Integrating De-duplication and Write for Increased Performance and Endurance of Solid-State Drives

Amit Berman and Yitzhak Birk
Electrical Engineering Department, Technion –Israel Institute of Technology, Haifa 32000, Israel
{bermanam@tx, birk@ee}.technion.ac.il

Abstract— **Modern NAND Flash-based Solid-State Drives (SSD) presents low latency, high throughput, low power consumption and solid-state reliability improvements comparing to traditional magnetic-disk based Hard Disk Drives (HDD). However, due to NAND Flash memory cell characteristics, update-in-place is impossible. Instead, the Flash software layer allocates new storage space whenever data is written, even if it is a slightly modified version of already stored data, e.g., a slightly modified file. Consequently, a logical overwrite entails extensive writing with resulting latency, power and chip endurance costs.**
Noting that reads are much faster than writes and that stale ("overwritten") data is seldom erased until a much later time, we present an SSD architecture with integrated de-duplication at the Flash software layer. Our concept is to use de-duplication tools to manage write and read operations in SSD so as to reduce redundant writes caused by data overwrite. Read operation incur a penalty in order to reconstruct the stored data from multiple data versions. Our proposed architecture's benefit grows in multi-block writes and in write-mostly applications. It moreover offers the side benefit of incremental backup, which enables to restore previous versions of the stored data.

Keywords-NAND Flash memory; Solid-State Drive; Performance; Endurance; De-Duplication; Architecture;

I. INTRODUCTION

Storage devices based on NAND flash non-volatile memory devices (flash SSD) have the following important properties: 1) very fast reading with fine-grain access capability; 2) much slower writes, which must moreover occur for an entire page, typically 2 KB and 3) it is only possible to add charge to memory cells, so (with the exception of advanced techniques like floating codes) one must erase cells prior to rewriting them, and this erasure can only be done at coarse granularities, typically 64 KB at a time. Also, writes and erasures damage the cells, whose endurance is currently between 10^4 and 10^5 cycles. As one increases capacity by permitting a larger number of charge levels per cell (MLC), endurance drops. This is in contrast with magnetic disk drives, in which read and write speeds are essentially equal and so are the granularities (512 byte sectors), erasure is not required, and endurance is effectively unlimited [1][2][3][4].

Rewriting data (e.g., saving a modified file) to SSD entails writing it in an erased area, adjusting the metadata to point to the new location and marking the old one as erasable. With time, as the SSD is almost full, sufficiently large contiguous regions that are entirely erasable are located, erased, and used for new writes. (If there are none, live blocks are copied to new areas in order to permit erasure.) All this is done regardless of the amount of change to the file. For example, inserting a character at the beginning of a text file alters all its blocks [5][6][7].

In this work, we propose an approach whereby, instead of writing the new version of the file in its entirety, we compute the difference between it and the older version, store this difference and update metadata accordingly. The actual algorithm can be any algorithm presently employed for incremental backup or de-duplication, and is not a contribution of this work. In order to do this, one must read the entire old version of the file (a fast operation), compute the difference (speed depends on the algorithm) and write only the difference. For small modifications to large files, the savings in writes are dramatic. This results in both an increase in the effective throughput of the SSD (as less of its time is taken by reading a file followed by writing of a small fraction than by writing the entire file) and in a decrease in the wear of the SSD chips: less data is written, so less needs to be erased over time. The latter results in longer lifetime of the SSD. Alternatively, in some cases the number of charge levels (and thus capacity) may be increased while achieving the original effective lifetime, at no cost. A side benefit of our approach is that multiple versions of the file are available. This can be harnessed to provide version support at low cost.

In order to read current data from the file, one may have to read the entire file along with the difference information, reconstruct the current version, and then retrieve the desired data. While this is obviously more complicated than directly accessing file data, it is important to note that many applications load the entire file into memory as soon as it is opened, so the amortized computational overhead is reasonable. Finally, one always has the option of writing the entire file (i.e., not using the differential approach). In practice, any decision policy is likely to be based on the number of differential blocks already existing and the resulting amount of computation required for reading current data, as well as on the total stored size (original version plus increments) relative to the current file size. Once a threshold derived from those is exceeded, the entire file would be written and the previously occupied space marked erasable. Another factor may be the amount of free space in the SSD.

Our scheme is best implemented at the lowest software level that is still aware of the semantic object that is being stored, e.g., files.

Placement of the integrated module in conventional SSD is shown in Fig. 1.

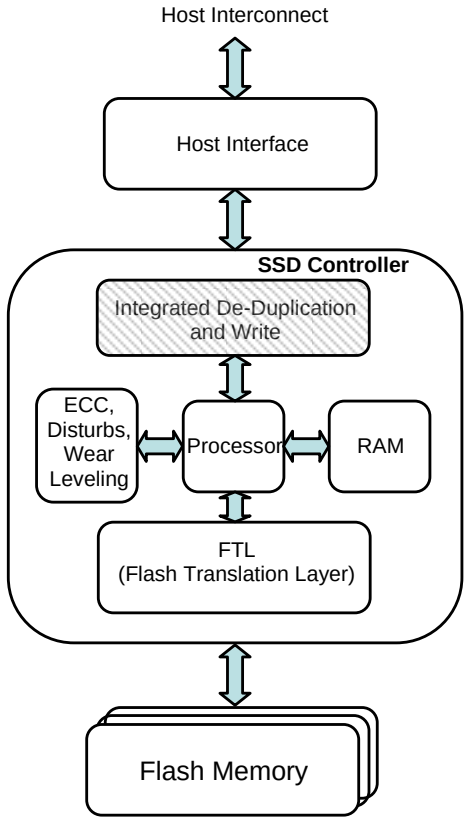

Figure 1. Placement of the integrated de-duplication and write module in conventional SSD.

II. Preliminary Analysis

We use the following notation:

- $X = \{x_1, x_2, ..., x_n\}$ - Set of file sizes [KB], where the file with index (*i+1*) is a modification of the file with index *i*.
- P - Page size [KB] of NAND Flash. (i.e., write and read granularity).
- B - Block size [KB] of NAND Flash. (i.e., erase granularity).
- d_i - size [KB] of the difference between files with indices (i+1) and i (de-duplication algorithm dependent).
- t_{WR}, t_{RD} - average page write and read time [μs], respectively.
- $D_{WR}(X), D_{RD}(X)$ - required time [μS] to calculate de-duplication difference during write, and reconstruction from multiple versions during read, respectively.

Consider the case wherein the file with index 1 is stored in the SSD, then modified and stored as file with index 2, and similarly up to index n.

The storage requirement [KB] of Set X (assuming no erasure), when using standard SSD, denoted $C_{std}(X)$ is:

$$C_{std}(X) = P \cdot \sum_{i=1}^{n} \left\lceil \frac{x_i}{P} \right\rceil \quad (1)$$

With integrated de-duplication and write, denoted $C_{int}(X)$, it is:

$$C_{\text{int}}(X) = P \cdot \left(\left\lceil \frac{x_1}{P} \right\rceil + \sum_{i=1}^{n-1} \left\lceil \frac{d_i}{P} \right\rceil \right) \quad (2)$$

Table 1 summarizes the comparison between the baseline scheme and our approach. Actual numbers depend on SSD parameters and on the achievable degree of de-duplication, which depends both on the data and on the algorithm. Write and read times neglect the controller calculation delay for both methods.

III. Conclusions

The use of a technique originally intended for backup systems and for the situation of multiple identical or similar copies of material that are stored in the same place, is shown to be beneficially adaptable to the case wherein a single copy of data is modified. For SSD in particular, this offers multiple important benefits.

This paper presented the basic idea. For a complete workable system, meta-data structure and processing must be considered, along with RAM management. Similarly policies for deciding when to create a stand-alone version of a file and when to reclaim space are required. These are interesting topics for further research.

References

[1] N. Agrawal, V. Prabhakaran, T. Wobber, "Design Tradeoffs for SSD Performance", USENIX Technical Conference, June 2008.

[2] A. Birrel, M. Isard, C. Thacker, T. Wobber, "A Design for High-Performance Flash Disks", ACM Operating Systems Review, 41(2), April 2007.

[3] S. Boboila, P. Desnoyers, "Write Endurance in Flash Drives: Measurements and Analysis", 8th USENIX Conference on File and Storage Technologies (FAST'10), 2010.

[4] J. Brewer et-al., "Nonvolatile Memory Technologies with Emphasis on Flash", IEEE Press Series on Microelectronic Systems, Chapter 6, pp. 223-310, 2008.

[5] T-S. Chung et-al. "System Software for Flash Memory: A Survey", Lecture Notes in Computer Science, Vol. 4096, pp. 394-404, 2006.

[6] C. Dirik, B. Jacob, "The Performance of PC Soild-State Disks (SSDs) as a Function of Bandwidth, Concurrency, Device Architecture, and System Organization", International Symposium on Computer Architecture (ISCA'09), June 2009.

[7] J-U. Kang et-al. "A Multi-Channel Architecture for High Performance NAND Flash-Based Storage System", Journal of System Architecture, Vol. 53, pp. 644-658, 2007.

Parameter	Common Arch.	Integrated Arch.
File Storage Occupancy [KB]	$P \cdot \sum_{i=1}^{n} \left\lceil \frac{x_i}{P} \right\rceil$	$P \cdot \left(\left\lceil \frac{x_1}{P} \right\rceil + \sum_{i=1}^{n-1} \left\lceil \frac{d_i}{P} \right\rceil \right)$
Write Time [μS]	$\frac{C_{std}(X)}{P} \cdot t_{WR}$	$\frac{C_{\text{int}}(X)}{P} \cdot t_{WR} + D_{WR}(X)$
Read Time [μS]	$\frac{\max\{x_1, ..., x_n\}}{P} \cdot t_{RD}$	$\frac{C_{\text{int}}(X)}{P} \cdot t_{RD} + D_{RD}(X)$
Endurance [Cycles]	$\left\lceil \frac{C_{std}(X)}{B} \right\rceil$	$\left\lceil \frac{C_{\text{int}}(X)}{B} \right\rceil$

Table 1. Scheme comparison.