Asymmetric Flash Volume Management

Minyoung Sung, *Member*, IEEE and Kanghee Kim, *Member*, IEEE

Abstract — *Logical volume management over an SLC (Single-Level Cell) / MLC (Multi-Level Cell) combined flash storage can provide improved performance and reliability. This paper proposes an asymmetric flash volume management (AFVM) scheme, which exploits the asymmetry of SLC and MLC flash memory in their service time characteristics. By hot data identification and periodic data migration, AFVM tries to maximize the overall read/write performance of the logical volume. Through extensive simulations with measured data on a real smartphone, we evaluate the performance of AFVM in comparison with other placement policies. Our experiments derive the best trade-off between the performance and overheads for various values of tunable parameters such as migration unit size, migration traffic limit, and migration period. The results show that AFVM could give better solutions, significantly reducing the execution time of a trace-driven workload at the expense of a negligible decrease in flash lifetime*[1].

Index Terms — Logical volume management, SLC/MLC flash memory, hot data identification, data migration.

I. Introduction

Traditionally, volume management has received large attention on storage systems with multiple disks. One large logical volume consisting of multiple disks gives a greater flexibility than a mere collection of individual disks in terms of storage management. That is, in a logical volume, we may allow data redundancy for higher reliability and/or exploit parallel access to the disks for higher performance [1]. For consumer electronics devices with multiple flash chips, volume management may still be useful if we consider peculiarities of the flash memory. Such peculiarities include page-based reads and writes, erase-before-write, block-based erases, and limited erase cycles.

One attractive application of volume management is a flash storage system combining SLC (Single-Level Cell) and MLC (Multi-Level Cell) flash memory. SLC means that each cell represents one bit with binary voltage levels while MLC means that each cell represents more than one bit with multiple voltage levels. In comparison with MLC flash memory, SLC flash memory has a smaller page size of 2 KB, a smaller erase block size of 128 KB, and a greater number of erases allowed for each erase block, i.e., 100,000 times [5]. Thus, it better suits fragmentation-prone data and data with high-reliability requirement. MLC flash memory has a greater page size of 4 KB, a greater erase block size of 512 KB, and a smaller number of erases allowed, i.e., 5,000 to 10,000 times. Thus, it better suits continuous media data and data with less reliability requirement. This suggests a chance of employing different data placement strategies for better performance and/or better reliability over a logical volume combing the two different types of flash memory. The need for volume management over an SLC/MLC combined flash storage system becomes more evident when we consider the consumer devices available in the market that can be configured with partitions of different types. For example, a hybrid flash chip [2],[3] used in smartphones can be divided into one SLC partition (under the name of "Enhanced User Data Area") and one MLC partition, the ratio of which can be arbitrarily set by the user. Therefore, for such a chip, we can create a logical volume covering the two partitions and employ a data placement strategy exploiting the asymmetry of the partitions. As another example, there are many consumer devices which use both SLC and MLC in separate chips. In their commercial brochure, the former is usually misleadingly referred to as 'ROM', which amounts to several hundred megabytes, while the latter is referred to as 'internal storage', which amounts to several dozens of gigabytes. In this case, the same idea of the volume management can be applied even if the sizes of the SLC and MLC chips are fixed.

This paper proposes an asymmetric flash volume management (AFVM) scheme, which exploits the asymmetry of SLC and MLC flash memory. In the proposed scheme, we try to maximize the overall read/write performance of the logical volume by (1) hot data identification and (2) periodic data migration between SLC and MLC. In hot data identification, to exploit the asymmetry of SLC and MLC areas in a logical volume, we divide the entire logical volume into zones of a fixed size and monitor the distribution of the sizes of write requests experienced in each zone for a certain period. By associating the collected request size distribution for each zone and pre-measured service times for each request size on the two types of flash memory, we can make a data migration plan that gives the highest overall read/write throughput for the next period. This planning is based on the observation that we can achieve better performance with more-frequent small writes placed on the SLC partition and less-frequent large writes placed on the MLC partition. Then, we execute the data migration plan in a non-intrusive manner in terms of both user-perceived performance and flash memory lifetime. That is, the data migration will be conducted exploiting idle periods of the system, for example, while the

[1] This research was supported by MKE (The Ministry of Knowledge Economy), Korea, under the Convergence-ITRC (Convergence Information Technology Research Center) support program (NIPA-2012-H0401-12-1004) supervised by the NIPA (National IT Industry Promotion Agency).

M. Sung is with the Department of Computer Software Engineering, Sangmyung University, Choenan, 330-720 Korea.

K. Kim is with the School of Electronic Engineering, Soongsil University, Seoul, 156-743 Korea (e-mail: khkim@ssu.ac.kr).

Contributed Paper
Manuscript received 04/15/12
Current version published 06/22/12

battery is being charged in the case of battery-operated devices, and the amount of the migration traffic will be kept below a certain threshold so that it can only have a marginal effect on the lifetime of the underlying flash memory.

Our scheme is distinguished from prior work in the following senses. *First, AFVM works in the block device driver layer of the underlying operating system, not in the FTL* (Flash Translation Layer), which is tightly bound to the geometry of the flash chip. This means that the scheme does not depend on a specific flash chip vendor and a specific filesystem, thus has wide applicability. *Second, AFVM can control the degree to which the data migration affects the lifetime of the underlying flash chip.* This is possible in our scheme since we try to find a best zone placement under a constraint on the total amount of data migration. With this constraint, we can provide the user with a spectrum between conservative incremental migration and aggressive burst migration.

To evaluate the proposed scheme, we perform trace-driven simulations and use as inputs the pre-measured request service times on a real smartphone having both SLC and MLC in separate chips. In this testbed, the measurement is conducted on a logical volume created with LVM (Logical Volume Manager) [4] that covers both the SLC and MLC chips. Our experiments show that we can find a best trade-off between the performance gains and overheads of data migration by tuning the parameters introduced in AFVM such as zone size, migration period, and migration traffic threshold. We also show that AFVM significantly improves the overall execution time of a trace-driven workload by up to 82.7% when compared with the fixed placement.

Our paper is organized as follows. In Section 2, we summarize the related work, and in Section 3, describe the proposed scheme in detail. Section 4 gives the experimental results and Section 5 concludes the paper.

II. Related Work

Organizing data according to the popularity over multiple storage devices has been studied in many ways. For traditional disk arrays, Pinheiro and Bianchini [17] propose a technique called "Popular Data Concentration (PDC)". The main idea of PDC is to dynamically migrate popular data to a subset of the disks in the array, so that the other disks can be sent to low-power modes. In PDC, the first disk has the most popular data and the second disk has the next most popular data, and so on. As a similar approach, Xie [18] proposes a disk array of two-speed disks, which is divided into two sets according to the data popularity: the hot disk set running in the high-speed mode and the cold disk set in the low-speed mode. Targeting for energy conservation, both approaches exploit the observation that network server workloads typically show files with widely different popularities.

For storage systems combining hard disks and flash memory devices, an augmented version of PDC called pattern-based PDC (PB-PDC) has been proposed [15]. It tries to further classify data popularity in terms of access type, i.e., read access and write access, while PDC does not distinguish them. Based on this classification, PB-PDC moves the popular write data to the hard disk and the popular read data to the flash memory device in order to achieve energy conservation. On the other hand, Xie and Sun [16] propose a dynamic data redistribution strategy called PEARL together with a hybrid disk array consisting of flash and hard disks. This approach is to monitor the popularity in terms of number of accesses for each zone in the hard disk array for a certain period and move the hottest zones to the flash disk.

For flash storage systems combining SLC and MLC flash memory, there are several approaches. A hybrid SSD (solid state disk) was proposed that consists of an SLC flash chip and a number of MLC flash chips [5]. In this hybrid SSD, the SLC flash chip serves small write requests as a log space based on the observation that most of the writes are either very small or very large, and are rarely medium sized. As a similar approach, SLC flash chips are used as non-volatile buffer space in order to boost the performance and lifetime of MLC flash based SSDs [6]. In this architecture, the performance can be improved since all the write data are first stored into SLC buffers, while the lifetime can be lengthened since the data in the SLC buffers are pre-merged wherever possible before merged into the MLC flash chips. As another example, it has also been proposed to use the SLC chips as log buffer space and the MLC chips as normal data blocks [7],[8]. In short, all of the above mentioned approaches try to direct small random writes to the SLC chips in order to exploit the performance asymmetry of SLC and MLC flash memory.

This performance asymmetry has also been exploited at different layers of the storage system. For example, a new buffer cache management scheme was proposed for SLC/MLC-combined flash memory as the secondary storage assuming that both byte-accessible NVRAM and conventional volatile RAM are used as the buffer cache [10]. As another example, SLC/MLC-combined flash memory has been used for non-volatile cache in order to enhance the performance and energy consumption of storage systems [9].

In order to organize data in a popularity-centric manner, it is important how to identify hot data in the storage system. In PDC [17] and PB-PDC [15], the data popularity is measured at file level. The data popularity was also measured at process level [14], thus write operations from a process with a high storage update frequency are marked 'hot' before forwarded to the flash storage system. Recently, a highly efficient method for online hot data identification is proposed that uses multiple independent hash functions [12]. This method has been used by Lee et al. [13] to exploit different mapping techniques for address translation in FTL such as sector mapping and block mapping according to hot/cold data, but not to make use of the asymmetry of different storage devices.

Our proposed scheme is distinguished from the above mentioned studies in that it is implemented in the block device driver layer of the underlying operating system, thus not dependent on a specific filesystem or a specific storage system. The data popularity is measured at block I/O request level

according to different I/O request sizes. Unlike file-level popularity measurement such as PDC and PB-PDC, we measure the request size distribution for each zone of a logical volume, assuming that the volume consists of one SLC area and one MLC area, which are divided into a number of zones of a fixed size. The request size distribution will be used in redistributing the zones over the volume since the request-size-aware data placement has great impact on the performance of the underlying flash memory.

III. Asymmetric Flash Volume Management

AFVM aims to maximize the overall read/write throughput of an SLC/MLC-combined logical volume. The volume is assumed to consist of one SLC area and one MLC area, both of which may reside in a single chip or two separate chips. From a filesystem point of view, the logical volume can be viewed as a flat storage, thus the filesystem will perform sector allocation in its own way, not being aware of the existence of the two areas. Only the block device driver abstracting the two areas into a logical volume knows it, which is called the device mapper in the LVM. Thus, if we implement the AFVM scheme with the LVM, we need to modify the device mapper, which is responsible for hot data identification, and write a daemon program, which is responsible for zone migration.

A. Device Mapper

The device mapper performs two main functions. The first function is to maintain a mapping table that translates the address of a logical zone to the address of a physical zone. The reason for this zone mapping table is that we should be able to track the location of every migrated zone so that we can serve the read/write requests coming from the upper filesystem, which has no idea about the locations of migrated zones. It is determined by the AFVM daemon which zone will be migrated to which area. In order for the daemon to specify this migration request, our device mapper provides an ioctl command to the daemon that swaps the contents of any two zones designated.

The second function of the device mapper is to monitor the distribution of the sizes of write requests falling in each zone. The range of the write request size is divided into a number of subranges whose sizes are in an increasing order of power-of-two numbers, e.g., 64 KB, 128 KB, 256 KB, and so on. In our device mapper, in order to treat small but sequential write requests as one large write request, we monitor the sequentiality of the write request stream. This is because the small but sequential write requests result from one large request split by the filesystem, which is an artifact in measuring the request size distribution. The reason why we only monitor the write request sizes is that write operations have significantly greater impact on the performance of flash chips than read operations. In order for the daemon to query the measured distribution of the sizes of the (logical) write requests, our device mapper provides an ioctl command to the daemon, so that the daemon can make a migration plan with pre-measured service times for each request size on both SLC and MLC areas. Fig 1. describes the pseudo code for monitoring the distribution of the write request sizes. Table 1 summarizes the notations used throughout the paper.

TABLE I
Notations

Notation	Description
N^v	The number of zones in flash area *v*
N	The total number of zones in both flash areas, *v0* and *v1*. $N = N^{v0} + N^{v1}$
S	The zone size, i.e., the number of sectors in a zone
$Z_{i,j}^{\text{count}}$	The access count for write size *j* at zone *i*
Z_i^{sector}	The address of the sector next to the most recently written sector at zone *i*
Z_i^{size}	The cumulative size of the recent consecutive write requests at zone *i*
T_j^v	The service time for a write request of size *j* on flash area *v*
G_i	The expected gain from placing zone *i* in *v0* when compared with placing in *v1*

```
AFVM_log_requests
input
R^sector // sector address of request R
R^size  // request size, i.e., the number of sectors to write
begin procedure
1    i = R^sector / S ; // get the zone number for R under zone size S
2
3    if ( Z_i^sector == R^sector ) // this is a consecutive request
4        Z_i^sector = Z_i^sector + R^size ;
5        Z_i^size = Z_i^size + R^size ;
6    else                    // a disjoint request
7        if ( Z_i^size > 0 )
8            j ← range index for request size Z_i^size;
9            Z_i,j^count = Z_i,j^count + 1;
10       end if
11       Z_i^sector = R^sector + R^size ;
12       Z_i^size = R^size ;
13   end if
end procedure
```

Fig. 1. Pseudo code for monitoring the write request sizes. It counts a sequence of successive write requests that are spatially consecutive as one large request.

In order to record the sizes of write requests falling in each zone *i*, the device mapper maintains $Z_{i,j}^{\text{count}}$, the count of the write requests of size *j* on zone *i*, Z_i^{size}, the consecutive write size, and Z_i^{sector}, the address of the sector next to the most recently written sector. When a new write request *R* arrives at zone *i*, its start address R^{sector} is compared with Z_i^{sector}. If they are equal, it means that *R* is spatially consecutive to the latest requests in the zone. We treat those requests as one large request, so we just increase Z_i^{sector} and Z_i^{size} by the size of *R* (lines 4 and 5). If R^{sector} and Z_i^{sector} differ, *R* is the first request of a new write request sequence. We increase the count by one corresponding to the write size Z_i^{size} (lines 8 and 9) and initialize Z_i^{sector} and Z_i^{size} for the new write sequence (lines 11 and 12). Note that instead of keeping separate counts for all

possible write sizes, we use a number of predefined ranges of request size and associate a single count with each range.

B. AFVM Daemon

The AFVM daemon periodically conducts zone migration between the SLC and MLC areas, which are denoted by *v0* and by *v1*, respectively. Whenever it is invoked, it queries the measured distribution of the write request sizes for each zone to the device mapper, and makes a migration plan by computing the gain of migration for each zone, assuming that the observed distributions in the previous period will not change significantly in the next period. We calculate the gain G_i, which is assumed to be zero initially, as the sum of the reduced service time that can be obtained from placing the zone *i* in *v0* when compared with placing in *v1*:

$$G_i = \sum_j Z_{i,j}^{\text{count}} \times (T_j^{v1} - T_j^{v0}), \tag{1}$$

where $Z_{i,j}^{\text{count}}$ refers to the access count for write requests of size *j* on zone *i* and T_j^v refers to the pre-measured service time for a write request of size *j* in flash area *v*. If the computed gain is greater than zero, it means that it is beneficial to place the zone in *v0*. However, since there are a large number of zones in the entire logical volume, the AFVM daemon sorts all the zones according to the computed gains and selects as migration candidates the zones with the higher gains. The number of migration candidates will be determined in such a way that the resultant migration traffic does not exceed a threshold. This is because we need to preserve the lifetime of the underlying flash memory by limiting the total amount of the migration traffic below a threshold.

Fig. 2 describes the pseudo code for a zone migration. Based on (1), for every invocation period, the AFVM daemon calculates the gains G_i's using the access counts $Z_{i,j}^{\text{count}}$'s and service time T_j^v (lines 1 to 6). The daemon then sorts the list of zones according to the computed gains in descending order (line 7). From the sorted list, it tries to migrate each of the first N^{v0} zones to area *v0* if it is not in the area *v0* (lines 9 to 13). If it is already in the area *v0*, it will not be migrated. Note that once the zone migration is completed, the distribution of write request sizes is reset (lines 15 to 19).

The AFVM scheme offers many benefits in terms of performance, flexibility, and durability. First, the essential feature of determining the best candidate zones is in the awareness of the service time characteristics that is intrinsic to flash storages. In order to handle the 'erase-before-write' property of flash memory devices, the FTL usually keeps newly committed writes in the temporary log blocks and merges them with the existing data pages when necessary. So, in the log, the larger and the less scattered the writes are, the shorter it takes to merge. The AFVM scheme tries to capture this service time characteristics by keeping track of the consecutive write size for each zone. The experimental results in Section IV validate its strength against other data migration schemes.

```
AFVM_do_migration
input
{Z_i}      // list of zones ( 0 ≤ i < N )
Z^count_i,j   // write counts for size range j at zone i
           // ( 0 ≤ i < N , 0 ≤ j < L )
T^v_j      // service time in area v for a write request in size range j
           // (v = v0 or v1, 0 ≤ j < L )
begin procedure
1   for i=0 to N-1 do // calculate the gains
2       G_i = 0;
3       for j=0 to L-1 do
4           G_i = G_i + Z^count_i,j × (T^v1_j − T^v0_j);
5       end for
6   end for
7   {Z_i} ← sorted list of zones according to G_i in descending order;
8
9   for k=0 to N^v0-1 do // relocate zones
10      if Z_k is in area v1
11          migrate Z_k into v0;
12      end if
13  end for
14
15  for i=0 to N-1 do // reset write counts
16      for j=0 to L-1 do
17          Z^count_i,j = 0;
18      end for
19  end for
end procedure
```

Fig. 2. Pseudo code for zone migration. Using the access counts and pre-measured service time, AFVM calculates the gain for each zone that is expected from placing the zone in *v0* when compared with placing in *v1*. The AFVM daemon migrates zones to a flash area giving a larger gain.

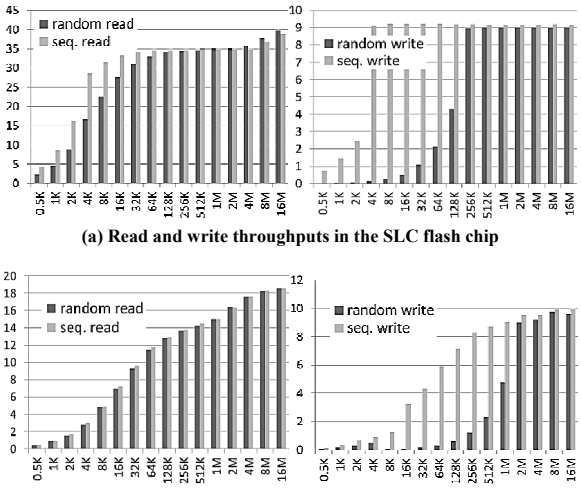
Second, the proposed scheme is very flexible in the sense that it allows to control the incidental overheads by varying the values of parameters. Since flash memory devices have a limited number of erase cycles, the additional write traffic due to migration may seriously damage the lifetime if it is not controlled carefully. For this control, we define three tunable parameters: migration period, zone size, and migration traffic limit. The migration period is defined in terms of the cumulative number of I/O requests before triggering a migration. The zone size is defined in sectors as the unit of the migration and monitoring the distribution of write sizes. The migration traffic limit is the amount of the total write traffic that arises due to the zone migration between two consecutive periods. Clearly, there exists a trade-off between the performance and overhead in terms of the parameter values. In the next section, in terms of *Pareto optimality* [19], we show how we can determine the values of tunable parameters such that they maximize the performance while keeping the overhead below a predefined level. Note that the overall lifetime of the underlying flash chips may not be shortened in proportion to the amount of the migration traffic. Since an MLC flash chip has a shorter lifetime, migrating hot zones to an SLC flash chip may balance the lifetimes of both SLC and MLC flash chips, thus lengthen the overall lifetime of the entire logical volume.

IV. PERFORMANCE EVALUATION

In this section, we evaluate the performance of AFVM through extensive simulations with the measurement data from a real smartphone.

A. Experimental Setup

For our measurement of the request service times, we use a commercial smartphone as our testbed. The testbed is equipped with a microprocessor running at 1 GHz, 512 MB DRAM, a 512 MB SLC flash chip, and a 16 GB MLC flash chip. In this testbed, we enabled the device mapper in the operating system and measured the request service time on each of the SLC and MLC areas.

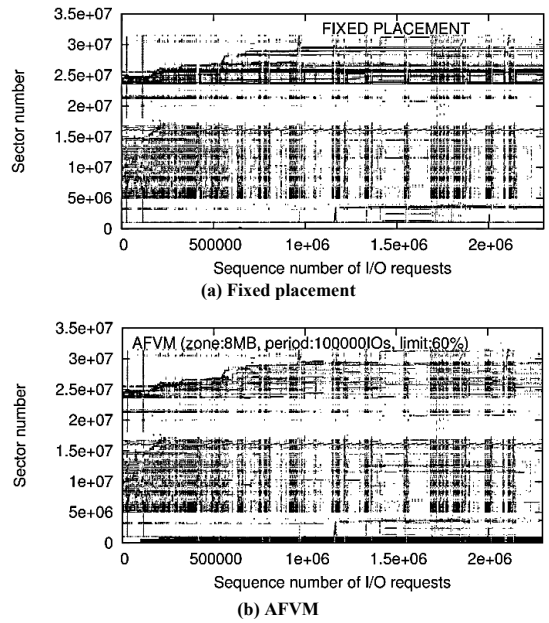


(a) Read and write throughputs in the SLC flash chip

(b) Read and write throughputs in the MLC flash chip

Fig. 3. The measured read and write throughputs according to the request size. The x-axis is the request size in bytes while the y-axis is the measured throughput in megabytes.

With the above testbed, we first measure the read and write throughputs for both types of flash memory. Fig. 3 depicts the measurement results. We can see that increased request size greatly improves the read and write throughputs. In overall, we observe similar throughputs for both the sequential and random reads. The write throughputs, however, show quite a different trend depending on the randomness of the requests. While random writes have extremely low performance for very small request sizes, their throughputs significantly grow in accordance with the increase of the request size, and beyond a certain size, they match the throughputs of sequential writes. The size from which the throughputs of both sequential and random writes become equal corresponds to the size of the logical block that the FTL internally uses as the basic unit for erase operations [11]. The determined logical block size in our case is 256 KB for the SLC flash chip and 2 MB for the MLC flash chip. As explained earlier, AFVM exploits the service time characteristics by treating small sequential writes as one large write, in order to better compute the migration gain for each zone. For minimizing the overhead of monitoring the distribution of write request sizes for each zone, we record write access counts in five subranges of request sizes, (0, 256KB], (256KB, 512KB], (512KB, 1MB], (1MB, 2MB], and (2MB, ∞). Note that we use the measured throughput for random writes in calculating the migration gain for each zone, since the real-life workload usually consists of random writes.



(a) Fixed placement

(b) AFVM

Fig. 4. Disk sectors accessed along the sequence of I/O requests. The two subfigures clearly demonstrate that AFVM effectively migrates heavily accessed sectors, e.g., those in the region of sector addresses around 2.5×10^7 into the SLC area of sector address below 1.1×10^6.

For the experimental workloads, we use a disk trace collected from an ext4 filesystem in a PC, which was used in the experiments by Chang [5]. The workloads consist of about 20 days of normal user activities such as web browsing, video streaming, typesetting, gaming, software installing, and email receiving and sending. The size of the ext4 filesystem was 16 GB. To understand the disk access pattern and the effect of the periodic migration, we conduct trace-driven simulations and show the accessed disk sectors along the sequence of I/O requests in Fig. 4. Without migration, as shown in Fig. 4(a), we can see that sectors in some regions, e.g., sector numbers around 2.5×10^7 are intensively accessed for the entire trace period while some other regions show changes in access frequency with time. Fig. 4(b) clearly illustrates the effect of AFVM. It plots the accessed sectors when the zone migration is conducted with a zone size of 8MB and a migration period of 100,000 requests. The migration traffic was limited to 60 % of the SLC-area capacity. We can observe that those intensively accessed sectors are progressively migrated into the SLC area, which corresponds to the region with sector address below 1.1×10^6.

B. Comparison with Other Schemes

In this subsection, we evaluate the performance of AFVM in comparison with PDC and WPDC, which is a variant of the PB-PDC [15]. The variant differs from the original PB-PDC in that it maintains the access count at zone level, not at file level, and that the access count only records the total number of write requests falling in each zone, excluding read requests. The reason that we use such a variant is that the SLC area in our testbed shows much better read performance than the MLC area, as shown in Fig. 3. Thus, if we migrate a zone into the SLC area, not only the performance of small writes, but also the read performance is improved. In this way, we can focus on evaluating the effect of the two different ways of data popularity measurement, i.e., zone access counting of WPDC vs. request-size-aware zone access counting of AFVM.

For performance comparison with PDC and WPDC, we set up the objectives of the migration policy in two folds. First, the policy should be able to minimize the time for reading and writing the logical volume. Second, in consideration of the flash lifetime, the policy should minimize the additional write traffic due to migration. In the experiments, we explore the trade-offs between the above mentioned performance and overhead using various values of tunable parameters, i.e., zone size, migration traffic limit, and migration period. As the metrics for the performance and overhead, we use the total service time and the total amount of migrated zones, respectively. For actual measurement, a tremendous time is required because we have to deal with a vast parameter space. Hence, rather than measuring on the real testbed, we decided to perform simulations with the pre-measured service time for each request size, shown in Fig. 3.

For the visualization of the trade-off between the two objectives, we introduce the concept of Pareto optimality [19]. A system variable vector $\mathbf{x}^* \in \mathbf{\Omega}$ is *Pareto optimal* if and only if there is no vector $\mathbf{x} \in \mathbf{\Omega}$ with the characteristics:

$f_i(\mathbf{x}) \le f_i(\mathbf{x}^*)$ for all i and

$f_i(\mathbf{x}) < f_i(\mathbf{x}^*)$ for at least one i (one objective),

where each f_i is an objective function. The *Pareto curve* is the set of $\mathbf{x}^*$ where there are no other solutions for which simultaneous improvement in all objectives can occur. In our case, we have two objectives, i.e., the total service time and the amount of migrated zones, whose functions are defined over the set of vector $\mathbf{x}$ = (zone size, migration traffic limit, migration period). Fig. 5 plots the estimated total service time and the corresponding total amount of migrated zones for different values of the tunable parameter vector. It is assumed that the 16 GB logical volume consists of 2% SLC area and 98 % MLC area in size. As the possible values of zone size, we used 2, 4, 8, 16, and 32 MB. The migration traffic limit is varied across 20, 40, 60, 80, and 100% of the SLC area. And the migration period was chosen from 5×10^4, 1×10^5, 2×10^5, and 4×10^5 requests.

In Fig. 5 the Pareto curve connects the Pareto-optimal points. The Pareto curve gives us intuitive solution when it is required to control the migration with a restriction on the migration overhead. From the figure, for example, we can find that if we wish to limit the amount of migrated zones to 2 GB, the combination of (2 MB, 20%, 100,000) is the optimal parameter set, which gives 1,472 seconds of estimated total service time with about 1.5 GB of additional write traffic for the migration.

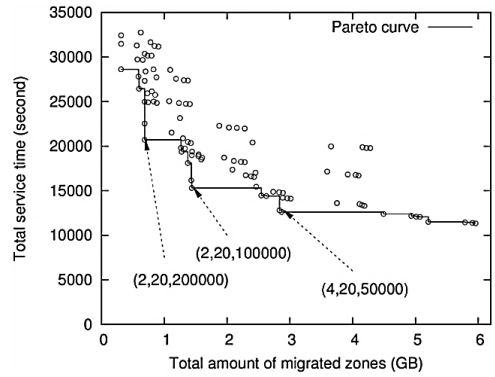


Fig. 5. Total service times and total amount of migrated zones estimated from the AFVM migration. Each point corresponds to a certain combination of control parameter values: zone size, migration traffic limit, and migration period. Some of the Pareto-optimal points are given with the values of the control parameters.

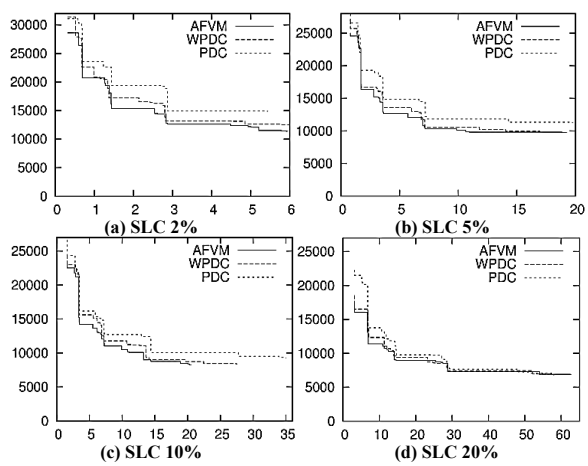


Fig. 6. Pareto curves according to the migration policies. The x-axis is the total amount of migrated zones in gigabytes while the y-axis is the total service time in seconds. The figures show their performance-overhead behavior when the SLC portion varies among (a) 2%, (b) 5%, (c) 10%, and (d) 20% of the entire 16 GB capacity.

As shown in Fig. 6, the Pareto curves derived from the experiment results validate the strength of AFVM in comparison with PDC and WPDC. The Pareto curves compare the performance-overhead behaviors of the AFVM, PDC, and WPDC schemes when we vary the portion of the SLC area out of the entire 16 GB storage capacity. We used the same set of parameter values as in the experiment shown in Fig. 5. In overall, the figures show that AFVM gives better solutions throughout various SLC capacities. The total service times with the fixed placement were 47,973, 47,481, 47,386, and 39,227 seconds for each respective SLC capacity of 2%, 5%,

10%, and 20%. Compared with the fixed placement, AFVM could reduce the service time by 76.3%, 79.5%, 82.7%, and 82.6% with the increase of the SLC capacity. It can be said that with AFVM, 10% of SLC capacity is the recommended design choice because beyond that, reduction in service time gets saturated and the advantage of AFVM over the other migration schemes becomes marginal. We can also observe that AFVM is very effective even when there exists only a small size of SLC area.

Also note that the overhead due to the migration is negligible even with the aggressive migration. The control point that gives the best service time performance in Fig. 6(c), for example, generates about 21 GB of additional write traffic in about 20 days, which corresponds to 120 times and 14 times of volume-wide erasure per year on the SLC and MLC devices, respectively. The resultant reduction in lifetime can be considered trivial when we recall their respective usual lifetimes of 100,000 and 10,000 erases.

V. Conclusion

We have proposed an asymmetric flash volume management scheme, which exploits the asymmetry of SLC and MLC flash memory in their service time characteristics. By hot data identification and periodic data migration, AFVM tries to maximize the overall read/write performance of the logical volume. Through extensive simulations, we showed that, in comparison with other relevant placement policies, AFVM could better reduce the execution time of a trace-driven workload at the expense of a negligible decrease in flash lifetime. We also explored the space of the tunable parameters such as migration unit size, migration traffic limit, and migration period, so that we could find a best trade-off between the performance and the flash lifetime. Note that our scheme is general enough to cover different objective functions, e.g., overall reliability, although we set in this paper our objective to the overall throughput.

It should be mentioned that the initial data placement is also important. As shown in our experiments, aggressive migration occurs during the initial phase of the given workload. This means that the migration traffic can be reduced significantly if we have a good initial placement decision. We will investigate this issue in our future research.

References

[1] D. A. Patterson, G. Gibson, and R. H. Katz. "A case for redundant arrays of inexpensive disks (RAID)," in *Proc. of ACM SIGMOD International Conference on Management of Data*, pp. 109-116, Jun. 1988.

[2] Toshiba America Electronic Components, Inc. "Advantages of e-MMC 4.4 based embedded memory architectures," *White paper*, Sep. 2010.

[3] JEDEC. "Embedded MultiMediaCard(eMMC) eMMC/Card product standard, high capacity, including reliable write, boot, sleep modes, dual data rate, multiple partitions supports, security enhancement, background operation and high priority interrupt (MMCA, 4.41)," *JEDEC Standard No. 84-A441*, Mar. 2010.

[4] M. Hasenstein. "The logical volume manager (LVM)," *White paper,* 2001.

[5] L.-P. Chang, "A hybrid approach to NAND-flash-based solid-state disks," *IEEE Trans. Computers*, vol. 59, no. 10, pp. 1337-1349, Oct. 2010.

[6] S. Jung and Y. H. Song, "Hierarchical use of heterogeneous flash memories for high performance and durability," *IEEE Trans. Consumer Electron.,* vol. 55, no. 3, pp. 1383-1391, Aug. 2009.

[7] S. Im and D. Shin, "Storage architecture and software support for SLC/MLC combined flash memory," in *Proc. of ACM Symposium on Applied Computing*, 2009.

[8] B.-W. Nam, G.-J. Na, and S.-W. Lee, "A hybrid flash memory SSD scheme for enterprise database applications," in *Proc. of Asia-Pacific Web Conference*, 2010.

[9] S. Hong and D. Shin, "NAND flash-based disk cache using SLC/MLC combined flash memory," in *Proc. of Workshop on Storage Network Architecture and Parallel I/Os*, 2010.

[10] J. Park, H. Bahn, and K. Koh, "Buffer cache management for combined MLC and SLC flash memories using both volatile and nonvolatile RAMs," in *Proc. of Conf. on Embedded and Real-Time Computing Systems and Applications*, 2009.

[11] J. Kim, Y. Oh, E. Kim, J. Choi, D. Lee, and S. H. Noh, "Disk schedulers for solid state drives," in *Proc. of Conf. on Embedded Software*, 2009.

[12] 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, Feb. 2006.

[13] H.-S. Lee, H.-S. Yun, and D.-H. Lee, "HFTL: hybrid flash translation layer based on hot data identification for flash memory," *IEEE Trans. Consumer Electron.,* vol. 55, no. 4, pp. 2005-2011, Nov. 2009.

[14] S. Jung, Y. Lee, and Y. H. Song, "A process-aware hot/cold identification scheme for flash memory storage systems," *IEEE Trans. Consumer Electron.,* vol. 56, no. 2, pp. 339-347, May. 2010.

[15] Y.-J. Kim, K.-T. Kwon, and J. Kim, "Energy-efficient file placement techniques for heterogeneous mobile storage systems," in *Proc. of Conf. on Embedded Software*, Oct. 2006.

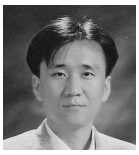
[16] T. Xie and Y. Sun, "PEARL: performance, energy, and reliability balanced dynamic data redistribution for next generation disk arrays," in *Proc. Of IEEE Symp. on Modeling, Analysis and Simulation of Computers and Telecommunication Systems*, Sep. 2008.

[17] E. Pinheiro and R. Bianchini, "Energy conservation techniques for disk array-based servers," in *Proc. of Conf. on Supercomputing*, Jun. 2004.

[18] T. Xie, "SEA: a striping-based energy-aware strategy for data placement in RAID-structured storage systems," " *IEEE Trans. Computers*, vol. 57, no. 6, pp. 748-761, Jun. 2008.

[19] H.T. Kung, F. Luccio, and F.P. Preparata. "On finding the maxima of a set of vectors," *Journal of the ACM*, vol. 22, no. 4, pp. 469-476, 1975.

BIOGRAPHIES



Minyoung Sung (M'03) received his B.S., M.S., and Ph.D. degrees in Computer Engineering from Seoul National University, Korea in 1995, 1997, and 2002, respectively. He is currently an assistant professor in the Department of Computer Software Engineering, Sangmyung University, Korea. Before joining Sangmyung University in 2006, he worked as a senior engineer at the Software Research Institute, Samsung Electronics, Korea, where he conducted research on the architecture and analysis of embedded software. He was also with the Department of EECS at University of Michigan, Ann Arbor as a visiting scholar in 2005. His research interests include embedded systems and power-aware computing.

Kanghee Kim (M'06) received the B.S., M.S., and Ph.D. degrees in computer engineering from Seoul National University, Korea, in 1996, 1998, and 2004, respectively. He is currently an assistant professor in the School of Electronic Engineering, Soongsil University, Korea. Previously, he was a senior engineer in the Mobile Communication Division, Samsung Electronics Co., Ltd. from 2004 to 2009. His current research interests include real-time and embedded systems, operating systems, flash storage systems, and mobile software platforms. He is the corresponding author.