FIOS: Feature Based I/O Stream Identification for Improving Endurance of Multi-Stream SSDs

Janki Bhimani*, Ningfang Mi*, Zhengyu Yang*, Jingpei Yang†,
Rajinikanth Pandurangan†, Changho Choi†, and Vijay Balakrishnan†
* Dept. of Electrical & Computer Engineering, Northeastern University, Boston, MA 02115
† Memory Solution Research Lab, Samsung Semiconductor Inc., San Jose, CA 95134

***Abstract*—The demand for high speed 'Storage-as-a-Service' (SaaS) is increasing day-by-day. SSDs are commonly used in higher tiers of storage rack in data centers. Also, all flash data centers are evolving to better serve cloud services. Although SSDs guaranty better performance when compared to HDDs, but SSDs endurance is still a matter of concern. Storing data with different lifetime in an SSD can cause high write amplification and reduce the endurance and performance of SSDs. Recently, *multi-stream* SSDs have been developed to enable data with different lifetime to be stored in different SSD regions and thus reduce write amplification. To efficiently use this new multi-streaming technology, it is important to choose appropriate workload features to assign the same streamID to data with similar lifetime. However, we found that streamID identification using different features may have varying impacts on the final write amplification of multi-stream SSDs. Therefore, in this paper we develop a portable and adoptable framework to study the impacts of different workload features and their combinations on write amplification. We also introduce a new feature, named "coherency", to capture the friendship among write operations with respect to their update time. Finally, we propose a feature-based stream identification approach, which co-relates the measurable workload attributes (such as I/O size, I/O rate, etc.) with high level workload features (such as frequency, sequentiality etc.) and determines a good combination of workload features for assigning streamIDs. Our evaluation results show that our proposed approach can always reduce the Write Amplification Factor (WAF) by using appropriate features for stream assignment.**

***Keywords*—Multi-Streaming, Write Amplification Factor (WAF), StreamID Identification, Coherency**

I. Introduction

The SSDs have become the main building blocks to support enterprise data centers and virtualized cloud environments due to their high I/O bandwidth compared to traditional hard disk drives (HDDs). With prevailing flash devices everywhere from specialized computing servers to user laptops, the special characteristics of flash such as Wear Leveling, Garbage Collection (GC) and Write Amplification (WA) are now well understood. Thus, although with many advantages of NAND flash technology, one major drawback lies in its internal *Write Amplification* (WA).

Multi-stream SSDs: In real I/O intensive applications, data often have high variability in their lifetime. This inevitably causes a large degree of data fragmentation due to data invalidation and then dramatically increases the WAF when garbage collection is triggered. To address this issue, the storage industry recently developed a new multi-streaming technology [2] that allows a host system to explicitly open different "streams" in SSD devices and allocate write requests to these streams according to the expected lifetime of data.

This work was initiated during Janki Bhimani's internship at Samsung Semiconductor Inc. [1]. This work was partially supported by National Science Foundation Career Award CNS-1452751, and Samsung Semiconductor Inc. Research Grant.

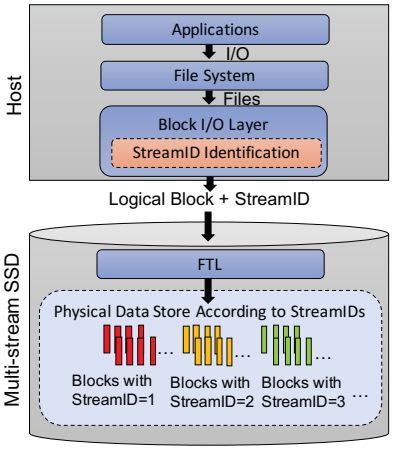


Fig. 1. Operation of Multi-stream SSDs with respect to I/O stack

Traditional SSDs have only one active append point where new data is written. Now, multi-stream technology enables the device to maintain more than one open erase blocks to append data writes in different physical locations of an SSD. In a multi-stream SSD [3], *streamIDs* are assigned to data according to their lifetime. The assignment of streamIDs can be done at any layers, such as the application layer, the file system layer or the block layer. Figure 1 shows the I/O stack of a multi-stream SSD where streamID identification is done at the block layer. Once each logical block is assigned a streamID, a list of logical blocks with their corresponding streamIDs will be sent as an input to the Flash Translation Layer (FTL) in the multi-stream SSD device. The FTL then stores data blocks with the same streamID to the same physical blocks. This ensures that data with the same lifetime (i.e., the same streamID) can be invalidated together, which thus reduces the garbage collection overhead and results into low write amplification by avoiding extra internal data movements.

Challenges of Using Multi-Stream Technology: An efficient streamID assignment in multi-stream flash drives can reduce write amplification, and improve endurance of SSDs. To achieve the best possible benefit of multi-stream SSDs, it is critically important to construct good streams. To frame good streams, data with similar lifetime should be assigned the same streamIDs. It is challenging to predict the data lifetime. The historical information of different features (such as frequency, sequentiality, etc.) for past data accesses can be used to predict the expected lifetime of data. However, we found that stream identification using different features may have different impacts on the WAF of multi-stream SSDs. Moreover, feature sets that can accurately capture the expected lifetime of data may vary with different applications and workloads. Using a combination of not useful features cannot offer good performance improvement of multi-stream SSDs over no-streaming legacy SSDs. Therefore, a good streamID

2159-6190/18/$31.00 ©2018 IEEE
DOI 10.1109/CLOUD.2018.00010

assignment technique to quantify the impact of different features and their combinations on the endurance of SSDs is important.

Novel StreamID Assignment Approach: In order to efficiently use multi-stream SSDs, we develop a portable and adoptable multi-stream framework to extract various I/O workload features and study the impacts of these features and their combinations on write amplification of multi-stream SSDs. Some well-known features, such as frequency, adjacent access, and sequentiality, are considered in the feature extraction for capturing the lifetime of data. But, we sometimes observed that these features are not sufficient to capture the lifetime of data when using them individually. We thus propose a new feature, named *coherency*, to capture the friendship between logical blocks. Coherency can be more closely related to the lifetime of data. By investigating all these different features, we found that (1) none of the features (such as frequency, adjacent access, sequentiality and coherency) can be claimed as the best for all I/O workloads, (2) different features have varying impacts on WAF, and (3) the benefit derived by using the combination of multiple features is not additive. Therefore, another big challenging issue in developing this multi-stream framework is *how to determine a combination of workload features that is best for assigning appropriate streamIDs under a given I/O workload*.

To address this issue, we build an analytical *correlation model* to capture the co-relation between easily obtained workload characteristics (such as I/O size, random write ratio, reuse ratio, and autocorrelation of write rates) with high-level workload features (such as frequency and coherency) and develop a novel Feature-based I/O Stream identification (`FIOS`) method to identify best set (or combination) of features suitable to a workload for streamID assignment. `FIOS` can obtain a good feature combination automatically, rather than experimenting all possible combinations. We modify the SSD module of DiskSim[1] [4] to simulate the multi-stream SSDs. We implement `FIOS` in modified DiskSim and compare `FIOS` with the existing equal-partition (EP) streamID identification algorithm that is currently used in multi-stream SSDs. We also consider the legacy SSDs that do not use the multi-streaming technology as the baseline. Our experimental results show that `FIOS` can always identify a good combination of appropriate features to decide streamIDs and thus be able to improve the lifetime of SSD devices by reducing WAF.

II. Framework Design

In this section, we explain the design and the main components of our framework.

A. `FIOS` Design

The block diagram of our `FIOS` technique is shown in Figure 2. `FIOS` consists of two main phases: *training* and *testing*. Training is the pre-processing step for streamID detection which needs to be performed only once. Testing represents the actual runtime phase of applications, during which streamID assignment is performed. Training and testing phases are performed in a cyclic pattern to capture the runtime workload changes. Here, we use a single cycle of these two phases to explain how `FIOS` operates. `FIOS` first uses `blktrace` and `blkparse` commands to obtain a real I/O block trace from the application platform as a training trace. This trace is then used by the training phase to extract the required features such as frequency, adjacent access, sequentiality and coherency. Later, we describe the details of how each feature is extracted from an I/O trace in Section II-B.

The captured features are enclosed in form of a feature matrix, such as the one shown in Figure 3. Accordingly, the feature matrix consists of $n \times m$ cells, where m is the number of features analyzed for streamID detection and n is the total number of sector_chunks in the storage volume. The storage volume may include a single SSD or multiple SSDs. Each sector_chunk comprises of several sectors. Here, we set 64 sectors on a disk as one sector_chunk under the consideration of the storage overhead of streamID computation and its efficiency. The sector_chunks in rows are arranged in an incremental numerological order such that the product of row number and each sector_chunk's size gives the sector_chunk address. Each cell in the feature matrix gives data quanta which reflects the importance of the i^{th} sector_chunk with respect to the j^{th} feature. Note that if a sector_chunk consists of only a single block address, then that sector_chunk would be the same as a logical block address (LBA). Thus, we use sector_chunk and logical block address interchangeably in this paper.

Through feature extraction, `FIOS` creates a feature matrix, which is then used for clustering write/update sector_chunks into different streams. To obtain high computational efficiency, we improved a multi-threaded K-means clustering algorithm [5] to cluster sector_chunks into K streams in parallel. In particular, each feature makes one dimension of clustering inputs and a relative weight factor can be used as an optional input to emphasize the relative importance of each feature in deciding streamIDs. By default, all features are considered to be equally important with the same weights. The number of clusters (i.e., K) of K-means algorithm maps to the number of streams supported by the SSD drive (e.g., 16 streams in the latest SSD drives). `FIOS` uses the K-means algorithm to group all data points in the feature matrix into the given number of streams. The clustering results are stored in the `LBA-StreamID` dictionary that consists of the pairs of sector_chunks and streamIDs.

In the testing phase, we have actual I/O operations (e.g., writes/updates) performed on storage devices. For a read, the operation of legacy and multi-stream SSDs remains the same. For a write or a update, a multi-stream SSD allows multiple append points. Thus, to decide data placement on a multi-stream SSD, `FIOS` assigns a streamID to each sector_chunk through a quick lookup in the `LBA-StreamID` dictionary. Thereafter, the assigned streamID is penetrated through the I/O stack until the data is actually written to the physical address space of that streamID. For an erase, both legacy and multi-stream SSDs work in the same way by searching all finalized blocks for a GC candidate and then copying out valid pages from the candidate.

B. Feature Extraction

Now, we turn to present how `FIOS` extracts workload features from the collected I/O traces and completes a feature matrix. As introduced above, the feature matrix comprises of the importance factor for each sector_chunk with respect to different features. In order to keep low instrumentation

[1] https://github.com/benh/disksim/tree/master/ssdmodel

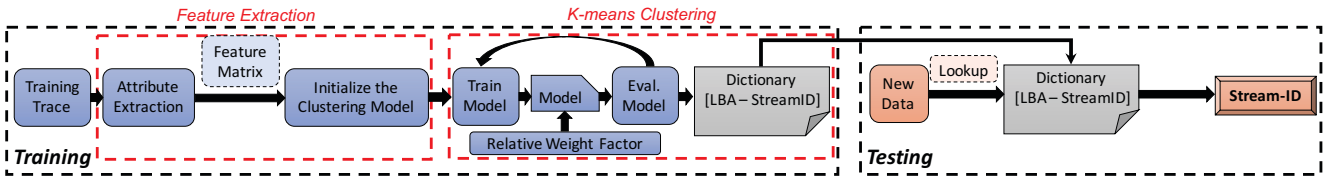


Fig. 2. Block diagram of our FIOS framework.

		Feature_1	Feature_2		Feature_j		Feature_m
1	Sector_chunk_1	0	1		1		0
2	Sector_chunk_2	1	0		0		0
·							
i	Sector_chunk_i	0	1		0		1
·							
n	Sector_chunk_n	1	0		0		1

Fig. 3. Feature matrix data structure.

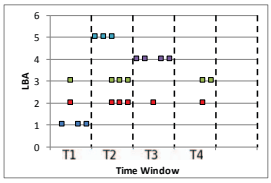


Fig. 4. A sampling graph for describing a novel feature of coherency.

overhead, we decide to represent the importance factor for a sector_chunk as a binary datum. Each entry (i,j) in the feature matrix can then be represented by a single binary bit (e.g., 1 or 0) that indicates whether j^{th} feature is considered to be important ("1") or not ("0"). We use a vector $\overrightarrow{\delta}$, to contain the thresholds for each feature, as criteria to determine the values (i.e., 0 or 1) for each entry.

For example, the feature of frequency indicates how often a particular sector_chunk is accessed. If the number of accesses of that sector_chunk is greater than the predefined threshold (e.g., $\delta[frequency]$ = 4 times), then the frequency entry of that particular address is set as 1, otherwise 0. We set $\delta[frequency]$ to the median value of the frequencies of sector_chunks assuming that the frequencies of sector_chunks follow a Gaussian distribution. The feature of the adjacent access indicates that sector_chunks that are adjacently accessed during a time window are more likely to be accessed together again. We set $\delta[adjacent\_access]$ to construct multiple groups according to the access time of sector_chunks. We can also have the feature of sequentiality to capture if an incoming I/O access is sequential to the previous one.

C. Coherency

We found that these existing well known features like frequency and sequentiality are not sufficient to capture the lifetime of data when using them individually. We thus propose a new feature, named *Coherency*, to capture the friendship between sector_chunks. Coherency can be more closely related to the lifetime of data. We refer to two addresses (i.e., sector_chunks) as friends if we observe that they are mostly updated together in multiple time windows. Intuitively, grouping coherent addresses in the same stream can be beneficial because all sector_chunks in that stream will have a similar update pattern.

Figure 4 illustrates the general idea of the coherency feature, where the x-axis corresponds to time windows and the y-axis corresponds to LBAs or sector_chunks. If a particular group of LBAs that are mostly updated together in multiple time windows, then these LBAs may be referred to as being coherent with each other. For example, if we snoop at a specific time interval which is shown by a vertical dashed line in Figure 4, we can say that LBA2 and LBA3 are coherent because they are concurrently updated in three of the four time windows, i.e., T1, T2 and T4. It indicates that these two sector_chunks (i.e., LBA2 and LBA3) have a tendency to be updated at the same time such that grouping them together into a single stream can help to reduce WAF.

In particular, we maintain lists of unique sector_chunks that have been accessed in each time window. We compare the list of each time window to identify the common sector_chunks that appears in at least two lists. We then mark these common sector_chunks as coherent (i.e., 1). The process is finished when all unique sector_chunks of every time window are allocated their coherency values. In our current model, because we consider the datum of our feature matrix as either 0 or 1, it is a limitation that we cannot differentiate different groups of friends while capturing coherency. As of now, our model only partitions sector_chuncks into two groups, i.e., one consisting of LBAs which are "friendly to somebody" and the other consisting of LBAs which are "friendly to nobody".

D. Combination of Multiple Features

As we discussed, an I/O workload has different features, such as frequency, coherency, etc. How to use one or multiple features to cluster data points into the desired number of streams is not trivial. We found using multiple features might give better performance than using a single feature. However, we also found that using unsuitable features to assign streamIDs may not be able to help improving the performance and even may cause performance degradation. Thus, a critical issue is how to find out an optimal combination of features which can enable `FIOS` to achieve high quality of streamID packetization with the minimum overhead. Given n features, we can have $2^n - 1$ possible combinations. We investigate and evaluate the impacts of these feature combinations on the write amplification of multi-stream SSDs in Sec. IV. Investigating the results for many workloads, we further propose a new approach to determine a good feature combination in Sec. IV-C.

III. Main Architecture

In this section, we present the I/O stack architecture overview of our `FIOS` implementation and introduce the basic data structures used in the implementation. Following that, we discuss the modifications on an existing SSD simulator, i.e., Disksim [4] to enable the multi-stream interface.

A. FIOS: Architecture Overview

Our prototype of `FIOS` can be implemented at any levels, such as the file system layer on the host side or the FTL layer

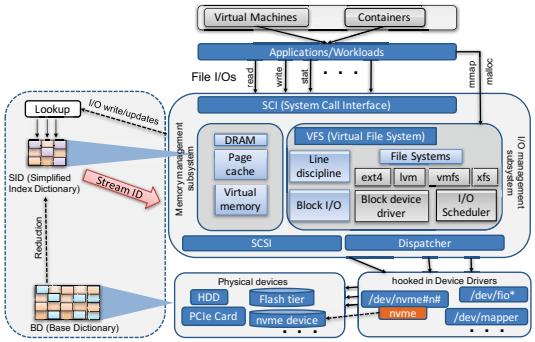


Fig. 5. The I/O stack of FIOS architecture.

inside the device. In our implementation, we choose to deploy our prototype at the block layer. Later in Section IV-E, we will discuss the benefits of this design choice. Figure 5 shows the I/O stack of FIOS architecture, which consists of two main components to assist in streamID assignment, i.e., (1) Base Dictionary (BD) and (2) Simplified Index Dictionary (SID). As shown in Figure 5, BD is persisted in flash memory and SID is stored within the memory management subsystem in DRAM. On a system failure like power outage, SID can be rebuilt from BD on rebooting. At the software level, host applications may run directly in the system or or containers like Docker or LxC. The application layer performs read and write I/Os, which are passed to underlying file system and flash memory through system calls. For every I/O write or update, a lookup operation is performed to SID by calling hash_get to get streamID for the corresponding block addresses. The obtained streamID is piggybacked on a reserved field of regular and queued write commands as specified in the ATA command set. Along with every write operation, the corresponding streamID is penetrated through all the below layers until that I/O is written or updated on the SSD.

B. Basic Data Structures

Base Dictionary: The Base Dictionary (BD) is the result of the training phase (see Figure 2) of our FIOS, which contains the mapping between streamIDs and sector_chunks. Recall that there are 64 sectors in one sector_chunk. Each sector_chunk has a streamID, as metadata associated with it, which is stored in BD on flash memory. Such a metadata uses 0.5 bytes to keep the streamID. In our implementation, we consider a logical volume of flash of a 3TB, which is stripped over the physical volume of three 1TB SSDs. Thus, for 3TB flash volume, we require less than 50MB in total to store all streamID metadata, which is only 0.001% of total flash disk space.

Simplified Index Dictionary: The Simplified Index Dictionary (SID) is used to perform a lookup for streamIDs at run time (i.e., the testing phase). SID is the compressed version of BD, where we reduce the size of base dictionary by combining all consecutive sector_chunks that have the same streamID and replacing multiple lines with one line. Following that, instead of sector_chunk number, we have the range of sector chunks for each row in SID. We can then store SID in DRAM memory for fast hash_get lookup. We observe that SID is 50% more efficient in terms of space when compared to BD. Once SID is created and stored, we can move BD to the back-end storage until we have a new dictionary from another round of the training phase. In our experiments, we have 128GB DRAM in the server. The SID footprint for different workloads consumes less than 25MB of main memory. Therefore, the SID footprint overhead is only about 0.02% of the size of main memory.

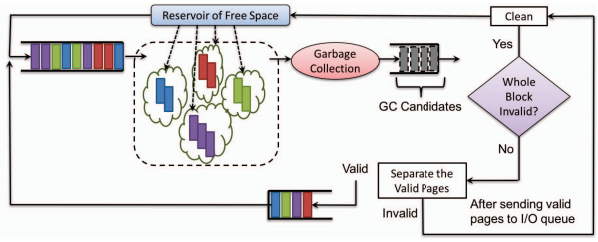


Fig. 6. Modification to garbage collection in order to enable multi-stream.

C. Multi-stream SSD Architecture

In an SSD, a single flash internal consists of multiple connected dies through a serial I/O bus and common control signals. Each die has its own chip enable and ready/busy signals. Thus, one of the dies can accept commands and data while the others are carrying out other operations. Furthermore, a die consists of multiple planes, and each plane is framed by multiple blocks containing pages where data is actually stored. Additionally, each plane has some register space to store data allocation and each block has one page reserved for metadata. We implement our FIOS in DiskSim's SSDSim plugin [4], which has been widely used to simulate SSDs. However, DiskSim does not support the multi-streaming technology. Thus, we modified this existing SSD simulator to allow us simulate multi-stream SSDs and evaluate our new FIOS method. Later, we discuss our implementation overhead in Section IV-E.

In particular, we mainly modify two modules, i.e., ssd_init and ssd_clean. The ssd_init module is modified to maintain a streamID attached to the hierarchy of active_page, active_block and active_plane. We ensure that for all dies, the total number of active_planes is always equal to the number of streams. Each active_plane has only one active_block and each active_block has only one active_page. Thus, the main difference between a legacy SSD and a multi-stream SSD is that we have multiple simultaneous open erase blocks (one for each stream) in the multi-stream SSD, but only a single open erase block in the legacy SSD.

If all pages in active_block are filled, then block_alloc_pos takes care of assigning a new block, which then becomes active_block with the same streamID as the previous active_block. Blocks are not pre-allocated to each stream. Instead, they are allocated on-demand to each stream after the previous one of the same stream is finalized. In Disksim, the ssd_clean module performs garbage collection (GC), which is a global component and not aware of the streams. So, when garbage collection starts, ssd_clean should search all the blocks to find one or multiple candidates to clean. But, there are still blocks with some valid pages whose streamIDs need to be preserved. The modified ssd_clean module of Disksim thus assigns these pages to a new physical plane that is ensured to belong to the preserved streamID. Figure 6 illustrates how the modified DiskSim works for supporting multi-streaming when garbage collection happens.

IV. Evaluation and Feature Selection

In this section, we present our experimental results to demonstrate the effectiveness of multi-stream using our FIOS.

We first study the impact of streamID identification using different features, such as frequency, sequentiality, on the WAF of multi-stream SSDs. Then, we analyze characteristics of different workloads and derive some implications for determining which feature or a combination of features can obtain the minimum WAF for a given workload. We leverage these implications to build an analytical model that co-relates workload characteristics with workload features. Finally, we validate the FIOS co-relation model to determine the best features for streamID identification.

A. Experimental Setup

Our evaluation environment of Disksim is calibrated based on the real testbed specs, summarized in Table I. We adopt the similar flash volume structure developed in our previous work [6], which consists of a logical volume of 3TB with full stripping of 128KB. We configure the parameter file of DiskSim [4] to support the On-Stack Replacement (OSR) write policy [7] and the wear-aware garbage collection cleaning policy [8]. We modify the SSDSim module of DiskSim to enable simulating operations of multi-stream SSDs.

TABLE I. TESTBED SPECS.

CPU Type	Intel(R) Xeon(R) CPU E5-2640 v3
CPU Speed	2.60 GHz
Number of CPU Cores	32 Hyper-Threaded
CPU Cache Size	20480 KB
CPU Memory	128 GB
OS Type	Linux
Kernel Version	4.2.0-37-Generic
Operating System	Ubuntu 16.04 LTS
Flash Storage	960 GB
Page Size	8 KB
Pages Per Block	64
Blocks Per Plane	351,562
Planes Per Die	8
Dies Per Element	2
Elements Per Gang	1
Flash Over-Provisioning	11%
Metadata Storage Reservation	1 Page Per Block

We conduct a trace-replay simulation in modified DiskSim by using 100+ enterprise workloads from University of Massachusetts (UMass) [9], Microsoft Research - Cambridge (MSR) [10] and Florida International University (FIU) [11] trace repositories. We analyze the performance using a wide variety of workloads with different characteristics. Table II shows some workload characteristics of the selected workloads. Brief descriptions about workload characteristics shown in Table II are as follows, ACF - the auto-correlation factor of inter-arrival time between two write/update requests; RW - the percentage of random writes among the total write I/Os; R - the reuse ratio of updates to new writes; S - the estimated average size of each write; WV - the working volume that represents the spatial capacity demand; λ - the daily logical write rate (GB/day); *Peak rate* - the peak throughput demand with the 5 min statistical analyses window; *Throughput* - the overall operation rate.

The primary goal of multi-streaming technology is to provide better data placement which can reduce the extra writes during garbage collection. Thus, the major evaluation metric is *Write Amplification Factor* (WAF), which is the ratio of physical writes (in bytes) on device to logical writes (in bytes) by applications running on host. Lower the WAF, better the lifetime and performance of the flash device.

B. Impacts of Workload Features

We inspect the impact of streamID identification using different features on the WAF. The baseline of our comparison is the WAF of no streaming legacy SSDs with same capacity and configurations. We also compare the results of our feature based streamID identification with one straightforward way of partitioning data into different streams. In this straight forward way, the total address space of SSDs is equally partitioned into the number of supported streams and streamID is assigned according to I/O address. This equal partition (EP) approach in our work is also the prevailing technique of streamID assignment in current multi-stream SSDs.

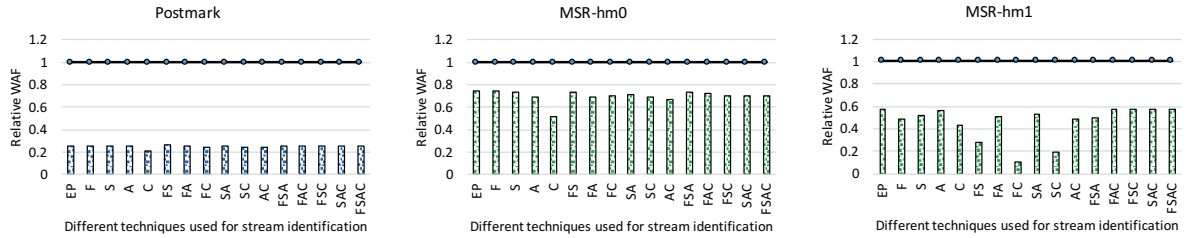
As discussed in Sec. II, our FIOS framework extracts various workload features and considers different feature combinations to cluster logical blocks into streams. Figure 7 shows the results of relative WAFs normalized by the WAF of no streaming legacy SSD. The blue horizontal line at 1 represents the relative WAF of legacy. Thus, the smaller the relative WAF is, the better the endurance of multi-stream SSDs can be obtained. Specifically, the first bar in each plot shows the WAF under EP, which performs streamID assignment by equally partitioning the available disk space. The remaining 15 bars represent some possible feature combinations used by FIOS. For example, *FSAC* stands for a combination of *Frequency*, *Sequentiality*, *Adjacent access* and *Coherency*.

We conduct our experiments with totally 100+ workloads and summarize our observations with 3 representative workloads in Figure 7. In overall, we observe that the multi-streaming technology offers a good opportunity to reduce WAF, e.g., at least 20% reduction in WAF for all workloads. For some write-intensive workloads e.g., Postmark (see Figure 7a) that have the majority of random writes, the WAF can be reduced by more than 70%. On the other hand, we also observe that none of these features can be always the best for different types of applications. For example, *Coherency* gives the best WAF reduction for MSR-hm0 while a combination of *Frequency* and *Coherency* (i.e., *FC*) is much more better for MSR-hm1. This is because different workloads have different characteristics. For example, by looking closely to the workloads we find that both MSR-hm0 and MSR-hm1 have more random writes and a high auto-correlation (i.e., ACF) of update time, see Table II. As a result, the feature of coherency that groups randomly occurred friendly sector_chunks into the same stream becomes a good choice. Additionally, MSR-hm1 has a high reuse ratio of updates. Thus, combining feature of frequency can capture the multi-touch reuse count. Moreover, from Figure 7c, we also find that further adding other features, such as sequentiality i.e., FSC does not work. This implies that the combination of *more features does not guarantee better reduction in WAF*.

In summary, our results show that (1) none of the features (such as frequency, adjacent access, sequentiality and coherency) can be claimed as the best for all I/O workloads, (2) different features have varying impacts on WAF, and (3) the benefit derived by using the combination of multiple features is not additive. Therefore, a big challenging issue in developing this multi-stream framework is *how to determine a combination of workload features that is best for assigning appropriate streamIDs under a given I/O workload*.

TABLE II. STATISTICS FOR POSTMARK, IOZONE AND SELECTED FIU, MSR-CAMBRIDGE AND UMASS WORKLOADS. (ACF - AUTO-CORRELATION OF INTER-ARRIVAL TIME, RW - RANDOM WRITE, R - REUSE RATIO, S - ESTIMATED SIZE OF EACH WRITE, WV - WORKING VOLUME SIZE, λ - WRITE RATE, BFC - BEST FEATURE COMBINATION)

Workload	ACF	RW (%)	R	S (KB)	WV (KB)	λ (GB/Day)	Peak rate (IOPS)	Throughput (IOPS)
Postmark	0.99	98.29	37.61	26	10244024	43.87	112.32	18.45
IOzone	0.98	49.45	1.21	197	5094328	28.08	109.53	14.08
FIU-1	0.78	75	157.35	9	1065000000	139.4	271.65	6.6
FIU-2	0.87	82	232.89	8	1084000000	114.72	156.79	1.02
FIU-3	0.97	57	182.04	8	1084000000	185.09	207.02	2
MSR-hm0	0.92	66	4.48	15	28000000	58.51	254.55	9.24
MSR-hm1	0.95	58	157.37	31	51000000	1.57	156.13	6.98
MSR-mds0	0.98	69	32.47	19	67000000	21.04	298.33	23.38
MSR-prn0	0.98	61	4.72	25	132000000	131.33	409.66	17.72
MSR-proj0	1	27	6.27	29	32000000	412.19	484.82	28.95
UMASS-1	0.21	64	1242.93	8	1289000000	575.94	218.59	122.05
UMASS-2	0.02	76	1.13	5	1156000	76.6	159.94	90.25



(a) (b) (c)

Fig. 7. Relative write amplification factor (WAF) w.r.t. legacy device for Postmark, IOzone and selected FIU, MSR-Cambridge and UMASS workloads by using different streamID identification techniques such as Equal Partition (EP), and our framework with dictionary framed from different features and their combinations. (F - Frequency, S - Sequentiality, A - Adjacent access

C. Our Approach - Automatic Feature Selection

To address the above issue, we build an analytical *correlation model* to capture the co-relation between easily obtained workload characteristics, such as I/O size (S), random write ratio (RW), reuse ratio (R), and autocorrelation (ACF) of write rates) with high-level workload features (such as frequency and coherency) and identify a good set (or combination) of features suitable to a workload for streamID identification. Our goal is to obtain such a good feature combination automatically, rather than experimenting all possible combinations.

The initial step in our approach is to determine the workload characteristics that can mainly affect the selection of features. Table III summarizes the implications that we develop regarding the relationship between workload characteristics and features used for streamID identification. For example, as shown in the first row of the Table III, if a workload has high ACF and high λ, then adjacent access (A) is one of the good features to select.

TABLE III. SUMMARY OF THE RELATION BETWEEN FEATURES AND WORKLOAD CHARACTERISTICS. (ACF - AUTO-CORRELATION OF INTER-ARRIVAL TIME, RW -RANDOM WRITE RATIO, R - REUSE RATIO, S - ESTIMATED SIZE OF EACH WRITES IN KB, λ - WRITE RATE IN GB/DAY)

Feature	ACF	RW	R	S	λ
Adjacent Access (A)	High	-	-	-	High
Coherency (C)	High	High	-	-	-
Sequentiality (S)	-	-	-	High	-
Frequency (F)	-	-	High	-	-

Based on the information in Table III, we construct a base matrix $\mathbb{B}$ as shown in Figure 8, where "1" corresponds to the "High" impact in Table III and "0" corresponds to the "Low" impact. In $\mathbb{B}$, some features (e.g., sequentiality (S) and frequency (F)) need to have 2 rows each. For example, no matter ACF of inter-arrival time is high or low, the feature of frequency is a good choice as long as the reuse ratio (R) is high. If we can map a workload's characteristics to a row in the base matrix, then we can choose the corresponding feature. However, we notice that the characteristics of a workload may not exactly map to one of the 6 possible rows of base matrix $\mathbb{B}$, and thus it is not straightforward to determine the best combination of features. Given this, we have the following problem objective and solution.

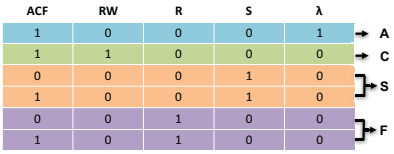


Fig. 8. Bit mapping log of base matrix $\mathbb{B}$.

Objective: Let us assume a map function f_{map} which determines the best possible combination of *features* ($\overrightarrow{\psi} \rightarrow F, T, S, C$) based on the workload characteristics ($\overrightarrow{\theta} \rightarrow ACF, RW, R, S, \lambda$), i.e.,

$$\overrightarrow{\psi} = f_{map}(\overrightarrow{\theta}). \quad (1)$$

Thus, our objective is to determine the above defined function f_{map}.

Solution: The attribute vector $\overrightarrow{\theta}$ is constructed based on users input of workload characteristics, where an attribute is assigned 1 if the given workload exhibits such a characteristic.

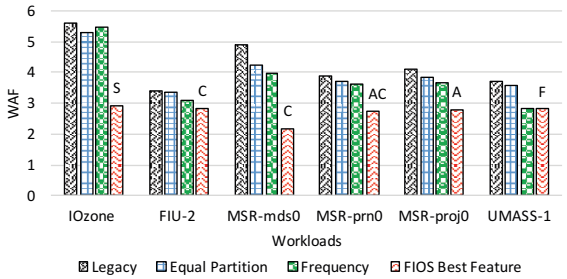


Fig. 9. Write amplification factor using the best feature selected using FIOS when compared to legacy no streaming, streamID identification with equal partition and with most prevently used feature of frequency. The best feature selected by FIOS is mentioned on top of the bar for each workload.

For example, lets consider workload *MSR-prn0*. After analyzing its workload characteristics, we have its attribute vector $\overrightarrow{\theta}$ *as* $[1 \quad 1 \quad 0 \quad 0 \quad 1]$. It represents that *MSR-prn0* has high ACF, RW and λ, and low R and S.

If the attribute vector $\overrightarrow{\theta}$ given by the user has "0" for a particular attribute, then any feature that has high impact of this attribute is definitely not a good candidate. For example, for the above considered $\overrightarrow{\theta}$, attributes such as reuse ratio (R) and I/O size (S) are "0". Thus, the features of sequentiality and frequency which has high impact of I/O size and reuse ratio (as seen from Figure 8) are not a good candidates. We construct the factorization vector $\overrightarrow{\alpha}$ that indicates the above information of whether or not each row in base matrix $\mathbb{B}$ is a good candidate for given $\overrightarrow{\theta}$. Such a factorization vector $\overrightarrow{\alpha}$ can be represented as below,

$$\overrightarrow{\alpha} = [\alpha_1,\ \alpha_2,\ \alpha_3,\ \alpha_4,\ \alpha_5,\ \alpha_6]^T, \tag{2}$$

where $\alpha_i \in \{0, 1\}$. If α_i is 0, then the i^{th} row of base matrix $\mathbb{B}$ is not a good candidate and vise-verse. Moreover, as at-least one of the features must be selected, we have $\sum_{i=1}^{6} \alpha_i > 0$. Thus, we obtain α by following elimination on base ($\mathbb{B}$) with respect to careful examination of input attribute vector $\overrightarrow{\theta}$. This elimination is represented by a "factorization" function f_{fact} as,

$$\overrightarrow{\alpha} = f_{fact}(\overrightarrow{\theta}). \tag{3}$$

Finally, the function f_{map} can be expressed by some other function f^{-1}_{deriv} such that $f_{map}(\overrightarrow{\theta}) = f^{-1}_{deriv}(\overrightarrow{\theta}) \quad \forall \overrightarrow{\theta}$ and $f^{-1}_{deriv}(\overrightarrow{\theta})$ gives all possible combinations of the rows of base $\mathbb{B}$ for which elements in $\overrightarrow{\alpha}$ is equal to 1, i.e., $f^{-1}_{deriv}(\overrightarrow{\theta}) = \bigvee_{i=0}^{6} (\overrightarrow{\alpha_i} \times \mathbb{B}_i)$. Then the resultant $\overrightarrow{\psi}$ gives the optimal feature combination. For our example of *MSR-prn0*, $\overrightarrow{\psi}$ = *AC*.

D. Validation of Feature Selection

Now, we turn to validate our approach for a given workload choosing the best combination of features under various workloads. We use WAF as the metric to evaluate our FIOS under six different I/O workloads, as shown in Figure 9. The best feature combination derived by FIOS are also mentioned on the top of the FIOS bars in the Figure 9. For comparison, we plot the results of legacy and multi-stream SSDs under the equal partition (EP) and the fixed feature (e.g., frequency) selection approaches. These results show that the existing methods that use the fixed features (such as frequency) or equally partitioning the available disk space (such as EP) for streamID assignment cannot effectively utilize the benefits of multi-stream SSDs. In contrast, for all these workloads FIOS is able to obtain the lowest WAF by using appropriate features for streamID assignment.

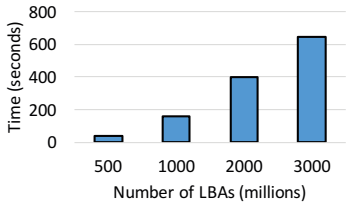


Fig. 10. Average time for streamID identification by FIOS using four features (frequency, adjacent access, sequentiality and coherency) as a function of the number of blocks in the workload.

E. Discussion

Implementation Layer: The identification of streamIDs can be done at any layers, such as the application layer, the file system layer, the block layer or the FTL layer. Stream identification at the application layer is to let applications be responsible to identify different streams (for e.g., commit logs, metadata, indices, etc.) in their workloads. However, when multiple applications are involved in a virtualized environment, stream management becomes more complex. Another option is to perform stream identification at the file system layer that appends each file with a corresponding streamID. This is portable to different applications without modifying them. But, this approach sometimes does not work when some applications bypass the file system to speed up data access. On the other hand, streamID identification at the FTL layer is limited by the computing and buffering capability of an SSD that is comparatively slower and smaller when compared to the computing speed and buffering capability of the host (i.e., CPU and main memory). We find that the block layer implementation can avoid the above issues, i.e., being portable to different applications and multi-tenant environments and being able to utilize the host resources for stream management. Thus, we choose to implement our stream identification framework at the block layer in this paper.

Implementation Overhead: We use base dictionary (BD) and simplified index dictionary (SID), as described in Section III, to maintain additional data of our framework. For a 3TB flash volume SSD storage, we require less than 50MB in total to store all streamID metadata, which is only 0.001% of the total flash disk space. In our experiments, we have 128GB DRAM in our server. The SID footprint for different workloads consumes less than 25MB of main memory. Therefore, the SID footprint overhead is only about 0.02% of the size of main memory. We mainly modified two modules, i.e., `ssd_init` and `ssd_clean` in DiskSim. In order to pertain the modification through out, we had to make subsequent changes in the `ssd_timing`, `ssd_gang` and `ssd_utils` modules. In total, we modified approximately 560 lines of codes to enable the operation of a general multi-stream SSD excludeing streamID identification.

The streamID identification time under FIOS is considered as the total additional operation time, including the time required for feature extraction, K-means clustering, construction of base dictionary (BD) and simplified index dictionary (SID), and queries of streamID to SID. Figure 10 shows the average time for streamID identification by FIOS using four features (i.e., frequency, adjacent access, sequentiality and coherency)

with respect to number of blocks in the workload. We can see that FIOS does add extra latency in order to identify the best feature combinations for improving SSD endurance. With an increase in dataset size (i.e., number of blocks), the time to identify and maintain streamIDs increases as well. In our experiment, a 3TB SSD was required to run a workload with 3 billions LBAs (blocks). Thus, we believe it is acceptable for FIOS to take around 600s for streamID identification. Finally, we note that there exists a trade-off between time efficiency and endurance improvement. Since a multi-stream SSD is developed to enhance the lifetime of SSDs, we believe that the main goal of a streamID identification algorithm should be to reduce write amplifications with the minimal time overhead.

V. Related Work

The evolution of flash has raised the research on SSDs as a hot topic in recent years. Because of high performance (compared to HDDs) and low cost (compared to RAM), SSDs were initially used as a buffer pool to cache data between RAM and hard disk [12], [13]. As the $/GB of flash drives kept decreasing, SSDs have been emerged as main storage [6], [14]–[16]. Nowadays, the use of SSDs in enterprise servers and data centers becomes more prominent [17]. One of the major issue of the flash technology is the limited lifetime of flash cells due to limited program and erase (PE) cycles. The work in [4] observed that SSD's performance and lifetime is highly sensitive to I/O workloads. The previous work in [18] designed new file systems, specifically for enhancing lifetime of the SSDs. These file systems concentrated on transforming all random writes at the file system level to sequential ones at the SSD level and considered a new data grouping strategy on writing by putting data blocks with similar update likelihood into the same segment. It was noticed that the reduction of the internal write amplification of SSDs may increase the lifetime of SSDs. The recent technology evolves a new product of multi-stream SSDs, which offers an intuitive storage interface to inform host system about the expected lifetime of the data [3]. The work in [19], [20] did modification in certain applications (i.e., FIO, Cassandra and RocksDB) to enable application assigned streamID. The study shows that with multi-streaming, SSDs can be more efficiently used for achieving consistently better performance and endurance. Recently, in [21] the authors proposes two automatic stream management algorithms that operate on temperature of data with respect to frequency and recency. To the best of our knowledge, there exists no framework or efforts to analyze the impact of stream identification for various I/O workloads using different data features (such as frequency, sequentiality) on the write amplification of multi-stream SSD.

VI. Conclusion

In this paper, we built a framework to investigate the impact of streamID assignment using different workload features on write amplification of multi-stream SSDs. FIOS is scalable to incorporate any number of features. Our experimental results show that compared to a legacy SSD, a multi-stream SSD is able to reduce WAF, and thus improve the device endurance and lifetime. We also found that different features have varying impacts on WAF of different workloads. Additionally, in order to better capture data lifetime, we proposed a new feature, called coherency, to capture the friendship between write operations with respect to their update time. We further investigated the correlation between workload attributes and workload features to automatically determine a combination of features that can offer the most WAF reduction. In the future, we plan to expand our stream identification technique with more workload features and develop a tool to analyze workload characteristics.

References

[1] J. S. Bhimani, J. Yang, C. Choi, and J. Huo, "Smart I/O stream detection based on multiple attributes," Mar. 16 2017, uS Patent App. 15/344,422.

[2] J.-U. Kang, J. Hyun, H. Maeng, and S. Cho, "The multi-streamed solid-state drive." in *HotStorage*, 2014.

[3] "Multi-Stream Technology," http://www.samsung.com/semiconductor/insights/article/25465/multistream.

[4] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. S. Manasse, and R. Panigrahy, "Design Tradeoffs for SSD Performance," in *USENIX Annual Technical Conference*, 2008, pp. 57–70.

[5] J. Bhimani, M. Leeser, and N. Mi, "Accelerating K-Means clustering with parallel implementations and GPU computing," in *HPEC'IEEE*.

[6] J. Bhimani, J. Yang, Z. Yang, N. Mi, Q. Xu, M. Awasthi, R. Pandurangan, and V. Balakrishnan, "Understanding performance of I/O intensive containerized applications for NVMe SSDs," in *Performance Computing and Communications Conference (IPCCC), 2016 IEEE 35th International*. IEEE, 2016, pp. 1–8.

[7] Y. Luo, Y. Cai, S. Ghose, J. Choi, and O. Mutlu, "WARM: Improving NAND flash memory lifetime with write-hotness aware retention management," in *2015 31st Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2015, pp. 1–14.

[8] A. V. Kuzmin and J. G. Wayda, "Multi-array operation support and related devices, systems and software," Jan. 5 2016, uS Patent 9,229,854.

[9] *UMass Trace Repository*, (accessed January 13, 2017). [Online]. Available: http://traces.cs.umass.edu

[10] D. Narayanan, A. Donnelly, and A. Rowstron, "Write Off-Loading: Practical Power Management for Enterprise Storage," *ACM Transactions on Storage*, vol. 4, no. 3, pp. 10:1–10:23, 2008.

[11] *SNIA Iotta Repository*, (accessed January 13, 2017). [Online]. Available: http://iotta.snia.org/historical_section

[12] L.-P. Chang, "Hybrid Solid-State Disks: combining heterogeneous NAND flash in large SSDs," in *Design Automation Conference, 2008. ASPDAC 2008. Asia and South Pacific*. IEEE, 2008, pp. 428–433.

[13] H. Jo, Y. Kwon, H. Kim, E. Seo, J. Lee, and S. Maeng, "SSD-HDD-hybrid virtual disk in consolidated environments," in *European Conference on Parallel Processing*. Springer, 2009, pp. 375–384.

[14] 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*. European Design and Automation Association, 2009, pp. 393–398.

[15] R. Chin and G. Wu, "Non-volatile memory data storage system with reliability management," May 25 2009, uS Patent App. 12/471,430.

[16] Z. Yang, J. Tai, J. Bhimani, J. Wang, N. Mi, and B. Sheng, "GREM: Dynamic SSD Resource Allocation In Virtualized Storage Systems With Heterogeneous IO Workloads," in *35th IEEE International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2016.

[17] Y. Wang, K. Goda, M. Nakano, and M. Kitsuregawa, "Early experience and evaluation of file systems on SSD with database applications," in *Networking, Architecture and Storage (NAS), 2010 IEEE Fifth International Conference on*. IEEE, 2010, pp. 467–476.

[18] C. Min, K. Kim, H. Cho, S.-W. Lee, and Y. I. Eom, "SFS: random write considered harmful in solid state drives," in *FAST*, 2012, p. 12.

[19] C. Choi, "Multi-Stream Write SSD: Increasing SSD Performance and Lifetime with Multi-Stream Write Technology," Flash Memory Summit 2016 Santa Clara, CA, http://www.flashmemorysummit.com/English/Collaterals/Proceedings/2016/20160809_FC12_Choi.pdf.

[20] F. Yang, K. Dou, S. Chen, J.-U. Kang, and S. Cho, "Multi-streaming RocksDB," in *Non-Volatile Memories Workshop*, 2015.

[21] J. Yang, R. Pandurangan, C. Choi, and V. Balakrishnan, "AutoStream: automatic stream management for multi-streamed SSDs," in *10th ACM SYSTOR*. ACM, 2017, p. 3.