Excessive SSD-Internal Parallelism Considered Harmful

Xiangqun Zhang
Syracuse University
xzhang84@syr.edu

Shuyi Pei
Samsung Semiconductor
shuyi.pei@samsung.com

Jongmoo Choi
Dankook University
choijm@dankook.ac.kr

Bryan S. Kim
Syracuse University
bkim01@syr.edu

ABSTRACT

Modern SSDs achieve high throughput by utilizing multiple independent channels and chips in parallel. However, we find that excessive parallelism inadvertently amplifies the garbage collection (GC) overhead due to the larger unit of space reclamation. Based on this observation, we design PLAN, a novel SSD parallelism management and data placement scheme that allocates different levels of parallelism to different workloads with different needs to minimize the GC overhead. We demonstrate the effectiveness of PLAN by evaluating it against other state-of-the-art designs across various real-world workloads. PLAN reduces write amplification with comparable or better performance to the other designs that are always at full parallelism.

CCS CONCEPTS

• **Information systems** → **Flash memory**; • **Computer systems organization** → *Firmware*; *Embedded software.*

KEYWORDS

Solid State Drive, Write Amplification, Garbage Collection

ACM Reference Format:
Xiangqun Zhang, Shuyi Pei, Jongmoo Choi, and Bryan S. Kim. 2023. Excessive SSD-Internal Parallelism Considered Harmful. In *15th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage '23), July 9, 2023, Boston, MA, USA*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3599691.3603412

1 INTRODUCTION

SSD requires time- and resource-consuming internal space reclamation, namely garbage collection (GC) [8, 9]. It is necessary to perform GC because (1) an SSD prohibits in-place updates and (2) the unit of an erase (flash memory block) is much larger than the unit of a program (flash memory page) [3, 31]. Due to these peculiarities, an SSD writes all updates to a new location and invalidates the prior physical copy. To reuse this invalidated physical space the flash memory block must be first erased after relocating all surviving valid data. Garbage collection refers to this SSD-internal data relocation process, and reducing its overhead has been the focal point of research in SSDs for many years [6, 15, 16, 27].

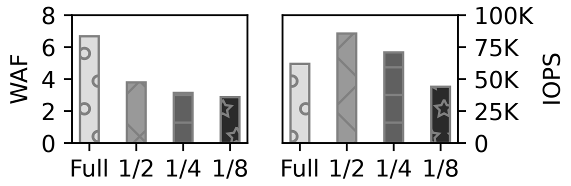

Figure 1: WAF and throughput of SSDs when four different write threads concurrently issue requests with respect to different levels of SSD-internal parallelism, conducted using a 256GiB, FEMU-emulated SSD with greedy GC algorithm.

While there have been significant advancements in the algorithmic aspect of GC, particularly on lifetime-based data characterization schemes [6, 15, 16, 27], the effect of flash memory technological trends on GC overhead is often overlooked. More specifically, the increasing flash memory page size, number of pages per block, and parallelism (number of channels and chips) effectively increase the unit of GC. As an effect, the amount of data copied per space reclaimed statistically increases for the same workload. Figure 1 illustrates this example. We concurrently run four FIO threads writing data of different characteristics to SSDs with different levels of internal parallelism: full parallelism stripes data across all channels and chips, 1/2 parallelism stripes data to all channels but only 1/2 of the chips for each channel, 1/4 to 1/4 of the chips, and 1/8 to 1/8 of the chips. We observe that the write amplification factor (WAF) increases to 230% as the parallelism increases from 1/8 to full. Paradoxically, full parallelism does not achieve the highest throughput due to the high WAF: half parallelism configuration achieves nearly 40% higher throughput than the full.

Based on this observation, we argue that excessive parallelism in SSDs is, in fact, harmful. While a large *superblock*

HotStorage '23, July 9, 2023, Boston, MA, USA
© 2023 Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 979-8-4007-0224-2/23/07…$15.00
https://doi.org/10.1145/3599691.3603412

(a collection of blocks from each chip in the SSD that are written together) improves the overall bandwidth [3, 12], this comes at the cost of high GC overhead, an overhead that affects not only the lifetime but also the performance of the SSD due to interference. The crux of the problem is that a fixed superblock size cannot attain both high peak performance and low GC overhead.

We address this problem by designing PLAN (Parallelism- and Lifetime-aware AllocatioN), a novel SSD parallelism management and data placement scheme that provides different levels of parallelism to different workloads with different needs. Unlike the traditional design with a single level of parallelism across the whole SSD, PLAN rightsizes the amount of parallelism: higher parallelism for bandwidth-intensive I/Os and lower for others. Our work thus improves the overall throughput by significantly reducing the GC overhead.

2 BACKGROUND AND MOTIVATION

In this section, we describe the algorithmic aspects of garbage collection (GC) and the common approach in reducing GC-induced write amplification factor (WAF). We then discuss further details on how parallelism affects GC performance.

2.1 Garbage collection essentials

The amount of data copied during GC is affected by (1) when GC is performed and (2) which flash memory block the algorithm selects to clean. Lazily cleaning allows the SSD to absorb more writes and to have more data invalidated, but this can block host writes if GC is unable to reclaim space on time[22, 24]. Thus, modern GC algorithms carefully select their trigger conditions (e.g., when available free blocks drop below a watermark) to reduce data copies while ensuring that free blocks are available for incoming host requests[8, 9].

To select a victim block to clean, an SSD not only maintains per-block information such as valid count and age, but also efficiently organizes blocks to facilitate the selection. A common garbage collection approach is to greedily choose the block with the least valid data as the victim, denoted as greedy GC[8, 9]. To reduce the overhead of traversing through per-block metadata for selection, mechanisms such as windowed GC or random selections are considered to realize a fast and efficient victim selection[13, 25]. In addition, algorithms such as cost-benefit consider the age of the data to avoid cleaning soon-to-be-updated hot data[24], and identifying infrequently-updated cold data and separating them from hot reduces the amount of data copies by creating a bimodal distribution of the valid data among blocks[22].

The separation of hot and cold data can be further assisted by host-level hints. The multi-stream interface allows the host to tag data with different stream IDs, which are used by the SSD to separate them during block allocation[15]. Its successor, AutoStream, provides the automatic assignment of stream IDs based on I/O characteristics[27]. Other stream-based approaches, including FStream[23], PCStream[16], and WARCIP[28], provide automatic stream ID assignment based on filesystem metadata, program context, and minimization of write interval differences. These approaches require support from the application, the host system, and/or the SSD, bringing extra overheads for their adoption.

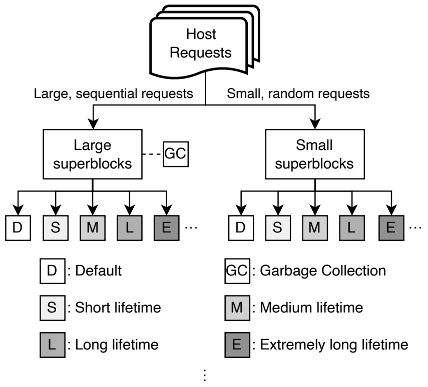

Figure 2: The overall design of PLAN. Data will be written to different superblocks based on their size and lifetime characteristics.

2.2 Effect of SSD parallelism on GC

With the significantly grown SSD capacity over the past decade, the unit of SSD internal management has also grown larger to reduce management overhead. The increasing number of pages per block and blocks per chip requires more information to be tracked for the FTL. Therefore, modern SSDs usually handle incoming write requests on the superblock level[14, 20, 21]. Data are striped over all available channels and chips to increase parallelism and reduce DRAM usage. The SSD also performs GC on the superblock level, where it chooses a superblock as the GC victim, relocates any valid pages in it, and erases the whole superblock[26].

However, a larger management unit trades off performance for space[11]. As larger superblocks lead to fewer GC candidates, the GC algorithm must erase the whole victim even if its valid page distribution is skewed since few workloads are entirely uniform; more valid pages will be relocated during the GC process, which causes performance degradation and shortens SSD lifetime. Therefore, SSDs should have smaller superblocks for better GC efficiency.

3 DESIGN

Figure 2 shows the overall structure of PLAN. SSD requires large superblocks with high parallelism to prevent large, sequential I/O from stalling but favors small superblocks for GC, so our design must satisfy both requirements. Large, sequential requests continue using large superblocks with full parallelism, while small, random requests are directed to small superblocks. On the other hand, Prior works have shown the effectiveness of lifetime-based data characterization schemes[15, 16, 27]. Based on the predicted data lifetime, the SSD chooses the most appropriate target superblock to write data of similar lifetimes to the same superblock.

Compared to prior works, our approach is implemented entirely inside the SSD. This means PLAN is transparent to users; it does not require any application[15, 30], filesystem[23], operating system[16, 27], or SSD protocol[15] modification when compared to prior works. Furthermore, no prior training is needed compared to machine learning-based approaches[5, 29]. This eliminated the need for changing anything except for the SSD itself in the storage stack.

3.1 Flexible Superblocks

Traditional SSDs organize blocks into a superblock, which contains a block from every single chip[14, 26]. This design ensures incoming data can be striped across all available channels and chips. Intuitively, this provides better parallelism and guarantees higher throughput, but this effectively increases the superblock size, which is also the unit of GC[26]. However, a larger GC unit causes higher WAF. Therefore, it is necessary to reduce the size of GC units.

How does an SSD reduce the size of a superblock? The size of a traditional SSD superblock striping across all channels and chips equals to *pages per block* × *chips per channel* × *# of channels* in the SSD, as each superblock uses one block per chip. Reducing one of these factors can reduce the size of the GC unit. *Pages per block* used per superblock cannot be changed as the smallest erase unit is a block. This leaves the only choices to be *chips per channel* and *# of channels*, affecting SSD parallelism. PLAN reduces the superblock size by reducing *chips per channel* used for each channel, as each channel can handle I/O requests independently[31]; reducing the *# of channels* used per superblock may cause several chips to share the same channel, which is undesirable since this can congest the channel and stall the request.

With this problem in mind, PLAN uses all available channels for all superblocks, similar to traditional SSDs, but it uses 2 chips per channel (1/4 parallelism) for small superblocks as the default configuration since further reducing parallelism degrades performance without significantly reducing WAF compared to the baseline. Using 1/4 parallelism for small superblocks also shows the best *overall* throughput in our evaluation. PLAN SSD keeps a given number of big and small superblocks concurrently open to host writes and uses full parallelism collectively to serve requests of different sizes. The SSD can easily obtain the number of pages required by the request by checking the starting/ending sectors and decide if a big or small superblock should serve the request. Big superblocks still span across all chips on all channels, similar to traditional SSDs; sequential requests spanning across 8 pages or more can continue enjoying high throughput with a higher level of parallelism, whereas the WAF of small, random requests spanning across less than 8 pages can be reduced with smaller superblocks.

The GC algorithm must also be modified since PLAN supports superblocks of different sizes. All superblocks in traditional SSDs are the same size, meaning the greedy GC can easily choose the victim by finding the superblock with the least number of valid pages. PLAN, on the other hand, have different superblock sizes for big and small superblocks. Naïvely choosing the superblock with the least number of valid pages leads to a constant selection of small superblocks as GC victim. PLAN addresses this issue by choosing the superblock with the least valid data ratio. This ensures the SSD always picks the superblock that frees up the most space.

3.2 Lifetime Predictor

Lifetime prediction has been proven effective in reducing GC overhead[15, 27]. By grouping data of similar lifetimes, they will be invalidated almost simultaneously, transitioning the superblock from a mostly valid state (where the superblock will not be chosen as GC victim) to a mostly invalid state (where the GC overhead is low). This requires metadata, including lifetime and previous I/O operation, to be recorded for host data.

To save valuable SSD DRAM space, we chunk the logical address space similar to prior works[27, 28]; each chunk requires a 4-byte structure to record its lifetime information, including the chunk's weighted average history lifetime in granularity of seconds, the time of the last write, and the previous operation. Similar to the observation in AutoStream, our evaluation shows smaller chunk size does not always bring better performance[27], and a chunk size of 32 pages performs the best overall. This requires a total of 8MiB of DRAM for a 1TiB SSD with a 16 KiB page size.

When a chunk is overwritten or invalidated, we calculate a weighted average using the old history lifetime and the chunk lifetime since the previous write as the new history lifetime. We use a weight of 0.1 for the old history lifetime as it performs the best overall by quickly adapting to changing workload characteristics. The newly calculated weighted average will be used as the expected lifetime of this write and be recorded as the new history lifetime.

After the lifetime predictor calculates the expected lifetime, the SSD chooses an open superblock to write to. Data with different lifetimes are written to different superblocks by this mechanism to create the desired bimodal distribution. As prior work shows data in different lifetime-based superblocks show exponential lifetime range[16], lifetime-based superblocks are designed to accept data of

Table 1: Evaluation configurations.

FEMU			
Channels	8	Page size	16KiB
Chips per channel	8	Physical capacity	256GiB
Planes per chip	1	Logical capacity	240GiB
Blocks per plane	1024	Over-provisioning	0.0625
Pages per block	256	GC Policy	Greedy
CPU cores per VM	12	DRAM per VM	8GB

exponentially-sized lifetime ranges: the short lifetime superblock accept data (denoted S in Figure 2) with a lifetime smaller than 1s; medium lifetime superblock (denoted M) accepts data with a lifetime between 1 and 3s; long lifetime superblock (denoted L) accepts data with a lifetime between 3 and 7s, and so on.

Another tunable parameter is the number of concurrently opened superblocks for data of different lifetimes. Figure 2 shows an example of 4 large superblocks and 4 small superblocks for host writes based on data lifetime; each of them accepts data with different lifetimes. More concurrently opened superblocks provide better granularity when separating data of different lifetimes[27, 28]. Therefore, we use 8 large superblocks and 8 small superblocks for PLAN, as commercial SSDs can have at least 16 concurrently opened superblocks[27]. There is also a big superblock and a small superblock as default superblocks for data without prior lifetime information since more concurrently opened superblocks are allowed if they are internal to the SSD[16]. Lastly, an extra superblock with full parallelism for GC prevents GC-incurred latency caused by the reduced parallelism.

With the lifetime predictor grouping data by their lifetime characteristics and PLAN providing a smaller GC unit, we can effectively reduce the GC overhead.

4 EVALUATION

We implement PLAN on our customized FEMU[17] with TRIM and multi-stream support[2]. Table 1 shows the parameters we use in our evaluation. We conduct the following evaluations with workloads of different I/O characteristics.

- **FIO** workload with 4 FIO processes running concurrently. Each process issues 256GiB of I/O in total (50% read, 50% write) to differently sized files (size ratio 1:10:30:64) on different partitions similar to AutoStream[27], creating data of different lifetime characteristics. All I/Os are uniformly random and 4KiB in size using `libaio` with `iodepth=32`.
- **TPC-C** workload with 30 terminals and 30 warehouses using BenchBase[10], running on MySQL with a 4KiB InnoDB page size.
- **YCSB**[7], which first inserts 100 million key-value pairs, then issues 50 million operations (workload A: 50% read, 50% update) on top of RocksDB[19].
- **Fileserver** workload from Filebench[1] with 1 million 128KiB files and 16 instances.
- **GCC** Linux kernel compilation workload with 100 iterations. Each iteration copies the Linux kernel source code to the SSD, compiles the kernel, and deletes the source code and compiled binary.

For every workload, we run it under the following settings with the same number of concurrently opened superblocks and use all 8 channels for every superblock.

- **Baseline**: Traditional SSD, no data classification scheme.
- **Partial GC**: Traditional SSD but using partial GC, i.e. the SSD still uses full parallelism for superblocks and stripes data across all available channels and chips, but the SSD chooses a block (instead of superblock) as the GC victim with the least amount of valid pages.
- **Multi-stream**: Traditional SSD with multi-stream[15].
- **AutoStream**: Traditional SSD with AutoStream (SFR)[27].
- **PLAN-NoPred-1/4**: PLAN without any data classification scheme. All large, sequential requests are sent to the default big superblock, whereas all small, random requests are sent to the default small superblock. Small superblocks use all channels and 2 chips per channel (1/4 parallelism).
- **PLAN-NoPred-1/8**: Similar to **PLAN-NoPred-1/4** but small superblocks use 1 chip per channel (1/8 parallelism).
- **PLAN-Pred-1/4**: PLAN with in-SSD lifetime prediction. Small superblocks use all channels and 2 chips per channel (1/4 parallelism). This is the recommended configuration.
- **PLAN-Pred-1/8**: Similar to **PLAN-Pred-1/4** but small superblocks use 1 chip per channel (1/8 parallelism).

To emulate an aged SSD with existing user files, we fill the SSD using valid data to certain levels before evaluations[6, 16, 22, 23]. For multi-stream evaluations, we assign a stream ID to a write request based on the issuing process name if the process did not provide a stream ID to the request; requests from different processes will be assigned with different stream IDs. If an application (i.e., FIO and YCSB) assigns stream ID to a request, we leave the application-assigned stream ID untouched.

4.1 Write Amplification Reduction

Figure 3 shows the WAF of the evaluations. Workloads issuing many small, random writes, including FIO, TPC-C, Fileserver, and GCC, reduces WAF by at least 25% on PLAN-Pred-1/4. FIO running on PLAN-Pred-1/4 reduces WAF by 42%, showing improved GC efficiency with smaller GC units. The most interesting evaluation is GCC, which reduced WAF by more than half to a negligible level between 1.04 and 1.07

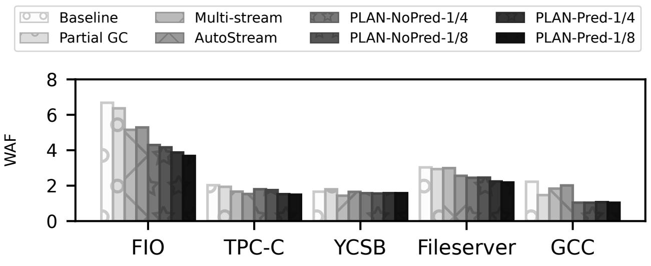

Figure 3: WAF of the workloads. Lower is better.

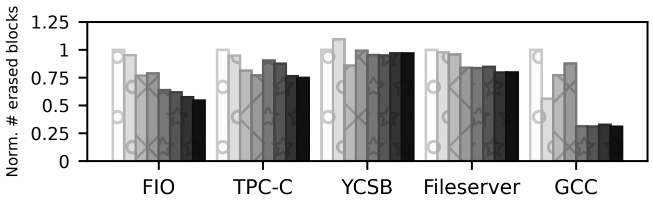

Figure 4: Total number of blocks erased. Lower is better.

using PLAN. The workload deletes the source code and compiled binaries after each iteration, which invalidates all data created during the iteration before the new iteration starts. Ideally, this should not incur any GC overhead. However, for all other approaches other than PLAN, files created by the GCC workload may be mixed with existing user data, causing existing user data to be relocated during GC. PLAN, however, separated cold user files from hot, ephemeral GCC data. After each iteration, superblocks containing GCC data can be fully invalidated as the workload data are deleted. On the other hand, PLAN reduced WAF by 5% for YCSB, which issues mostly large, sequential writes. Similar to the observation from Figure 1, reducing parallelism from full to 1/4 shows significant WAF reduction, but reducing parallelism from 1/4 to 1/8 reduces WAF marginally.

Figure 4 shows the number of blocks erased during the workload evaluation. PLAN shows the most benefit in FIO and GCC, reducing the number of erased blocks by up to 46% and 69% compared to the baseline. Figure 5 shows the CDF of the valid data ratio for the victim superblocks during each GC. Curves of PLAN are more toward the left compared to other approaches in FIO, TPC-C, Fileserver, and GCC, showing better GC performance. For the YCSB workload, multi-stream performs the best since the RocksDB database has the best knowledge of data lifetime expectations; the database directly tags write requests with a stream ID, which the SSD will use to group the incoming data accordingly. Together with fewer blocks erased and a less valid data ratio in each victim superblock, PLAN can reduce WAF significantly.

Last but not least, Figure 3 to Figure 5 all show partial GC by single blocks does not provide much benefit compared to greedy GC by superblocks. This is because each block only receives shards of correlated data from a request, as data is striped among all channels and chips used by the superblock to prevent stalls during writes. This indicates data from the same GC unit (i.e., block) will not be invalidated at similar times, preventing the desired bimodal distribution of the GC units. On the other hand, if we characterize data by lifetime and write them to individual *blocks*, the GC will be efficient, but this wastes the SSD's internal parallelism, which leads to performance degradation. Therefore, keeping the GC unit at the superblock level and also with some parallelism is essential.

4.2 Performance

As PLAN reduces SSD parallelism, Evaluating PLAN on performance is essential to ensure the design does not hinder performance. Figure 6 shows the throughput information. Although PLAN reduces parallelism for some superblocks, the throughput was not affected even for the YCSB workload with a significant number of large, sequential I/Os.

For CPU-bounded workloads including TPC-C, YCSB and GCC, PLAN keeps (if not improves) the performance while reducing write amplification. Meanwhile, Fileserver benefits more from PLAN with an improvement of around 10%, as the improved GC efficiency frees up SSD internal resources, allowing the SSD to handle more host I/O requests. FIO on PLAN-Pred-1/4 improved throughput by 30%.

The parallelism used for small superblocks also affects performance. Using 1/4 instead of 1/8 parallelism for small superblocks significantly improves FIO performance, as the workload issues 4KiB random requests intensely. Fileserver first issues a large, sequential write request, then issues a small, random write request in each iteration. The extra WAF reduction using 1/8 parallelism for small superblocks allows large, sequential I/O using full parallelism to perform better, leading to a slightly better performance than using 1/4 parallelism. Nevertheless, it is recommended to use 1/4 parallelism than 1/8 for PLAN: FIO, TPC-C, YCSB, and GCC all perform the same or better using 1/4 parallelism even compared to non-PLAN schemes, especially when under intense I/O pressure (i.e., FIO).

5 RELATED WORK

5.1 Multi-stream SSD

Multi-stream SSDs are created to enable on-host I/O characteristic classification schemes[15]. These schemes assign a stream ID to the data based on data I/O characteristics, as the host has more information regarding the workload. SSD will then place data with the same stream ID to the same erase unit (i.e., superblock). This requires change for the operating system, SSD interface, and SSD. Aside from these changes, a stream ID delegate is also required to manage stream ID assigned to the I/O requests. The original

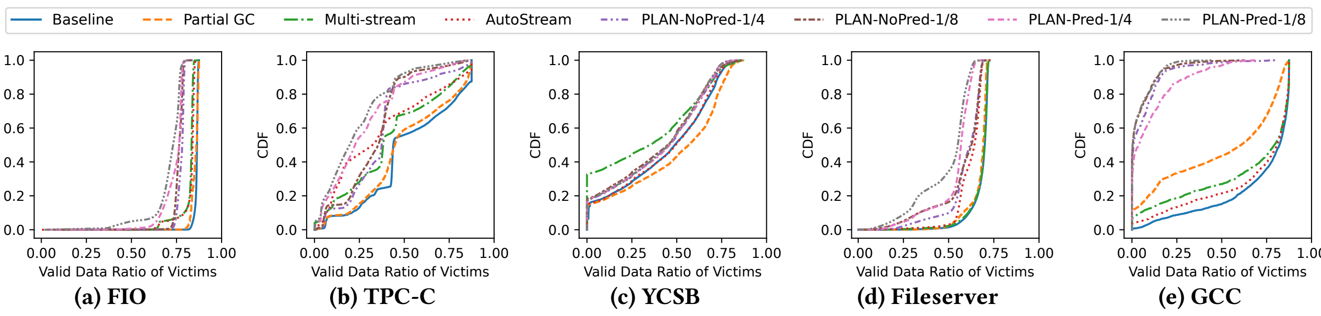

Figure 5: CDF of GC victim's valid data ratio during each GC. A line towards the left indicates fewer valid pages relocated during GC, i.e., better GC efficiency.

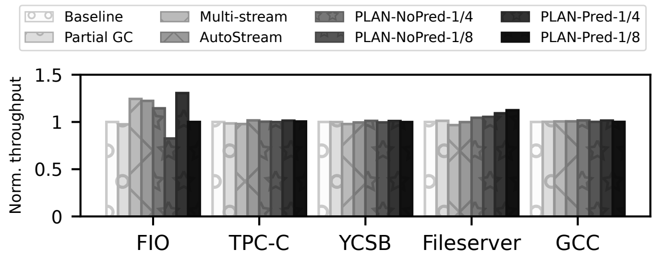

Figure 6: Normalized throughput. Higher is better.

multi-stream requires changes on existing applications, as the applications must explicitly assign a stream ID to their I/O requests[15]. AutoStream provides a daemon for automatically assigning stream ID based on data characteristics, including access sequentiality, frequency, and recency[27]. FStream automatically separates different filesystem metadata and user data[23] to different streams. PCStream automatically separates requests from different program contexts (PCs) based on the call stack by arguing write requests from different PCs show different characteristics[16].

Despite its promises, Linux kernel recently removed the support for multi-stream, disconnecting the communication link between the on-host data characterization schemes and multi-stream SSDs. The maintainers stated a lack of support from the drive vendors and the applications[18]. Due to the removal of the multi-stream interface, the host cannot coordinate with the SSD on I/O characteristics easily as before. However, prior works show I/O characterization is effective in reducing WAF. To leverage the benefits of I/O characterization, the job should be offloaded to the SSD (e.g., PLAN) so that the users can immediately enjoy lower WAF and higher throughput without user intervention and system changes.

5.2 Zoned Namespaces (ZNS) SSD

Similar to multi-stream SSDs, ZNS SSDs also allow hosts to participate in the data placement process, as the host has more data characterization information. ZNS SSDs expose zones to the host, which is similar to the stream in multi-stream SSDs. Similar to erase units of a traditional SSD, data can only be appended to zones in a log-like manner, and a zone has to be erased to reclaim space after it is full. The host is in charge of not only data placement based on some mechanism (e.g., data characterization schemes) to zones but also garbage collection, which reduces the WAF and increases the SSD throughput by improving GC efficiency[4].

Also similar to multi-stream SSDs, major changes in the whole storage stack are required to support ZNS SSDs. application, OS, filesystem, SSD protocol, and SSD are all required to be changed to support ZNS SSDs. The application and/or the filesystem also have to choose a zone to write to, and the OS and the SSD protocol have to be modified to support ZNS SSDs[4]. Therefore, PLAN requires less change than ZNS SSDs and less user intervention.

6 CONCLUSION

In this paper, we present PLAN, a novel SSD parallelism and data placement scheme that provides different levels of parallelism to different workloads with different needs. PLAN reduces WAF by up to 52% and increases throughput up to 30% for I/O-bounded workloads with small, random writes without sacrificing the performance of large, sequential writes.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable suggestions. This work is supported in part by the Ministry of Science and ICT (MSIT) of Korea under the High-Potential Individuals Global Training Program (RS-2022-00154903) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation), and Samsung Memory Solutions Lab through the National Science Foundation I/UCRC ASIC (Alternative Sustainable and Intelligent Computing) Center (CNS-1822165).

REFERENCES

[1] 2011. filebench/filebench: File system and storage benchmark that uses a custom language to generate a large variety of workloads. https://github.com/filebench/filebench

[2] 2017. NVMe streams directive function support · multi-stream/qemu@9888fff. https://github.com/multi-stream/qemu/commit/9888fffbf3695f7de4170e97f49231ea18fa8cd4#. (Accessed on 03/29/2023).

[3] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark Manasse, and Rina Panigrahy. 2008. Design Tradeoffs for SSD Performance. In *USENIX 2008 Annual Technical Conference* (Boston, Massachusetts) *(ATC'08)*. USENIX Association, USA, 57–70.

[4] Matias Bjørling, Abutalib Aghayev, Hans Holmberg, Aravind Ramesh, Damien Le Moal, Gregory R. Ganger, and George Amvrosiadis. 2021. ZNS: Avoiding the Block Interface Tax for Flash-based SSDs. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 689–703. https://www.usenix.org/conference/atc21/presentation/bjorling

[5] Chandranil Chakraborttii and Heiner Litz. 2021. Reducing Write Amplification in Flash by Death-Time Prediction of Logical Block Addresses. In *Proceedings of the 14th ACM International Conference on Systems and Storage* (Haifa, Israel) *(SYSTOR '21)*. Association for Computing Machinery, New York, NY, USA, Article 11, 12 pages. https://doi.org/10.1145/3456727.3463784

[6] M.-L. Chiang and R.-C. Chang. 1999. Cleaning policies in mobile computers using flash memory. *Journal of Systems and Software* 48, 3 (1999), 213–231. https://doi.org/10.1016/S0164-1212(99)00059-X

[7] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing* (Indianapolis, Indiana, USA) *(SoCC '10)*. Association for Computing Machinery, New York, NY, USA, 143–154. https://doi.org/10.1145/1807128.1807152

[8] Peter Desnoyers. 2012. Analytic Modeling of SSD Write Performance. In *Proceedings of the 5th Annual International Systems and Storage Conference* (Haifa, Israel) *(SYSTOR '12)*. Association for Computing Machinery, New York, NY, USA, Article 12, 10 pages. https://doi.org/10.1145/2367589.2367603

[9] Peter Desnoyers. 2014. Analytic Models of SSD Write Performance. *ACM Trans. Storage* 10, 2, Article 8 (mar 2014), 25 pages. https://doi.org/10.1145/2577384

[10] Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudré-Mauroux. 2013. OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases. *PVLDB* 7, 4 (2013), 277–288. http://www.vldb.org/pvldb/vol7/p277-difallah.pdf

[11] Aayush Gupta, Youngjae Kim, and Bhuvan Urgaonkar. 2009. DFTL: A Flash Translation Layer Employing Demand-Based Selective Caching of Page-Level Address Mappings. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems* (Washington, DC, USA) *(ASPLOS XIV)*. Association for Computing Machinery, New York, NY, USA, 229–240. https://doi.org/10.1145/1508244.1508271

[12] Jun He, Sudarsun Kannan, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2017. The Unwritten Contract of Solid State Drives. In *Proceedings of the Twelfth European Conference on Computer Systems* (Belgrade, Serbia) *(EuroSys '17)*. Association for Computing Machinery, New York, NY, USA, 127–144. https://doi.org/10.1145/3064176.3064187

[13] Xiao-Yu Hu, Evangelos Eleftheriou, Robert Haas, Ilias Iliadis, and Roman Pletka. 2009. Write Amplification Analysis in Flash-Based Solid State Drives. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference* (Haifa, Israel) *(SYSTOR '09)*. Association for Computing Machinery, New York, NY, USA, Article 10, 9 pages. https://doi.org/10.1145/1534530.1534544

[14] Jian Huang, Anirudh Badam, Laura Caulfield, Suman Nath, Sudipta Sengupta, Bikash Sharma, and Moinuddin K Qureshi. 2017. FlashBlox: Achieving Both Performance Isolation and Uniform Lifetime for Virtualized SSDs.. In *FAST*, Vol. 17. USENIX Association, 375–390. https://www.usenix.org/conference/fast17/technical-sessions/presentation/huang

[15] Jeong-Uk Kang, Jeeseok Hyun, Hyunjoo Maeng, and Sangyeun Cho. 2014. The Multi-streamed Solid-State Drive. In *6th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 14)*. USENIX Association, Philadelphia, PA. https://www.usenix.org/conference/hotstorage14/workshop-program/presentation/kang

[16] Taejin Kim, Duwon Hong, Sangwook Shane Hahn, Myoungjun Chun, Sungjin Lee, Jooyoung Hwang, Jongyoul Lee, and Jihong Kim. 2019. Fully Automatic Stream Management for Multi-Streamed SSDs Using Program Contexts. In *17th USENIX Conference on File and Storage Technologies (FAST 19)*. USENIX Association, Boston, MA, 295–308. https://www.usenix.org/conference/fast19/presentation/kim-taejin

[17] Huaicheng Li, Mingzhe Hao, Michael Hao Tong, Swaminathan Sundararaman, Matias Bjørling, and Haryadi S Gunawi. 2018. The CASE of FEMU: Cheap, accurate, scalable and extensible flash emulator. In *16th USENIX Conference on File and Storage Technologies (FAST 18)*. USENIX Association, 83–90. https://www.usenix.org/conference/fast18/presentation/li

[18] Torvalds Linus. 2022. Merge tag 'for-5.18/write-streams-2022-03-18' of git://git.kernel.dk/... · torvalds/linux@561593a — github.com. https://github.com/torvalds/linux/commit/561593a048d7d6915889706f4b503a65435c033a. [Accessed 06-Feb-2023].

[19] Meta Platforms, Inc. and Contributors. [n.d.]. RocksDB | A persistent key-value store | RocksDB. https://rocksdb.org/. (Accessed on 03/20/2023).

[20] Micron. 07. TN-29-28: Memory Management in NAND Flash Arrays. https://www.micron.com/-/media/client/global/documents/products/technical-note/nand-flash/tn2928.pdf. (Accessed on 03/21/2023).

[21] Micron. 18. TN-FD-22: Client SATA SSD SMART Attribute Reference. https://www.micron.com/-/media/client/global/documents/products/technical-note/solid-state-storage/tnfd22_client_ssd_smart_attributes.pdf. (Accessed on 03/21/2023).

[22] Changwoo Min, Kangnyeon Kim, Hyunjin Cho, Sang-Won Lee, and Young Ik Eom. 2012. SFS: Random Write Considered Harmful in Solid State Drives. In *10th USENIX Conference on File and Storage Technologies (FAST 12)*. USENIX Association, San Jose, CA. https://www.usenix.org/conference/fast12/sfs-random-write-considered-harmful-solid-state-drives

[23] Eunhee Rho, Kanchan Joshi, Seung-Uk Shin, Nitesh Jagadeesh Shetty, Jooyoung Hwang, Sangyeun Cho, Daniel DG Lee, and Jaeheon Jeong. 2018. FStream: Managing Flash Streams in the File System. In *16th USENIX Conference on File and Storage Technologies (FAST 18)*. USENIX Association, Oakland, CA, 257–264. https://www.usenix.org/conference/fast18/presentation/rho

[24] Mendel Rosenblum and John K. Ousterhout. 1992. The Design and Implementation of a Log-Structured File System. *ACM Trans. Comput. Syst.* 10, 1 (feb 1992), 26–52. https://doi.org/10.1145/146941.146943

[25] Benny Van Houdt. 2013. A Mean Field Model for a Class of Garbage Collection Algorithms in Flash-Based Solid State Drives. In *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems* (Pittsburgh, PA, USA) *(SIGMETRICS '13)*. Association for Computing Machinery, New York, NY, USA, 191–202. https://doi.org/10.1145/2465529.2465543

[26] Shunzhuo Wang, Fei Wu, Chengmo Yang, Jiaona Zhou, Changsheng Xie, and Jiguang Wan. 2019. WAS: Wear Aware Superblock Management for Prolonging SSD Lifetime. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*. 1–6. https://ieeexplore.ieee.org/document/8806851

[27] Jingpei Yang, Rajinikanth Pandurangan, Changho Choi, and Vijay Balakrishnan. 2017. AutoStream: Automatic Stream Management for Multi-Streamed SSDs. In *Proceedings of the 10th ACM International Systems and Storage Conference* (Haifa, Israel) *(SYSTOR '17)*. Association for Computing Machinery, New York, NY, USA, Article 3, 11 pages. https://doi.org/10.1145/3078468.3078469

[28] Jing Yang, Shuyi Pei, and Qing Yang. 2019. WARCIP: Write Amplification Reduction by Clustering I/O Pages. In *Proceedings of the 12th ACM International Conference on Systems and Storage* (Haifa, Israel) *(SYSTOR '19)*. Association for Computing Machinery, New York, NY, USA, 155–166. https://doi.org/10.1145/3319647.3325840

[29] Pan Yang, Ni Xue, Yuqi Zhang, Yangxu Zhou, Li Sun, Wenwen Chen, Zhonggang Chen, Wei Xia, Junke Li, and Kihyoun Kwon. 2019. Reducing Garbage Collection Overhead in SSD Based on Workload Prediction. In *11th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 19)*. USENIX Association, Renton, WA. https://www.usenix.org/conference/hotstorage19/presentation/yang

[30] Hwanjin Yong, Kisik Jeong, Joonwon Lee, and Jin-Soo Kim. 2018. vStream: Virtual Stream Management for Multi-streamed SSDs. In *10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 18)*. USENIX Association, Boston, MA. https://www.usenix.org/conference/hotstorage18/presentation/yong

[31] Jinsoo Yoo, Youjip Won, Joongwoo Hwang, Sooyong Kang, Jongmoo Choi, Sungroh Yoon, and Jaehyuk Cha. 2013. VSSIM: Virtual machine based SSD simulator. In *2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*. 1–14. https://doi.org/10.1109/MSST.2013.6558443