c + 1$;
2: **if** (D is found in the SLC-flash hash table) **then**
3: return ACCEPT;
4: **else**
5: Insert virtual sectors of D to the hash table;
6: **end if**
7: **if** ($c \geq 1{,}000$) **then**
8: $k \leftarrow k - 100$; $c \leftarrow 0$;
9: {Decrease k for aggressive rejection of writes}
10: **end if**
11: return REJECT;

3.5 MLC-Flash Management

In a hybrid SSD, we propose managing the MLC flash as if the SLC flash were not there. As in a conventional SSD, the MLC flash in a hybrid SSD is responsible for disk emulation. All the disk sectors, except those accepted by SLC flash, are mapped to MLC-flash pages. However, SSD firmware cannot afford the RAM-space overhead for sector-level mapping in MLC flash. Instead, disk emulation for the MLC flash adopts a two-level mapping scheme. Let a *physical block* refer to a flash-memory block, which includes a number of pages. Partition all the disk sectors into *logical blocks*, each of which is as large as a physical block. Label these logical blocks and physical blocks as logical-block addresses (LBAs) and PBAs, respectively. For ease of presentation, let a disk sector be as large as an MLC-flash page.

During first-level mapping, logical blocks are mapped to physical blocks in a one-to-one fashion, as Fig. 5a indicates. A physical block is referred to as a *data block* if it is mapped to a logical block. The sectors of a logical block are sequentially mapped to pages of the corresponding data block. A *spare block*, which is a physical block entirely of free space, is allocated on the first sector write to a logical block because

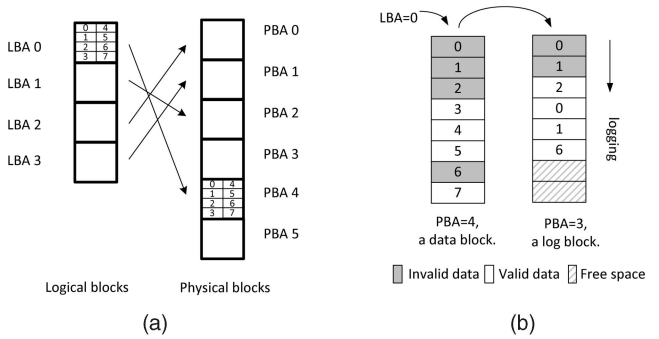

Fig. 5. (a) The mapping of LBAs to PBAs. PBAs 0, 1, 2, and 4 are data blocks. (b) LBA 0 is mapped to PBA 4, and sector updates are carried out in a log block at PBA 3. The numbers in the boxes are disk-sector addresses.

in-place updating is prohibited. The new data are written to the first page of the spare block before the old copy of the sector data is invalidated. Subsequent sector-write data are appended to the free space of the spare block. This block is called a *log block* of the logical block because it logs updates. Fig. 5b shows that LBA 0 is mapped to PBA 4, and updates to LBA 0 take place in a log block at PBA 3. If the log block runs out of free space, another log block is created. A *block chain* of a logical block refers to an ordered list of one data block and a number of log blocks.

Second-level mapping maps the disk sectors of a logical block to MLC-flash pages in the corresponding block chain. A small cache of second-level mapping information can be used to take advantage of localities in access patterns. Whenever a logical block is accessed, the pages of its data block and log blocks are scanned to establish the mapping of the logical block's sectors to pages in the logical block's block chain. This mapping information is then cached so that the pages do not need to be scanned again on subsequent accesses to the same logical block. The mapping cache is a performance optimization option, and is not mandatory.

This management scheme uses two RAM-resident tables. Fig. 6a shows a scenario in which there are four logical blocks and nine physical blocks. The L2P table (i.e., logical-to-physical table), indexed by LBAs, maps logical blocks to data blocks. The LB table (i.e., log-block table), indexed by PBAs, maintains block chains. For example, the L2P table in Fig. 6b shows that a logical block at LBA 1 is mapped to a data block at PBA 4, which is followed by two log blocks at

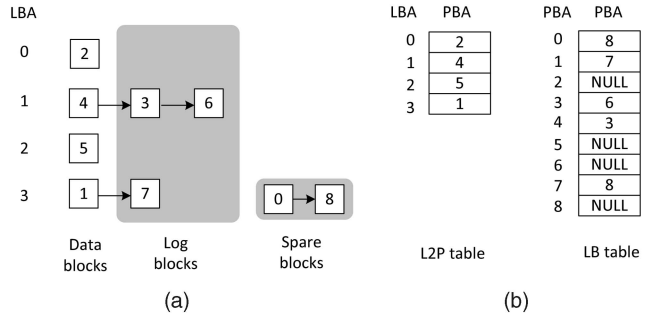

Fig. 6. The L2P table and the LB table: (a) Block chains of logical blocks, and (b) the corresponding L2P table and LB table.

PBA 3 and PBA 6. The block chain ends with NULL in the LB table. Note that the mapping information in the SLC flash's hash table precedes that in the L2P table and LB table. This means that if a piece of data previously stored in the SLC flash is written to the MLC flash, all the mapping information related to the data must first be removed from the hash table.

As writes continue to arrive, the number of spare blocks decreases. This necessitates garbage collection, which is carried out by means of *block-chain folding*. To fold the block chain of a logical block, a spare block must first be allocated. All the valid sector data are then collected from pages in the block chain and copied to the spare block. The spare block serves as a data block after the copying. Again, sectors in the data block must be sequential in terms of sector addresses. Some techniques can be used to reduce garbage-collection overhead and optimize performance. Specifically, if a log block is full and all the sector data appear in sequential order, the log block can be switched to a data block without any data copying. Reclaiming log blocks with low space utilization for a long period of time is also a priority.

This management scheme for MLC flash is standard and inherits many key ideas from existing NFTLs [4], [12], [21], [30], [31]. The goal of a hybrid SSD is to provide a low-cost incremental improvement over conventional SSDs. We believe that this layered approach is preferable from the software-engineering point of view.

3.6 Write-Back Strategies

As shown in prior sections, data on the SLC flash and the MLC flash are separately managed. This section discusses how joint strategies can be taken to further improve system performance.

Before erasing the tail block of the SLC flash for garbage collection, all valid data in the tail block must be moved to the MLC flash. This study proposes a new write-back policy for the SLC flash: Before moving a valid sector from the SLC flash to the MLC flash, the MLC flash checks whether or not the logical block related to the sector has been allocated to any log blocks. If so (e.g., LBA 1 in Fig. 6), then the sector is written to the MLC flash. Otherwise (e.g., LBA 0 in Fig. 6), the sector is copied back to the head block of the SLC flash.

This new policy reduces garbage-collection overhead on the MLC-flash side. Because MLC flash adopts block-level mapping, garbage collection can become very inefficient for random write patterns. This new policy delays writing a piece of data to the MLC flash if the write increases the randomness of the write pattern. However, this new policy accumulates valid data in the SLC flash, reducing the SLC flash's accommodation of hot data.

This study also proposes a complementary policy for the MLC-flash side: Whenever an MLC-flash block chain is being folded for garbage collection, valid data are collected from not only the log blocks of the chain but also from the SLC flash. This policy opportunistically removes valid data from the SLC flash. Sectors related to the removed data on the SLC flash are then marked as virtual, as they are forcibly removed.

A possible problem with this approach is that some data cannot retire from the SLC flash because the logical block of the MLC flash related to the data has not been allocated to any log blocks. To deal with this problem, the write-back policy on the SLC-flash side monitors the fraction of write traffic contributed by update. Whenever the ratio notably decreases, either the SLC flash has accumulated too many valid data, or the temporal localities in workloads have changed. In these situations, the write-back policy of the SLC flash is deactivated until the update traffic returns to a stable level.

4 System Analysis

4.1 Performance Analysis

Before presenting experimental results, this study analytically reviews how a hybrid SSD improves upon a conventional SSD in terms of response.

For ease of explanation, assume that a disk sector, SLC-flash page, and MLC-flash pages are the same size. Let the MLC flash take t_m^r, t_m^w, and t_m^e units of time to read a page, write a page, and erase a block, respectively. Let the SLC flash take t_s^r, t_s^w, and t_s^e units of time to read a page, write a page, and erase a block, respectively. Let there be g_s pages and g_m pages in an SLC-flash block and an MLC-flash block, respectively. The following analysis is concerned with write costs only. That is because writes are slow and may involve expensive garbage collection.

Consider a write which is one page large. If the write is directed to the SLC flash, then the basic expense is t_s^w units of time. Now consider garbage collection for the SLC flash. Let n_1 be the average number of pages found valid in the tail block. Out of n_1 valid pages, an average of n_2 pages are copied to the head block, and $(n_1 - n_2)$ pages are written to the MLC flash. In other words, one SLC-flash block is erased for every other $(g_s - n_2)$ page writes. The expected write cost is

$$\left(n_1 t_s^r + n_2 t_s^w + t_s^e\right)/(g_s - n_2) + t_s^w$$

units of time.

If the write arrives at the MLC flash, then the write costs involve 1) writing one page, 2) maintaining the sector-mapping information, and 3) reclaiming some free space. One page write costs the MLC flash t_m^w units of time. As mentioned in Section 3.5, if the write goes to an LBA which does not have cached mapping information, then the LBA's block chain must be scanned again. Let n_3 be the average number of pages scanned on a cache miss. The average overhead for scanning a block chain is $O_1 = t_m^r n_3$.[2] Next, consider the garbage-collection overhead. Let n_4 be the average number of blocks erased by folding a block chain. The overhead of folding one block chain is $O_2 = g_m(t_m^r + t_m^w) + n_4 t_m^e$. Let p_1 be the miss ratio of the mapping-information cache, and let p_2 be the probability that a page write will trigger garbage collection. The estimated write cost is thus

$$p_1 O_1 + p_2 O_2 + t_m^w$$

units of time.

Let α be the ratio of the amount of user data written to the SLC flash to the amount of all user data written to a

2. According to our measurement, reading only the spare area of a page requires nearly the same amount of time as reading the entire page. The result has also been reported in [12].

hybrid SSD. The expected response of writing one page to a hybrid SSD is then

$$\alpha\big(\big(n_1 t_s^r + n_2 t_s^w + t_s^e\big)/(g_s - n_2) + t_s^w\big) + (1-\alpha)\big(p_1 O_1 + p_2 O_2 + t_m^w\big).$$

The above analysis shows that a hybrid SSD improves upon a conventional SSD by means of 1) switching slow MLC-flash operations to fast SLC-flash operations and 2) reducing the management overheads of the MLC flash, i.e., O_1 and O_2. Of course, how well a hybrid SSD performs largely depends on n_1, n_2, n_3, n_4, p_1, and p_2, which are determined by workload characteristics.

4.2 RAM-Space Requirements

This section shows the RAM-space requirements of the essential data structures for managing conventional SSDs and hybrid SSDs.

Consider the following configuration: A conventional SSD reports itself as a 20 GB block device, and it has a 21 GB MLC-flash array. Let one MLC-flash block be 512 KB. The 20-GB disk space has 40,960 512-KB logical blocks, and the MLC flash has 43,008 physical blocks. Thus, there are $43{,}008 - 40{,}960 = 2{,}048$ spare blocks. Logical blocks and physical blocks are both addressed by 2 bytes. Based on the same configuration, a hybrid SSD adds an extra 128-MB SLC flash to the 21-GB MLC-flash array. Let the SLC-flash page size be 2K bytes. The SLC flash has 65,536 pages.

The RAM-space requirements of a conventional SSD are determined by an L2P table and an LB table, as Section 3.5 indicates. The L2P table has 40,960 entries for all the LBAs. The LB table hash 43,008 entries for all PBAs. Each entry in these two tables is a 2-byte PBA. Therefore, the two tables need $(40{,}960 + 43{,}008) \times 2 = 164$ KB.

A hybrid SSD requires the same data structures for MLC-flash management. Unlike a conventional SSD, the SLC flash requires a hash table. The hash table is for sector-level address translation, as mentioned in Section 3.3. Because hot data quickly leave many invalid data in the SLC flash, only a small number of SLC-flash pages are mapped to valid sector data and the hash table does not need to be large. Let the hash table has 32,768 entries, which is half the total number of the SLC-flash pages. For ease of explanation, let a disk sector be as large as an SLC-flash page. A hash-table entry stores a disk-sector number and an SLC-flash page number as a (key, value) pair. The key needs 24 bits to address all the sectors of a 20-GB block device, and 16 bits to address 65,536 2-KB pages. The hash table requires $32{,}768 \times (24 + 16)/8 = 160$ KB of RAM space.

Compared to a conventional SSD, a hybrid SSD requires only an extra hash table for SLC-flash management. The experimental results below show that this investment is very rewarding in terms of performance improvement.

5 Experimental Results

5.1 Experimental Setup and Performance Metrics

A simulator was built to conduct experiments on the proposed hybrid SSDs and conventional SSDs. As SSDs present themselves as logical disks, our experimental workloads are disk traces of read and write. Disk traces were collected from the daily use of a UMPC (Ultra Mobile PC) ASUS R2H. This UMPC is equipped with a Celeron-M ULV processor, 768 MB of RAM, and a 20 GB disk. We installed Windows XP and Linux on the UMPC to collect different types of disk workloads. The disk was formatted in NTFS under Windows XP and ext3 under Linux. Applications run on the UMPC include web browsers, e-mail clients, movie players, FTP clients, office suites, and games. Trace collection continued for one month for each operating system. For each run of our experiments, workload traces were replayed 10 times on our simulator to accumulate more erasure-cycle counts. The replay provides no artificial benefits to a hybrid SSD, because any workload can entirely write the SLC flash more than twice under any experimental setting.

TABLE 2
Three Experimental Configurations of Hybrid SSDs

		Geometry			Timing characteristics		
		page size	block size	block endurance	page read	page write	block erase
C1	SLC	2 KB	128 KB	100 K	25 μs	200 μs	1.5 ms
	MLC	2 KB	128 KB	10 K	60 μs	800 μs	1.5 ms
C2	SLC	2 KB	128 KB	100 K	25 μs	200 μs	1.5 ms
	MLC	4 KB	512 KB	10 K	60 μs	800 μs	1.5 ms
C3	SLC	2 KB	128 KB	100 K	25 μs	200 μs	1.5 ms
	MLC	4 KB	512 KB	5 K	60 μs	800 μs	1.5 ms

Table 2 show the three different configurations of hybrid SSDs considered in our experiments. Basically, the specifications of the SLC flash and the MLC flash follow those of real products [17], [18]. The total size of the MLC flash in the experiments remains at 21 GB, so the space utilization of the MLC flash is never higher than $20/21 = 95$ percent. The size of the SLC flash in a hybrid SSD is an independent variable. Configuration C_1 evaluates a scenario in which the SLC flash and the MLC flash have the same geometry. In configuration C_2, the blocks and pages of the MLC flash are larger. In configuration C_3, the endurance of MLC-flash blocks is lower. These experiments do not consider parallelism among the NAND-flash chips. Before each run of experiments, the MLC flash maps every logical block to a physical block. Unless explicitly specified, the default settings are as follows: The total number of the hash-table entries for the SLC flash is half the total number of the SLC-flash pages. The MLC flash adopts the dual-pool algorithm for wear leveling [13] with parameter $TH = 16$.

Dynamic density control proposed by Kgil et al. [27] is evaluated for performance comparison. This method assigns a frequently accessed LBA to the SLC mode. An LBA in SLC mode requires double MLC-flash capacity. When an SLC-mode LBA is accessed infrequently, the LBA can be switched back to MLC mode to increase free capacity.

The experiments in this study also consider various metrics of performance improvement, flash-memory lifetime, and hardware costs. Performance improvement is evaluated in terms response: Let there be total n requests (reads and writes) in an experimental workload. Let a conventional SSD and a hybrid SSD respond to the ith request in r_i^s and r_i^h units of time, respectively. The average response of a hybrid SSD can be denoted as

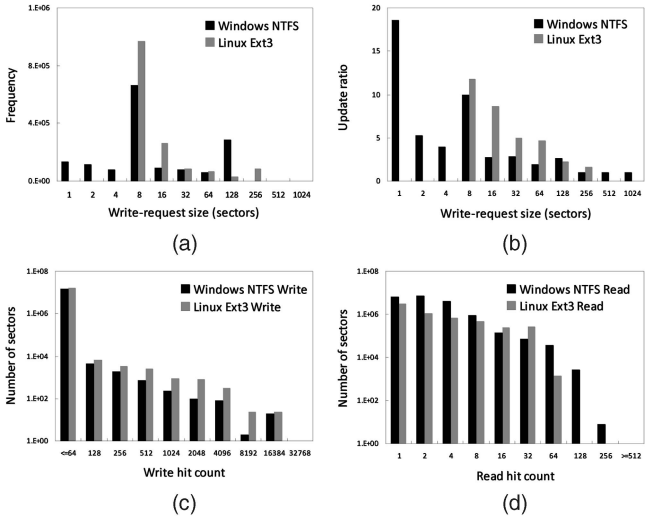

Fig. 7. Profiles of disk workloads collected from Windows XP and Linux operating systems. (a) Write counts with respect to different request sizes, (b) update ratios of writes respect to different request sizes, (c) sector counts with respect to different write hit counts, and (d) sector counts with respect to different read hit counts. Scales of Y-axes of (c) and (d) are logarithmic.

$(\sum_{i=1}^{n} r_i^h)/n$, and that of a conventional SSD as $(\sum_{i=1}^{n} r_i^s)/n$. The response-speedup ratio (i.e., **RS ratio**) is defined as

$$\frac{\sum_{i=1}^{n} r_i^s}{\sum_{i=1}^{n} r_i^h}.$$

Note that the common factor n has been reduced. The larger the RS ratio is, the better a hybrid SSD performs.

The flash-memory lifetime issue is considered in the following terms: 1) The evenness of the wearing of blocks of the same type, and 2) the ratio of the average erasure-cycle count of SLC-flash blocks to that of the MLC-flash blocks. The second metric is referred to as block-wearing ratios (**BW** ratios). If the SLC flash is well protected, the BW ratios will be no higher than the ratio of the block endurance of SLC flash to that of MLC flash. As for hardware costs, this study adopts energy-saving ratios (**ES** ratios) and extra-cost ratios (**EC** ratios). The ES ratio represents the ratio of the total energy consumed by a hybrid SSD to that consumed by a conventional SSD. The **EC** ratio denotes the ratio of the total cost of flash-memory chips of a hybrid SSD to that of a conventional SSD. Table 1 includes the costs and the energy models for both SLC flash and MLC flash.

5.2 Profiles of Real-Life Disk Workloads

This section provides profiles of the experimental workloads. This information helps reveal the rationale behind the proposed hybrid SSD design and the experimental results.

This section first focuses on writes because they concern flash-memory management most. Fig. 7a shows the distributions of write counts with respect to different request sizes. The X-axis refers to the request size in terms of disk sectors, and the Y-axis represents the total number of write requests. The distribution of NTFS is bimodal. In most cases, the disk is accessed by small writes of eight sectors. The second largest part includes writes of 128 sectors. The histogram of the ext3 workload looks very similar to that of the NTFS workload. However, ext3 generates more small writes of eight sectors than NTFS. It also generates large writes of 256 sectors.

The relationship between the sizes of write requests and data attributes (i.e., hot or nonhot) may be an interesting point. Let *update ratio* be the ratio of the total number of sectors written to the total number of distinct sectors ever written. If this ratio is related to a request size, then only writes of that size are counted. Fig. 7b shows that, under NTFS, small writes have very high update ratios, while large writes are of low ratios. The update ratios of ext3 are similar to those of NTFS.

Next, this section investigates how many sectors are updated at high frequency. Let the write *hit count* be the number of times a disk sector is written in the entire workload. Fig. 7c shows the sector counts with respect to different write hit counts. Note that the scale of the Y-axis is logarithmic. In both the workloads of NTFS and ext3, a small amount of sectors are updated no less than 16,384 times. In summary, small writes repeatedly go to a small collection of disk sectors, while large writes barely touch a piece of data multiple times. Therefore, if a write is small, it may very well go to hot data. These observations are consistent with those reported in [5], [23], [28], [30].

Readers may question why the disk still receives so many writes of hot data, as these should have been absorbed by the host OS's write buffer. However, closer inspection shows that writes of hot data are contributed by not only file systems, but user applications as well. In our disk traces, writes to file-system metadata frequently arrive at the disk storage. In particular, the Master File Table of NT file system and cluster-group headers of ext3 file system are updated at high frequencies. Even for read-only operations, the time stamp of a file is updated every time the file is opened. Additionally, NTFS and ext3 are journal file systems. For fast crash recovery, the on-disk journal must be updated before dirty blocks are committed onto the disk [15]. As for user applications, many of them maintain their own index structures for fast data retrieval. For example, a web browser maintains its own webpage cache, and it frequently flushes the index file onto the disk to keep it up-to-date.

Unlike writes, strong temporal localities do not occur among reads. Fig. 7d shows sector counts with respect to different read hit counts. Over a one-month time-span, no sectors are read more than 256 times under NTFS, and 64 times under ext3. This is because frequent reads of a sector are absorbed by the disk cache in the host's operating system. Because read shows no noticeable temporal localities, whether or not the SLC flash can efficiently handle, writes of hot data is the key to the success of hybrid SSDs.

5.3 Performance Improvement of Hybrid SSDs over Conventional SSDs

This section examines the performance improvement of a hybrid SSD over a conventional SSD. Diverse conditions are considered, including different SLC-flash sizes, different SLC/MLC configurations, and different disk workloads.

5.3.1 Size of SLC Flash

This section evaluates the RS ratios that can be achieved by a hybrid SSD. The SLC flash of a hybrid SSD varies from 32 to 512 MB. This study also evaluates dynamic density control [27] for comparison. Because the density of MLC flash is

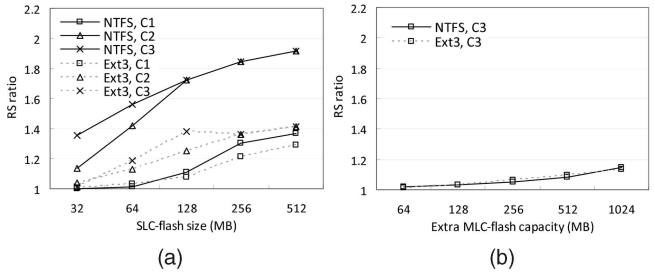

Fig. 8. Response-speedup ratios (RS ratios) achieved by (a) the proposed hybrid SSD and (b) dynamic density control [27]. See Table 2 for definition of configurations C_1, C_2, and C_3. There is no SLC flash in (b).

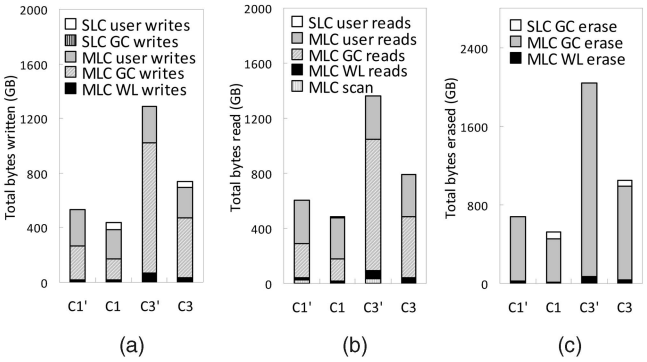

Fig. 9. Under the NTFS workload (a) write traffic, (b) read traffic, and (c) erasure overheads of configurations C_1 and C_3. "GC," "WL," and "scan" refer to garbage collection, wear leveling, and block-chain scan, respectively.

twice as high as that of SLC flash, the extra capacity of MLC flash adopted by dynamic density control is between 64 and 1,024 MB for fair comparison. Different configurations (i.e., C_1, C_2, and C_3) and different disk workloads (i.e., NTFS and ext3) are considered. Figs. 8a and 8b show the results of a hybrid SSD and the density-control method, respectively. The Y-axes of the two figures stand for the RS ratios.

In Fig. 8a, the RS ratio appears to steadily increase as the SLC flash increases in size. The improvement saturates when the SLC flash reaches 256 MB for two reasons: First, when the SLC flash is large enough, a piece of hot data can rarely be phased out from the SLC flash. Second, large SLC flash offers plenty room to collect valid sectors and then writes them back to the MLC flash with proper timing (see Section 3.6).

When the SLC flash is small, and smaller than 128 MB in particular, performance improvement is limited for a few reasons: First, many hot data are forcibly phased out from the SLC flash before they are updated again. Copying hot data from the SLC flash to the MLC flash is of little benefit and may significantly increase response time. Second, to maintain reasonable update traffic, the SLC flash is eager to phase out nonhot data. The MLC flash then suffers from random write patterns. Third, as many erasure-cycle counts are accumulated on a small number of SLC-flash blocks, the utilization throttle will start rejecting writes to protect the SLC flash. Overall, 256 MB is a good size for SLC flash, as the benefits diminish beyond this size.

Fig. 8b shows the RS ratios obtained using dynamic density control. These results are not as good as those of a hybrid SSD. Density control may be beneficial to disk cache in the host operating system due to read temporal localities. However, there are no such localities on the storage-device side, as they have been eliminated by the host-side disk cache. Thus, read response is not much improved. A major fraction of write response is contributed by garbage collection, but density control has no means to deal with it. Density control also has large space requirements: With 1,024 MB of extra MLC-flash capacity, the RS ratio in Fig. 8b is smaller than the ideal speedups (i.e., 1.2 times on read and two times on write, as Table 1 indicates).

5.3.2 SLC/MLC Configurations

This section investigates how the performance of a hybrid SSD can be affected by different SLC/MLC configurations. Only results under the NTFS workload are considered. Fig. 9 assists with this discussion, as it shows how user data, garbage collection, and wear leveling contribute overheads of read, write, and erasure. The SLC flash is 256 MB in configurations C_1 and C_3, while configurations C_1' and C_3' have no SLC flash. Fig. 9 does not include configuration C_2 because it behaves nearly the same as Configuration C_3 when the SLC flash is 256 MB.

First, consider configurations C_1 and C_1'. As a hybrid SSD directs small writes to the SLC flash, Fig. 9a shows that in configuration C_1, the SLC flash absorbs a portion of write traffic. Even though small writes do not make a major contribution to write traffic, since SLC flash writes four times faster than MLC flash, configuration C_1 still achieves an RS ratio of 1.3. Additionally, of all the 1,534,627 write requests, 68 percent are handled by the SLC flash. This implies that the user experience can be largely improved, as the host issues small writes to a disk most of the time.

Fig. 8a shows that the RS ratio achieved by configuration C_3 is 1.84, which is much higher than that of configuration C_1. Comparing configurations C_3 to C_3' in Fig. 9 shows that the improvement is due to a significant reduction in garbage-collection overhead of the MLC flash. Configurations C_3 and C_3' have a large MLC-flash block size, and thus the total number of spare blocks on the MLC flash is small. With keen competition for spare blocks, a block chain might be forcibly folded before the free space in the last log block of the chain is fully utilized. As free space is wasted in this way, garbage collection becomes very inefficient. The write-back policy of the SLC flash addresses this problem: Until a logical block of the MLC flash is accessed by a large write, the SLC flash keeps valid data written by small requests related to the logical block. This delays the allocation of log blocks to the logical block, which greatly alleviates the competition for spare blocks.

5.3.3 Effects of Workloads

Fig. 8a shows that the RS ratios of a hybrid SSD improve consistently under different disk workloads. However, readers may notice that a hybrid SSD performs better under the NTFS workload than under the ext3 workload. The reason for this remains unclear, as in Section 5.2 the two workloads' profiles are shown similar. The following

TABLE 3
Performance Factors of Configurations C_3 and C'_3 Measured under Workloads of NTFS and ext3

		α	n_1	n_2	n_3	n_4	p_1	p_2
C'_3	NTFS	0	—	—	48.4	2.0	0.034	0.028
C_3	NTFS	0.17	12.58	10.29	44.08	1.92	0.011	0.017
C'_3	ext3	0	—	—	81.5	1.97	0.012	0.011
C_3	ext3	0.23	10.38	8.63	82.18	1.92	0.005	0.009

discussion uses the analytical model presented in Section 4.1 for performance characterization.

This analytical model is first validated by comparing it to the results reported in Fig. 8a. Table 3 shows performance factors n_1, n_2, n_3, n_4, p_1, and p_2 measured in experiments. Different configurations (i.e., C_3 and C'_3) and different workloads (i.e., NTFS and ext3) are also considered. The SLC flash of configuration C_3 is 256 MB, and there is no SLC flash in configuration C'_3. The estimated response of one page write for configuration C_3 under the NTFS workload is

$$w_{c_3} = 0.17 \times \left(\left(12.58t_s^r + 10.29t_s^w + t_s^e\right)/(g_s - 10.29) + t_s^w\right) + 0.83 \times \left(0.011\left(44.08t_m^r\right) + 0.017\left(g_m\left(t_m^r + t_m^w\right) + 1.92t_m^e\right) + t_m^w\right).$$

Because configuration C'_3 has no SLC flash, its estimated response under the same workload is

$$w_{c'_3} = 0.034\left(48.4t_m^r\right) + 0.028\left(g_m\left(t_m^r + t_m^w\right) + 2.0t_m^e\right) + t_m^w.$$

Using timing parameters in Table 1, we have $w_{c'_3}/w_{c_3} = 1.76$. Based on the same calculation, we have $w_{c'_3}/w_{c_3} = 1.4$ for the ext3 workload. The above calculated ratios 1.76 and 1.4 are very close to their counterparts 1.8 and 1.38 in Fig. 8a, respectively. This validates the performance analysis above.

The performance factors in Table 3 show two major differences between the NTFS workload and the ext3 workload. First, ext3 generates more small writes than NTFS, as the α of configuration C_3 is 0.23 under ext3 and is 0.17 under NTFS. Fig. 7a also reports this result. Second, when SLC flash is not in use, garbage collection is more intensive under NTFS ($p_2 = 0.028$) than it is under ext3 ($p_2 = 0.011$). However, using SLC flash greatly improves p_2 from 0.028 to 0.017 under NTFS, which is much better than under ext3 (from 0.011 to 0.009). Apparently, the NTFS workload has some behaviors that the ext3 workload does not have.

Fig. 10 shows write requests arriving at the MLC flash in a hybrid SSD. Figs. 10a and 10b show the results of considering an arbitrary trace fragment extracted from the NTFS workload. The results of Fig. 10a are for configuration C'_3, while Fig. 10b is for configuration C_3. Figs. 10c and 10d show their counterparts under the ext3 workload. Both of these trace fragments generate nearly 3.5 GB of write traffic.

Notice that in Fig. 10a, writes are widely spread over the entire disk space. The NTFS traces are collected from a one-year old Windows XP installation. Not only are small files distributed throughout the disk space, but large files are fragmented into pieces as well. This widespread write pattern is at odds with the block-level mapping scheme of the MLC flash. Fig. 10b shows that write-pattern randomness is largely weakened by adopting the SLC flash. Under

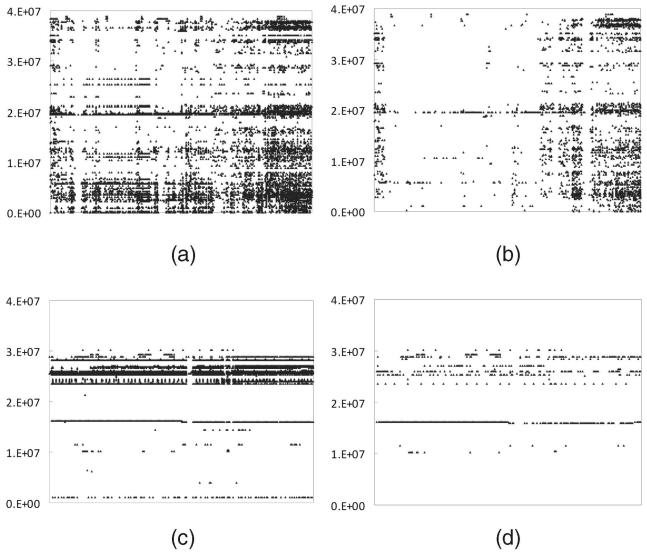

Fig. 10. Write traffic to the MLC flash in a hybrid SSD under different workloads. X-axes are sequence of writes, and Y-axes are disk-sector numbers. (a) C'_3, NTFS; (b) C_3, NTFS; (c) C'_3, ext3; (d) C_3, ext3. The SLC-flash size is 0 MB in (a) and (c), and 256 MB in (b) and (d).

the ext3 workload, writes appear in a region of disk space, as Fig. 10c shows. This is because the ext3 traces are collected from a brand-new Linux installation, which utilizes only 2.2 GB of the 20 GB disk capacity. We believe that the write distribution of ext3 will become similar to that of NTFS after a long period of use.

5.4 Hash-Table Designs for the SLC Flash

The SLC flash requires a hash table for address translation. This section evaluates different hash schemes and hash-table sizes.

This part of the experiment is based on configuration C_3 with a 256-MB SLC flash. The SLC flash has 131,072 2-KB pages. The sector numbers of data are used as hash keys, and the hash function seeks to find the remainder of dividing the hash key by the largest prime number immediately smaller than the hash-table size. If there is a collision, linear probing or quadratic probing can be used. The hash table size varies from 16,384 to 65,536 table entries. To limit the time spent on probing, only a limited number of probes can be taken. This number varies from 1 (never resolving collision) to 32. Performance of a hash scheme is evaluated in terms of 1) the total number of writes rejected due to unresolved collisions and 2) the RS ratio of the hybrid SSD. Fig. 11 shows these results. The lines in this figure are labeled with the probing method and the hash-table size. For example, "2^{16}, Linear" corresponds to a hash table using linear probing over 65,536 entries.

The first part of our observation is on the hash-table size. When the hash table has only 16,384 entries, a lot of writes are rejected and the RS ratios are low. The hash table rejects more writes under ext3 than under NTFS. This is because the ext3 workload consists of more small writes than the NTFS workload, as Fig. 7a indicates. When the hash-table size is increased to 32,768 entries, the number of rejected writes decreases significantly and the response improves greatly. This improvement begins to level off when the

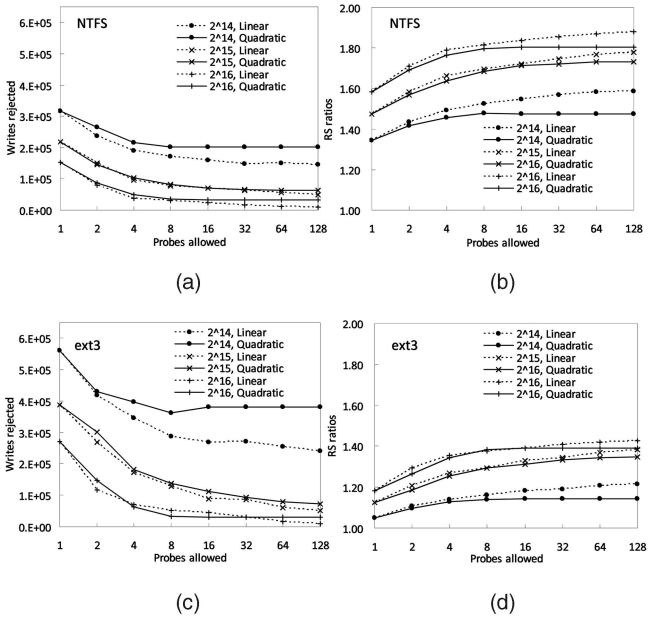

Fig. 11. The total number of writes rejected by the hash table under (a) the NTFS workload and (c) the ext3 workload. The RS ratio under (b) the NTFS workload and (d) the ext3 workload.

hash-table size is greater than 32,768. This implies that the total amount of valid data on the SLC flash is usually close to $32{,}768^{*}512 = 16$ MB (a sector is of 512 bytes), which is only one-sixteenth the SLC-flash capacity. Sufficiently large SLC flash provides enough buffer space to keep valid sectors away from the tail block.

The second part of our observation is on the probe method. Linear probing and quadratic probing show similar performance when the hash table is sufficiently large. When a small hash table of 16,384 entries is considered, quadratic probing rejects more writes than linear probing. That is because, even though a write request is classified as small, it may involve a number of contiguous sectors. A write can be accepted only if all the written sectors can have their mapping information inserted in the hash table. Linear probing tends to maintain the continuity of a key sequence in the hash table, which reduces the possibility of rejecting a write request. Another parameter is how many probes can be made. Results show that a small hash table (i.e., 16,384 entries) may require more probes to resolve collisions. With a reasonably large hash table, the benefits diminish when more than eight probes are allowed. The observations are consistent under different workloads. In summary, linear probing over 32,768 entries with up to eight probes seems a good choice.

5.5 Flash-Memory Lifetime Issues

This section examines 1) wear leveling over blocks of the same type (i.e., SLC or MLC) and 2) the efficacy of the utilization throttle. The second item is evaluated in terms of the BW ratio (refer to Section 5.1). Let *endurance ratio* of a configuration be the ratio of the endurance of SLC-flash blocks to that of MLC-flash blocks. Table 2 shows that the endurance ratios of configuration C_2 and configuration C_3 are 10:1 and 20:1, respectively. This section considers the NTFS workload.

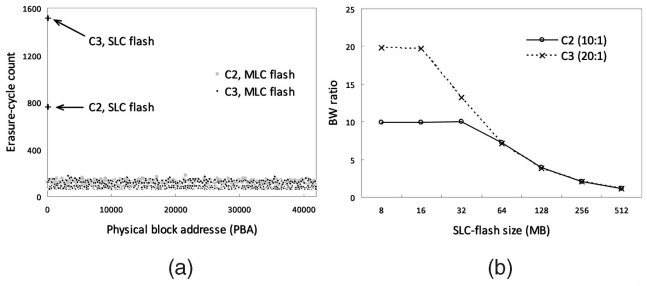

Fig. 12. Under the NTFS workload (a) distributions of erasure-cycle counts of SLC-flash blocks and MLC-flash blocks, and (b) BW ratios.

Fig. 12a shows the distributions of the erasure-cycle counts of SLC-flash blocks and MLC-flash blocks. The X-axis and the Y-axis denote flash-memory blocks' PBAs and their erasure-cycle counts, respectively. In this experiment, the SLC flash is 256 MB in both configurations C_2 and C_3. Fig. 12a shows that, as expected, wear leveling of SLC-flash blocks is perfect. This is because the proposed scheme erases all the SLC-flash blocks in a round-robin fashion. Wear leveling over MLC flash is independent of the idea of hybrid SSDs. This study adopts the dual-pool algorithm [13] with parameter $TH = 16$ for wear leveling over MLC-flash blocks. Fig. 12a shows that the wearing of MLC-flash blocks is even.

The second issue of this experiment is whether or not the utilization throttle properly balances the lifespan of SLC flash and MLC flash. If the utilization throttle works properly, the BW ratios should be no higher than the endurance ratio (i.e., 10:1 for C_2 and 20:1 for C_3). The X-axis and the Y-axis of Fig. 12b denote the SLC-flash sizes and the BW ratios, respectively. Results show that when the SLC flash is no larger than 16 MB, the BW ratios are strictly controlled at 10 and 20 for configurations C_2 and C_3, respectively. This is because the SLC flash is too small to sustain all the incoming writes without violating the endurance ratios. In this case, the utilization throttle starts rejecting writes to protect the SLC flash. In particular, configuration C_2 rejects more writes than configuration C_3 because configuration C_2 has a smaller endurance ratio. This also explains why the RS ratios of configuration C_2 in Fig. 8 are smaller than those of configuration C_3 when the SLC flash is small. Fig. 12b shows that when the SLC flash is larger than 32 MB, the BW ratios are lower than the endurance ratios of configurations C_2 and C_3 because a large number of SLC-flash blocks share the total erasure-cycle counts. This proves the effectiveness of the utilization throttle.

5.6 Energy Consumption and Hardware Costs

This section focuses on the energy consumption and hardware cost of hybrid SDDs. The hardware cost of a conventional SSD and a hybrid SSD are calculated according to Table 1. The energy models of SLC flash and MLC flash are based on the specifications in Table 1. For example, the energy required to write an MLC-flash page is $3.3\ \mathrm{V} \times 15\ \mathrm{mA} \times 800\ \mu\ \mathrm{seconds} = 39.6\ \mu\mathrm{J}$. The following discussion is based on configuration C_3, because the energy model, geometry, and costs of the SLC flash and MLC flash listed in Table 1 directly apply to configuration C_3.

TABLE 4
The EC Ratios and ES Ratios of Configuration C_3 with Different SLC-Flash Sizes

SLC-flash size	EC ratios	ES ratios NTFS	ES ratios ext3
32	100.5%	83.8%	94.2%
64	101.0%	68.9%	87.0%
128	102.0%	58.0%	78.6%
256	104.1%	54.2%	73.6%
512	108.2%	52.2%	70.9%

Table 4 shows the EC ratios and ES ratios of a hybrid SSD with respect to configuration C_3 (refer to Section 5.1 for the definition of ES ratios and EC ratios). These results show that hybrid SSDs consume less energy than conventional SSDs, especially when the SLC flash is large. Generally speaking, the SLC flash helps to conserve energy like it reduces response time. When the SLC flash is 256 MB, the ES ratios are 54.2 and 73.6 percent under the workloads of NTFS and ext3, respectively. As for extra costs, results show that the cost of hybrid SSDs with an 256 MB SLC flash is only 1.04 times more expensive than a conventional SSD. We believe that this investment is affordable and very rewarding.

6 Conclusion

As high-capacity NAND flash is becoming affordable, replacing power-hungry disks with large and cheap solid-state disks is an increasing trend. However, there is always a compromise between storage density and performance, as reflected by the physical characteristics of SLC flash and MLC flash. This paper investigates the combination of heterogeneous NAND-flash chips in large SSDs. The idea is to have SLC flash and MLC flash complement each other. This study proposes data placement/migration and wear leveling strategies for heterogeneous NAND flash. The rationale behind this hybrid-SSD design is closely related to the observations of real-life disk workloads. The primary finding of this study is that a hybrid SSD greatly improves upon a conventional SSD in terms of response, throughput, and even energy consumption. All of these improvements are achieved by adding a small SLC flash to a conventional SSD. The extra cost of a hybrid SSD is limited, and worth the investment.

Our goal for future study is to develop multitier architectures. For example, a small nonvolatile RAM serving as a write buffer, SLC flash acting as a high-speed logging device, and MLC flash functioning as the final repository of data.

Acknowledgments

This work is in part supported by research grant NSC-96-2218-E-009-017 from the National Science Council, Taiwan, R.O.C., and a research grant from A-DATA Technology Co., Ltd. This paper is an extended version of [14].

References

[1] A. Kawaguchi, S. Nishioka, and H. Motoda, "A Flash-Memory Based File System," *Proc. USENIX Technical Conf.*, 1995.
[2] B. Marsh, F. Douglis, and P. Krishnan, "Flash Memory File Caching for Mobile Computers," *Proc. 27th Hawaii Conf. Systems Science*, 1993.
[3] C. Park, P. Talawar, D. Won, M.J. Jung, J.B. Im, S.S. Kim, and Y.J. Choi, "A High Performance Controller for NAND Flash-Based Solid State Disk (NSSD)," *Proc. 21st IEEE Non-Volatile Semiconductor Memory Workshop (NVSMW)*, 2006.
[4] C.H. Wu and T.W. Kuo, "An Adaptive Two-Level Management for the Flash Translation Layer in Embedded Systems," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design*, 2006.
[5] D. Roselli, J.R. Lorch, and T.E. Anderson, "A Comparison of File System Workloads," *Proc. 2000 USENIX Ann. Technical Conf.*, 2000.
[6] E. Forgy, "Cluster Analysis of Multivariate Data: Efficiency versus Interpretability of Classifications," *Biometrics*, vol. 21, pp. 768-769, 1965.
[7] F. Chen, S. Jiang, and X. Zhang, "SmartSaver: Turning Flash Drive into a Disk Energy Saver for Mobile Computers," *Proc. 11th ACM/IEEE Int'l Symp. Low Power Electronics and Design*, 2006.
[8] E.J. O'Neil, P.E. O'Neil, and G. Weikum, "The LRU-k Page Replacement Algorithm for Database Disk Buffering," *ACM SIGMOD Record*, vol. 22, no. 2, pp. 297-306, 1993.
[9] H.G. Lee and N. Chang, "Energy-Aware Memory Allocation in Heterogeneous Non-Volatile Memory Systems," *Proc. Int'l Symp. Low Power Electronics and Design*, 2007.
[10] J.H. Yoon, E.H. Nam, Y.J. Seong, H.S. Kim, B.S. Kim, S.L. Min, and Y.K. Cho, "Chameleon: A High Performance Flash/FRAM Hybrid Solid-Sate Disk Architecture," *IEEE Computer Architecture Letters*, vol. 7, no. 1, pp. 17-20, Jan.-June 2008.
[11] J.W. Hsieh, T.W. Kuo, and L.P. Chang, "Efficient Identification of Hot Data for Flash Memory Storage Systems," *ACM Trans. Storage*, vol. 2, no. 1, pp. 22-40, 2006.
[12] J. Kim, J.M. Kim, S.H. Noh, S.L. Min, and Y. Cho, "A Space-Efficient Flash Translation Layer for Compactflash Systems," *IEEE Trans. Consumer Electronics*, vol. 48, no. 2, pp. 366-375, May 2002.
[13] L.P. Chang, "On Efficient Wear-Leveling for Large-Scale Flash-Memory Storage Systems," *Proc. 22nd ACM Symp. Applied Computing*, 2007.
[14] L.P. Chang, "Hybrid Solid-State Disks: Combining Heterogeneous NAND Flash in Large SSDs," *Proc. 13th Asia and South Pacific Design Automation Conf.*, 2008.
[15] M.E. Russinovich and D.A. Solomon, *Microsoft Windows Internals: Microsoft Windows Server 2003, Windows XP, and Windows 2000*, fourth ed. Microsoft Press, 2004.
[16] O. Avissar, R. Barua, and D. Stewart, "An Optimal Memory Allocation Scheme for Scratch-Pad-Based Embedded Systems," *ACM Trans. Embedded Computing Systems*, vol. 1, no. 1, pp. 6-26, Nov. 2002.
[17] Samsung Electronics Company, "K9F2G08U0A 256M * 8 Bit SLC NAND Flash Memory Data Sheet." 2006
[18] Samsung Electronics Company, "K9GAG08U0M 2G * 8 Bit MLC NAND Flash Memory Data Sheet (Preliminary)." 2006
[19] Samsung Electronics Company, "NAND Flash-Based Solid State Disk Data Sheet," http://www.samsung.com/Products/Semiconductor/FlashSSD/download/Standard_type.pdf, 2010.
[20] T. Bisson and S. Brandt, "Reducing Energy Consumption with a Non-Volatile Storage Cache," *Proc. Int'l Workshop Software Support for Portable Storage (IWSSPS)*, 2005.
[21] S.W. Lee, D.J. Park, T.S. Chung, D.H. Lee, S.W. Park, and H.J. Song, "A Log Buffer-Based Flash Translation Layer Using Fully-Associative Sector Translation," *ACM Trans. Embedded Computing Systems*, vol. 6, no. 3, pp. 18:1-18:27, 2007.
[22] W.G. Wang, Y.P. Zhao, and R. Bunt, "HyLog: A High Performance Approach to Managing Disk Layout," *Proc. Third USENIX Conf. File and Storage Technologies*, 2004.
[23] W. Vogels, "File System Usage in Windows NT 4.0," *Proc. 17th ACM Symp. Operating Systems Principles*, 1999.
[24] Y.J. Kim, K.T. Kwon, and J. Kim, "Energy-Efficient File Placement Techniques for Heterogeneous Mobile Storage Systems," *Proc. Sixth ACM and IEEE Int'l Conf. Embedded Software*, 2006.
[25] Y.M. Hu and Q. Yang, "DCD-Disk Caching Disk: A New Approach for Boosting I/O Performance," *Proc. 23rd Int'l Symp. Computer Architecture*, 1996.

[26] Y.W. Park, S.H. Lim, C. Lee, and K.H. Park, "PFFS: A Scalable Flash Memory File System for the Hybrid Architecture of Phase-Change RAM and NAND Flash," *Proc. 23rd ACM Symp. Applied Computing,* 2008.
[27] T. Kgil, D. Roberts, and T. Mudge, "Improving NAND Flash Based Disk Caches," *Proc. 35th Int'l Symp. Computer Architecture,* 2008.
[28] A. Wang, P.L. Reiher, G.J. Popek, and G.H. Kuenning, "Conquest: Better Performance through a Disk/Persistent-RAM Hybrid File System," *Proc. Ann. Conf. USENIX Ann. Technical Conf.,* 2002.
[29] T.M. Wong and J. Wilkes, "My Cache or Yours? Making Storage More Exclusive," *Proc. Ann. Conf. USENIX Ann. Technical Conf.,* 2002.
[30] S. Lee, D. Shin, Y. Kim, and J. Kim, "LAST: Locality-Aware Sector Translation for NAND Flash Memory-Based Storage Systems," *ACM SIGOPS Operating Systems Rev.,* vol. 42, no. 6, pp. 36-42, 2008.
[31] C. Park, W. Cheon, J. Kang, K. Roh, W. Cho, and J. Kim, "A Reconfigurable FTL (Flash Translation Layer) Architecture for NAND Flash-Based Applications," *ACM Trans. Embedded Computing Systems,* vol. 7, no. 4, pp. 38:1-38:23, 2008.
[32] D. Arthur and S. Vassilvitskii, "How Slow is the k-Means Method?" *Proc. 22nd Ann. Symp. Computational Geometry,* 2006.
[33] H. Kim and S. Ahn, "BPLRU: A Buffer Management Scheme for Improving Random Writes in Flash Storage," *Proc. Sixth USENIX Conf. File and Storage Technologies,* 2008.
[34] H. Jo, J.U. Kang, S.Y. Park, J.S. Kim, and J. Lee, "FAB: Flash-Aware Buffer Management Policy for Portable Media Players," *IEEE Trans. Consumer Electronics,* vol. 52, no. 2, pp. 485-493, May 2006.

Li-Pin Chang received the BE degree in computer science and information engineering from I-Shou University, Taiwan, in 1995, and the MSE and PhD degrees in computer science and information engineering from the National Taiwan University, in 1997 and 2003, respectively. He is an assistant professor in the Department of Computer Science, National Chiao-Tung University, Taiwan. His research interest is in operating systems, storage systems, and real-time system. He had served on the editorial board of the *Journal of Signal Processing Systems* (Springer, SCI-E) and the technical committees of international conferences, including the ACM Symposium on Applied Computing (ACM SAC), the IEEE Real-Time Systems Symposium (IEEE RTSS), the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (IEEE RTCSA), and the IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC).