ANALYSIS OF TRADE-OFFS IN V2P-TABLE DESIGN FOR NAND FLASH

Borja Peleato, Rajiv Agarwal and John Cioffi

Electrical Engineering Department
Stanford University
Stanford CA 94305

ABSTRACT

Flash memory uses relocate-on-write, also called out-of-place write for performance reasons. Data files from the host are spread across several non-sequential NAND physical pages. In order to retrieve host data at a later point a virtual-to-physical address table mapping the files to their physical addresses must be maintained. This process entails two basic steps. The first is to divide the NAND physical space in a hierarchical manner for efficiency of address lookup. The second is to store the resulting address lookup table, also called a virtual-to-physical (V2P) table in an efficient manner on the flash. This paper explores different architectures for constructing such table and storing it, thereby characterizing the trade-off that they offer in terms of complexity, write speed, and endurance of the flash memory.

1. INTRODUCTION

NAND-flash has unique characteristics that pose challenges to the SSD system design. The basic unit of NAND physical space is a block, consisting of a fixed number of pages, typically 64 pages of 4 KB each. A block is the elementary unit for erase operations, whereas reads and writes are processed in terms of pages. Before data can be written to a page (i.e., the page is programmed with that data), the block must have been erased. Moreover, NAND flash memories have a limited program-erase (PE) cycle count, or equivalently there is a limit to the number of times information can be re-written.

Flash memory uses relocate-on-write, also called out-of-place write [8], mainly for performance reasons: If write-in-place is used instead, flash will exhibit high latency due to the necessary reading, erasing, and re-programming of the entire block in which data is being updated. However, relocate-on-write requires maintaining a virtual-to-physical (V2P) table mapping the files to their physical addresses. This process entails two basic steps. The first is to divide the NAND physical space in a hierarchical manner for efficiency of address lookup. The second is to store the V2P table in an efficient manner on the flash. The architecture used for constructing such table and storing it plays a significant role in the overall performance of the memory.

Additionally, relocate-on-write requires a garbage-collection process involving additional read and write operations [4]. This effect is known as write amplification and it reduces both endurance and write throughput [6]. Different garbage collection policies result in different write amplification performance. Here, endurance refers to the lifetime of the memory and write throughput refers to the rate at which incoming host data is transfered to Flash memory.

Write amplification, which depends on overprovisioning[1], can be reduced by writing sequentially, since any file is then condensed in a small number of blocks. Additionally, when a file, or a portion thereof, occupies physically sequential pages, it can be located using only its starting location and length. As a result, writing sequentially would also reduce the size of the virtual-to-physical address table (V2P-table) mapping the files to their physical addresses. However, spreading the file across several blocks allows for faster reading and writing, since different blocks can be read in parallel via different channels. Additionally, the latter provides better error protection: if a block gets corrupted or loses information due to passage of time, the errors are spread across several files, and hence any single block can be corrected using a small amount of redundancy for each file.

Another important factor to consider is wear leveling. The program and erase operations damage the dielectric barrier in the flash cells, until they reach a point in which they can no longer store information reliably. When a certain number of the blocks reach this point, the memory is considered dead. If a subset of blocks are programmed and erased more often than the others they will suffer more damage and die earlier, effectively decreasing the lifetime of the memory. In practice, it is common for a memory to store both cold data, which is very rarely updated, and hot data, which is updated often. Blocks storing hot data are programmed and erased more often than those storing cold data. Wear leveling algorithms are therefore used to increase the memories' lifetime by occasionally moving cold data between blocks. The amount of data that needs to be moved during wear leveling, as well as how often this needs to be done depends on the architecture used. Wear

[1]Overprovisioning is defined as the difference between the total space in the drive and the user-accessible space, normalized by the user-accessible space

leveling itself uses additional writes thus reducing both write throughput and endurance.

Hence, different architectures have different advantages and drawbacks. In this work, we study these trade-offs through computer simulations. The number of possible designs is too large to address them all, so this paper will just give a flavor of the existing tradeoffs in flash architecture design by studying a small subset of architectures. Each architecture is tested for several garbage collection strategies, hot to cold data ratio, file and block sizes, etc. Ultimately, our goal is to produce a set of directives as to which architecture and garbage collection strategy should be used for a desired trade-off between the different figures of merit.

2. PREVIOUS WORK

In the existing literature, there exist two main types of addressing schemes depending on the granularity of the mapping between virtual and physical locations: block and page level addressing. In page level addressing the V2P table has as many entries as valid pages in the drive, whereas in block level addressing it has as many entries as blocks in the drive. In page level addressing each entry in the v2p table can reference any physical page, while in block level addressing the page offsets of the logical and physical blocks should be identical. Page level addressing is more flexible and results in lower write amplification, but it requires a very large amount of memory in both RAM and flash. Block level addressing, on the other hand, could require a very large number of garbage collections if the same logical page was repeatedly updated. Both schemes are explained and extensively analyzed by Park et al. in [5]. The simulations in this paper employed a V2P table with page level addressing.

This paper groups physical blocks into super-blocks to reduce the amount of metadata required and simplify the book-keeping. A similar approach was proposed by Kang et al. in [3], with the difference that they propose grouping the logical blocks instead of the physical ones into super-blocks.

Arpaci-Dusseau et al. were able to achieve significant gains in [1] by removing indirection from the mapping. Unlike usual write requests, which specify the data as well as the address where it should be written, they propose a scheme where write requests only include the data to be written. The device then finds a suitable location to write it and informs the host of its choice.

3. SYSTEM MODEL

Figure 1 shows one possible way of structuring the physical space of a flash drive. The drive is divided into dies, each consisting of a fixed number of blocks. Each die is mapped to a read and write channel. Blocks in different channels can be read or written in parallel, but blocks in the same die always share the same channel.

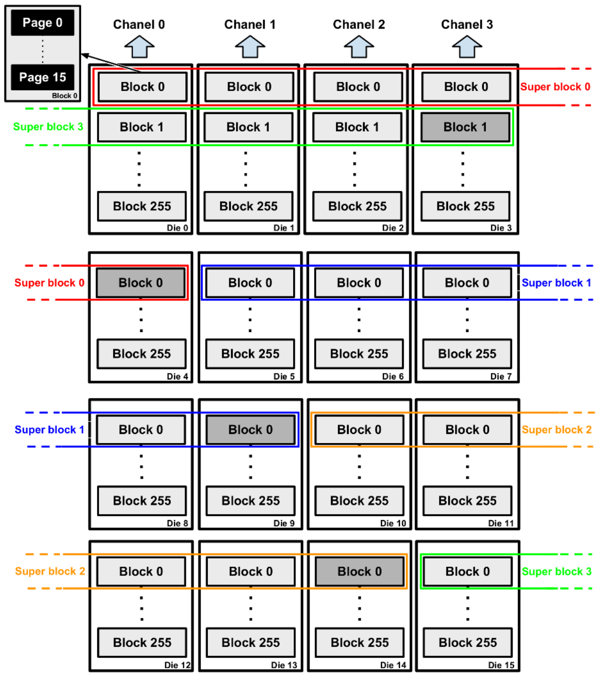

Fig. 1. Diagram of a Flash drive with 16 dies, 256 blocks per die, and 16 pages per block. The last block of each super-block is used for RAID protection.

Sometimes, a block, or even a whole die, might fail, rendering the information stored in it inaccessible. In order to avoid loosing that information, blocks are grouped into RAID (Redundant Array of Independent Disks) structures, known as super-blocks (SB). A super-block consists of a set of blocks from different dies where one of them stores the XOR of all the others [2]. This way, if one block or a die fails, the information that it was storing can be very efficiently recovered. Two blocks from the same die cannot be part of the same super-block. It is generally preferred that a super-block spans all channels, so that as many of its blocks as possible can be read/written simultaneously. Similarly, the pages from the blocks in a super-block are grouped into super-pages, also known as stripes, where one page in each stripe stores the XOR of all the others and no two pages from the same die can be in the same super-page. In order to simplify the book-keeping, flash controllers are often designed to perform the read and write operations in super-page units.

Once the physical space has been structured, an example for which is shown in Figure 1, the next step is to store metadata about its state [7]. The state for the physical space corresponds to information such as the number of PE cycles, the time when each block or super-block was last written, the number of valid pages in a super-block, etc. Maintaining state information or its book-keeping is critical for processes such as garbage collection or wear leveling. This information can be overwhelming to maintain for large SSDs where the number of blocks may exceed several hundreds of thousands. Furthermore, all this information needs to be stored onto the NAND flash, wherein it occupies space which otherwise could have been used for storing host data. Consequently, there is an incentive to minimize the amount of state information stored. In order to do so, super-blocks and super-pages are yet again grouped into super-block-groups (SBG) and super-page-groups (SPG), respectively. The simulations in this paper performed garbage collection and wear leveling in terms of SBGs.

4. DESIGN PARAMETERS

From the host perspective, the flash is a list of logical blocks, which can be overwritten at will. All the practical complexities associated to the physical block-level erasing, wear-leveling and out-of-place write are hidden from the host and seamlessly addressed by the flash system-level controller or the flash translation layer (FTL). The flash controller needs to keep and frequently update a table mapping logical addresses to physical addresses on the flash. Additionally, it must make several design choices that will affect performance metrics such as endurance, speed and robustness.

This paper will address the following performance metrics:

P_1 : Write amplification (related to write latency and endurance)

P_2 : Host writes to failure

P_3 : Average number of PE cycles across all SBGs in the drive,

the following usage scenarios:

U_1 : No cold data on the drive vs 50% cold data

U_2 : Small files (10 pages per file) versus large files (1000 pages per file),

and the following design parameters:

D_1 : Physical space on the NAND where the table is stored: in a dedicated SBG versus as a header on each SPG

D_2 : Maintaining a single versus two separate queues for host data and garbage collected data

D_3 : Storing files in sequential physical pages versus scattering them throughout the drive

D_4 : Block picking strategy for garbage collection.

This paper uses a hybrid garbage collection and wear leveling policy where SBGs are selected for garbage collection in decreasing order of the following metric:

$$M(\text{block } b) = \frac{r_b + \mu}{1 + \mu n_b},$$

where n_b and r_b respectively represent the number of PE cycles and the fraction of invalid data for SBG b. The parameter μ can be used to control the trade-off between wear leveling and write amplification. Specifically, greedy garbage collection, a policy that always picks the SBG with the smallest fraction of valid data, would correspond to $\mu = 0$ and perfect wear leveling, a policy that always picks the SBG with the least number of PE cycles, to $\mu >> 1$.

5. SIMULATION RESULTS

The simulations in this paper are done for a drive consisting of 16 dies, with 256 blocks per die, and 16 pages of 8kB each per block. The stripe depth (number of blocks per super-block) is 5, the number of super-blocks per SBG is 4, and overprovisioning is 25%. The maximum number of PE cycles that a block could sustain was assumed to be 10^4. Once any block reaches that number of PE cycles, the drive is considered dead. Uncompressed logical pages received from the host are 8kB in size, but are subsequently assumed to be compressed, so that the resulting logical page size is uniformly distributed between 1kB and 8kB.

This section will evaluate the tradeoffs that exist between the previously mentioned performance metrics (P_1-P_3) based on the choices made for the design parameters (D_1-D_4) under different user scenarios (U_1-U_2). The number of combinations of the possible choices for the design parameters are too many to compare in this section. Therefore, only a few candidate configurations are studied, the ones that illustrate the extreme points on the trade-off curves. Specifically, the following configurations are studied:

C$_1$: The first configuration that we study has 50% cold data, the garbage collection process maintains two separate queues for incoming host data and relocated data, the block picking algorithm uses $\mu = 0.005$, the V2P table is located in a separate SBG in the drive, the file size is 10 (logical pages per file), and files are scattered.

C$_2$: The second configuration that we study has 50% cold data, the garbage collection process does not distinguish between incoming host data and relocated data, the block picking algorithm uses $\mu = 0.005$, the V2P table is located in a separate SBG in the drive, the file size is 10, and files are scattered.

C$_3$: The third configuration that we study has no cold data, the garbage collection process does not separate incoming host data from relocated data, the block picking algorithm uses $\mu = 0$, the V2P information is stored as a header on each super-page-group, the file size is 10, and each file is stored in sequential physical pages.

C$_4$: The fourth configuration that we study has 50% cold data, the garbage collection process maintains two separate queues for incoming host data and relocated data, the block picking algorithm uses $\mu = 0.6$, the V2P table is located in a separate SBG in the drive, the file size is 1000, and files are

strategy	host writes to failure	write amplification	average PE cycle count
C_1	2117 GB	2.44	7020
C_2	1925 GB	3.3	8640
C_3	2709 GB	2.73	9960
C_4	2437 GB	2.57	8500

Table 1. Performance metrics evaluated for different Flash memory architectures

scattered.

Table 1 shows the three performance metrics studied in this paper for the above mentioned four configurations. Comparing C_2 and C_3, we find that the host writes until failure is much higher for C_3 because in C_3 there is no cold data, sequential writes result in a lower write amplification, and the V2P table location requires less space on the drive (more space is available for host data). Similarly, by comparing C_1 and C_2 we find that write amplification is much smaller for C_1 because in C_1 the garbage collection policy maintains separate queues for host data and relocated data (cold data). By doing this, the cold data gets compacted into a few SBGs on the drive, thereby increasing the effective overprovisioning for the remainder of the drive, which is being used for writing host data. Lastly, by comparing C_1 and C_4, we find that the average number of PE cycles at the time of failure is much higher for C_4 because in C_4 the host usage comprises of larger file sizes. Because of large file size, during the time of garbage collection, the fraction of invalid data is generally larger. Additionally, the larger value of μ results in wear leveling to happen as a part of the garbage collection process, therefore the overall wear leveling overhead is reduced as well. These two factors thus lead to a higher number of PE cycles until failure in C_4. Numerous other such performance trade-offs resulting from the choices of the design parameters can be seen in Table 1.

It is also interesting to study how the policy for storing the V2P information has influenced in the results. The write amplification factor compares the number of writes caused by new user data with the number of writes caused by relocation of that data, but does not account for other writes such as V2P information, RAID protection, metadata, etc. One possible way of accounting for those read is by comparing the total number of user data writes, given by the product of the host writes and the write amplification, with the average number of PE cycles. Dividing the last column in Table 1 by the product of the first two shows that C_3 is more efficient than C_1, C_2, and C_4.

Configurations C_1, C_2, and C_4 devote a whole SBG to storing a V2P table with a physical address for each logical page. The simulated drive can hold 80.000 compressed logical pages, each requiring a 4 byte physical address, so a minimum of 320 kB (about two and a half blocks) are required to store the V2P table. Furthermore, the V2P table needs to be frequently updated as old logical pages get updated and relocated to new physical locations. Most of the time, these updates happen in a separate memory but they periodically need to be transferred onto the flash to avoid information loss in case of power loss. Instead of erasing and re-writing the V2P table every time it needs to be updated, the controller reserves additional space (a whole SBG in this case) for storing updates. Once all the space has been used, the SBG can be erased and re-written.

Configuration C_3, on the other hand, does not store a V2P table explicitly. Instead, it devotes a few bytes in each super-page-group to store information about the logical pages written on that super-page-group. Specifically, 3 bytes are stored for each logical page: a page identifier and its length after compressing. This information only needs to be updated when the SBG is re-written, so it can be done seamlessly during the garbage collection process. Furthermore, configuration C_3 stores each file in sequential physical pages, so the only information required for locating a file is the compressed length of all its logical pages and the address of its first one.

Configuration C_3 is more efficient than C_1, C_2, and C_4 in terms of space and throughput, but it requires significantly longer boot time. During boot-up, it is necessary to use the information on the flash to construct a full V2P table in DRAM. If the V2P table is already constructed and stored in a SBG, it can be read directly. With configuration C_3, however, constructing the V2P table requires reading and processing all super-page-groups.

6. CONCLUSION

The unique characteristics of flash memories require using relocate-on-write, which entails maintaining a mapping between virtual and physical addresses as well as garbage collection and wear leveling processes. Different policies for flash memory architectures have different advantages and drawbacks in terms of performance. This paper gives a flavor of the existing trade-offs by simulating a small subset of architectures for several usage scenarios and analyzing the results that they provide in terms of speed and endurance.

7. REFERENCES

[1] A.C. Arpaci-Dusseau, R.H. Arpaci-Dusseau, and V. Prabhakaran. Removing the costs of indirection in flash-based ssds with nameless writes. In *Proceedings of the 2nd USENIX conference on Hot topics in storage and file systems*, pages 1–1. USENIX Association, 2010.

[2] S. Im and D. Shin. Flash-aware raid techniques for dependable and high-performance flash memory ssd. *Computers, IEEE Transactions on*, 60(1):80–92, 2011.

[3] J.U. Kang, H. Jo, J.S. Kim, and J. Lee. A superblock-based flash translation layer for nand flash memory. In *Proceedings of the 6th ACM & IEEE International conference on Embedded software*, pages 161–170. ACM, 2006.

[4] J. Menon and L. Stockmeyer. An age-threshold algorithm for garbage collection in log-structured arrays and file systems. *KLUWER INTERNATIONAL SERIES IN ENG. AND COMP. SCIENCE*, pages 119–132, 1998.

[5] 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. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(4):38, 2008.

[6] B. Peleato, R. Agarwal, and J. Cioffi. On distribution of valid pages with greedy garbage collection for nand flash. In *Proc. of IEEE Statistical Signal Processing Workshop.*, 2012.

[7] B. Peleato, R. Agarwal, and J. Cioffi. Probabilistic graphical model for flash memory programming. In *Proceedings of the IEEE Statistical Signal Processes Workshop*, pages 788–791, 2012.

[8] M. Rosenblum and J.K. Ousterhout. The design and implementation of a log-structured file system. *ACM SIGOPS Operating Systems Review*, 25(5):1–15, 1991.