p-OFTL: An Object-based Semantic-aware Parallel Flash Translation Layer

Wei Wang, Youyou Lu, and Jiwu Shu *
Department of Computer Science and Technology, Tsinghua University, Beijing, China
Tsinghua National Laboratory for Information Science and Technology, Beijing, China
Email: {wangwei11,luyy09}@mails.tsinghua.edu.cn, shujw@tsinghua.edu.cn

***Abstract*—With increased density and decreased price, flash memory has been widely used in storage systems for its low latency and low power features. However, traditional storage systems are designed and excessively optimized for magnetic disks, and the potential of flash memory is not brought into full play in the form of Solid State Drives (SSDs). In this paper, we propose p-OFTL, an object-based semantic-aware parallel flash translation layer (FTL). p-OFTL removes the mapping table in the FTL and directly manages the flash memory in file objects, which enables optimization of data layout in the flash using object semantics. While the removing of the mapping table improves system performance, a challenge remains to exploit the internal parallelism when maintaining the continuity of logical addresses in each object, which is essential for efficient garbage collection. To address this challenge, p-OFTL statically remaps the addresses by shifting the bits in the addresses, which spreads writes to different internal parallel units without another mapping table. Also, p-OFTL employs a semantic-aware data grouping algorithm to group data pages by trading off the hot-cold clustering for the continuity of logical addresses, so as to reduce the page movement in garbage collection. Experiments show that p-OFTL improves system performance by 4.0% ˜ 10.3% and reduces garbage collection overhead by 15.1% ˜ 32.5% in semantic-aware data grouping compared to those in semantic-unaware data grouping algorithms.**

I. Introduction

Magnetic disks, which have dominated the storage market for decades, are now threatened by the flash storage, which has low access latency and low power consumption. With decreased price and increased density, flash-based Solid State Drives (SSDs) are widely used in both enterprise and embedded storage systems. But the access to flash memory is different from that to magnetic disks. Flash memory cannot be overwritten, which needs an erase operation before overwriting the data (in-place update). And it is read or written in the unit of pages (e.g., 4KB) and erased in the unit of blocks (e.g., 64 pages) [2]. To hide the long latency of erase operations before overwriting, flash translation layers (FTLs) are used to remap the writes to free pages, which is called out-of-place update. With the block device emulation of FTLs, flash-based SSDs can easily substitute magnetic disks without changes to existing software or hardware.

However, the block interface between software systems and SSDs prevents the optimization of data layout in either side, while the data layout is important to both the performance and the endurance of flash storage. Flash memory has limited program/erase (P/E) cycles, which is known as the endurance problem. And the problem is getting worse with the increasing density, e.g., the P/E cycles of SLC are 100,000, and MLC 10,000 [9]. In flash storage, valid pages in flash blocks should be moved to free blocks before erasing these blocks, which is known as garbage collection. This movement incurs more read and write operations, and thus hurts both performance and endurance.

There are two mapping tables in traditional SSD storage systems. One is the mapping from file offset to logical page address in the file system, and the other is the mapping from logical page address to physical page address in the FTL. Unfortunately, the indirection found in many FTLs comes at a high price [11], which manifests as performance cost, space overhead, or both. Recent researches have proposed to leverage the storage management in FTLs of SSDs in the storage management of file systems to remove the duplicated mapping tables [5], [7], [11]. Unfortunately, file/object semantics (the type of and the relationship between data pages, e.g., pages belonging to the same file/object) have not been used to guide the distribution of data pages to different blocks and banks in flash memory, which is know as data layout. Since pages in the same object have higher probability to be created, deleted and modified at the same time, the continuity of logical addresses is important to cluster the pages and improve the efficiency of garbage collection. Failure to cluster the pages in the same object increases the fragmentation in blocks, which increases the average number of valid pages in each block and further decreases the garbage collection efficiency. However, optimizing data layout to exploit the internal parallelism while maintaining the continuity of logical continuity remains a challenge. First, accesses to flash memory can be distributed to different planes, banks or channels (i.e., internal parallelism). But aggressively allocating pages from different blocks breaks the continuity of logical addresses. Second, legacy data grouping algorithms [8] separate the hot pages from the cold, which makes the fragmentation problem more serious.

Our goal in this paper is to exploit the internal parallelism without additional mapping table and to improve garbage collection efficiency by optimizing the data layout using file/object semantics, so as to improve the performance and endurance of flash-based storage systems.

Observations and Key Ideas: We propose an object-based parallel flash translation layer, p-OFTL, to directly manage the flash memory without the FTL mapping table.

*Corresponding author: Jiwu Shu (shujw@tsinghua.edu.cn).

This work is supported by the National Natural Science Foundation of China (Grant No. 60925006), the State Key Program of National Natural Science of China (Grant No. 61232003), the National High Technology Research and Development Program of China (Grant No. 2013AA013201), Shanghai Key Laboratory of Scalable Computing and Systems, the research fund of Tsinghua-Tencent Joint Laboratory for Internet Innovation Technology, Huawei Technologies Co. Ltd., and Tsinghua University Initiative Scientific Research Program.

Object-based management enables data layout optimization using object semantics, and we make two major changes in the object-based FTL to take this advantage. (1) Page addresses can be remapped statically without introducing another mapping table, which would otherwise incur extra maintenance overhead. In order to exploit internal parallelism of flash storage, we shift the address bits to remap pages to different parallel units statically, and pages with continuous addresses can be written simultaneously. (2) Pages in the same object have high probability to be modified simultaneously. When designing data grouping algorithms to group writes, not only the hot-cold separation but also the logical continuity should be considered. In order to improve the garbage collection efficiency, we propose a semantic-aware data grouping algorithm, which only merges the adjacent pages, to keep the continuity while separating the hot data from the cold.

Our contributions are summarized as follows:

1) We propose p-OFTL to directly manage the flash memory by removing the mapping table from the FTL of SSDs and use a static address remapping method to provide internal parallelism without additional mapping table.
2) We design a new data grouping algorithm, which makes trade-offs between hot-cold clustering and the page clustering of same objects, so as to improve garbage collection efficiency using object semantics.
3) Our evaluation of p-OFTL shows that the static address remapping effectively exploits the internal parallelism of flash storage, and the access latency is reduced nearly inversely proportional to the parallelism degree. Also, the data grouping algorithm using object semantics makes further efforts to optimize data distribution and gains an improvement in system performance by 4.0% ˜ 10.3% and a reduction in garbage collection overhead by 15.1% ˜ 32.5%.

II. Motivation and Related Work

Flash memory has significant differences from magnetic disks. Flash-based SSDs not only hide the characteristics of flash memory from the system, but also prevent the semantics from the device. We propose an object-based FTL to optimize the FTL design with both file/object semantics and flash memory characteristics into consideration.

A. Internal Parallelism

One major difference between magnetic disks and flash-based devices is the internal parallelism. Magnetic disks have only one head and have to access the data serially. In contrast, most SSDs are built on an array of flash memory packages, which are connected through multiple channels to flash memory controllers, and provide internal parallelism at different levels [3]. FTL mapping table is flexible to redirect data writes to different banks by modifying the page mapping. However, the object-based flash translation layer aims to remove the FTL mapping table in order to eliminate its translation and storage overhead. In this paper, we use static address remapping in object-based flash translation layer to provide flexible parallelism without additional mapping table.

B. GC Induced Write Amplification

Another major difference is the endurance problem. Flash memory cells wear out with more program/erase cycles, while

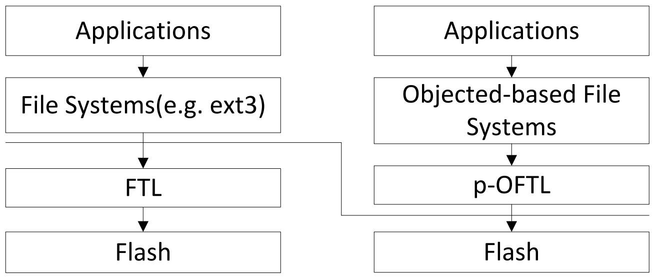


Fig. 1: Architecture Comparison of p-OFTL and normal FTL in SSDs.

magnetic disks do not. Garbage collection (GC) has a large impact on endurance when the average number of valid pages in flash blocks is high. GC selects blocks for erase and moves valid pages to free blocks before erasing these blocks. The page movement causes extra writes. To minimize the pages needed to move, FTLs usually classify data according to their update frequencies into hot and cold [8]. Data with the similar update frequencies are written to the same blocks and will probably turn invalid in the same time. When the block is picked out for erasure, few valid pages need to be migrated. In addition, because pages in the same file/object tend to be updated or deleted at the same time, the logical address continuity of data pages in each file/object also has impacts on GC and thus the write amplification. As such, we aim to achieve better data layout by using object semantics.

C. Related Work

Nameless Writes [11] and Direct File System [5] propose to remove the mappings of the file systems to allow the FTL to directly manage the flash storage space for the purpose of better performance. But the file semantics fail to be passed to the FTL for intelligent storage data layout. p-OFTL adopts the concept of object-based storage instead of traditional local file systems to export the object interface to the objected-based file systems, which can easily take advantage of the file semantics for intelligent data layout. File-system-aware FTL [10] takes the use of a filter mechanism to separate the access request of file metadata and contents for better performance based on the knowledge of the layout of file systems. Aimed at reducing the overhead introduced by file systems, it has to get well understanding of the file systems layout, however, it does little in the data layout in SSDs. OFSS [7] takes use of an object-based design, OFTL, and significantly reduces the write amplification from file systems with the system co-design with flash memory. But it ignores the internal parallelism of flash and file/object semantics. With the file/object semantics of data ignored, SFS [8] simply divides the data into cold data and hot data, and gathers the data with the same update frequency together. With a view to the internal parallelism, p-OFTL goes a step further and has the file/object semantics in mind to optimize the data distribution. With the help of semantics, p-OFTL can gain a better data distribution and reduce the GC induced write amplification. Moreover, p-OFTL can also make the best of the sequential access optimization of the traditional storage systems.

III. p-OFTL Design

In this section, we present an object-based parallel flash translation layer, p-OFTL, to directly manage flash memory using object semantics. The design goals of p-OFTL are: (1) internal parallelism support without additional mapping table; (2) semantic-aware data grouping to improve the garbage collection efficiency.

As shown in Figure 1, p-OFTL merges the storage management in file systems with the FTL and manages the flash storage space directly. The address mapping function is removed from the devices. Without address mapping, the device provides a simple interface for read, write, and erase operations. As a consequence, p-OFTL accesses the flash pages directly. Therefore, p-OFTL is able to determine the locations that the data writes are performed to. Other functions like garbage collection and wear leveling are moved to p-OFTL as well. Simple wear leveling and garbage collection mechanisms are implemented in p-OFTL and are not discussed much in this paper.

p-OFLT uses an object storage framework following the design of OFSS [7]. It uses objects to manage the data layout and exports the interface to the file systems for accessing the object data and attributes. An object is a collection of data and attributes with a unique ID. File data are stored in one or more objects, and file metadata are stored in the object attributes. As such, p-OFTL is able to gain the object semantics and optimize the data layout.

A. *Parallelism with Static Address Remapping*

In p-OFTL, we exploit the bank-level internal parallelism of flash devices. The address of a flash physical page can be described as a 32-bit integer and divided into three parts (A, B, C). The higher address part A is the bank address and identifies which bank the page belongs to. The middle part is the block address in the bank, while the lower part is the page address in a block. For sequential access, the physical addresses of the pages accessed are in sequence and will be probably located in the same bank. This leads to poor performance. To exploit the parallelism of flash without destroying the semantics, we use a simple static address mapping and then provide a sequential address space with no additional mapping table. The core idea is to combine physical blocks from several banks into super-blocks and data access will be spread to the pages in super-blocks for read and write operations, and an entire super-block for erasure.

The static mapping approach can be easily implemented by shifting and is described as follows. The bank address A can be divided into two parts A1, A2, so the page address becomes (A1, A2, B, C). We then map it to (B, A1, C, A2). For instance, we set the degree of parallelism to be 4, then A2 is the low two bits of bank address and A1 is the higher bits. The parallelism degree is flexible by shifting different number of bits. We combine the 4 banks (e.g. bank 0,1,2,3) into a group, and blocks in the banks with the same block address make up a super-block which is four times larger than a normal block. A sequential access to multiple pages in a super-block will be spread to the four banks, which can benefit from the parallelism. Legacy storage systems have been focusing on sequential access optimization. No additional mapping table is needed, this static mapping approach works much better for the current sequential-access-oriented optimization model adopted in system and application designs, which is particularly important for data-intensive applications.

As to the GC, we can erase either the original flash block or the entire super-block at a time. Erasing an original block of a super-block with other blocks remained, will destroy the continuity of the logical address, making it difficult to maintain the file content addressing. Once erasing the entire super-block, the write amplification will grow as the super-block size increases. We choose the second way. As a result, a more efficient data grouping algorithm is needed for better data distribution and less write amplification. Moreover, the degree of parallelism can be easily adjusted for different needs and compromise between latencies and write amplification.

B. *Semantic-aware Data Grouping*

In the process of garbage collection, the valid pages within a block must be migrated to other regions before the block can be erased, which leads to extra cost. And the more invalid pages in a victim block, the higher cost. The main reason is that the pages of a block turn invalid in different periods of time and when the block is picked out for erasure, some pages still remain valid. As a result, we can put data which will be rewritten in the same time together and store them in the same block, then pages in that block will turn invalid in the same period of time which leads to less data movement during GC.

Traditional ways [8] obtain the update frequencies of data from history and then simply classify data as hot and cold according to their update frequencies. Then data with different degrees of hot and cold will be stored in different regions. However, the history information of the data is not accurate without file/object semantics. For example, when a file is deleted from file systems, the logical pages the data stored in will be reclaimed and reused for other files. The history information becomes unreliable. Moreover, the data appended to a file, which have never been accessed before have no history, and cannot be clarified according to the degree of hot and cold.

Eliminating the dual mapping of system software and FTL-s, p-OFTL gains a better perception of file/object semantics which contributes to a better data distribution. First, p-OFTL can obtain the various data types of the flash pages, such as data pages, metadata pages. Data of different types can be separated and conducted to different regions. Second, p-OFTL combines small writes to a large continuous write and then writes them simultaneously. When data are written to flash memory from the main memory, they are grouped according to their update frequencies and the logical address continuity of data pages. With logical continuity of data pages, fewer IO requests are needed for accessing the same amount of data, which reduces the time spent on the software and makes better use of internal parallelism of SSDs. The continuity of data pages also reduces the size of mapping table of an object in p-OFTL, which will benefit the management and addressing of object data in turn. Because of the locality of data access, some data are accessed, and then the contiguous part may be most probably accessed soon. The adjacent data not only have the similar update frequency, but also have a fair chance of being overwritten and turning invalid in the same time. Therefore, when written back to the flash, the data need to be divided into groups and written piecewise continuously. Making the most use of the semantics of file/objects, an semantic-aware data grouping algorithm is proposed to distribute the data in flash storage according to both the update frequencies and the data pages' semantics.

1) Modeling : The above question can be modeled as follows:

A write to the flash involves multiple flash pages, let the number of pages be L. As to a flash page, the degree of hot and cold is defined as h_p:

$$h_p = \begin{cases} W_p/(T - T_p^m), & (W_p > 0) \\ h_0, & (W_p = 0) \end{cases}$$

Among which, W_p is the write count of the logical pages in objects, T is the current time, T_p^m is the last modified time of that page, and h_0 is the degree of the adjacent pages (limited to 5 pages at most).

The degree of a page mainly depends on the access history of that page, which can be evaluated by the write count and last modified time. But for appending operations, the pages are written the first time and have no history. Their degrees of hot can be defined as the average hot of adjacent pages h_0:

$$h_0 = \frac{1}{n} * \sum_{1 \le i \le n} h_i, (n \le 5)$$

Then the goal is to divide the L pages into a few continuous groups and achieve the following goals.

- For each page in a group, its degree of hot and cold h_p should be as close as possible;
- The number of groups should be as few as possible, that is the pages should be as continuous as possible.

Assume the pages are divided into k groups, $G_1, G_2, G_3, ..., G_k$. Each group is a collection of a number of continuous pages. The number of groups is k and the hot diversity of pages within each group is $\delta(G_i)$.

$$H_i = \frac{1}{m} \sum_{j=1}^{m} h_{p_j}, p_j \in G_i$$

$$\delta(G_i) = \frac{1}{m} \sum_{p \in G_i} |h_p - H_i|$$

Let the monotone increasing function, $f(k) = k^2$, be the weight of group number k. For a certain grouping g, define its total weight W_g.

$$W_g = \sum_{i=1}^{k} \delta(G_i) + \lambda f(k)$$

λ is constant, and g is the grouping $G_1, G_2, G_3, ..., G_k$, in other words, the pages are grouped into $G_1, G_2, G_3, ..., G_k$. The total weight consists of two parts: the total hot diversity of each group and how many groups data are divided into. The goal becomes finding out the grouping that has the minimum weight, which is the best grouping. For the ith group, it contains m flash pages $p_1, p_2, p_3, ..., p_m$, the hot of each is $h(p_j)$, and the average is h_i.

Achieving the best grouping, the mass of data can be written to the flash sectional continuously according to the above grouping.

2) Algorithm: On the basis of the above model, the data grouping algorithm can be described as the following steps:

(a) Calculate the degrees of hot and cold of the L flash pages $h_1, h_2, h_3, ..., h_L$;
(b) Initialize the grouping $g_0 = \{G_1, G_2, G_3, ..., G_k\}$, among which, $G_i = \{i\}$, that is to say each page forms one group;
(c) For grouping g_j, we can try to combine two continuous groups, G_a, G_b, into one group G_c, and get a new grouping $g_j^{'}$. Then the decrease of weight from g_j to $g_j^{'}$ is

$$\begin{aligned}\Delta j(a,b) &= g_j - g_j^{'} \\ &= \delta(G_a) + \delta(G_b) - \delta(G_c) + \lambda f(k_j) - \lambda f(k_{j-1}) \\ &= \delta(G_a) + \delta(G_b) - \delta(G_c) + 2\lambda f(k_j) - \lambda\end{aligned}$$

(d) Find out the two groups that can maximize $\Delta j(a,b)$, merge them and get the next grouping g_{j+1}, and go to Step (c). Otherwise, each of the $\Delta j(a,b)$ is no more than 0, in other words, the total weight cannot be reduced any more;
(e) Now we get the best grouping $g = \{G_1, G_2, G_3, ..., G_{k'}\}$, which leads to the minimum weight. As to each group, find the region that is the most proximal in update frequency from the whole write regions (in p-OFTL we set it as four). Some groups may be mapped to the same region, and we can write the data to the corresponding region together.

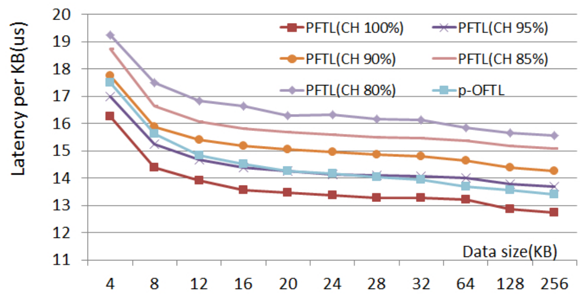
Because we only merge the adjacent pages, the continuity of logical address will be able to endure in some ways. In this way, we can gain the continuity and hot-cold separation at the same time. When data are written to flash piecewise continuously, each piece of data will probably be overwritten and turn invalid in a similar period of time. As such, we can balance the hot and cold separation and the data continuity.

IV. Evaluation

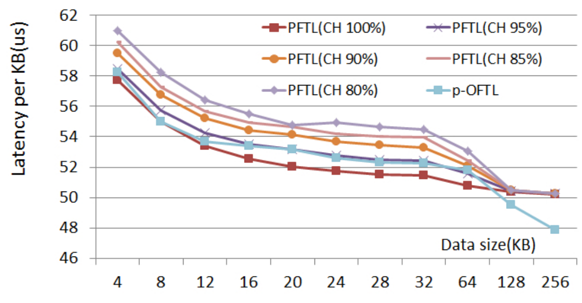
In this section, we first evaluate the overall performance of p-OFTL compared to existing SSD-formed flash storage. And then we evaluate the benefits in exploiting the internal parallelism from static address mapping and the benefits in garbage collection from semantic-aware data grouping.

A. SSD Simulator

We conduct our evaluation on an SSD simulator, which is implemented based on Flashsim [6]. We create a virtual block device in Linux kernel which simulates the hardware architecture of SSD: package, die, plane, block, page and channel for data transmission. In the simulator, flash memory packages are connected to the controller through multiple channels. Each channel is shared by multiple packages and is operated independently. Various parallelism degrees are provided in the simulator. In addition, it also simulates the out-of-band (OOB) area of flash page. OOB is the extra space alongside each flash page, which stores a variety of information including error correction code for the page data. In the simulator, OOB is exposed to the software for direct reads or writes. The latency of each I/O request is calculated to evaluate the performance in the simulator. For each I/O request, the data transmission path has to be locked for certain time. The latency of each operation is the sum of each read/write latency and the data transmission latency. And the latencies are collected using the high-precision clock in the Linux kernel.

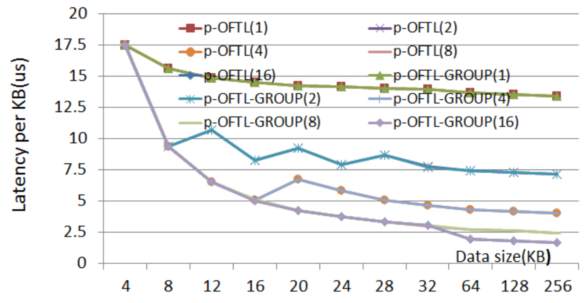


(a) Read Latency

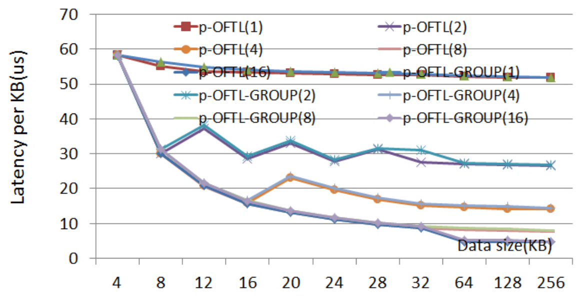


(b) Write Latency

Fig. 2: Performance Comparison of p-OFTL and ext3+PFTL. Note that CH-n means the cache hit rate of mapping table is n%.



(a) Read Latency



(b) Write Latency

Fig. 3: Effects of Static Address Remapping. Note that p-OFTL(n) means the parallelism degree is n, and GROUP means that the semantic-aware data grouping algorithm is used.

TABLE I: Simulator Configurations

Operation	Read	Write	Erase
Latency	0.05ms	0.20ms	1.50ms
Unit size	4KB	4KB	256KB

B. Experimental Setup

We use the system level I/O traces home02 [1], which are collected from the main campus general-purpose servers. We extract read and write operations from these traces and then replay them on p-OFTL. We collect the runtime, the erase count and the number of pages moved during garbage collection to evaluate the performance and the endurance of p-OFTL. We also simulate a page-level FTL (PFTL), which transforms the logical address to the physical address using a page-level mapping table, as a kernel module to evaluate the traditional file systems.

We implement p-OFTL, including the static address remapping and the semantic-aware data grouping algorithm, on the SSD simulator in Linux kernel 3.2.9. In the simulator, the page size is set as 4KB, and each block contains 64 pages with a total size of 256KB [2]. The size of the whole virtual disk size is 3.64GB. The parallelism degree is 16, i.e., at most 16 operations can be performed independently at the same time. The latencies for reading and writing a single page, erasing a block are configured as shown in Table I. The experiments are conducted on Fedora 16 with kernel 3.2.9 running on a computer with a 4-core Intel i5 2.4GHz processor and 16GB memory.

C. Overall Performance

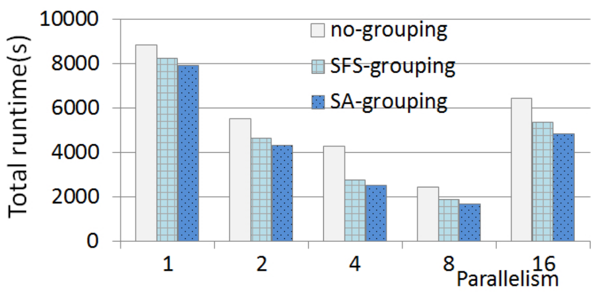
In this section, we evaluate the performance of p-OFTL against ext3 built on PFTL in terms of read and write latencies.

In the simulation, ext3 is mounted with data journal option. We collect the average read and write latencies under the situation that there is enough free space in flash so as not to be affected by the garbage collection. As the FTL is usually implemented in the controller with limited DRAM cache, the mapping table cannot be completely cached. A lot of research is conducted on optimizing SSD mapping algorithm to improve the cache hit ratio [4]. To evaluate the performance under different cache hit ratios, we artificially set the cache hit rate ranging from 100% to 80%, and then write various sizes of data to the flash and collect the average read and write latencies.

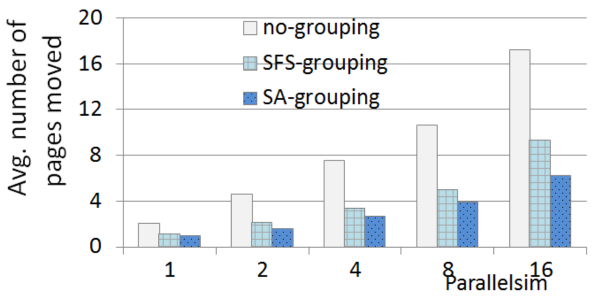
Figure 2 shows the read and write latencies of p-OFTL and ext3 on PFTL. From the figure, we have two observations. First, the performance of p-OFTL is a little poorer than that of ext3 on PFTL when the cache hit ratio is 100%. Second, when the cache hit ratio get lower, which is common for embedded FTLs, the performance of p-OFTL gets better than ext3 on PFTL. In this case, the overhead of FTL mapping table gets visible to the system, making the overall performance poorer. In contrast, p-OFTL takes use of the redundant host memory and reduces the latencies by eliminating the mapping table in the FTL.

D. Effects of Static Address Remapping

In this section, we evaluate the impact of static address remapping on p-OFTL by measuring the latencies under various parallelism degrees. We collect the latencies by varying the



(a) Total Runtime



(b) Write Amplification

Fig. 4: Effects of Semantic-Aware Data Grouping.

read/write size from 4KB to 256KB and the parallelism degree from 1 to 16. Figure 3 shows that both the read and write latencies are nearly inversely proportional to the parallelism degree. The latencies are observably reduced as the parallelism degree increases, and the static address remapping is proved to be an efficient approach to exploit the internal parallelism of flash.

E. Effects of Semantic-Aware Data Grouping

With the increase of parallelism degree, the latency is obviously reduced. However, the size of the super-block also increases, which leads to more data movement when erasing one super-block and gives rise to the write amplification. To measure the impact on write amplification and system performance, we write totally 20GB data to the 3.64GB SSD simulator. And we collect the runtime, the number of pages moved and the erase count during garbage collection.

Figure 4 shows the total runtime as well as the averaged number of pages moved in erasing one flash block (i.e., the number of pages moved divided by the erase count). In the figure, we first observe the trend of performance and endurance with the increase of parallelism degree. As shown in Figure 4(a), the performance is improved with higher parallelism degree when the parallelism degree is not larger than 8, but it becomes worse when the parallelism degree is 16. This performance loss is induced by the overhead of garbage collection, which has eaten up the benefits from parallelism. This cause is further explained in Figure 4(b). Figure 4(b) shows the averaged number of pages moved for erasing one flash block. We can see the write amplification increases observably as the parallelism increases.

In Figure 4, we also compare the semantic-aware grouping (SA-grouping) algorithm with SFS data grouping (SFS-grouping) algorithm [8], which is designed only with hot-cold separation. No-grouping means that the data are written to the flash sequentially in spite of that whether the data are hot or cold. The SFS grouping algorithm simply divides the data into hot and cold, then written them to the flash separately. Figure 4(a) and (b) show that semantic-aware data grouping algorithm is more efficient than the traditional algorithm. Moreover, we can see from Figure 3 that the data grouping algorithm has a very tiny overhead on read and write latencies which can be ignored. The algorithm reduces the write amplification by about 15.1% ˜ 32.5%. and brings 4.0% ˜ 10.3% improvements to the whole system performance compared to SFS data grouping algorithm.

V. Conclusion

Dynamic address mapping in the FTL simplifies the design and implementation of the software, however, it builds strong barriers between software and devices. To take better use of flash storage, an object-based semantic-aware parallel flash translation layer, p-OFTL, is proposed to tear up the semantic barriers by directly managing the flash memory. In p-OFTL, a simple static address mapping approach is used to exploit the internal parallelism of flash storage. In addition, a semantic-aware data grouping algorithm is designed to optimize the data layout with object semantics. The write amplification is effectively reduced, so as to benefit both the performance and the endurance of flash storage.

References

[1] Sos project traces. http://www.eecs.harvard.edu/sos/traces.html.

[2] 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*, 2008.

[3] F. Chen, R. Lee, and X. Zhang. Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing. In *Proceedings of the 17th IEEE International Symposium on High-Performance Computer Architecture(HPCA)*. IEEE, 2011.

[4] A. Gupta, Y. Kim, and B. Urgaonkar. *DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings*. ACM, 2009.

[5] W. K. Josephson, L. A. Bongo, D. Flynn, and K. Li. DFS: a file system for virtualized flash storage. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies(FAST)*, 2010.

[6] Y. Kim, B. Tauras, A. Gupta, and B. Urgaonkar. Flashsim: A simulator for nand flash-based solid-state drives. In *First International Conference on Advances in System Simulation(SIMUL)*. IEEE, 2009.

[7] Y. Lu, J. Shu, and W. Zheng. Extending the lifetime of flash-based storage through reducing write amplification from file systems. In *Proceedings of the 11th USENIX Conference on File and Storage Technologies(FAST)*, 2013.

[8] C. Min, K. Kim, H. Cho, S. Lee, and Y. Eom. SFS: Random write considered harmful in solid state drives. In *Proceedings of the 10th USENIX conference on File and Storage Technologies(FAST)*, 2012.

[9] C. Wang and W.-F. Wong. Extending the lifetime of nand flash memory by salvaging bad blocks. In *Proceedings of the Conference on Design, Automation and Test in Europe(DATE)*, 2012.

[10] P.-L. Wu, Y.-H. Chang, and T.-W. Kuo. A file-system-aware ftl design for flash-memory storage systems. In *Proceedings of the Conference on Design, Automation and Test in Europe(DATE)*. IEEE, 2009.

[11] Y. Zhang, L. P. Arulraj, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. De-indirection for flash-based ssds with nameless writes. In *Proceedings of the 10th USENIX Symposium on File and Storage Technologies(FAST)*, 2012.