Unveiling Page Bloating in SSDs: File Blocks Stored Across Unnecessarily Many Pages

Yuhun Jun
yuhun@skku.edu
Sungkyunkwan University
Suwon, Korea

Euiseong Seo
euiseong@skku.edu
Sungkyunkwan University
Suwon, Korea

Advancements in design and manufacturing processes have significantly enhanced the data storage capacity of flash memory chips. To maintain efficiency as chip capacity expands, manufacturers have opted to increase the size of individual pages [1]. This adjustment is aimed at managing the growing complexity and overhead associated with the flash translation layer (FTL) in solid-state drives (SSDs). Driven by the twin goals of technological progress and cost-effectiveness, SSD page sizes have progressively grown, reflecting the industry's push towards higher density and lower costs. Presently, the standard page size in SSDs falls between 16 KB and 32 KB, whereas initially, it was only 512 Bytes [3].

While the page size of an SSD has been gradually increasing, the filesystem block sizes have remained almost constant over the decades. Although the default file system block size of Linux was 4 KB two decades ago [5], it is still 4 KB. This constancy in file system block size creates a mismatch between SSD page sizes and filesystem block sizes. This discrepancy results in a phenomenon we term as *page bloating*, where consecutive filesystem blocks, despite potentially fitting within a single flash memory page, end up spanning multiple pages due to interleaved writes from different files. This can significantly impact read performance, a concern that has not been adequately addressed in the literature.

When reading files stored with page bloating, blocks that could have been read on a single page in a non-bloated scenario now require reading from multiple pages. Although reading pages from different dies can occur simultaneously, making page bloating less impactful in scenarios with intermittent reads of a few filesystem blocks, continuous reading operations or sustained high SSD workload conditions can lead to significant throughput degradation due to amplified read operations caused by page bloating.

For example, the read performance of Filebench's fileserver workload [4] conducted at a 16 KB size, slowed down by 33%. This issue arises even when large files are pre-allocated to ensure continuity at the file system level. Pre-allocation of a file at the file system level is usually used to avoid fragmented files when the application is supposed to write a small amount of data frequently to the file. When a file is pre-allocated at the file system level, no pages are actually allocated in an SSD, and the actual page allocation is conducted when the application writes the data to the file later. Even in this case, concurrent small writes across multiple files can trigger page bloating. It is most commonly observed in systems such as log servers and databases, where the nature of the workload involves numerous small transactions.

The most straightforward solution to mitigate page bloating involves aligning file system block sizes with SSD page sizes. However, applying this solution would result in significant space wastage due to the prevalence of small files [2] and the non-uniformity of file sizes relative to the page size. For example, in a Linux desktop PC installed with Ubuntu 20.04, if we increase the file system block size to 32 KB from 4 KB, 38.8% of additional storage space is necessary.

Based on these observations, we are currently conducting research on the scheme to detect and rewrite data spread across multiple pages due to filesystem block size and SSD page size mismatches, effectively addressing the root cause of page bloating without requiring modifications to existing filesystem block sizes. Rearranging files that have been impacted by page bloating into their optimal layout is a straightforward task, as it simply involves sequentially rewriting the files after reading them, provided there are no concurrent write operations. However, identifying which filesystem blocks are dispersed due to page bloating requires information beyond what's available to either the filesystem or the SSD alone. Therefore, we are exploring methods to detect these cases with minimal entanglement between the filesystem and the FTL. This approach, still in the exploratory phase, holds promise for eliminating performance degradation attributed to page bloating.

References

[1] Yuhun Jun, Jaehyung Park, Jeong-Uk Kang, and Euiseong Seo. 2022. Analysis and mitigation of patterned read collisions in flash SSDs. *IEEE Access* 10 (2022), 96997–97009.

[2] Marshall K McKusick, William N Joy, Samuel J Leffler, and Robert S Fabry. 1984. A fast file system for UNIX. *ACM Transactions on Computer Systems (TOCS)* 2, 3 (1984), 181–197.

[3] Kang-Deog Suh, Byung-Hoon Suh, Young-Ho Lim, Jin-Ki Kim, Young-Joon Choi, Yong-Nam Koh, Sung-Soo Lee, Suk-Chon Kwon, Byung-Soon Choi, Jin-Sun Yum, et al. 1995. A 3.3 V 32 Mb NAND flash memory with incremental step pulse programming scheme. *IEEE Journal of Solid-State Circuits* 30, 11 (1995), 1149–1156.

[4] Vasily Tarasov, Erez Zadok, and Spencer Shepler. 2016. Filebench: A flexible framework for file system benchmarking. *USENIX; login* 41, 1 (2016), 6–12.

[5] Stephen C Tweedie et al. 1998. Journaling the Linux ext2fs filesystem. In *Proceedings of the The Fourth Annual Linux Expo.* Durham, North Carolina.