

FTL Design Exploration in Reconfigurable High-Performance SSD for Server Applications

Ji-Yong Shin^{†‡}
Xiong-Fei Cai[†]

Zeng-Lin Xia[†]
Seungryoul Maeng[‡]

Ning-Yi Xu[†]
Rui Gao[†]
Feng-Hsiung Hsu[†]

[†]Hardware Computing Group
Microsoft Research Asia
Beijing, China, 100190
{zlxia, ningyixu, ruigao, xfcai,
fhh}@microsoft.com

[‡]Korea Advanced Institute of
Science and Technology
Daejeon, Korea, 305-701
{jyshin, maeng}
@camars.kaist.ac.kr

ABSTRACT

Solid-state disks (SSDs) are becoming widely used in personal computers and are expected to replace a great portion of magnetic disks in servers and supercomputers. Although many high-speed SSDs are present in the market, both the design of hardware architecture and the details of the flash translation layer (FTL) are not well known. Meanwhile, in the systems requiring high-end storages, specially tuned SSDs can perform better than the generic ones, because the applications in such environment are usually fixed.

Based on the architectural design of our reconfigurable high-performance SSD prototype and by using a trace-driven simulator, we explore the key factors and tradeoffs that must be considered when designing a customized FTL. FTL related issues, such as data allocation, cleaning, and wear leveling, are analyzed in detail presenting suitable design decisions for different workload characteristics. The experimental result shows that the figures for the performance metrics will vary from several percent to more than tens of times among each other depending on the decision made for designing each FTL functionality.

Categories and Subject Descriptors

B.1.4 [Microprogram Design Aids]: Firmware engineering; B.3.3 [Performance Analysis and Design Aids]: Simulation

General Terms

Design, Measurement and Performance

Keywords

NAND flash memory, FTL design, solid state disk, high performance SSD

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ICS'09, June 8–12, 2009, Yorktown Heights, New York, USA.
Copyright 2009 ACM 978-1-60558-498-0/09/06 ...\$5.00.

1. INTRODUCTION

Recent advancement of NAND-flash based solid-state disks (SSD) is expected to impact greatly on the future of storage subsystems [7, 1]. The reason for this is SSDs provide superior features, such as higher performance and lower power consumption, compared to magnetic disks. However, to utilize SSDs a variety of design criteria must be considered. Specifically, design of the flash translation layer (FTL) must be carefully reviewed. Even with the same hardware organizations, the performance and the lifetime of each SSD can significantly differ depending on the FTL design. In addition, the performance of flash memory-based devices is highly workload sensitive [1], making the design decision for FTL more crucial.

In the environments that require high-end storages, such as in servers and supercomputers, applications are likely to be fixed. Thus, FTL design for specific applications will leverage the performance of the system. However, to the best of our knowledge there hasn't been any research publication focusing on the application specific FTL design in SSDs for high-performance environment. In this paper, we analyze the key factors and tradeoffs for customizing FTL regarding workload characteristics in a high performance SSD (HPSSD) environment, and propose guidelines for FTL design decisions.

Efforts have made to utilize NAND flash memories in areas requiring high performance storages. The overall bandwidth of generic SSDs in the market is usually bounded by the host-to-SSD interface, such as SCSI, PATA, and SATA. Thus, multiple chips in the SSDs will usually share a flash channel, or a bus connecting flash chips and flash controller, and the channels will collectively have similar bandwidth to the interface. However, high-end SSDs for enterprise servers [8] take advantage of a host-to-SSD interface with gigabytes of bandwidth. Acceleration platforms using NAND flash chips [27, 18], employ independent channels per flash chips to gain maximum parallelism. Accordingly, we define the HPSSD as SSDs with high-bandwidth host-to-SSD interface as large as several gigabytes per second and tens or hundreds of flash chips and channels that can be parallelized to fully utilize the interface. Yet, in research using the acceleration platforms, very generic FTLs are used, and for most of the SSDs in the market the designs of FTLs are kept as the manufacturers' intellectual properties.

Previous research has focused mostly on FTLs in limited resource environments, such as embedded devices or low cost SSDs. In particular, considering the limited size of random access memory (RAM), the focus has been on log-based block mapping algorithms [20, 21, 14, 24]. Because high-overhead block merge operations are involved, they perform worse than page mapping algorithms. In addition, bandwidth to SSD is often limited, so only a few flash chips on shared channels are employed [1, 17]. Thus, not many ways to manage wear level and to maximize parallelism among a large number of independent chips as in HPSSDs are studied.

Throughout our work we have discovered some tradeoffs in designing FTLs for HPSSDs. The experimental results show that, different from the embedded flash storage environments utilizing few flash chips, tradeoff lies between gaining high performance and reducing the number of erase operations for longer lifetime of SSDs. Introducing wide striping-based allocation in unit of page or greedy-based dynamic methods may maximize parallelism, but high locality of write, which reduces the number of erase operations, can be achieved when large striping units than a page size are used for allocations. By introducing a novel concept of wear leveling cluster we also have verified the tradeoffs between achieving even wear level in a large area and gaining high performance.

Considering the workload characteristics, sequential IO intensive workloads benefits from wide striping based allocation methods, while random IO intensive workloads run better on dynamic based allocations. When the data and requests are distributed evenly within a SSD, hot/cold data separation can help reduce the cleaning overhead. When data are skewed, however, proposed load balancing can resolve the situation with low overhead. The experimental results collectively show that the workload characteristics and the corresponding design decisions for each FTL functionality changes the figures of the performance metrics from several percent to more than thousands of percent.

The rest of the paper is organized as follows. Background information about NAND flash memory, reconfigurable HPSSD (RHPSSD), which provides the base architecture for HPSSD in this paper, and related works will be introduced in Section 2. Section 3 will provide the exploration of design considerations in FTL. Section 4 will present the evaluation work regarding the discussions in the previous section. Our study will be summarized in Section 5, and finally a conclusion will be given in Section 6.

2. BACKGROUND AND RELATED WORK

2.1 Characteristics of NAND Flash Memory

NAND Flash memory consists of multiple blocks which are composed of pages. Flash blocks must be erased before data is written and the blocks wear out as they are being erased. The unit of read and write operations is a page while that of erase is a block, with the latencies for each operation differing depending on the manufacturer.

There are two types of NAND flash memories: single level cell (SLC) and multi level cell (MLC). The capacity of MLC doubles that of SLC, but SLC has lower operational latency and higher endurance. The erase count that a block in SLC can endure is typically 100K, while it is 10K in MLC.

In this paper, we used a configuration of SLC NAND flash

Operation	Time required
Page read to register	25 μ s
Page write from register	250 μ s
Block erase	500 μ s
Page register access through flash channel	100 μ s

Table 1: Timing parameters for NAND flash chips

chips in Table 1. SLC chips are used to take advantage of the high performance and endurance in IO intensive environments.

2.2 NAND Flash Chip Organization and Operations

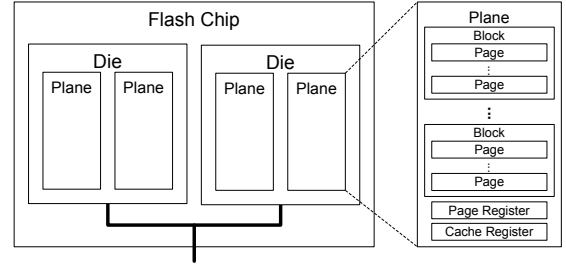


Figure 1: Flash chip organization

The capacities of flash chips used in this paper are 4GB and 8GB. Although they share the same latencies for the basic operations, the internal organizations of the chips are different. Figure 1 shows the general organization of a flash chip. A NAND flash chip consists of dies which contains several planes. Within a die planes can be interleaved only when the same types of commands are issued together. A die is an independent unit that has its own ready/busy and chip-enable signals. The dies in a chip can execute different operations from each other in an interleaved manner. The difference between the two chips is that the 4GB chip consists of 2 dies while the 8GB chip consists of 4 dies. The rest of the organizational settings used in this paper are summarized in Table 2.

For most of NAND flash chips interleaved plane and die commands, and read and write for copy commands are supported. The interleaved commands provide parallelism within a chip, and the read and write for copy commands enable a page to be copied within a plane without utilizing the flash channel.

	4GB Chip	8GB Chip
Page size	4KB	4KB
Pages per block	64	64
Blocks per plane	4096	4096
Planes per die	2	2
Dies per chip	2	4

Table 2: Internal organizational settings of NAND flash chips

Open NAND flash interface (ONFI) 2.0 [23] compliant flash chips provide cache mode operations using cache registers. If each plane has only one page register, flash channel operation and plane access operation cannot be pipelined, because the page register must be exclusively available to the either operation. However, introducing the cache register

and the cache mode commands enables the both operations to be executed independently.

2.3 Role of FTL

The primary role of FTL is to provide abstraction of the flash memory characteristics to the host machine: maintaining an address mapping table, cleaning the blocks for reuse, and conducting wear leveling are some of the basic functions. As employing multiple NAND flash chips as in SSDs becomes common, FTL should also consider the parallelism and the load balancing of requests among flash modules such as chips, dies, and planes.

2.4 Reconfigurable High Performance SSD

The discussion about the FTL design in the rest of the paper will be based on the architecture of the RHPSSD prototype that we have designed and are currently testing.

In our RHPSSD, PCI Express is used as the host-to-SSD interface. A total of 4GB/s uplink and downlink provides sufficient bandwidth to fully utilize tens of flash chips in parallel. The high level view of the SSD prototype is shown in Figure 2. The SSD has a FPGA between the PCI Express bus and the flash channels. The FPGA is where the FTL and the flash channel controllers are implemented. FTL can be implemented and utilized as software by adding a general purpose processor on the FPGA or as hardware by using HDL code. This flexibility makes our prototype reconfigurable and facilitates the FTL to be customized. The flash-channel controllers are independently implemented in FPGA for each channel to maximize parallelism. Each flash chip possesses an independent flash channel and is placed on a daughter board that can be disconnected from the main board of the RHPSSD. There are more than 30 flash chips on the daughter board and the bandwidth between the daughter and the main board is sufficient enough to utilize all the flash channels at the same time. RHPSSD also has a large amount of RAM that can be utilized by the FTL. Although the RAM can be used as cache buffers, we leave using it as the buffer for future work.

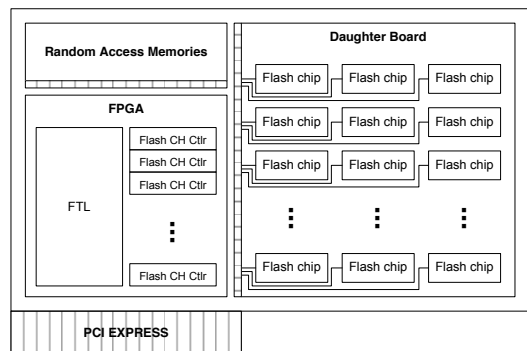


Figure 2: High-level view of reconfigurable high-performance SSD

Compared to the generic SSDs the performance of RHPSSD will be greatly affected by the algorithms used to keep the chips busy. Wear leveling is considered more important than in the generic ones, because RHPSSD is likely to be placed under IO intensive environments. Additionally, maintaining even wear level among flash modules is

important, because each module contributes to the parallel performance.

Although the RHPSSD might not be the standard design for HPSSDs, we expect that most HPSSDs will have similar high level organizations.

2.5 Related Work

Agrawal et al. [1] have studied the tradeoffs that lie in SSDs focusing on the different organization of flash chip arrays and corresponding basic FTL issues. However, we fix the SSD architecture to the extreme to gain the highest performance, and concentrate on the detailed analysis and tradeoffs in each FTL functionality considering the workload characteristics.

Park et al. [24] proposed a dynamically reconfigurable FTL in a block mapping scheme, but the focus of the work was reducing the number of erase operations using different block mapping algorithms without considering the multi-chip environments. In this paper, page mapping is used, FTL related issues in multi-chip environments are covered more comprehensively, and concerns of statically reconfiguring FTL are studied.

Data allocation to multiple chips in SSDs are very similar to the issues studied in the RAID [25] system. The adaptive striping method for a flash chip with banks, or dies employing one plane each, has been investigated by Chang and Kuo [4]. Different from our paper the investigation involved only one chip with multiple banks and posed passive measures to deal with load imbalance among banks.

There were also approaches to optimize the array of disks in accordance with workload characteristics, such as in HP AutoRAID [31] and by using RAID configuration tools [33]. Yet, SSDs do not have moving mechanical parts, and cleaning and wear leveling operations distinguishes the factors that must be considered.

3. EXPLORATION OF DESIGN CONSIDERATIONS FOR SPECIALIZED FTL

3.1 Page Mapping Algorithms

Taking advantage of abundant RAM in the RHPSSD architecture, we focus on page mapping methods. Logical pages in flash memories are not stored in a fixed position. A Concept of allocation pool is used to limit the assignable physical locations of logical pages to physical flash modules. Within the allocation pools logical pages are assigned to physical pages in runtime. Depending on the size of the allocation pool and the allocation policy the performance of SSD can vary.

Because a plane is the smallest unit that can process a page request, we set the minimum allocation pool as plane and the maximum as SSD in this paper. Thus, we define allocating page addresses statically to planes as static allocation and using larger allocation pools as dynamic allocation since a logical page can be allocated to any plane within the pool in runtime.

3.1.1 Static Allocation

Utilizing a static allocation can be very convenient if the allocation can distribute the requests evenly to each plane, because flash modules will be automatically parallelized and the wear levels among the modules will be balanced. However, an improperly skewed distribution of requests to few

flash modules can rather degrade the overall performance. Organization and properties of flash chips, and workload characteristics must be considered before determining how to allocate the addresses. Taking these into account, there can be a variety of striping-based allocation methods. Although the plane is the unit that will be in charge of processing each stripe, striping the addresses to planes can be carried out in different sequences presenting different performances.

Striping the address in a fine grain manner to gain parallelism may seem to result in good performance, but considering the wear leveling and cleaning operation the choice may not always be optimal. A small striping unit may not benefit from spatial locality within blocks and result in more valid page migrations during cleaning operations. This can again do harm to the lifetime of SSD because increased page migrations causes more erase operations.

3.1.2 Dynamic Allocation

As studied earlier in the RAID systems, workloads with small and random IO access patterns will not benefit much from using static striping-based allocation, because small requests cannot usually take advantage of parallel operations and the requests may be skewed to a small number of disks [22, 11, 9]. Dynamically allocating the requests to storages can be a potential solution, but again the amount of data in each storage location can be imbalanced. For example, if write requests are skewed to only a few storage devices, the devices will be full and no longer capable of receiving new data. Then the storage system will lose parallelism and from a SSD point of view the wear leveling will also be harmed.

One of the easiest ways to avoid the load imbalance in dynamic allocations is migrating data to storage devices with relatively more free space [28]. For SSDs migrating cold pages to the flash module with a small number of valid pages will balance the number of hot and cold data in each flash module and will likely benefit the wear leveling at the same time.

Considering planes as disks the flash modules in an allocation pool can be considered as an IO system in charge of distributing the request dynamically to planes. When the allocation pool is set to a large unit it will have more flexibility to assign the requests, but unless a larger degree of freedom is managed in a very intelligent way high performance will not always be guaranteed.

3.2 Cleaning and Hot/Cold Data Separation

Because the erase operation is conducted in a unit of blocks, valid pages in a block to be erased must be moved to another location. Thus, an important consideration for cleaning operations is to minimize the operational latency by reducing the number of page migrations so as not to delay the IO requests.

Selecting the block with the minimum number of valid pages for cleaning is one of the most commonly used cleaning policies. It would be the best if we can find blocks with no valid pages, but it is not always the case. However, separating hot and cold data can facilitate reducing the number of page migrations [6, 12, 19]: blocks with only hot pages are likely to be full of invalid pages in the near future, while blocks with only cold pages are likely to maintain the number of valid pages.

However, most methods for identifying data's hotness or coldness are on the basis of history based prediction. They work well with most cases, but depending on the workload characteristics or the allocation policies the prediction may lead to misclassification of data, further degrading the cleaning performance.

3.3 Wear leveling

Wear leveling and cleaning operations cannot be separated from each other, because cleaning will wear out the blocks. Consequently, wear leveling algorithms are commonly triggered during cleaning or write operations [16, 5, 3]. There have also been studies on triggering wear leveling algorithms periodically [32], but it is not desirable to use periodic measures in environments, where it is difficult to predict when the flash modules will be idle.

Following Pareto's 80/20 rule, there are usually blocks of data that have rare access frequencies. Wear leveling usually involves migrating this data to the blocks with less remaining lifetime. Because migrating a block of pages requires tens of reads and writes, it can degrade the IO response time.

Here we introduce a concept of wear leveling cluster. The cluster is similar to the allocation pool. The wear leveling algorithms will balance the lifetime of blocks only within the same cluster. Thus, the remaining lifetime of blocks from different cluster can have large gaps. When the cluster is large, the lifetime of blocks will be distributed evenly in SSD, but the maintenance overhead can be larger than using smaller cluster.

The wear leveling cluster must be smaller than or equal to the allocation pool, because logical pages cannot be moved to the outside of the allocation pool. Therefore, when static allocations are used, the wear level of different flash modules can vary greatly, so selecting the static allocation with an even distribution of erase operations is necessary. In contrast, the size of the cluster can be selected for dynamic allocations depending on the size of allocation pool. The performance and wear leveling goal of the SSD must be considered to determine the size of the cluster.

4. IMPLEMENTATION AND EVALUATION

In this section simulation-based evaluation of the issues discussed in the previous section and a more detailed analysis of FTL matters will be presented.

4.1 Simulation Environment

Our simulation environment was built after extensive modification of the DiskSim [10] based SSD simulation plugin [26]. Cache mode operation, a module aging flash chips, and various FTL functionalities that are discussed in the following sections were added. The aging module carries out write requests similar to each workload's IO pattern until the cleaning operation is triggered in all planes before simulation starts. This module enables us to simulate an aged SSD, where there are few clean blocks to write and requires cleaning operations to secure writable blocks. Although SSD without aging shows better performance, the aging configuration reflects a more realistic situation.

The base FTL follows the design of [2]. Other fixed configurations besides the latencies in Table 1 are summarized in Table 3. The amount of over-provisioned region and cleaning thresholds were fixed because the effect of controlling the former was already studied [1], and the latter does not

	Postmark (22GB)			IOzone (16GB)			WebDC (234GB)		
	Overall	Read	Write	Overall	Read	Write	Overall	Read	Write
IOPS	126.944	105.581	21.363	102.058	45.129	56.928	331.085	190.934	140.152
Avg. inter arrival time (ms)	8.006	9.625	41.798	6.752	15.502	11.958	3.032	5.247	7.163
Total data requested*	27,682K	17,455K	10,227K	642,110K	414,512K	227,598K	382,528K	161,109K	221,419K
Data request per millisecond*	56.444	35.591	20.853	54.961	35.480	19.481	42.131	17.744	24.387
Avg. req size*	444.635	337.092	976.150	538.527	786.180	342.203	127.251	92.934	174.002

	TPC-C (154GB)			SQL (223GB)			Exchange (478GB)		
	Overall	Read	Write	Overall	Read	Write	Overall	Read	Write
IOPS	3,635.196	2,348.523	1286.673	61.872	39.617	22.255	90.519	49.740	40.779
Avg. inter arrival time (ms)	0.275	0.426	0.777	20.904	32.295	58.117	11.056	20.110	24.542
Total data requested*	112,108K	41,483K	70,625K	37,939K	11,230K	26,709K	1,836K	835K	1,001K
Data request per millisecond*	59.647	22.071	37.576	12.388	3.667	8.721	2.039	0.927	1.112
Avg. req size*	16.408	9.398	29.204	200.221	92.560	391.876	22.525	18.638	27.268

* in unit of number of 512B disk sectors

Table 4: Workload characteristics

Parameter	Value
Over-provisioning	13%
Cleaning threshold	5%
Use of cache mode operation	true
Use of read and write for copy operation	true
Use of interleaved plane operation	true
Use of interleaved die operation	true
Flash chips per flash channel	1
Number of flash channels and chips per SSD	36

Table 3: Fixed configurations of SSD in simulations

affect the performance much when SSD is aged. For high performance, the special operations supported by the chip are used as default when they can be issued. The SSDs are configured to hold 36 chips with independent channels. Other varying configurations will be introduced in each of the following subsections.

4.2 Workload Analysis

The disk traces of workloads used in the experiments are as follows: Postmark [15], IOzone [13], WebDC, TPC-C [30], SQL, and Exchange. Postmark, IOzone, TPC-C, and Exchange traces are the same ones used [1], which we received from the authors. Postmark and IOzone are IO-intensive file system benchmarks, where the traces were gathered by running the benchmarks on workstation-class PCs. TPC-C is a database benchmark for comparing the performance of online transaction processing and Exchange is a workload collected from a Microsoft Exchange server. WebDC is taken from running a highly IO-concentrated algorithm used to classify massive data collected from the web. SQL was obtained by running SQLIOSim [29], a software simulating IO behavior of Microsoft SQL Server 2005. SQLIOSim was configured to simulate a server with over 200 users on a 40GB database.

Some of the characteristics of the traces are summarized in Table 4. The amount of the minimum address spaces necessary for the workloads are shown next to their names. Depending on the address space required, we used the configuration of one 144GB SSD with 4GB chips, one 288GB SSD with 8GB chips, and two 288GB SSDs with 8GB chips.

IO operations per second (IOPS) of each trace and the amount of data requested per millisecond show how much IO stress leans toward the storage. Postmark, IOzone, WebDC, and TPC-C show highly IO intensive behavior compared to SQL and Exchange. Considering the request size, TPC-C and Exchange are random workloads. SQL shows a similar proportion of random and sequential IO requests and the

other workloads are mostly dominated by sequential IOs. Although not described in the table, the requests in IOzone shows a very high locality compared to others. The majority of the logical pages in IOzone are accessed more than 32 to 40 times while those in others are accessed less than 16 to 32 times.

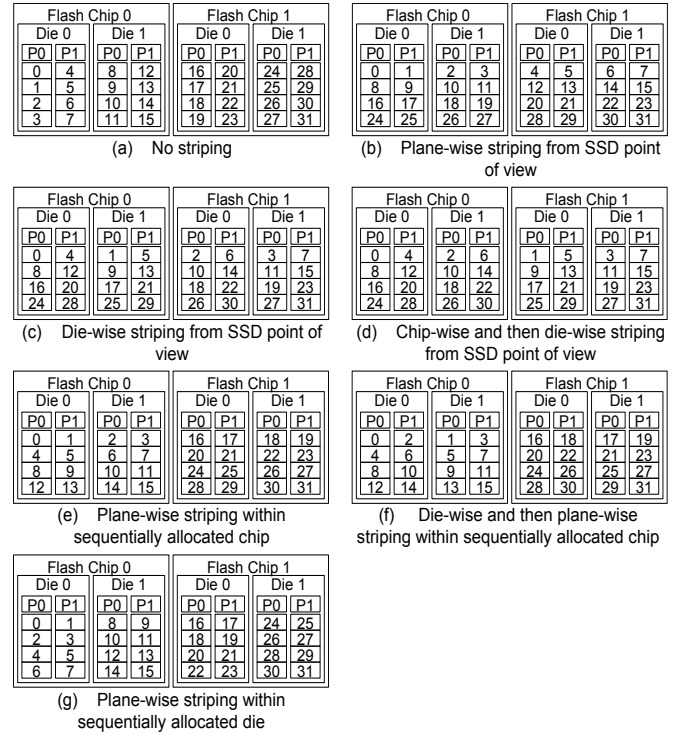


Figure 3: Striping-based static allocation methods

4.3 Different Allocation Methods

4.3.1 Striping-Based Static Allocations

For each workload, we tested seven different static allocations described in Figure 3. All the methods are based on striping except for (a), which simply allocates logical page addresses sequentially to each plane. (b), (c), and (d) are striping the addresses from a viewpoint of SSD: (b) is striping addresses to each plane not considering the die or chip; (c) is striping to dies first and again to each plane; and (d) is striping to each chip first, then to dies and finally to planes.

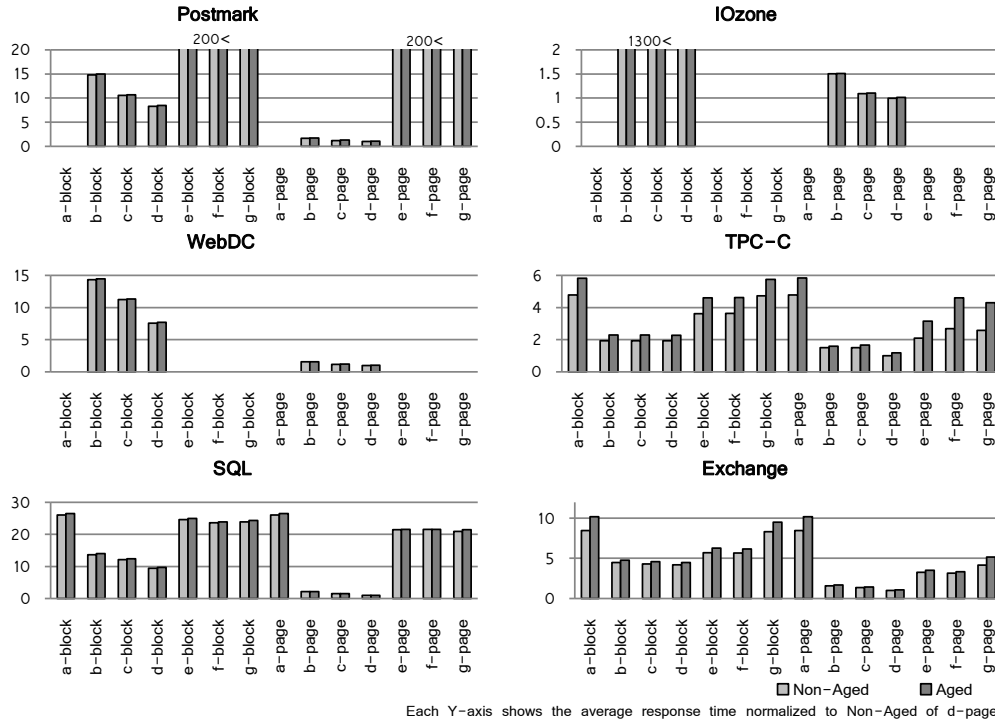


Figure 4: Average response time of static allocation

(e) and (g) are stripings based on chips: when continuous addresses are allocated to each chip, they will be allocated internally in the striping-manner as in (b) and (c). Finally, (g) is again allocating sequential addresses to each die and striping to each plane. We will call (b), (c), and (d) as wide striping and (a), (e), (f), and (g) as narrow striping for convenience.

Page and block are used as striping units when allocating the addresses. Thus, the numbers in the figure can be logical page addresses, or logical block numbers depending on which unit we use. According to the base FTL configurations [2], the logical page address and the block number are determined by the following formulas:

$$\text{logical page address} = \frac{\text{logical sector number}}{(\text{page size} / \text{disk sector size})}$$

$$\text{logical block number} = \frac{\text{logical page address}}{(\text{block size} / \text{page size}) - 1}$$

The order of the pages within a plane can change, because the numbers only indicates in which plane the page addresses are allocated to.

The response times of the allocations for the workloads are shown in Figure 4 in a normalized form to non-aged of d-page, which performs the best for all workloads. We ran the traces with and without the aging process. The entry without any bar indicates that we have failed to run the trace due to the saturation of the IO queue. The page and block in the figure shows the striping unit used for the allocation, and the alphabet in the front indicates the allocation method. The numbers above the bars that reached the top shows that the bars originally points to the values of the Y-axis that are larger than the numbers.

For all cases, narrow stripings perform notably worse than wide stripings, and using the striping unit of block for initial

allocation performs worse than using that of page. This is clearer for workloads with sequential IO and high data rates, such as Postmark, IOzone, and WebDC.

If continuous page addresses are allocated to flash modules, page requests can be concentrated to few modules. The standard deviations for the number of requests that each flash module received explain this. Except for TPC-C and Exchange, the standard deviations from narrow stripings are ten to several hundred times as large as that of wide stripings and this applies the same to using different striping units.

When SSDs are aged the gap becomes larger because of the erase operations entailed. Similarly, IOzone failed to run on narrow stripings because of cleaning operations from frequent data update, while Postmark, which has more data writes per millisecond, has succeeded.

The response time of different allocation methods in TPC-C and Exchange do not have larger gaps than the others. This is because the workloads inherently consist of small random requests. The requests will be distributed evenly to flash modules even when continuous addresses are mapped to the modules. However, this randomness makes the workloads take less advantage of the striping based allocations.

The standard deviations for request distribution of wide stripings are very similar to each other in all workloads. However, (d) performs the best for all cases because it utilizes the maximum number of independent modules to process an IO request. Yet, in a comparison with dynamic allocations and considering the cleaning and wear leveling, (d) allocation may not be the ultimate optimal choice, which will be shown in the later sections.

4.3.2 Hot/Cold Data Separation

To test the impact of separating hot and cold data pages, the dynamic data clustering (DAC) [6] algorithm is modified

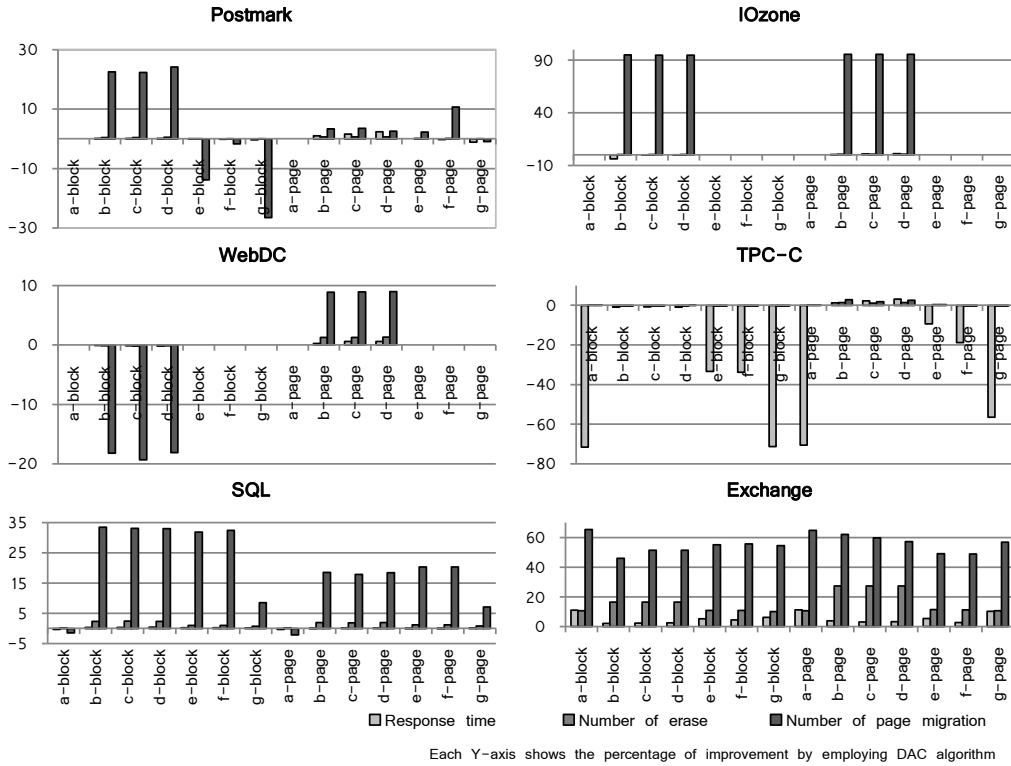


Figure 5: Improvement from hot and cold data separation

and implemented for SSDs. Different from other algorithms, DAC classifies data into multiple degrees of temperatures. The classification of data takes place within each plane and the temperatures for the pages are traced globally in SSDs. The simulation was conducted in aged SSDs and the experimental result in Figure 5 shows the improvements on the average response time, number of erase operations, and number of valid page migrations in percentages when using the classification.

SQL, Exchange, and IOzone benefit more from hot and cold data separation for most of the cases compared to the others. SQL and Exchange workloads have very few logical pages that are written more than once while more than 90 percent of pages in IOzone are overwritten more than 40 times. Therefore, hot data is easily distinguished from rarely accessed pages, and they can benefit much from the data separation.

Postmark, WebDC and SQL exhibit an increased number of page migrations and a slightly worsened response time when continuous addresses are concentrated in flash modules. In these cases there are higher possibilities for misclassification because it is difficult to perceive the changing temperature over time when many requests are mixed. Consequently, it is important to spread the hot and cold data evenly to all flash modules. This is why all cases in the wide stripings using page-sized unit show improvements.

Although there aren't any dramatic changes in the number of erase and migration operations, the average response time in most of allocations of TPC-C is degraded. The IO intensiveness and the small request patterns make TPC-C the workload with the largest write-to-cleaning ratio among all workloads. Moreover, the logical addresses in TPC-C are accessed almost uniformly over time making it difficult

to capture the temperature of pages. These characteristics and misclassification unevenly distributed the erase and page migration, with the result that the response time was increased.

4.3.3 Dynamic Allocation and Load Balancing

We first propose greedy-based dynamic allocation and load balancing methods, and then evaluate them using various initial data allocations.

Dynamic Allocation and Load Balancing Methods. As observed in Section 4.3.1, utilizing as many independent SSD parts when processing a request leads to a small response time. It would be best if the dynamic allocation methods utilize independent SSD parts as in the best-performed static allocation. However, if requests arrive when all the dies are busy, it is not clear which would be better; whether to issue the requests as soon as a die becomes idle or to wait and issue them to multiple dies.

Thus, we propose an operational-latency-based estimation method. To dynamically issue the requests to maximize parallelism, we maintain a request queue storing groups of interleaved plane requests and an estimated time value for each die. We sum up the flash channel and memory cell access latencies and add the value to the estimated time when inserting a group of requests to the queue.

Based on the estimated time, page requests are greedily assigned to the dies with the smallest estimated time. When the estimated time values among dies are the same, first the chip and then the die will be selected in a round-robin manner similar to the static allocation (d) of the page-sized striping unit.

As previously stated, once a die is selected, the request will be assigned to a plane in a round robin manner. When requests are assigned to all the planes in a die, a group

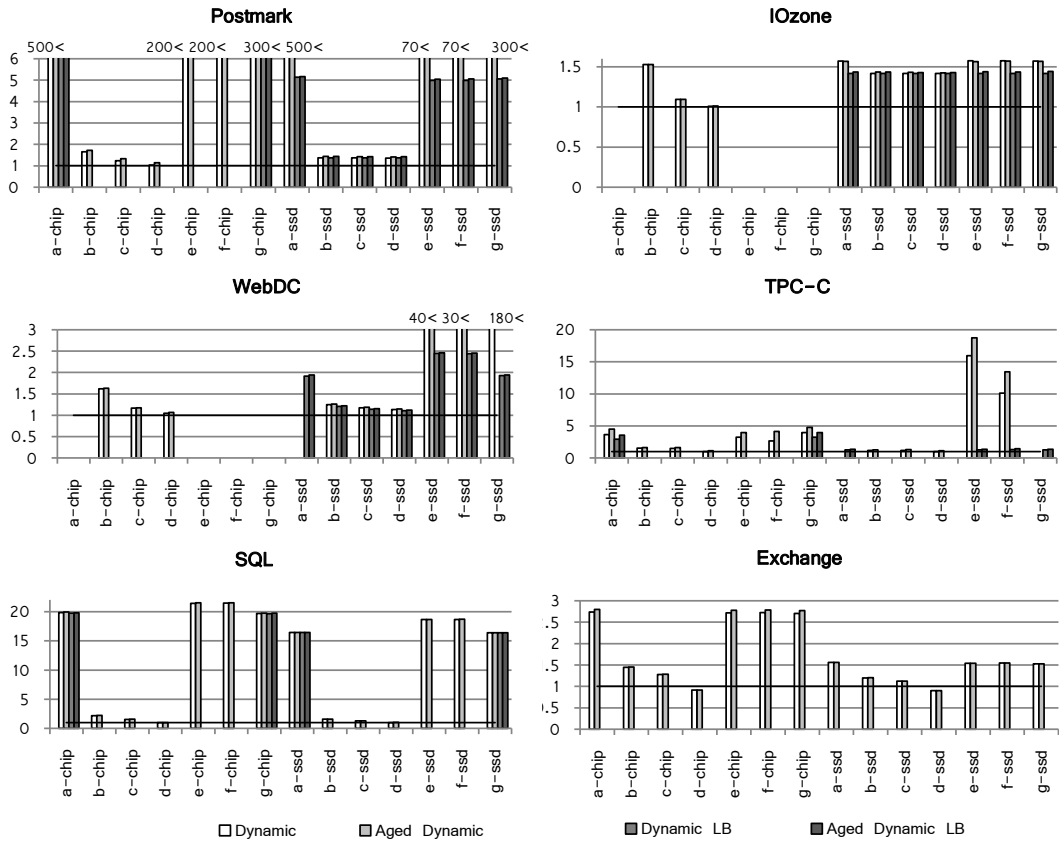


Figure 6: Average response time of dynamic allocation

of interleaved requests is formed and will be added to the queue. When the requests are finished the estimated time will be decreased.

We use the SSD or chip as the allocation pool in dynamic allocation. When the chip is the allocation pool addresses are allocated statically to the chips in one of the ways in Figure 3, and the write requests are issued using the same algorithm without chip selection.

To actively react on the skew of data, load balancing is used. When the number of valid pages that are assigned to a plane exceeds a certain threshold, a block from the lowest temperature group of the DAC algorithm is migrated to the plane with the smallest number of valid pages. The migration takes place only in the same allocation pool and the block that contained cold pages will be erased after the migration. So as not to disturb the IO requests the process will be carried out as a background operations.

Performance of Dynamic Allocation and Load Balancing. In the experiments we used the allocations in Figure 3 employing a page-sized striping unit as initial data placement. Hot/cold separation is used as default, because it reduced the erase and page migration, and affected the response time less than 1 percent for the majority of the cases in dynamic allocations.

Figure 6 demonstrates the average response time for each run normalized to non-aged of the static d-page allocation in Figure 4. Chip and SSD in the figure indicate the allocation pool, the alphabets shows the initial data placement, and LB means load balancing is used. When the number of valid pages of a flash module exceeds the load balancing threshold, the module will not become a write candidate unless load balancing is used. We configure the threshold as 91 percent

of the total number of physical pages in a plane considering the middle point of percentages for an over-provisioned region and cleaning threshold. Non-aged dynamic LB or aged dynamic LB without any bar in the figure are the cases, where no load balancing was triggered.

		Overall	Read	Write
Postmark	static	26.24	20.11	20.71
	dynamic	123.09	55.43	86.59
IOzone	static	1860.26	91.41	1844.26
	dynamic	1638.82	745.45	1255.28
WebDC	static	13571.13	10087.50	3642.22
	dynamic	33308.77	18673.43	14655.24
TPC-C	static	2861.02	1800.98	1065.56
	dynamic	1200.72	1201.19	48.31
SQL	static	576.88	517.60	94.82
	dynamic	506.02	465.13	84.70
Exchange	static	1233.18	618.59	615.57
	dynamic	145.13	73.35	76.54

Table 5: Standard deviation of number of requests per die

Workloads with a low request arrival rate or random IO access pattern, such as TPC-C, SQL, and Exchange, tend to work well with dynamic allocations. D-ssd of TPC-C and Exchange, and d-chip of SQL and Exchange, which effectively parallelized random IOs, showed better performance than the static allocations regardless of aging. The standard deviations of the number of requests processed by each die in the best performed static and the dynamic allocations are shown in Table 5. The table verifies that it is better to use dynamic allocation to evenly utilize the parallel parts in

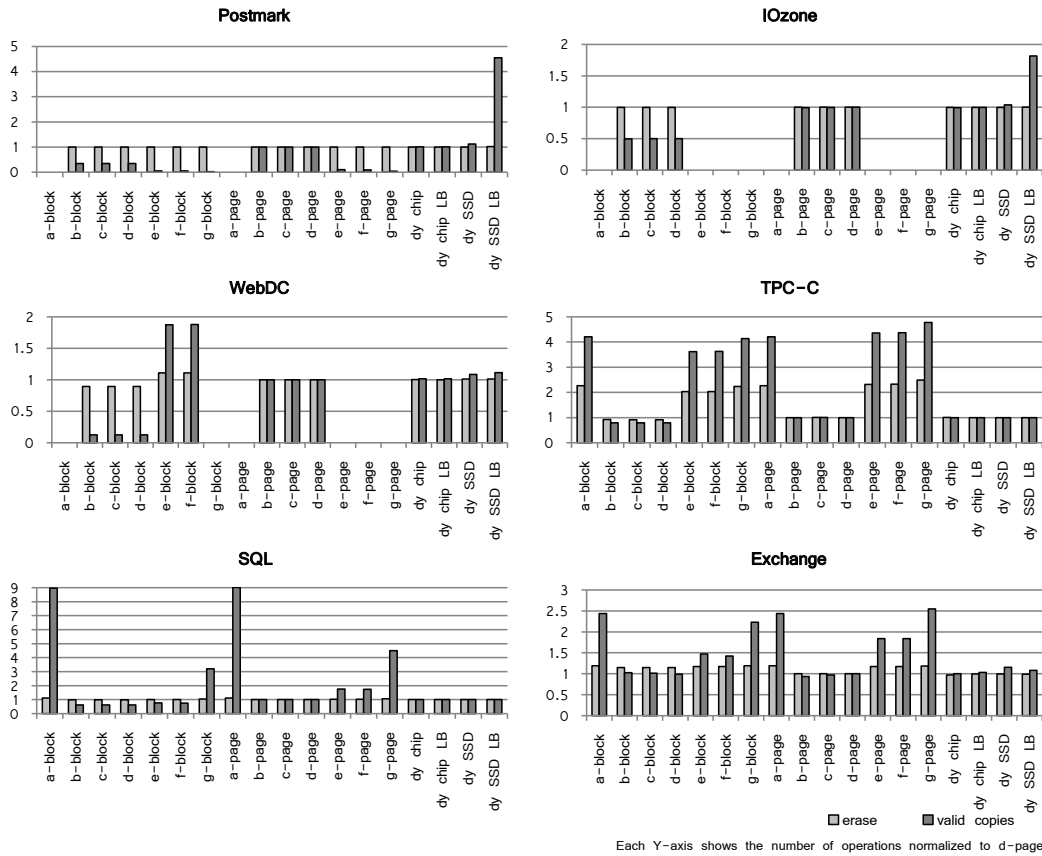


Figure 7: Number of erase and valid page copying

SSD for workloads that consist of a large amount of random IOs.

When allocation pool of chip is used we can see that it partially benefits from statically allocating addresses to each chip; allocation pool of chip shows a similar trend of response time to static allocations more than that of SSD does. Because SQL has a similar ratio of sequential and random IOs, it shows the best performance for this case. Sequential IOs benefit from statically allocating addresses to chip and random IO benefits from dynamically allocating addresses to plane creating a synergy effect.

The response time of the SSD allocation pool does not vary among different initial data placement as much as in using chip allocation pools. Additionally, improved performance are mostly presented for initial data placement using narrow stripings compared to utilizing chip allocation pools.

Load balancing was not triggered for TPC-C, SQL, and Exchange unless large chunks of continuous addresses were allocated to each flash module. This again shows that dynamic allocation methods spread the addresses evenly to each flash modules when request arrival rate is low or small requests are dominant.

For Postmark, IOzone, and WebDC load balancing took place in all SSD allocation pools. Comparing the dynamic and dynamic LB of initial data placement using wide stripings we can see that the overhead of load balancing is not large. Yet, as shown in Table 5 and Figure 6, static allocation is a better choice from a performance point of view for workloads with dominant sequential IOs.

SSD allocation pool using narrow striping as initial data

placement in Postmark, IOzone, and TPC-C demonstrates the importance of load balancing. When load balancing is not employed, some flash modules were not utilized well. Employing load balancing solves this problem and performs several tens of times better.

4.4 Cleaning and Wear Leveling Analysis

Efforts have been made to reduce the cleaning operation by introducing hot/cold data separation, but again the number after the reduction and the distribution of erase operation among flash modules must be considered for wear leveling. We only use the simulation results with aging and hot/cold data separation in this section.

4.4.1 Cleaning Analysis

Although some allocations showed smaller response time than others, the number and the distribution of erase and valid page copy operations are a different story. The number of erase operations and valid page copying operations for dynamic allocations with the smallest response time and all static allocations are in Figure 7.

Because of the locality of data pages, static allocations using a block-sized striping unit shows a similar or lesser number of erase and page copying compared to using page-sized striping unit for most cases. However, when too many continuous addresses are concentrated as in narrow stripings, the benefit from the locality rather decreases by the skewed requests. Nevertheless, Postmark benefits from narrow stripings, because it has the largest average write request size compared to the others.

Meanwhile, Figure 7 and the response times from other figures show that parallelizing the IO requests are more relevant to the response time than decreasing the number of erase operations when there are many flash modules.

For dynamic allocation methods the number of erase operations differs by a maximum of 2 percent from d-page and the number of page copying is also similar. However, when load balancing is triggered the number of copying operations increases notably. This is because the load balancing operation includes copying a block of data. Dy SSD LB of Postmark, IOzone, and WebDC in the figure are the ones that the load balancing took place. Yet, comparing the dynamic allocations with and without load balancing, we can see that the erase operation during load balancing does not greatly increase the total number of erases. Erasing blocks during load balancing prevents erase operation that would have happened in the future making the total number of erases even. In addition, different from the copying process triggered during the regular cleaning operation load balancing creates a clean block while reducing the number of valid pages in a plane with skewed requests. This helps reducing the number blocks, which are fragmented with valid and invalid pages and have potentials to trigger additional cleaning operations in the future.

4.4.2 Wear Leveling Analysis

The distributions of erase operations in planes without wear leveling is shown in Figure 8. Only the three best performed static allocations with different striping units, and the best performed dynamic allocations of the two allocation pools with load balancing, are placed in the figure.

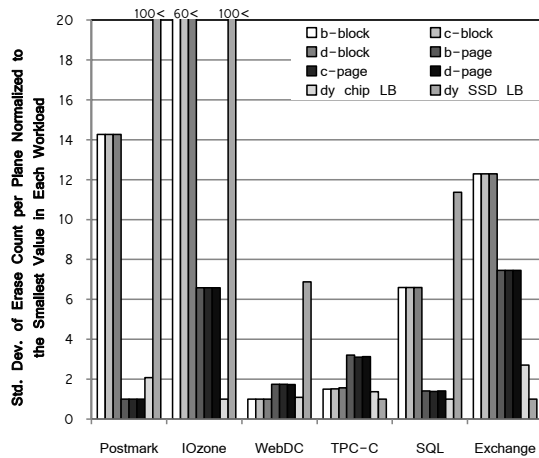


Figure 8: Standard deviation of erase operation per plane

The trend presented in the figure is reflected by the distribution of write requests. For static allocations the standard deviation of the average lifetime of each plane follows the same trend in the figure even with wear leveling. The number of erase operations in Figure 7 and the distribution must be thoroughly considered when choosing a static allocation method. Because the wear leveling cannot be conducted across planes, selecting the allocation with the most even distribution or alternatively choosing dynamic allocations should be considered depending on the target environment.

While any existing wear leveling algorithms can be used within the allocation pools, balancing the wear level among the allocation pools cannot be achieved by such algorithms. We first propose a wear leveling algorithm utilizing the DAC algorithm and based on it we explore the further issues.

Wear Leveling Algorithm. According to our proposal, the wear leveling takes place only during the cleaning operation, where the blocks with the largest number of invalid pages are selected for erased. As in [16, 3] we use the difference between the remaining lifetime of the block selected for erase and the largest remaining lifetime of a block in the wear leveling cluster as the trigger for wear leveling. If the difference is larger than or equal to a given threshold, a cold block from the lowest temperature region of the DAC algorithm will be copied to the block selected for erase. The block from the cold region will then be erased. To minimize the overhead, wear leveling is conducted as a background job like load balancing.

Wear Leveling in Different Clusters. We set the lifetime of each block to 50 and test the wear leveling algorithm using 2 as the threshold to trigger the algorithm. However, even with aging process the total number of erases is not large enough to study wear leveling for most traces, except for IOzone.

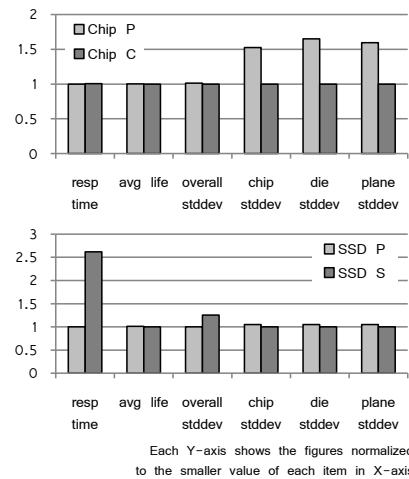


Figure 9: Wear leveling using different cluster

Therefore, to explore the effect of choosing different wear leveling cluster in dynamic allocations, the IOzone workload is used. When wear leveling was not employed, the average remaining lifetime of all blocks in SSD was around 48.6 and the standard deviation was over 19 for all allocations. After wear leveling was utilized the average lifetime became around 47.3 and the standard deviation dropped to less than 1.3. Because the proposed algorithm is a simple modification of general existing wear leveling methodologies, the numbers can be considered as indications for the effectiveness of the algorithm. Figure 9 shows the average and the standard deviation of the remaining lifetime of all blocks in SSD, and the standard deviation for the average remaining lifetime of blocks in chip, die, and plane in a normalized form for dynamic allocation using chip and SSD allocation pools. We have tested the clusters as large as a plane and the selected allocation pool for each dynamic allocations. P, C, and S stands for the size of the cluster which are plane, chip, and SSD respectively.

FTL design choices	Best conditions to use	Pros	Cons
Static allocation (b, c, d)	Dominant sequential IO requests	High parallelism, Even distribution of requests	Less benefit for random IO, uncontrollable wear level among flash modules
Static allocation (a, e, f, g)	No benefit for most cases	N.A.	Bad figures for most of evaluation metrics
Page striping unit	High IO rate	Utilizes many flash modules in parallel	Small advantage from data locality increasing cleaning operation
Block striping unit	Low IO rate	Small number of cleaning operations	Uneven request distribution increasing response time
Dynamic allocation with chip allocation pool	Similar ratio of random and sequential IO	Parallelism for both sequential and random IOs, moderate performance for most cases	Potential for data and request skew
Dynamic allocation with SSD allocation pool	Dominant random IO requests	High parallelism for random IO, Even distribution of random requests	Potential for data and request skew, relatively worse performance for sequential IOs
Load balancing	Skewed IO request to few flash modules	Even utilization of flash modules	Increased page migration
Hot/Cold Separation	Evenly distributed IO request and hot/cold data	Less erase, and page copying	Misclassification of data degrades performance
Large wear leveling cluster	Unevenly distributed IO and erase requests, requirements for even wear level throughout SSD	Even wear level throughout large cluster	Larger overhead and response time
Small wear leveling cluster	Evenly distributed IO and erase requests, Requirements for small IO response time	Small overhead for wear leveling	Potential for uneven wear level among flash modules

Table 6: Summary of FTL exploration and tradeoffs

As shown in Figure 8, the erase distribution for utilizing chip allocation pool in IOzone is very even while that for utilizing SSD as the pool is the opposite. We can see in Figure 9 that the response time is larger and the average remaining lifetime is smaller when using larger cluster than using a smaller one. This is because the larger a cluster is the more the wear leveling is triggered to balance the larger number of blocks. However, when erase operations are evenly distributed, as in the chip allocation pool, the overhead for using large cluster is relatively small.

The standard deviation of average the lifetime in each flash module in Figure 9 demonstrates that using small clusters will create larger gaps on the remaining lifetime among the modules compared to using larger clusters. Although the overall standard deviation of SSD S is large because it covers larger area, the value is expected to be maintained while that of SSD P is expected to grow larger in the future.

5. PUTTING IT ALL TOGETHER

Using six different workloads we have tested various FTL issues including data allocation, hot/cold data separation, and wear leveling in distinctive clusters. The summary of the exploration and tradeoffs of FTL from our study is outlined in Table 6.

6. CONCLUSION

We have discovered the design considerations and tradeoffs for designing a customized FTL regarding the workload characteristics in HPSSD using a simulation-based method. Based on the discovery, guidelines for FTL design, such as page allocations, load balancing, hot/cold data separation, and wear leveling cluster, have been proposed.

Wide striping-based static allocations using page as striping unit cope with sequential IOs very well, while dynamic allocations deal with random IOs well from a performance point of view. However, because tradeoffs exist between in-

creasing parallelism within SSD and gaining write locality, utilizing static allocations with block striping unit is better than employing page striping unit or dynamic allocations to reduce the number of cleaning operations.

Hot/cold data separation is known to be effective for reducing the number of cleaning operations in environments with evenly distributed requests. Load balancing is proven to be helpful to resolve data skewness in dynamic allocations. To increase the degree of even wear level in SSDs, we need to employ large wear leveling clusters. However, large wear leveling clusters have the potentials to degrade the response time, so using small clusters are recommended when performance is a crucial issue.

As we have shown, the decisions for designing customized FTLs in HPSSDs can vary depending on workload characteristics, system requirements, and the remaining lifetime of SSDs. Our study provides the basis to understand, design, and utilize such FTL for HPSSDs in different conditions, and also yields insights into designing a high-performance generic SSD.

7. ACKNOWLEDGMENTS

We thank Vijayan Prabhakaran, Ted Wobber and their co-workers for providing us with some of the workload traces and the SSD plug-in for DiskSim. We are also grateful for their opinion in the initial stage of this work. Finally, we would like to express our appreciation to Bojun Huang for reviewing and commenting on our work.

8. REFERENCES

- [1] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy. Design tradeoffs for SSD performance. In *Proceedings of USENIX Annual Technical Conference*, pages 57–70, Jun 2008.
- [2] A. Birrell, M. Isard, C. Thacker, and T. Wobber. A design for high-performance flash disks. *SIGOPS Oper. Syst. Rev.*, 41(2):88–93, 2007.

- [3] L.-P. Chang. On efficient wear leveling for large-scale flash-memory storage systems. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, pages 1126–1130, New York, NY, USA, 2007. ACM.
- [4] L.-P. Chang and T.-W. Kuo. An adaptive striping architecture for flash memory storage systems of embedded systems. *Real-Time and Embedded Technology and Applications Symposium, 2002. Proceedings. Eighth IEEE*, pages 187–196, 2002.
- [5] M.-L. Chiang, P. Lee, and R.-C. Chang. Managing flash memory in personal communication devices. *Consumer Electronics, 1997. ISCE '97., Proceedings of 1997 IEEE International Symposium on*, pages 177–182, Dec 1997.
- [6] M.-L. Chiang, P. C. H. Lee, and R.-C. Chang. Using data clustering to improve cleaning performance for flash memory. *Softw. Pract. Exper.*, 29(3):267–290, 1999.
- [7] R. F. Freitas and W. W. Wilcke. Storage-class memory: The next storage system technology. *IBM Journal of Research and Development*, 52(4/5):439–447, 2008.
- [8] Fusion IO. Next generation server storage. <http://www.fusionio.com/PDFs/Fusion>
- [9] G. Ganger, B. Worthington, R. Hou, and Y. Patt. Disk subsystem load balancing: disk striping vs. conventional data placement. *System Sciences, 1993, Proceeding of the Twenty-Sixth Hawaii International Conference on*, i:40–49 vol.1, Jan 1993.
- [10] G. Ganger, B. Worthington, and Y. Patt. The DiskSim simulation environment, version 4.0. <http://www.pdl.cmu.edu/DiskSim/>.
- [11] J. Gray, B. Horst, and M. Walker. Parity striping of disk arrays: Low-cost reliable storage with acceptable throughput. In *VLDB '90: Proceedings of the 16th International Conference on Very Large Data Bases*, pages 148–161, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.
- [12] J.-W. Hsieh, L.-P. Chang, and T.-W. Kuo. Efficient on-line identification of hot data for flash-memory management. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 838–842, New York, NY, USA, 2005. ACM.
- [13] IOzone Filesystem Benchmark. <http://www.iozone.org>.
- [14] J.-U. Kang, H. Jo, J.-S. Kim, and J. Lee. A superblock-based flash translation layer for nand flash memory. In *EMSOFT '06: Proceedings of the 6th ACM & IEEE International conference on Embedded software*, pages 161–170, New York, NY, USA, 2006. ACM.
- [15] J. Katcher. Postmark: A new file system benchmark. *Technical Report TR3022, Network Appliance*, Oct 1997.
- [16] H.-J. KIM and S.-G. LEE. An effective flash memory manager for reliable flash memory space management. *IEICE Transaction on Information System*, E85-D(6):950–964, 2002.
- [17] J. H. Kim, S. H. Jung, and S. Y. Ho. Cost and performance analysis of nand mapping algorithms in shared-bus multi-chip configuration. *International Workshop on Software Support for Portable Storage*, 2008.
- [18] D. Lavenier, L. Xinchun, and G. Georges. Seed-based genomic sequence comparison using a fpga/flash accelerator. *Field Programmable Technology, 2006. FPT 2006. IEEE International Conference on*, pages 41–48, Dec. 2006.
- [19] S. Lee, D. Shin, Y.-J. Kim, and J. Kim. Last: locality-aware sector translation for nand flash memory-based storage systems. *SIGOPS Oper. Syst. Rev.*, 42(6):36–42, 2008.
- [20] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, and H.-J. Song. A log buffer-based flash translation layer using fully-associative sector translation. *Trans. on Embedded Computing Sys.*, 6(3):18, 2007.
- [21] Y.-G. Lee, D. Jung, D. Kang, and J.-S. Kim. μ -ftl: a memory-efficient flash translation layer supporting multiple mapping granularities. In *EMSOFT '08: Proceedings of the 7th ACM international conference on Embedded software*, pages 21–30, New York, NY, USA, 2008. ACM.
- [22] S. Ng. Some design issues of disk arrays. *COMPCON Spring '89. Thirty-Fourth IEEE Computer Society International Conference: Intellectual Leverage, Digest of Papers.*, pages 137–142, Feb-3 Mar 1989.
- [23] Open NAND Flash Interface. ONFI 2.0 specification. <http://www.onfi.org/docs/ONFI>
- [24] C. Park, W. Cheon, J. Kang, K. Roh, W. Cho, and J.-S. Kim. A reconfigurable FTL (flash translation layer) architecture for nand flash-based applications. *Trans. on Embedded Computing Sys.*, 7(4):1–23, 2008.
- [25] D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (RAID). *SIGMOD Rec.*, 17(3):109–116, 1988.
- [26] V. Prabhakaran and T. Wobber. SSD extension for DiskSim simulation environment. <http://research.microsoft.com/en-us/downloads/b41019e2-1d2b-44d8-b512-ba35ab814cd4/default.aspx>.
- [27] R. Rivera. Distributed data acquisition and storage architecture for the supernova acceleration probe. *IEEE Nuclear Science Symposium Conference Record*, 2:1082–1093, 2007.
- [28] P. Scheuermann, G. Weikum, and P. Zabback. Data partitioning and load balancing in parallel disk systems. *The VLDB Journal*, 7(1):48–66, 1998.
- [29] SQLIOSim, Microsoft Corporation. <http://support.microsoft.com/kb/231619>.
- [30] Transaction Processing Performance Council. TPC benchmark C, standard specification. http://www.tpc.org/tpcc/spec/tpcc_current.pdf.
- [31] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. The HP AutoRAID hierarchical storage system. *ACM Trans. Comput. Syst.*, 14(1):108–136, 1996.
- [32] D. Woodhouse. JFFS: The journalling flash file system. *Ottawa Linux Symposium*, 2001.
- [33] P. Zabback, J. Menon, and J. Riegel. The RAID configuration tool. *High Performance Computing, 1996. Proceedings. 3rd International Conference on*, pages 55–61, Dec 1996.