Exploration and Exploitation for Buffer-Controlled HDD-Writes for SSD-HDD Hybrid Storage Server

SHUCHENG WANG and ZIYI LU, Wuhan National Laboratory for Optoelectronics, HUST
QIANG CAO, Key Laboratory of Information Storage System of Ministry of Education, HUST
HONG JIANG, University of Texas at Arlington
JIE YAO, School of Computer Science and Technology, HUST
YUANYUAN DONG and PUYUAN YANG, Alibaba Group
CHANGSHENG XIE, Key Laboratory of Information Storage System of Ministry of Education, HUST

Hybrid storage servers combining solid-state drives (SSDs) and hard-drive disks (HDDs) provide cost-effectiveness and μs-level responsiveness for applications. However, observations from cloud storage system Pangu manifest that HDDs are often underutilized while SSDs are overused, especially under intensive writes. It leads to fast wear-out and high tail latency to SSDs. On the other hand, our experimental study reveals that a series of sequential and continuous writes to HDDs exhibit a periodic, staircase-shaped pattern of write latency, i.e., low (e.g., 35 μs), middle (e.g., 55 μs), and high latency (e.g., 12 ms), resulting from buffered writes within HDD's controller. It inspires us to explore and exploit the potential μs-level IO delay of HDDs to absorb excessive SSD writes without performance degradation.

We first build an HDD writing model for describing the staircase behavior and design a profiling process to initialize and dynamically recalibrate the model parameters. Then, we propose a Buffer-Controlled Write approach (BCW) to proactively control buffered writes so that low- and mid-latency periods are scheduled with application data and high-latency periods are filled with padded data. Leveraging BCW, we design a mixed IO scheduler (MIOS) to adaptively steer incoming data to SSDs and HDDs. A multi-HDD scheduling is further designed to minimize HDD-write latency. We perform extensive evaluations under production workloads and benchmarks. The results show that MIOS removes up to 93% amount of data written to SSDs, reduces average and 99^{th}-percentile latencies of the hybrid server by 65% and 85%, respectively.

CCS Concepts: • **Software and its engineering → Secondary storage**; • **Information systems → Magnetic disks**; **Cloud based storage**; **Hierarchical storage management;**

Additional Key Words and Phrases: Hybrid storage, IO scheduling, tail latency

This work is supported in part by National Key Research and Development Program of China (No. 2018YFA0701800), NSFC (No. 61821003 and 61872156), the US NSF under grant numbers CCF-1704504 and CCF-1629625, and Alibaba Group through Alibaba Innovative Research (AIR) Program.

Authors' addresses: S. Wang and Z. Lu, Wuhan National Laboratory for Optoelectronics, HUST, 1037 Luoyu Road, Wuhan, Hubei, 430074, China; emails: wsczq@hust.edu.cn, luziyi@hust.edu.cn; Q. Cao (corresponding author) and C. Xie, Key Laboratory of Information Storage System of Ministry of Education, HUST, 1037 Luoyu Road, Wuhan, Hubei, 430074, China; emails: caoqiang@hust.edu.cn, cs_xie@hust.edu.cn; H. Jiang, Department of Computer Science and Engineering, University of Texas at Arlington, 701 S. Nedderman Drive, Arlington, TX 76019; email: hong.jiang@uta.edu; J. Yao, School of Computer Science and Technology, HUST, 1037 Luoyu Road, Wuhan, Hubei, 430074, China; email: jackyao@hust.edu.cn; Y. Dong and P. Yang, Alibaba Group, Hangzhou, Zhejiang, 311121, China; emails: yuanyuan.dyy@alibaba-inc.com, puyuan.ypy@alibaba-inc.com.

1553-3077/2022/01-ART6 $15.00
https://doi.org/10.1145/3465410

ACM Reference format:
Shucheng Wang, Ziyi Lu, Qiang Cao, Hong Jiang, Jie Yao, Yuanyuan Dong, Puyuan Yang, and Changsheng Xie. 2022. Exploration and Exploitation for Buffer-Controlled HDD-Writes for SSD-HDD Hybrid Storage Server. *ACM Trans. Storage* 18, 1, Article 6 (January 2022), 29 pages.
https://doi.org/10.1145/3465410

1 INTRODUCTION

Storage clouds have prevalently deployed hybrid storage servers integrating **solid-state drives (SSDs)** and **hard-drive disks (HDDs)** in their underlying uniform storage infrastructure, such as Microsoft Azure [9], Amazon [45], Facebook [42], Google [23], and Alibaba Pangu [10]. Such hybrid storage servers employ an SSD-HDD tiered architecture to reap the benefits of both SSDs and HDDs for their superior IO performance and large capacity, respectively, thus achieving high cost-effectiveness. Incoming writes are quickly persisted in the SSD tier and acknowledged, and then flushed to the HDD tier at a later time.

Our observations from real-world production workloads of hybrid storage servers in Alibaba Pangu indicate that, SSDs are generally over-used while HDDs are less than 10% utilized on average, missing the opportunity to exploit HDDs' performance and capacity potentials. Writes are known to be unfriendly to SSDs for two reasons. First, SSDs have limited **Program/Erase (P/E)** cycles [7, 43] that are directly related to the amount of writes. Second, SSDs suffer from unpredictable, severe performance degradation resulting from **garbage collections (GCs)** [30, 62]. To guarantee stable write performance of storage servers in write-heavy workloads, cloud providers have to deploy more and/or larger SSDs, significantly increasing their total investment capital.

Our extensive experimental study on HDD write behaviors, conducted on various HDD products and with results shown in Figure 1, suggests that a series of continuous and sequential small HDD writes (e.g., 4 KB) exhibit low latency (e.g., 35 μs) for about 60 ms, and then a sharply elevated high latency (e.g., 12 ms), which is followed by middle latency (e.g., 55 μs) for about 40 ms. The three states of write behaviors, or *write states* in short, are referred to in this article as *fast, mid*, and *slow* writes, respectively. The former two types of writes can provide μs-level responsiveness, because incoming writes are considered complete and acknowledged (to the host) once their data have been written into the built-in buffer in the controller. However, when the write buffer is full, host writes have to be blocked until the buffered data are flushed into the disk, causing slow writes. This finding inspires us to fully exploit performance potentials offered by buffered writes of HDD, improving the performance while mitigating write-penalty on SSDs. Our goal is to enable hybrid storage servers to achieve higher performance and reliability without introducing extra hardware.

However, the key challenge for adopting buffered writes in HDDs to take advantage of the fast and mid writes is the difficulty in predicting precisely when these write states would occur. The internal buffer and other components of HDDs are completely hidden from the host. Host can only identify the current write state according to its own delay, but not future write states. To address this issue, we build a prediction model for sequential and continuous write patterns that predicts the next HDD write state. The insight is that the write states of continuous and sequential HDD write requests are periodical. The prediction of next write state can be achieved with the information of buffered-write period and current write state. To build the HDD prediction model, we conduct a profiling process that should be executed when adding or replacing an HDD device and takes low overhead that just consumes a few seconds. It is also conditionally or periodically triggered to recalibrate the parameters of the prediction model in runtime with respect to the disk aging and disk characteristics. We also assess the prediction accuracy of the write-state predictor and results shows that the predictor correctly identifies at least 99.8% of the slow write state.

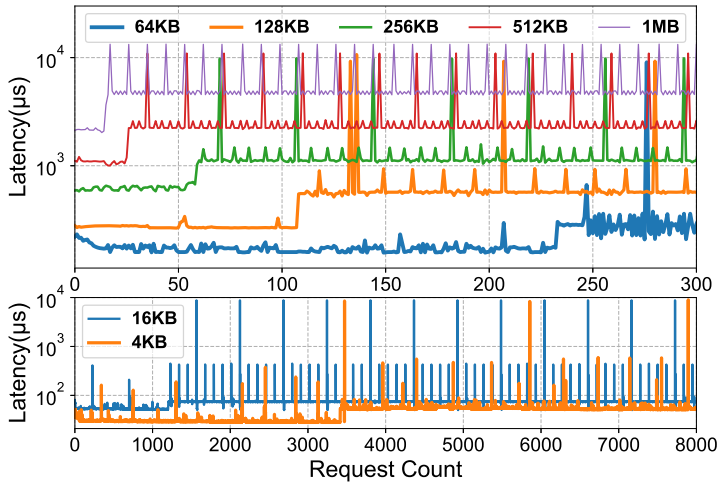


Fig. 1. Sequential writing in a 10 TB western digital HDD.

Then, we propose a **Buffer-Controlled Write (BCW)** approach. BCW can proactively and effectively control the buffer write behavior according to the predictor and runtime IO monitoring. Besides, BCW also actively "skip" slow writes by filling padded data during HDD slow writes. We further propose a **mixed IO scheduler (MIOS)** for SSD-HDD hybrid storage by leveraging the BCW approach. MIOS adaptively redirects incoming writes to SSDs or HDDs depending on write states, runtime queue length, and disk status. Under high IO intensity, MIOS can be triggered to reduce IO pressure, the amount of data written, and write-penalty on SSDs while improving both average and tail latency. In addition, MIOS supports multi-HDD scheduling, which selectively redirects requests to HDD with the lowest predicted latency. Particularly, the multi-HDD scheduling mechanism could ensure the *fast* write state is always available by actively syncing the HDD built-in buffer in a round-robin way.

The main contributions of this article are as follows:

— Through extensive experimental studies on HDD write behaviors, we discover that there exists a periodic staircase-shaped write latency pattern consisting of μs-level write latency (fast and mid write states) followed by ms-level write latency (slow write state) upon continuous and sequential writes, because of the buffered write feature in HDDs. To facilitate the full exploitation of this write latency pattern, we build a predictor to pre-determine what the next write state is.
— We propose a BCW approach, which proactively activates continuous and sequential write patterns as well as effectively controls the IO behavior, according to the predictor and runtime IO monitoring. BCW also employs data padding to actively avoid or skips slow writes for the host. We further analyze the prediction accuracy and running stability of BCW.
— We design an SSD-HDD MIOS leveraging BCW to improve the overall performance of SSD-HDD hybrid storage servers, while substantially reducing write traffic to SSDs. We further present a multi-HDD scheduling of MIOS to fully exploit potential of multiple HDDs.
— We prototype and evaluate MIOS under a variety of production workloads. We also compare them with different HDD write mechanisms and the state-of-art IO scheduling approaches. The results demonstrate that MIOS reduces average and tail latency significantly due to dramatic decrease in the amount of data written to SSDs.

The rest of the article is organized as follows. Section 2 provides the necessary background for the proposed BCW approach. Section 3 analyzes the behaviors of HDD buffered writes. Section 4 describes design and implementation of BCW and MIOS. We evaluate the effectiveness of MIOS in Section 5. Finally, Section 6 describes related works and Section 7 concludes the article.

2 BACKGROUND AND MOTIVATION

2.1 Primary Storage

Nowadays, primary storage involves popular SSD and traditional HDD. SSDs have become a mainstream storage media due to its superior performance and lower power consumption than HDDs [2, 59]. However, the limited write endurance has become a critical design issue in SSD-based storage systems [41]. Furthermore, SSDs suffer from performance-degrading GCs, which recycle the invalid pages by moving valid parts to new blocks and then erasing old blocks [32, 44]. GCs with ms-level delays can block incoming user requests, thus leading to long tail latency [20]. On the other hand, both large IO blocks and high IO intensity can lead to sudden increment in SSD queue, also resulting in high tail latency [30]. Therefore, recent studies [60] indicate that SSDs do not always exhibit their ideal performance in practice.

HDDs have large capacity at low cost without the wear-out problem. However, HDDs have relatively low performance compared to SSDs. A random HDD IO has 2~3 orders of magnitude higher latency than an SSD IO. This is primarily because of the *ms*-level mechanical seeking of disk head.

2.2 SSD-HDD Hybrid Storage

To accommodate exponentially increasing storage requirement while achieving overall cost-effectiveness, SSD-HDD hybrid storage has emerged to be an inevitable choice for cloud providers [47, 56]. Most of them, such as Google [23], Amazon [45], Facebook [42], and Microsoft's online services [9], expect larger storage capacity and better performance but at lower cost. To meet this demand, they increasingly embrace storage heterogeneity by deploying variable types and numbers of SSDs and HDDs. The former offers lower IO latency [16] as the primary tier, the latter provides larger capacity at low cost as the secondary tier. The better performing SSD tier generally plays the role of a write buffer to quickly persist incoming write data, which are eventually flushed to the slower but larger HDD tier. As a result, the SSD tier absorbs most of the write traffic from foreground applications.

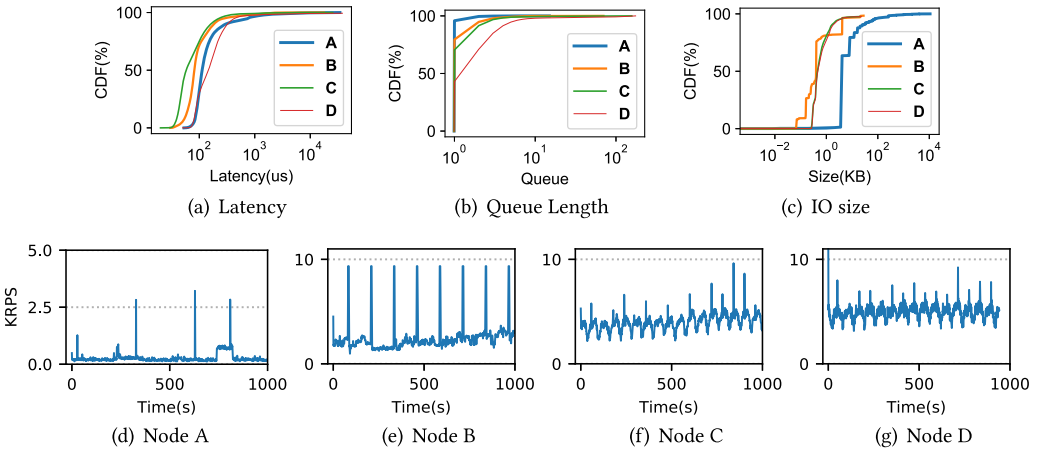
2.3 Write Behavior of Hybrid Storage

Write-intensive workloads widely exist in many production environments, such as enterprise applications, supercomputing, and clouds. Enterprise servers are expected to rapidly persist production data in time, such as business databases. Burst buffer [4, 34] in supercomputing systems also deploys high-performance SSDs to temporarily store instantaneous highly-intensive write data. More commonly, many backend storage servers in cloud must accommodate write-dominated workloads, as observed in Alibaba Pangu [38]. Pangu [10] is a distributed large-scale storage platform and provides cost-effective and unified storage services for Alibaba Clouds [26, 37] and Ant Financial. As such, Pangu needs to minimize the total cost of ownership while meeting strict QoS requirements like tail latency [6, 18].

Table 1 gives the workload characteristics of production trace data from Pangu. We observe that some storage nodes (servers) in Pangu rarely serve reads from the frontend and instead must handle amounts of highly-intensive writes. For Alibaba Cloud, the upper-level latency-critical online services generally build their own application-aware caches to ensure service responsiveness and

Table 1. The Workload Characteristics of Pangu Traces Recorded from One SSD and One HDD in Four Different Nodes, A, B, C, and D, That Support Online Services

Node Type	Duration (min)	Writes (GB)	Reads (GB)	Avg. Req. Size (KB)	Peak KRPS	Avg. KRPS	Avg. HDD IO Uti.(%)	Avg. SSD IO Uti.(%)
A	45	18.5	1.4	56.0	3.4	0.23	7.6	11.9
B	30	74.4	2	17.7	9.3	2.5	9.8	28.5
C	30	10.7	2.1	4.2	9.6	2.7	4.1	24.6
D	26	10.1	1.7	4.1	11.1	3	4.8	25



(a) Latency (b) Queue Length (c) IO size

(d) Node A (e) Node B (f) Node C (g) Node D

Fig. 2. Behaviors of production workloads on four representative hybrid storage nodes in Pangu in terms of latency, queue length, request size and IO intensity.

reserve local fast-storage to cache hot data for user reads. Therefore, the backend storage nodes are required to persist new and updated data from frontend nodes as soon as possible. To better understand this write-dominated workload behavior, we analyze four typical workloads on Pangu storage nodes A (Cloud Computing), B (Cloud Storage), and C and D (Structured Storage). We count these workloads from one SSD and one HDD in each node because the workload behavior of all storage devices is basically the same in one node. Observations are drawn as follows. A comprehensive workload analysis of Pangu can be found in the previous study [38].

— Most requests are writes. Table 1 shows that more than 77% and up to 95% of requests are writes in these nodes, and the amount of data written is 1–2 orders of magnitude larger than that of data read from them. Actually, nearly 3 TB data are written to every SSD each day, which is close to drive writes per day that strictly limits the amount of SSD data written daily for reliability.
— The IO intensity distribution has bursty patterns. As shown in Figure 2(d) through Figure 2(g), SSDs experience bursty intensive write workloads (e.g., 11K request per second in workload D).
— The amount of data written to SSDs and HDDs differ dramatically. For instance, the average SSD IO utilization is up to 28.5% in workload B while it is less than 10% in HDD. Even so, most of the HDD utilization is used in dumping SSD data, rarely serving user requests.
— There exists long tail IO latency. As shown in Figure 2(a), SSDs with high IOPS suffer from heavy-tail IO latency (e.g., the 99.9^{th} percentile latency is 50 ms) due to the queue blocking

Table 2. The Device Information about Five Representative HDD Products

Manufacturer	Capacity	Model	Recording Technology	Sequential Write Bandwidth (MB/s)
West Digital	10 TB	WD100EFAX [15]	PMR	200
	8 TB	WD8004FRYZ [14]	PMR	180
	4 TB	WD40EZRZ [15]	PMR	180
Seagate	8 TB	ST8000DM0004 [53]	PMR	180
	4 TB	ST4000DM004 [54]	SMR	180

shown in Figure 2(b). This is caused in part by (1) large writes (e.g., 1 MB), and (2) frequent SSD GCs induced by high write intensity.

— Small size IOs account for a large proportion. As shown in Figure 2(c), more than 75% of write requests are smaller than 10 KB, and the average request size is nearly 4 KB in C and D.

2.4 Challenge

To relieve the SSD pressure from write-dominate workloads, a simple solution is to increase the number of SSDs in the hybrid nodes. However, this is a costly solution as it increases the total cost of ownership.

An alternative is to exploit the severely underutilized HDD IO capacity in hybrid storage nodes when SSDs are overused. The state-of-art solution SWR [38] redirects large SSD writes to idle HDDs. This approach can alleviate the SSD queue blocking issue to some extent. However, the IO delays experienced by write requests redirected to HDDs are shown to be 3–12 times higher than those written to SSDs. This is clearly undesirable, if not unacceptable, for most write requests that demand μs-level latency. The key challenge is how to reduce HDD IO delay to the μs-level that is close to SSDs, which is a seemingly impossible task at first glance. Fortunately, as we look closer into the write behaviors of HDDs, this is indeed possible, which we will elaborate in the next section.

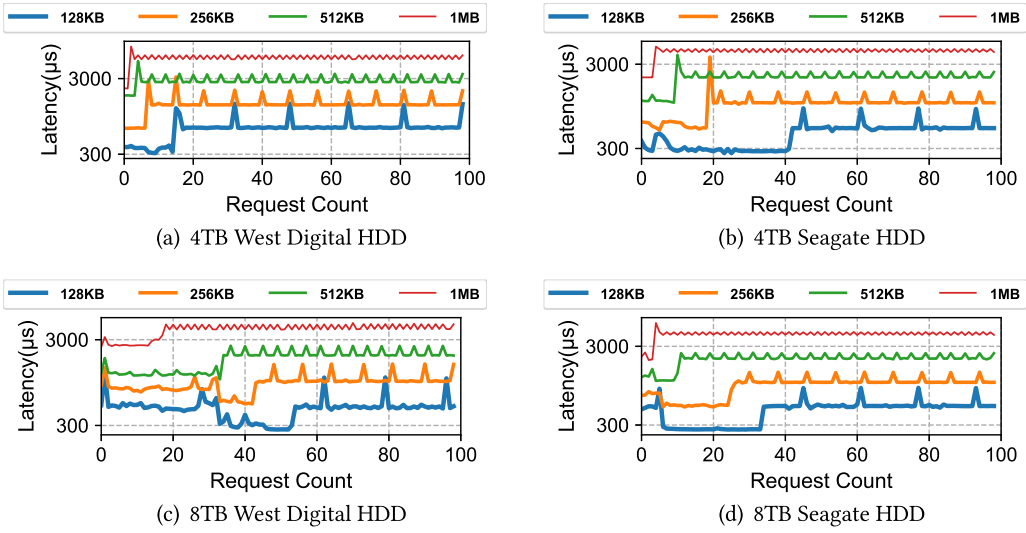
3 HDD WRITE BEHAVIORS

To have a comprehensive understanding of HDD write behaviors, so as to assess the possibility of achieving μs-level write latency on HDDs, we perform continuous and sequential writes, which is the most friendly write pattern for HDDs.

3.1 Buffered Writes

We conduct a "profiling" process to observe detailed HDD behaviors. The profiling of the HDD model executes a series of continuous and sequential writes with the same IO size to an HDD, which roughly consumes 5–10 seconds. The profiling process must be executed when adding or replacing an HDD device. Certainly, it can be conditionally or periodically triggered to recalibrate the parameters of BCW model. We select five representative HDD products: West Digital 10 TB, 8 TB, and 4 TB, and Seagate 8 TB and 4 TB. Table 2 lists their device parameters. The recording technology of 4 TB Seagate HDD is **Shingled Magnetic Recording (SMR)** and the other four HDDs are **Perpendicular Magnetic Recording (PMR)**. We draw three observations from the profiling results shown in Figures 1 and 3.

— For each tested HDD, the series of continuous sequential write requests experience a similar sequence of three-level write latency, i.e., low, mid, and high latencies, forming



(a) 4TB West Digital HDD (b) 4TB Seagate HDD

(c) 8TB West Digital HDD (d) 8TB Seagate HDD

Fig. 3. Sequential writing on four types of HDDs.

a staircase-shaped time series. For example, in the 10 TB HDD, the three levels of write latency of 16 KB writes are about 66 μs, 135 μs, and 12 ms, respectively.

— The observed HDD write behavior is periodic. At the beginning (right after the buffer becomes empty), low-latency writes last for a period (e.g., 60 ms in 10 TB), which is followed by a spike (high-latency writes) and then mid-latency writes. If the write pattern is continuous, high-latency writes and mid-latency writes will appear alternately.

— The number of low-latency continuous writes in a sequence relies on their I/O size. Smaller write size leads to a larger number of writes. For example, the number of 16 KB and 64 KB writes is about 1,200 and 240 on the 10 TB HDD, respectively.

The reasons behind these observed HDD write behaviors are as follows. Modern HDDs deploy a built-in **Dynamic Random Access Memory (DRAM)** (e.g., 256 MB for the 10 TB and 8 TB HDDs, and 64 MB for the two 4 TB HDDs). However, only a part of the DRAM (e.g., 16 MB for 10 TB WD and 8 TB Seagate HDD, 4 MB for 8 TB WD HDD and 4 TB Seagate HDD, 2 MB for 4 TB WD HDD) can be used to buffer incoming write IOs based on external observation. The remaining capacity of the HDD built-in DRAM can be used as read-ahead cache, ECC buffer [11], sector remapping buffer, or prefetching buffer [21, 52]. However, the exact buffering and flushing mechanisms of built-in DRAM in HDD are completely hidden from the host and heavily depend on the specific HDD models. Fortunately, we can measure the buffered write feature of an HDD externally and experimentally based on the aforementioned experiments.

Upon successful buffering of a write, HDD immediately informs the host of request completion. When the buffered data exceed a threshold, the HDD must force a flushing of the buffered data into their locations in the disk media. During this period, incoming writes must be blocked until the buffer is freed up again. It is worth noting that, after an idle period, the HDD buffer may become empty implicitly as a course of flushing data to the disk. However, to explicitly empty the buffer, we can actively invoke *sync*() to force flushing.

3.2 An HDD Buffered-Write Model

To formally characterize the HDD write behavior, we build an HDD buffered-write model. Figure 4 illustrates the schematic diagram of this model in the time dimension. The x-axis represents the

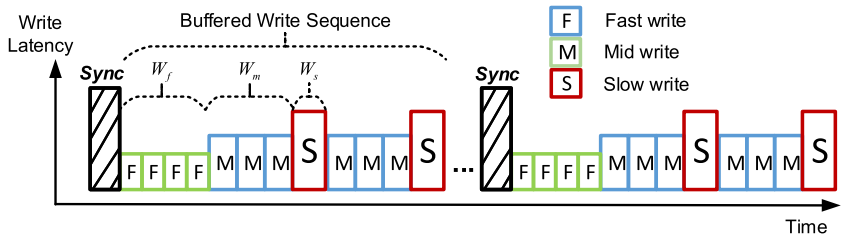


Fig. 4. The HDD buffered-write model with two complete buffered write sequences.

Table 3. The List of Descriptions about All the Parameters in the HDD Buffered-Write Model

Parameters	Description
$L_{f/m/s}$	The IO delays of write requests in the F/M/S write states
$W_{f/m/s}$	The cumulative amount of written data for the Fast/Mid/Slow Stages
$T_{f/m/s}$	The time duration of the Fast/Mid/Slow Stages
s_i	The IO size of write request i

time sequences of transitions among the three write levels, with each sequence being started by "Sync" event.

A Buffered-Write Sequence consists of three aforementioned types of HDD buffered writes, i.e., Fast (low-latency), Mid (mid-latency), and Slow (high-latency) writes, which denotes as F, M, and S, respectively. In the model, F, M, and S can be thought of as the states a write request can be in (i.e., experiencing the fast, mid or slow write process). As described in Table 3, these IO delays are denoted as L_f, L_m, and L_s, respectively. The F state means that an incoming write request w_i with the size of s_i can be buffered completely in the built-in DRAM buffer of HDD. The M state means that the write buffer is close to being full. The S state means that the write buffer is full and any incoming write request is blocked.

A Buffered-Write Sequence lasts a Fast stage, followed by one or more Slow-and-Mid stage-pairs. The sequence begins when there is sufficient buffer available for Fast stage (e.g., close to empty). It ends when current series of continuous writes ends. The Fast, Mid, and Slow stage last for T_f, T_m, and T_s respectively, which are determined by the cumulative amount of written data W_f, W_m, and W_s in the respective states. Actually, $W_f = T_f * s_i/L_f$ and it is applied to W_m.

We can profile the HDDs to identify such key parameters. For example, Figure 1 shows the profiling results of 10 TB WD HDD with varying write request sizes. The value of L_f is 180 μs, L_m is 280 μs and L_s is 12 ms. The value of T_f is 60 ms, T_m is 37 ms and T_s is 12 ms. W_f is 16 MB and W_m is 8 MB. W_s depends on the IO size s_i. According to the HDD buffered-write model, the Fast and Mid writes of HDD have 100-μs-level latency, which can approach the write latency of SSDs. This motivates us to design a controllable buffer write strategy for HDDs to reduce writes on SSDs in hybrid storage systems without sacrificing the overall performance.

4 DESIGN

To fully exploit HDD buffered writes, two critical challenges must be addressed. The first is how to determine which write state that a write request will be Fast (F), Mid (M), or Slow (S), in order to properly schedule the write request. The second is how to steer an incoming write request to HDD without performance degradation.

For the first problem, we build a *Write-state Predictor* to pre-determine the next write state based on current write state and buffer state. The ability to determine the subsequent write state of HDD

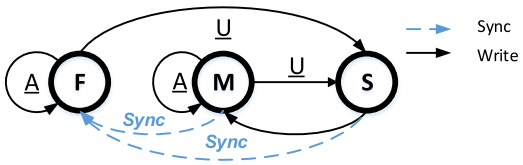


Fig. 5. The state predication diagram. Each write request can only be one of the three write states, F, M, and S. Letter "A" means that the current data written in the F and M states are less than the W_f and W_m values, respectively. Otherwise, the write buffer is "U." The *Sync* operation takes the next write state back to F.

is critical to scheduling incoming write requests. Based on that, we propose BCW, a writing approach to proactively activate continuous and sequential write patterns that the predictor relies on, as well as effectively control the IO behavior according to the predictor and runtime IO monitoring. To avoid performance degradation caused by S writes, we propose a proactive padding-write approach to "skip" the S state by executing slow writes with padded non-user data.

To overcome the second problem, we propose an SSD-HDD MIOS that adaptively controls queue lengths of SSDs and HDDs in hybrid storage nodes, and determines where to steer a user write request.

4.1 Write-State Predictor

The next write state could be predicted according to write buffer's free space and the write state of the current request. In the HDD buffered write model, each write request state should be one of F, M, and S state. The HDD buffer state is considered by buffered-write model to be in either A (available) or U (unavailable). The "A" state means current **Accumulative Data Written (ADW)** in the F and M states are less than W_f and W_m respectively. Otherwise, the write buffer is in the "U" state. Figure 5 shows how the next write state is determined based on the current buffer state and write state in a State Predication Diagram, which is described as follows:

- F/A: The current write state is F and the buffer is available. Next write state is most likely to be F.
- F/U: Although the current write state is F, the buffer is unavailable. Next write state is likely to change to S.
- M/A: The current write state is M and the buffer is available. Next write state is most likely to remain M.
- M/U: Although the current write state is M, the buffer is unavailable. Next write state should be S.
- S: The current write state is S. Next write state will be M with a high probability.
- The *Sync* operation will force the next write state and buffer state back to be F/A in all cases.

Based on that, we design a Write-state Predictor described in Algorithm 1. It identifies what the current write state is, F, M, or S by monitoring the IO request size and latency, and calculating the free space in the write buffer. That is, the ADW in the current write state (F or M) is recorded and compared with W_f or W_m for predicting the next write state.

4.2 Buffer-Controlled Writes

BCW is an HDD writing approach that ensures user writes using F or M write state and avoids allocating Slow writes. The key idea of BCW is to make buffered write controllable. Based on the Write-state predictor, we design BCW as described in Algorithm 2.

ALGORITHM 1: The Algorithm of Write-state Predictor

Input: Current write request size: $size$;
The last write state: $state$; Current accumulative amount of data written: ADW;
The amounts of data written in the F state and M state: W_F and W_M
Output: Write-state prediction for the next request (F, M or S)

```
function Predictor()
if state == F then
    if (ADW + size) < W_f then : return F
    else: return S
    end if
else if state == M then
    if (ADW + size) < W_m then : return M
    else: return S
    end if
else: return M
end if
```

ALGORITHM 2: The Algorithm of Buffer-Controlled Write

Input: The max loop of Buffered Write Sequence: $Loop_{max}$
Size of request R_i : $size_i$; Current written amount: ADW;
The state of last write: $state$;
Active padded writes and their size: PS,PF and $size_{PS}$,$size_{PF}$

```
sync()
while loop < Loop_max do
    if request R_i in the HDD write queue then
        write R_i to HDD, update ADW and state
    else
        if Predictor() == S then
            flag_HDD = False    // Stop receiving
            while state == S do
                write PS to HDD, update ADW and state
            end while
            flag_HDD = True    // Start receiving
            reset ADW; loop++
        end if
        if Predictor() == M then
            write PF to HDD; update ADW and state
        end if
    end if
end while
```

Upon activating BCW, a *sync*() operation is invoked to force synchronization to empty the buffer actively. BCW dispatches sequential user writes to HDD if it is predicted to be in F or M state, otherwise pads non-user data to HDD, until it reaches the max setting loop (or unlimited) of Buffered Write Sequence. If there are user requests in the queue, BCW writes them serially. After a write is completed, BCW adds its write size to ADW, and updates the write-state accordingly.

During light or idle workload periods with sparse requests, the HDD request queue will be empty from time to time, making the write stream discontinuous. To ensure the stability and certainty of buffered writes in a sequential and continuous pattern, BCW will proactively pad non-user data to the HDD. There are two types of padded data, PF and PS. The former is used to fill the F and M states with 4 KB non-user data; the latter is to fill the S state with larger block size, e.g., 64 KB

of non-user data. A small *PF* can minimize the waiting time of user requests. A large *PS* helps trigger Slow write quickly. Note that BCW still executes the write-state predictor algorithm for each padded write.

More specifically, BCW continuously calculates ADW in the current state (*F* or *M*). When ADW is close to W_f or W_m, it means that the HDD buffered write is at the tail of the Fast or Mid stage. The *S* write state may occur after several writes. At this point, BCW notifies the scheduler and proactively triggers the Slow write with *PS*. To avoid long latency for user writes, at this period, all incoming user requests have to be steered to other storage devices, such as SSDs. When an *S* write is completed, the next write will be *M* according to the write-state predictor. Then BCW resets the ADW and accepts user requests again.

We also find it unnecessary to proactively fill padded writes in the *F* stage before ADW exceeds W_f. When ADW does not reach W_f, the physical disk operation is not yet triggered and the buffer can absorb user requests for this period. When ADW exceeds W_f in a short time of period, it means that the buffer will begin to flush the data to the disk and the next write state will be changed to *S*. On the other hand, when ADW is less than W_f for a long time of period, the disk can flush the buffered data automatically so that the next write state may be *F*. However, it does not affect performance. We apply this observation to the scheduler design in the next section.

In most cases, the sequential and continuous write pattern induced by BCW is reasonably stable. However, this pattern can be broken, e.g., HDD reads. Besides, the *S* write states may be triggered in advance by user requests or *PF* writes. To regain the control of buffered writes, BCW continuously executes *PS* until a *S* write state is detected. As a result, the write-state predictor will be recalibrated. The cost of this strategy is the waste of IO and storage of BCW to perform *PS* writes to HDD. In addition, we can also issue *sync*() to reset the buffered write state in BCW. Howerver, a *sync*() can take several hundred milliseconds, during which the HDD cannot accept any writes. Fortunately, in the experiment, we find that the BCW interrupted cases are rare.

BCW stores incoming data to HDD in a log-append manner. This differs from traditional logging mechanism in existing file systems like ext4 [40], which allocates writes to the tail logical address of the log, ensuring address continuity. However, it doesn't ensure IO continuity and does not determine the next write state. In contrast, BCW maintains both address and IO continuity, making the buffer writing control effectively.

Accuracy and Stability. We assess the prediction accuracy of the write-state predictor and the robustness of BCW. We define three synthetic workloads and describe them as follows. The total data written volume for each workload is 100GB. We perform predictions for each request, and calculate both the correctness and error rate of prediction on the *F*, *M*, and *S* write states respectively.

- Synthetic workload (1): *Sequntial write*. It generates sequential and continuous write IO stream with random sizes from 4 KB to 128 KB. We invoke *sync()* after each 1GB data written. This workload represents the normal state of BCW.
- Synthetic workload (2): *Write dominated*. It randomly invokes a small number (i.e., less than 1% of total requests counts) of read requests or *sync* operations to the HDD, which occasionally interrupts BCW.
- Synthetic workload (3): *Mixed read/write*. Read requests account for 50% of the total workload, which causes frequent BCW interruptions and predictor resets. Note that BCW should stop running in this case. However, we can still estimate the accuracy of write-state predictor and the stability of BCW.

The results in Table 4 show that the predictor correctly identifies at least 97.1% of the *F* state, 97.9% of the *M* state, and 99.8% of the *S* state. The high probability for mis-predicting the *F* and *M* write-states to *S* is because the prediction policy tends to prevent the *S* write-state, thus avoiding

Table 4. The Correctness and Error Rate of the Prediction on F, M, and S Write States

Actual state	F			M			S		
Predicted state	**F**	**M**	**S**	**F**	**M**	**S**	**F**	**M**	**S**
Workload 1	99.89%	0.00%	0.11%	0.10%	97.88%	2.02%	0.01%	0.12%	99.88%
Workload 2	97.19%	0.89%	1.92%	0.19%	97.91%	1.90%	0.03%	0.17%	99.79%
Workload 3	99.68%	0.00%	0.32%	-%	-%	-%	-%	-%	-%

We use the West Digital 10 TB HDD (model WD100EFAX shown in Table 2) as a typical example.

performance degradation. It is better than mis-predicting an actual S write-state to other state. In the synthetic workload 2, the error prediction rate of F write-state is 1.8% higher than in the workload 1, while that of M and S write-states is almost unchanged. This is because BCW could be occasionally interrupted and then restart from the F stage. However, mis-predicting a F state does not degrade the performance. In the workload 3, BCW is frequently interrupted by read requests and cannot access M or S stages in a buffered write sequence. Besides, we carry out the same synthetic test for other types of HDDs and the results are the same basically.

Moreover, BCW performs recalibration in runtime based on the runtime IO results with respect to disk aging and variation on disk characteristics. BCW recalibrates at the end of each complete HDD buffered write sequences. It records key parameters (i.e., W_f, W_m and W_s) in the HDD Buffered-Write Model from both the actual disk IO feedback and the prediction results in each sequence. When the difference of the parameter values between the prediction model and the actual IO feedback exceeds a PF size (i.e., 4 KB), BCW will adjust these parameters in the prediction model to the actual value. In addition, to smooth adjust parameters in recalibration, we record the actual parameters from the last ten write sequences, and use their average value as a new parameter value. Secondly, the recalibration will not be triggered if a buffered write sequence is unexpectedly interrupted by a read request or a *sync*() operation.

Persistency and Reliability. The built-in buffer in almost all of the existing HDDs is volatile, which could cause data loss in the face of unexpected power outages. Meanwhile, it is enabled to buffer write data at the default factory settings. Therefore, in the ordinary case, all HDD-write requests would write into the buffer first and then flush to the disc. However, it is hard for the hosts to explicitly determine whether the data remain in the built-in buffer. Compared with that, BCW monitors and controls the states of the built-in buffer in an active manner, which makes the unreliability time window of the buffered data deterministic and measurable.

4.3 MIOS

BCW provides a proactive and controllable buffer writing approach for HDDs. In this section, we further propose an MIOS for SSD-HDD hybrid storage to leverage BCW effectively. The scheduler decides whether or not to steer user writes to an HDD queue according to the results of the write-state predictor and current queue status.

Architecture. The architecture of MIOS is shown in Figure 6. Generally, a typical hybrid storage node could contain multiple SSDs and HDDs in practice, and the number of HDDs is usually higher than that of SSDs. In MIOS, all disks are divided into independent SSD-HDD pairs, each of which contains an SSD and one or more HDDs, and is managed by an independent MIOS scheduler instance. MIOS monitors all request queues of SSDs and HDDs at runtime, judiciously triggers the BCW process, and determines whether a user write should be directed to a selected HDD or SSD. MIOS creates a device file to each HDD in the configuration process, which stores BCW writes in an append-only manner. Before MIOS scheduling, a profiling is performed to determine the key parameters (W_f, W_m, etc.) for the write-state predictor.

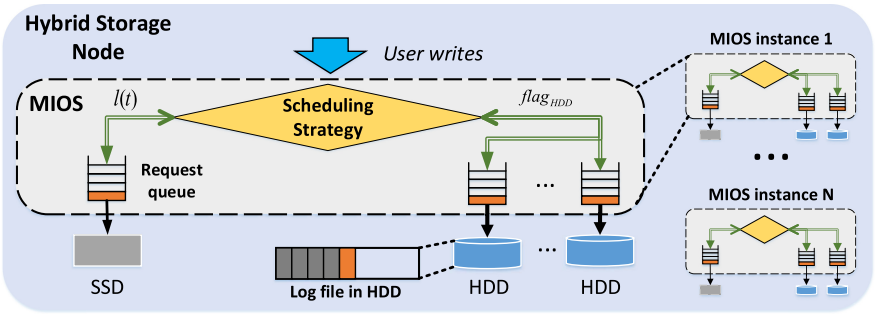


Fig. 6. Architecture of the MIOS. MIOS monitors all request queues of SSDs and HDDs in SSD-HDD pairs. The user writes meeting the conditions are redirected to appropriate HDDs.

ALGORITHM 3: The Scheduling Strategy of MIOS

Input: SSD queue length at time t: $l(t)$;
Queue length threshold: L; HDD available flag: $flag_{HDD}$;
Schedule Strategy: $MIOS\_D$ or $MIOS\_E$

```
if (flag_HDD == True) then
    if l(t) > L && Predictor() == F or M then
        Send to HDD queue
    else if MIOS_E && Predictor() == F then
        Send to HDD queue
    else: Send to SSD queue
    end if
else: Send to SSD queue
end if
```

Scheduling Strategy. In Algorithm 3, the SSD request queue length $l(t)$ at time t is a key parameter in MIOS. When $l(t)$ is larger than a predefined threshold L, the scheduler steers user writes to an HDD with the prediction of it being F or M write state. The threshold L is pre-determined according to the actual performance measurements on SSD. Specifically, we measure the write latency under different SSD queue lengths. If the request with queue length l has latency larger than the IO delay of HDD in the M state, we simply set the threshold L to the minimum l. The rationale is that when the SSD queue length is larger than L, the SSD writes' latencies will be at the same level as their latencies on an HDD in the F or M write state with BCW. L can be determined and adjusted experimentally according to workload behaviors and storage device configurations at runtime. This strategy mitigates, though not avoids, the long-tail latency upon workload bursts or heavy GCs on SSD [44, 57]. In these cases, the SSD request queue length can be 8–10 times longer than its average. Therefore, redirected HDD writes not only relieve SSD pressure imposed by bursty requests and heavy GCs, thus curbing the long-tail latency, but also lower the average latency.

Additionally, when the queue length of SSD is less than L, triggering BCW is optional. Enabling or Disabling BCW in this case is denoted as $MIOS\_E$ or $MIOS\_D$, respectively. In other words, $MIOS\_E$ strategy allows redirection with BCW when the queue length of SSD is lower than L. $MIOS\_D$ strategy, by contrast, disables redirection when the SSD queue length is lower than L. We will experimentally analyze the positive and negative effects of $MIOS\_D$ and $MIOS\_E$ in Section 5.

Finally, BCW requires the complete control over HDDs that cannot be interfered by other IO operations. When an HDD is executing BCW and a read request arrives, MIOS immediately suspends

ALGORITHM 4: The Multi-HDD Scheduling Algorithm of MIOS

Input: Request R_i; Number of HDD N;
The available flag of HDD_i: $flag_{HDD_i}$;

```
1: if R_i == read then
2:     stop BCW in HDD_i
3:     flag_HDD_i = false
4:     send to HDD_i queue
5: else
6:     if flag_HDD_1-N == false then
7:         write back to SSD
8:     end if
9:     predict-state[] = Predictor(HDD_1-N)
10:    if predict-state[HDD_i] has the fastest write state above other HDDs && (flag_HDD_i) == True then
11:        send to HDD_i queue
12:    end if
13:    if predict-state[HDD_1-N] != F then
14:        enable full BCW
15:    end if
16:    if predict-state(HDD_i) != F && predict-state(HDD_t(t!=i)) == F then
17:        sync(HDD_i)
18:    end if
19: end if
```

BCW and serves this read, and then try to redirect all writes to other idle disks with a multi-HDD scheduling. We will explain this in the next section. For read-dominated workloads, BCW can also be disabled to avoid interfering with reads.

Multi-HDD Scheduling. MIOS and BCW actually increase the IO burden of HDDs. First, MIOS redirects a large number of SSD write requests to HDDs. Second, BCW has to write padded data into HDDs. Single HDD has limited IO capacity, making the HDD difficult to be always IO-available when performing SSD redirection. Meanwhile, BCW has to suspend to serve read requests in advance. All of these could make the HDDs bottleneck and affect the efficiency of MIOS scheduling. In addition, it is hard for MIOS to redirect requests using only the best-performing *F* write states to achieve better latency. This is because the *F* stage has limited capacity and could only be used after *sync*(). When storage nodes suffer intensive writes, the *F* stage will be used up quickly. MIOS has to invoke sync to obtain the *F* write state in a round-robin way for all idle HDDs. We find in our experiments that the *sync* operation is costly, which could cause MIOS to lose the redirecting opportunities even if SSDs are busy.

To improve the effectiveness of MIOS and to further exploit the low latency feature of HDD buffered write, we propose a multi-HDD scheduling mechanism in MIOS and describe it as Algorithm 4. MIOS is extended to monitor all HDDs in an SSD-HDD pair. It prioritizes HDD that is idle (i.e., queue length is 0) and has the lowest predicted latency to maximize write performance. Particularly, this mechanism is proposed to increase the utilization of the best-performing *F* write states in HDD. First of all, the multi-HDD scheduling will preemptively free the HDD buffers to ensure MIOS always has available *F* write states for scheduling. It will invoke the *sync* operation on the HDD that totally depletes *F* stage, on condition that available *F* write states still exist in other HDDs. Then, once the *F* write state is available for any HDD, MIOS will enable the full BCW process to boost IO capacity under high write intensity. If all HDDs are busy, requests will write back to SSD. Finally, if BCW is interrupted by a read request or other operation, MIOS could write request to the other HDDs and restore the availability of BCW as soon as possible.

Implementation. BCW is a runtime write-scheduling mechanism running on SSD-HDD hybrid storage. BCW is orthogonal to local file system running on HDDs. MIOS can be implemented in either file-level or volume-level to jointly manage SSDs and HDDs in a hybrid storage. In this work, MIOS provides a simple yet flexible file-level request scheduling scheme atop of file systems. HDD-logs of MIOS can directly map to specified areas within raw HDD without native file system, or be configured as large log-files of the local file system (e.g., Ext4 and F2FS) running on the HDD to leverage their mature file-to-storage mapping mechanism. To reduce overhead of the underlying file system, MIOS employs direct IO mode to access the log by calling Linux kernel functions such as *open*, *close*, *read*, and *write*. When enabling BCW, its involving HDDs should be fully controlled and cannot serve any IO of other applications. To meet this demand, MIOS carefully coordinates multiple SSDs and HDDs in a hybrid storage server. It merely writes user data to the HDDs while storing the log-metadata to the SSDs to ensure that metadata IO does not interrupt BCW. Besides, for MIOS based on file system, modifying a file could trigger its metadata update and leads to uncontrolled file-system metadata IOs, which interrupts BCW. Therefore, we pre-create and allocate the space of all log-files with a fixed size and cache its addresses. MIOS only updates their file data in place, which cannot generate extra file-system metadata IOs.

MIOS manages the data-chunk location to an SSD-log file in each MIOS instance. We design a request node as metadata structure that records and tracks each redirected chunk data in the HDD-log. Each request node represents a request pointed to its relevant chunk. It contains five fields: the chunk file name of a redirected write request (*File ID*), length of the request (*Length*), location of the request that is persisted in the log file (*Physical offset*), position of the request in the corresponding chunk file (*Logical offset*), and the request arrival time (*Timestamp*) to keep the sequential order of requests. When the chunk data are redirected into the HDD-log, the relevant request node as metadata is periodically written into the SSD-log. We aggregate multiple request nodes to 512 B before flushing to match the sector size of SSD. In addition, if there are no IO writes for more than 100 ms, the unaggregated request nodes will also be forcibly flushed. MIOS set a single thread to serially write new request nodes to the SSD-log in the order of their Timestamp field. Therefore, the data location manager will not suffer a lock contention issue even if there are multiple threads updating a chunk file at the same time. Besides, all request nodes are also cached in the memory to accelerate reads for locating the redirected chunks. After a crash, the recovery procedure scans all SSD-logs to rewrite all redirected chunks to their own original locations. It only uses intact request nodes with five fields and broken request nodes will be discarded. Therefore, the data location manager will not be inconsistent after system restart.

We periodically perform **HDD garbage collection (HDD-GC)** to recycle the logs on HDD. When an HDD is idle, all file-data stored in the log are re-written to their own original chunk files, and then the log are completely recycled. HDD-GC also should be forcedly triggered when the log size exceeds a predefined threshold (e.g., 16 GB). HDD-GC first sequentially and continuously reads all data in the log to reduce seeks, and then extracts and merges the user data to update their correspond files. These chunk file updates can be performed in batch [63]. Meanwhile, the multi-HDD scheduling mechanism can distribute requests to other HDDs during the HDD-GC execution to maintain the available BCW process. Additionally, HDDs have not write-induced wear-out problem unlike SSDs. Therefore, the temporary HDD-storage wastage caused by BCW is not a burden.

5 EVALUATION

5.1 Experiment Setup

We run experiments for performance evaluation on a server with two Intel Xeon E5-2696 v4 processors (2.20 GHz, 22 CPUs) and 128 GB of DDR4 DRAM. To understand the impact of different

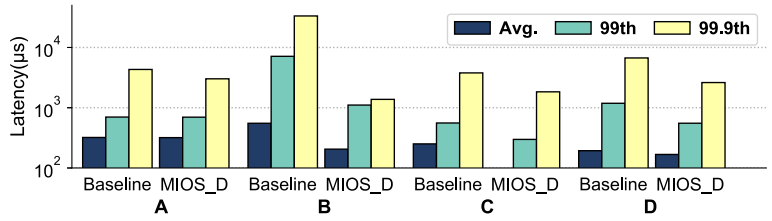


Fig. 7. The average, 99^{th} and 99.9^{th}-percentile latency under four Pangu production workloads, comparing *Baseline* with *MIOS_D* (a logscale is used for the y-axis).

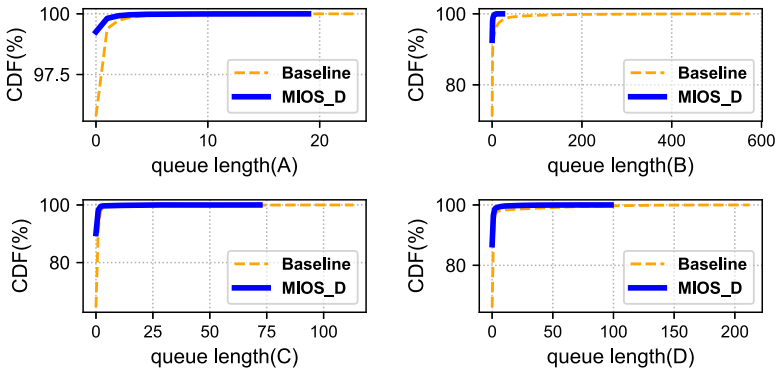


Fig. 8. The CDF of SSD queue length.

storage configurations on the performance, we choose two models of SSDs, a 256 GB Intel 660p SSD [13], a 256 GB Samsung 960EVO SSD [19], and a 480 GB Intel Optane 905P SSD [12]. Their long-term stable write throughputs are 0.6 GB/s, 1.5 GB/s, and 2.1 GB/s, respectively. Three models of HDDs are WD 10 TB, WD 4 TB, and Seagate 4 TB, as described in Table 2.

By profiling those HDDs as described in Figure 1, we could obtain the key parameters of BCW. For example, the 10 TB WD HDD has a W_f of 16 MB and W_m of 8 MB. Using the process to predetermine the queue length threshold L explained in Section 4, we set L to 1 for workload of node A, 3 for node B, and 2 for node C and D, where the workloads of nodes A, B, C, and D are described in Table 1 of Section 2. As discussed earlier, MIOS has two schemes, *MIOS_D* and *MIOS_E*. When the SSD queue length is less than L, the former conservatively disables request redirection; the latter allows request redirection but only redirects user write requests to the F write state. The *Baseline* for the evaluation is writing all user data into the SSDs. In addition, a complete BCW sequence consists a series of one Fast stage and ten Mid/Slow stage-pairs (Figure 4). At last, the number of HDDs N is set to 1 through 4 in an SSD-HDD pair. When $N = 1$, MIOS cannot perform multiple HDD scheduling. In experiment, we deploy all HDDs as the same model (i.e., WD 4 TB).

5.2 MIOS Under Production Workloads

We first evaluate the effectiveness of *MIOS_D* under four Pangu production workloads on the WD 10 TB HDD.

Write Performance. Figure 7 shows that the average and tail latency (99^{th} and 99.9^{th}) of all four workloads are significantly reduced by *MIOS_D*. Among four workloads, B gains the most benefit. Its average, 99^{th} and 99.9^{th}-precentile latencies are reduced by 65%, 85%, and 95% respectively. On the contrary, these three latencies in A are only reduced by about 2%, 3.5%, and 30%,

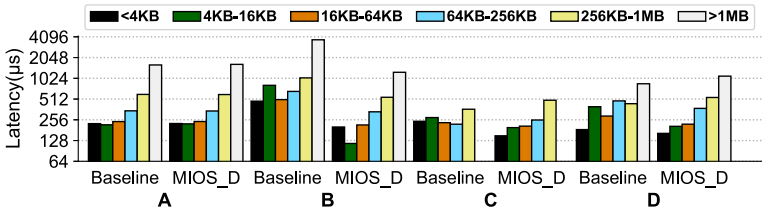


Fig. 9. The average request latency in six request-size groups that are classified by IO size with *MIOS_D*.

respectively, which is far less than the other workloads. The reason is that the redirection in *MIOS_D* is only triggered when the queue length is high, but *A* has the least intensity and thus the least queue blocking, which renders MIOS much less useful.

To better understand the root causes for the above experimental results, the **cumulative distribution functions (CDFs)** of SSD queue lengths for four workloads are shown in Figure 8. *MIOS_D* significantly shortens queue lengths compared to *Baseline*. B and A have the maximum (95%) and minimum (15%) reduction in their queue lengths. Therefore, the overall queueing delay is reduced significantly.

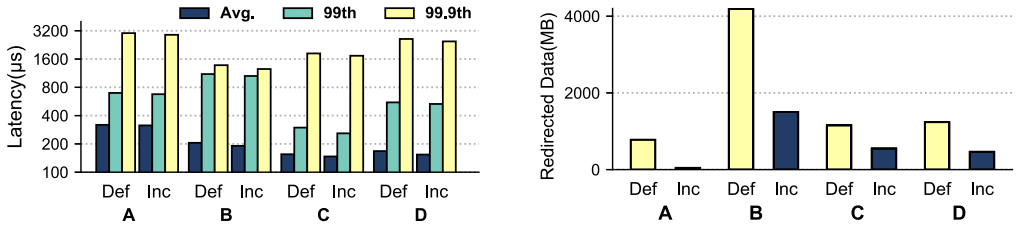
Request Size. To deeply understand impact of write size in MIOS and BCW, we break down all redirected requests into six groups with different ranges of IO sizes, and measure the average latency with *MIOS_D* in each group.

Figure 9 shows that, *MIOS_D* reduces the average write latency of size below 64 KB in all four workloads, and workload B benefits the most. The average latencies of three groups of small-sized requests (<4 KB; 4 KB–16 KB; 16 KB–64 KB) are reduced by 61%, 85%, and 59%, respectively. The other three workloads also reduce their latencies differently. In *Baseline*, small and intensive requests result in queue blocking more frequently (Figure 2) than in *MIOS_D*. Therefore, *MIOS_D* is the most effective in reducing latency in such cases.

However, in groups of requests larger than 256 KB, the average latency is increased in all workloads except B. For workload D, the average latency is increased by 31.7% in the >1 MB group, and 12.1% in the 256 KB–1 MB group. The average latency of the 256 KB–1 MB group in C is also increased by 20.1%. The reason is twofold. First, large SSD writes under light load have better performance than HDDs because SSDs have high internal-parallelism that favors large requests. Second, large writes are relatively sparse and not easy to be completely blocked. For example, the average latency of the >256 KB request-size groups in *Baseline* is very close to the raw SSD write performance.

Queue Length Threshold L. To evaluate the effect of L selection, we compare the pre-defined L value (**Def**) determined by the process described Section 4.2, with $L + 1$ (**Inc**). Note that the pre-defining process for queue length threshold is designed to tradeoff between decreasing the write latency and reducing the write traffic to SSD.

Figure 10(a) shows that *Inc* slightly reduces average, 99^{th} and 99.9^{th}-percentile latency compared to *Def*. Among the four workloads, the maximum reduction in average latency is less than 10%. This is because the higher queue length is, the longer waiting delay a request experiences. Therefore, *Inc* can acquire more latency gains by redirection than *Def*. However, the choice of L value can greatly affect the amount of redirected data. In Figure 10(b), the number of redirected requests is much smaller in *Inc* than in *Def*. The amount of redirected data for workloads A~D is decreased by 94%, 64%, 52%, and 62%, respectively. These results are consistent with the implications of Figure 8 that longer queue length in SSD triggers much fewer SSD overuse alerts, significantly reducing chances for request redirecting to HDD.



(a) The average and 99^{th} tail latency with different L values (b) The redirected written data with different L value

Fig. 10. *MIOS_D* with different queue length threshold L.

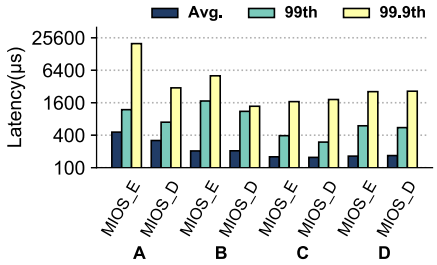
Table 5. The Amount of Redirected Writes Data and Requests with the *MIOS_D* and the *MIOS_E* Strategies

Workload Type	A	B	C	D
Writing Method	Baseline / MIOS_D / MIOS_E			
SSD Writes (GB)	14.7 / 13.9 / 1.2	61.2 / 57.1 / 48.1	7.2 / 6.1 / 2.1	7.5 / 6.3 / 2.1
HDD Writes (GB)	- / 4.1 / 61.6	- / 18.4 / 56	- / 4.5 / 22.3	- / 4.4 / 25.6
SSD Requests (millions)	0.43 / 0.36 / 0.04	4.4 / 3.7 / 1.3	4.8 / 3.7 / 1.6	4.7 / 3.8 / 1.3

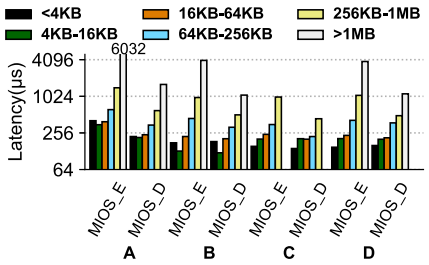
MIOS_D vs. MIOS_E. We compare *MIOS_D* with *MIOS_E* in terms of the amount of data written to SSD and HDD, and the number of redirected write requests. Results are shown in Table 5. Workload A has the highest percentage of redirected data and requests with *MIOS_E*. It reduces the SSD written data by up to 93.3% compared with *Baseline*, which is significantly higher than *MIOS_D*. Since workload A has lower IO intensity, *MIOS_E* has more chances to redirect even when the queue length is low. Note that we also count the padded data in BCW as the amount of data written in HDD. In such a case the total amount of data written can vary a great deal. Workload B has the lowest percentage of redirection with *MIOS_E*, which reduces SSD written data by 30%. Nevertheless, the absolute amount of redirected data is very large because the SSD written data in *Baseline* is larger than any of the other three workloads. Compared with *MIOS_D*, *MIOS_E* can greatly decrease the amount of data written to SSD. Therefore, it is more beneficial to alleviate SSD wear-out.

However, the negative effect of *MIOS_E* is the increase of average and tail latency. In Figure 11(a), *MIOS_E* leads to generally higher average latency than *MIOS_D* by up to 40% under workload A. Although for other three workloads, the average latency remains basically unchanged. This is because much more writes (i.e., >90%) are redirected by *MIOS_E* than by *MIOS_D* in workload A, and requests in HDD experience longer latency than that in SSD. Moreover, the 99.9^{th}-percentile latency of *MIOS_E* is increased by 70% in A, 55% in B, 31% in C, and 8% in D compared to *MIOS_D*. The results can be explained by Figure 11(b). *MIOS_E* increases the average latency for nearly all the IO size groups, especially for the groups with requests of size larger than 256KB.

Comparison with Other HDD Writing Mechanisms. We compare BCW with other HDD writing mechanisms. The first one is *BCW_OF* that only uses the F write state by proactively issuing *sync*() when the ADW reaches W_f in a HDD. The second one is *Logging* [36, 48], which is an ordinary way to improve the performance of HDD-writes by storing written data in an append-only manner. The difference between *Logging* and *BCW* is that the former cannot predict or selectively determine to use low-latency HDD buffered write states while the latter can. We measure

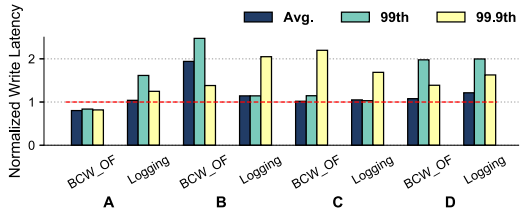


(a) Average and tail latency with *MIOS_D* and *MIOS_E*

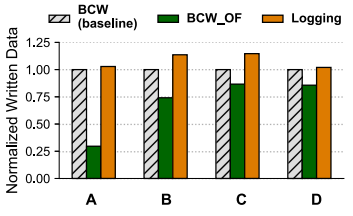


(b) Average latency for different request-size group

Fig. 11. *MIOS_D* vs. *MIOS_E*.



(a) Average and tail latency



(b) SSD write reduction

Fig. 12. Latency and SSD written data reduction with *BCW_OF* and *Logging* (normalized to *BCW*).

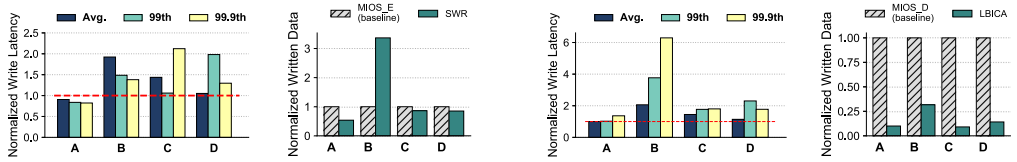
the average, 99^{th}, 99.9^{th}-percentile latency and the SSD written data reduction with *BCW_OF* and *Logging*. We combine all these HDD write mechanisms with MIOS scheduler that uses the MIOS_E strategy. We take MIOS_E with *BCW* as the baseline and present performance normalized to it.

Figure 12(a) shows that the 99.9^{th}-percentile latency of *BCW_OF* is increased by 2.19× over *BCW* in workload C. The 99^{th}-percentile latency in the B, C, and D workloads also increases by 2.5×, 1.1×, and 1.9×, respectively. It means that *BCW_OF* performs less efficiently on reducing tail latency when the workload becomes heavier. This is because such mechanism redirects less requests when SSD suffers queue blockage. Figure 12(b) shows that, the data volume of SSD redirection is reduced by 71% in workload A and 26% in workload B. As mentioned in Section 4.2, *sync*() is a high cost (e.g., tens of milliseconds) operation to flush the HDD buffer and HDDs cannot serve any requests during this time window.

Furthermore, *Logging* can reduce 10% more SSD written data than *BCW* at the cost of explicit write latency increase. The 99.9^{th}-percentile latency in the B, C, and D workloads is 2.0×, 1.9×, and 1.6× higher than *BCW*, reaching several milliseconds. This is because *Logging* cannot prevent write requests to encounter the *S* write states during intensive workloads.

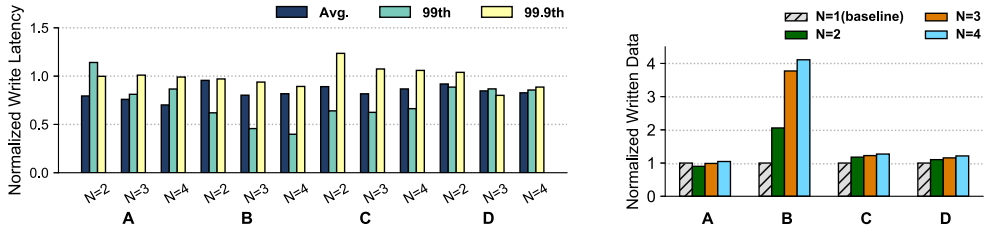
Comparison with Existing IO Scheduling Approaches. LBICA [1] and SWR [38] are the state-of-the-art IO scheduling approaches in SSD-HDD hybrid storage. In the write-dominated workload, LBICA redirects requests from the tail of the SSD queue to the HDDs only when SSD has long queue length. Besides, SWR redirects large size write requests preferentially and allows redirecting when the SSD queue length is low. We normalize the performance of *LBICA* and *SWR* to MIOS_D and MIOS_E, respectively.

Figure 13(c) illustrates that *LBICA* increases the average and tail latency by up to 2.0× and 6.2× over MIOS_D in workload B. *LBICA* calculates the redirect threshold *L* by comparing the average latency of random HDD-writes with that of the SSD-writes in tail of the queue, leading to a high



(a) Latency of SWR (Normalized to MIOS_E) (b) SSD write reduction with SWR (c) Latency of LBICA (Normalized to MIOS_D) (d) SSD write reduction with LBICA

Fig. 13. Latency and SSD written data reduction with SWR and LBICA.



(a) Average and tail latency with different number of HDDs (i.e., N value). (b) SSD write reduction.

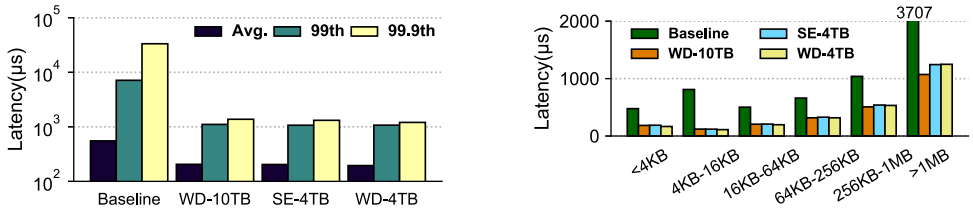
Fig. 14. Latency and SSD written data reduction with multiple-HDD scheduling (normalized to *MIOS_E* with $N = 1$).

L value. It means that *LBICA* can only redirect requests when SSD suffers high queue length, which is a rare condition. Figure 13(d) shows that the SSD data reduction of *LBICA* is decreased 70%–90% than MIOS_D. Compared to *LBICA*, MIOS_D performs SSD-write redirection under a lower queue length, because BCW could provide μs-level latency for HDD-writes.

Figure 13(a) shows that the average and tail latency of *SWR* is 1.2× −2.1× higher than MIOS_E. There are two main reasons. First, *SWR* prefers to redirect requests with large IO size. However, Figure 9 indicates that the performance degradation of redirecting large size request is more than that of a small one, due to the high internal parallelism of SSDs. Second, *SWR* cannot take full advantage of the low-latency HDD write states and results in increased write latency. The advantage of *SWR* scheduling scheme is that it could efficiently perform redirection under high IO intensity. Figure 13(b) shows that the amount of SSD write reduction of *SWR* is 3.4× more than that of MIOS_E in workload B.

Multi-HDD Scheduling. We measure the average, 99^{th} and 99.9^{th}-percentile latency of write requests with N values from 1 through 4. When $N = 1$, the multi-HDD scheduling mechanism cannot be performed. We take the *MIOS_E* with $N = 1$ as the baseline and present performance and data redirection normalized to it, because multi-HDD scheduling is more adaptable to high-intensity workloads.

Figure 14(a) shows that the multi-HDD scheduling provides a consistent improvement in request latency over single-HDD. The average latency of $N = 4$ is 30% lower compared to the baseline (i.e., $N = 1$) in workload A, and that is 17% higher in other three workloads. This is because the multi-HDD scheduling undertakes most redirection requests with the best-performing F write state. Meanwhile, the normalized 99^{th}-tail latency on $N = 4$ is decreased 50% than baseline in workload B, 20% in workload A and C, and 10% in workload D. It redirects more requests under high IO intensity, which further relieves SSD pressure. However, the 99.9^{th} tail latency is not significantly reduced, which even presents an increase of 3% and 23% in workload A and



(a) Average, 99^{th} and 99.9^{th}-percentile latency

(b) Average latency of different request-size group

Fig. 15. *MIOS_D* performance with three different models of HDDs under workload B.

Table 6. Amount of Data Written to and Number of Requests Processed in SSD with Different Models of HDDs Under Workload B

	Baseline	WD-10TB	WD-4TB	SE-4TB
SSD written data (GB)	61.2	57.1	57.0	56.8
SSD write requests (thousands)	4453	3733	3729	3684

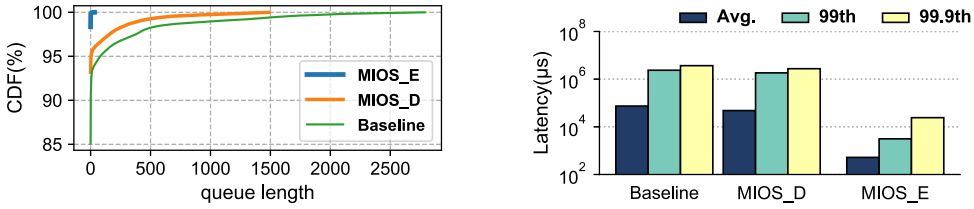
C, respectively. In the multi-HDD scheduling, MIOS redirects more requests to HDDs when the number of HDDs increases. Particularly, the number of redirected writes with the 0.5% largest IO size increase, which experience longer latency when written to HDD than that to SSD. It results in the increase of 99.9^{th}-percentile latency.

Figure 14(b) demonstrates that the multi-HDD scheduling further improves SSD data reduction. In workload B with the most amount of written data, $N = 4$ could redirect 4× more data than $N = 1$, while $N = 2$ or $N = 3$ is enough to redirect more than 92% of requests in other three workloads. We also measure the performance of multi-HDD scheduling with the *MIOS_D* strategy. We find that two HDDs are typically enough for redirection in all four workloads, and more disks present little improvement. This is because *MIOS_D* just performs redirection on IO bursty.

Experiment with Other HDDs. We use the 4 TB WD, 4 TB Seagate, and the 10 TB WD HDD to replay workload B, comparing *MIOS_D* (with the default L value) with the *Baseline* in terms of the request latency and the amount of data written to SSD. Workload B is chosen for this experiment, since it has the most SSD written data and the most severe SSD queue blockage, clearly reflecting the effect of IO scheduling.

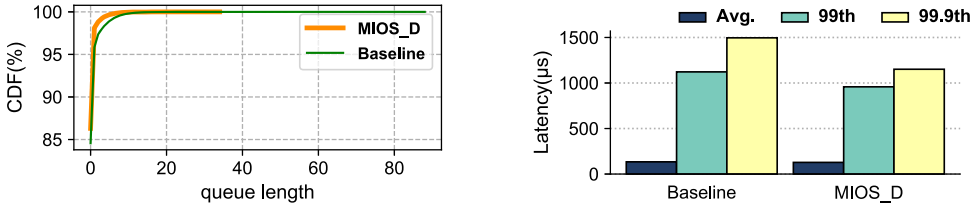
Figure 15 shows that different models of HDDs do not have a significant impact on the effect of *MIOS_D*. First, the average and tail latencies for all the three HDDs are virtually identical, with a maximum difference of less than 3%. In addition, in the six request-size groups, only the >1 MB group exhibits a large difference among different HDD models. The average latency of the 10 TB WD HDD is 14% lower than that of the other 4 TB HDDs. This is because of the native write performance gap between them. It can be found from Table 6 that different models of HDDs do not notably affect the amount of data redirected, with little difference of less than 5%.

Experiment with Other SSDs. Next, to further explore the effect of MIOS with different types of SSDs, we first deploy a lower-performance 660p SSD. We replay the same workload A that provides the lowest IO pressure, and employ the *MIOS_D* and *MIOS_E* strategies, respectively. From the latency CDF shown in Figure 16(a), when using the lower-performing SSD, more than 7% of the requests are severely affected by long queuing delay and the maximum queue length reaches up to 2,700. It surpasses the experiment result with the better-performing 960EVO (e.g., 23 shown in Figure 8(b)). This is because when the IO intensity exceeds the ability of 660p SSD to accommodate, the SSD queue length builds up quickly. As a result in Figure 16(b), the average and



(a) Queue length CDF (b) Average and tail latency

Fig. 16. Queue length CDF and latency under Pangu workload A, with the 660p SSD for three scheduling strategies.



(a) Queue length CDF (b) Average and tail latency

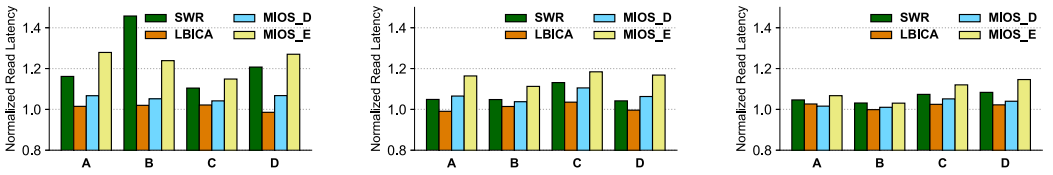
Fig. 17. Queue length CDF and latency under Pangu workload B with Optane SSD.

tail latencies in *Baseline* rise sharply compared with 960EVO SSD shown in previous Figure 7. The average latency in *Baseline* is 90 ms and the 99^{th}-percentile latency exceeds 5 second.

With such high pressure on the 660p, MIOS can help reduce some IO burden on SSD by redirecting queued requests to HDDs. As seen from Figure 16, *MIOS_D* decreases the queue blockage with 45% queue length reduction to the maximum. At the same time, the average latency in *MIOS_E* returns to μs-level (e.g., 521 μs), and the 99^{th} and 99.9^{th}-percentile latencies are reduced to an acceptable range of 2.4 ms and 87 ms, respectively. Because *MIOS_E* redirects much more SSD requests with low queue length, it prevents queue blockage in SSD, particularly for a lower-performing one. By comparing this experiment on a lower-performing SSD with an earlier one on a higher-performing SSD, we believe that when the SSD in hybrid storage cannot support the intensity of a write-dominated workload, MIOS and BCW can provide an effective way to improve the overall IO capacity by offloading part of the workload from SSD to HDD.

To further explore the effect of MIOS with the Intel Optane SSD, we replay the workload B that provides the highest IO pressure and employ the MIOS_D strategies. We do not use MIOS_E here because it is an aggressive strategy that redirects more requests to HDDs than MIOS_D. The excessive latency performance gap between HDDs and Optane SSDs will result in unacceptable performance degradation. Figure 17 shows that MIOS_D is still effective in the hybrid storage when using Optane SSD as primary write buffer. The 99.9^{th}-tail latency is reduced by up to 19% and it also decreases the queue blockage with 60% queue length reduction to the maximum in workload B. Although the Optane SSD has less GC overhead than NAND-based SSD, they have similar ability to handle large size IOs (e.g., Optane SSD takes 230 μs while 960EVO takes 260 μs for a 512 KB write IO in average). Therefore, the intensive write workload can also block queues and increase tail latency in Optane SSD.

In addition, we compare BCW to a system that simply adds an extra SSD. We equally distribute the workloads to two SSDs. The system can achieve the same or even better latency than *MIOS_E*, but at a significantly increased hardware cost.



(a) Average latency (b) 99^{th}-percentile latency (c) 99.9^{th}-percentile latency

Fig. 18. The average and tail latency of read requests.

Read Performance. We measure the average, 99^{th} and 99.9^{th}-percentile latency of external read requests (i.e., user reads) with *MIOS_D*, *MIOS_E*, *SWR*, and *LBICA* in all four workloads. We present all approaches normalized to the *Baseline* where all user data are directly written into the SSDs and then dutmped to HDDs.

The read performance of MIOS decreases slightly compared with that of the Baseline. Figure 18 indicates that the average read latency in *MIOS_E* is increased by 27%, 23%, 14%, and 28% in workload A, B, C, and D, respectively. The reasons are twofold. First, compared to Baseline, MIOS bypasses more SSD-writes directly to HDDs in advance, indicating that the HDDs have to serve more read operations. However, the majority of read requests in origin Pangu workloads are HDD-reads. For example, the proportion of SSD-reads in the total number of requests are less than 1.5% in all workloads. This is because Pangu periodically flushes data from SSD to HDD, and most of the data are already stored in HDDs. Therefore, *MIOS_E* turns up to 31% SSD-reads to HDD-reads and affects these average read latency slightly. Second, BCW uses the append manner to improve write performance and actively pads non-user data to HDDs. It results in the file data being scattered on HDD. However, we find that the overhead of BCW on HDD-reads is relatively small. This is because most read requests in these Pangu workloads are discrete spatially and temporally, i.e., random HDD-reads. In addition, the average size of the read requests is similar to that of writes, e.g., 4-16 KB, so that BCW rarely divides a read request into multiple read IOs. Note that, as described in Section 4.3, MIOS periodically writes logging data back to their interrelated chunk files, meaning that the long-term read performance in MIOS is identical with that in *Baseline*. Moreover, the increases of the 99^{th}- and 99.9^{th}-percentile latency are less pronounced with *MIOS_D* and *MIOS_E*. Because a large part of the tail read latency is caused by queue blocking, which could get to hundreds of milliseconds. We also observe that read latency increases more with MIOS than that with *SWR*, since MIOS leads to more HDD-reads and has worse data locality in HDD. Besides, *LBICA* has little effect on both average and tail read latency because it redirects far less SSD write requests than other approaches.

We discuss the tradeoffs between improved write performance and degraded read performance caused by MIOS and BCW. In short, *MIOS_D* reduces the average and tail latency of write requests, which account for 84%-98% of the total requests, by 65% and 95%, respectively. In contrast, those of read requests, which account for 2%–16% of the total requests, rises by 7% and up to 10% respectively. For the *MIOS_E* strategy, the average and tail latency of write requests is reduced by up to 60% and 85%, and those of read requests increase by 28% and 18%, respectively. Therefore, we indicate MIOS approach gains more benefits for applications with write-dominated IO patterns, in which a few reads with limited latency increase could be accepted. Users can determine the appropriate scheduling policy according to current IO patterns. *MIOS_E* is more suitable for the burst and intensive-write environments, while *MIOS_D* can be used for write-dominated workloads. In addition, MIOS can be disabled in read-dominated situations.

Table 7. The HDD Utilization with *MIOS_E* and *MIOS_D* in Single-HDD and with *MIOS_E* in Multi-HDD

Workload Type	Duration (s)	Net Uti. Baseline	Net Uti. MIOS_D	Net Uti. MIOS_E	Gross Uti. MIOS_E	Avg. Net Uti. ($N = 4$) MIOS_E	Avg. Gross Uti. ($N = 4$) MIOS_E
A	2700	7.6%	7.9%	11.9%	27.9%	8.8%	8.8%
B	1800	9.8%	18.2%	26.8%	56.9%	20.0%	20.5%
C	1800	4.1%	10.7%	16.2%	35.8%	7.6%	7.6%
D	1560	4.8%	12.3%	17.3%	39.5%	8.4%	8.5%

5.3 Cost of BCW

We further analyze and evaluate the wasted storage-space of BCW due to the padded data for keeping continuous HDD written pattern and skipping the *S* write state.

Amount of Padded Data. We first analyze the amount of padded data written to HDD. In Table 5, we measure the data amount with *MIOS_D* and *MIOS_E* when a BCW sequence contains one Fast stage and 10 Mid/Slow stage-pairs. The statistics in the table clearly indicates that *MIOS_E* generates substantially more HDD write data than *MIOS_D*. When more requests are redirected, the amount of padded data increases proportionally. For example, the padded data with *MIOS_E* is 15× that with *MIOS_D* in workload A, 3× in workload B, 4× in workloads C and D. Frequently triggering BCW increases the occurrences of padded data. Furthermore, when the amount of redirected data increases, the Fast stage without padded data will be used up faster and more Mid/Slow stage-pairs with padded data will be executed.

HDD Utilization. The original Pangu traces exhibit low HDD utilization, which is defined by the percentage of time that an HDD is actually working on processing IO requests. More specifically, Tables 1 and 7 show that HDDs generally keep very low utilization (e.g., <10%) in all four workloads. Using MIOS and BCW, the HDD utilization has been increased with different degrees. The gross utilization is defined to be the percentage of the total execution time when HDD is working on IO requests (including *sync*() operation), which is the real usage of the disks. The highest gross utilization is 56.9% under workload B. This means that the disk still has enough free time for HDD-GC. To analyze the amount of time HDD is effectively working for user requests, we define *net utilization* as the percentage of the total execution time that the HDD spends exclusively serving user requests, excluding the time spent on padding data in BCW. Thus, it is positively correlated to the amount of redirected data. The net utilization of HDD in *MIOS_E* is higher than that in *MIOS_D*. Under workload B and *MIOS_E*, HDD has the highest net utilization improvement over *Baseline*, by 2.7×, while the same is enhanced to 1.8× under *MIOS_D*. In addition, if MIOS enables multi-HDD scheduling, the HDD utilization would drop significantly. First, multiple HDDs would serve the data volume of SSD redirect requests. Second, multi-HDD scheduling does not require BCW to pad data frequently, which results in less padded data and lower gross utilization.

5.4 Write Intensity

The efficiency of BCW heavily depends on the write intensity. To better understand this relationship, we test the average and tail latency as a function of write intensity (in terms of IO transmission interval) with three scheduling strategies as the *Baseline* (**_B**), *MIOS_D* (**_D**), and *MIOS_E* (**_E**). We initialize the IO size to 32 KB, and continuously issue write requests using (Flexible I/O tester) [5]. Since FIO cannot adjust IO intensity, we set the generated IOs with a fixed transmission interval from 20 to 320 μs. We use 960EVO SSD and 10 TB WD HDD, and set the *L* value to 1.

Figure 19 shows that when the request interval is between 20 and 60 μs, the SSD writes are severely blocked and the 99^{th}-tail latency reaches as high as 5.2 second. In this case, both *MIOS_D*

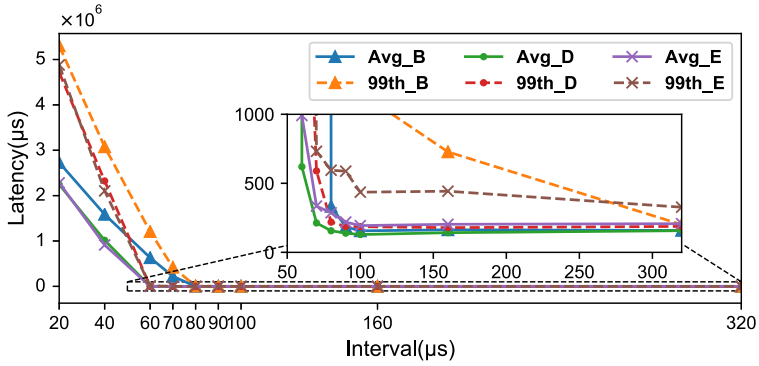


Fig. 19. FIO benchmark to experiment with three strategies. The IO transmission interval is set to 20–320 μs.

and *MIOS_E* can significantly reduce the tail latency. And *MIOS_E* is slightly better than *MIOS_D* because the former handles more burst writes. When the interval is 60–80 μs, *Baseline* still exhibits very high latency in SSD. However, the latency has returned to an acceptable μs-level after MIOS scheduling. When the interval exceeds 100 μs, the average and 99^{th}-percentile latencies are stable, because there is very little SSD queue blockage under such request intensity. In this case, *MIOS_D* and *Baseline* have the lowest average latency and remain the same as request interval grows. However, the average and tail latency of *MIOS_E* is higher than others. This is because even if there is no queue in SSD, *MIOS_E* will still redirect requests, and the performance gap between SSD and HDD can lead to high latency.

6 RELATED WORKS

IO Scheduling. The IO scheduling methods in host systems have been adequately studied. The Linux CFQ [49], Anticipatory, Deadline [8], and NCQ [64] provide fairness and high efficiency scheduling on HDDs. Nowadays, with wide adoption of SSDs, more recent researches address IO characteristics of the flash media as read/write performance asymmetry and internal parallelism. FIOS [46] employs a fair IO timeslice management to attains fairness and high efficiency of SSD disks. HIOS [27] gives GC-aware and QoS-aware scheduler in host. PIQ [22] and ParDispatcher [55] minimize access conflicts between IO requests by exploring the rich internal parallelism of SSDs. In contrast to these efforts, BCW models and exploits the feature of HDD built-in buffer, and is proposed to provide μs-level low latency HDD writes.

Moreover, a large body of research offer fine-grained scheduling inside of SSD devices to reduce interference between IO flows [51], write amplification [32], and GC overhead [20, 24, 28]. FLIN [51] proposes a lightweight IO scheduler for modern multi-queue SSDs [50]. It provides fairness among concurrent IO flows inside SSDs by distributing the GC overhead proportionally across different IO flows. However, FLIN cannot eliminate GC or contention between different flows. GC preemption [31, 60] postpones GC to avoid conflicts between GC and user requests. Moreover, an SSD P/E suspension mechanism [33] suspends the on-going SSD P/E to service incoming user IOs. However, these approaches could not completely remove or hide GC overhead under a long-running intensive load in clouds.

Some high-end **all flash array (AFA)** consists of plenty of SSD disks and raw flash chips with internal redundant SSD resources. There are some studies to address GC impact in AFAs by scheduling across SSD disks. SWAN [30] partitions SSDs into multiple zones to separately serve write requests and perform GC. Similarly, GC-Steering [58] partitions dedicates SSDs, called staging disks,

to absorb the host writes while the other SSDs are busy performing GC. These works focus on homogeneous-device block-level scheduling. In contrast, MIOS schedules writes upon SSD-HDD hybrid storage.

Hybrid Storage. In SSD-HDD hybrid storage, most works use SSDs as a read cache or/and write buffer [3, 9, 25, 29, 35, 48, 66], and HDDs as the secondary or backup storage [36], due to the large performance gap between SSDs and HDDs. However, the write-penalty and GC-penalty inherent in SSDs could impact the actual performance in practical intensive scenarios. Moreover, emerging NVM is supposed to replace SDDs as the primary write buffer in hybrid storage. Ziggurat [65] designs a tiered file system across NVM, DRAM, and disks. It stores small or random writes to NVM, and then coalesces many small writes into large and sequential writes to disks. Compared with these works, MIOS optimizes the latency of intensive SSD writes and updates by conditionally and directly redirects some writes to the secondary HDDs, bypassing the primary SSD buffer.

Prior works also employ HDDs as a write cache to reduce the amount of data written to SSDs [48, 61]. LBICA [1] redirects write requests from the tail of SSD queue to HDDs, preventing the SSDs from becoming bottleneck under IO bursty. SWR [38] prefers to redirect write requests with large IO size or in the tail of SSD queue to HDDs. Griffin [48] employs the write back mode by using HDDs as persistent write buffer for MLC-based SSDs. SeRW [17] uses HDDs to absorb SSD writes at high workload to reduce read/write and read/GC competition in SSDs. Besides, SSD-HDD mixed RAID [39] also has been studied to complement their disadvantages with advantages. It designs a log-structured HDD buffer to temporarily absorb small and random write requests, reducing wear-outs of SSDs. Compared with them, BCW further exploits HDD buffer to redirect synchronous small writes while avoiding performance degradation.

7 CONCLUSION

Some hybrid storage servers serve write-dominate workloads, which lead to SSD overuse and long-tail latency while HDDs are underutilized. However, our extensive experimental study reveals that HDDs are capable of μs-level write IO latency with appropriate buffered writes. It motivates us to use HDDs for offloading write requests from overused SSDs by request redirection. To this end, we present a BCW approach to proactively control buffered writes by selecting fast writes for user requests and padding non-user data for slow writes. Then, we propose an MIOS to automatically steer incoming data to SSDs or HDDs based on runtime monitoring of request queues. The multi-HDD scheduling mechanism further selects HDD with the lowest predicted latency, which achieves better performance than MIOS with single-HDD. Driven by real-world production workloads and benchmarks, our extensive evaluation of MIOS and BCW demonstrates their efficacy.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous referees for their valuable comments and helpful suggestions.

REFERENCES

[1] Saba Ahmadian, Reza Salkhordeh, and Hossein Asadi. 2019. LBICA: A load balancer for I/O cache architectures. In *Proceedings of the 2019 Design, Automation & Test in Europe Conference & Exhibition*. 1196–1201.

[2] David G. Andersen and Steven Swanson. 2010. Rethinking flash in the data center. *IEEE Micro* 30, 4 (2010), 52–54.

[3] Manos Athanassoulis, Shimin Chen, Anastasia Ailamaki, Phillip B. Gibbons, and Radu Stoica. 2011. MaSM: Efficient online updates in data warehouses. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 865–876. DOI:https://doi.org/10.1145/1989323.1989414

[4] Guillaume Aupy, Olivier Beaumont, and Lionel Eyraud-Dubois. 2018. What size should your buffers to disks be? In *Proceedings of the 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS'18)*. IEEE, 660–669.

[5] AXBOE. FIO: Flexible I/O tester. Retrieved from https://github.com/axboe/fio.

[6] Oana Balmau, Florin Dinu, Willy Zwaenepoel, Karan Gupta, Ravishankar Chandhiramoorthi, and Diego Didona. 2019. SILK: Preventing latency spikes in log-structured merge key-value stores. In *Proceedings of the 2019 USENIX Annual Technical Conference (USENIX ATC'19)*. USENIX Association, Renton, WA, 753–766. Retrieved from https://www.usenix.org/conference/atc19/presentation/balmau.

[7] Simona Boboila and Peter Desnoyers. 2010. Write endurance in flash drives: Measurements and analysis. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies*. 115–128.

[8] Daniel P. Bovet and Marco Cesati. 2005. *Understanding the Linux Kernel: From I/O Ports to Process Management*. O'Reilly Media, Inc.

[9] Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, Jaidev Haridas, Chakravarthy Uddaraju, Hemal Khatri, Andrew Edwards, Vaman Bedekar, Shane Mainali, Rafay Abbasi, Arpit Agarwal, Mian Fahim ul Haq, Muhammad Ikram ul Haq, Deepali Bhardwaj, Sowmya Dayanand, Anitha Adusumilli, Marvin McNett, Sriram Sankaran, Kavitha Manivannan, Leonidas Rigas, 2011. Windows azure storage: A highly available cloud storage service with strong consistency. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*. ACM, 143–157.

[10] Alibaba Clouder. 2018. Pangu – the high performance distributed file system by Alibaba cloud. Retrieved from https://www.alibabacloud.com/blog/pangu_the_high_performance_distributed_file_system_by_alibaba_cloud_594059.

[11] Intel Corporation. 2016. Enterprise-class versus desktop-class hard drives. Retrieved from https://www.intel.com/content/dam/support/us/en/documents/server-products/Enterprise_vs_Desktop_HDDs_2.0.pdf.

[12] Intel Corporation. 2018. Product brief of intel optane solid state drive 905P. Retrieved from https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/optane-ssd-905p-product-brief.pdf.

[13] Intel Corporation. 2019. Product brief of Intel 660p series. Retrieved from https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/660p-series-brief.pdf.

[14] Western Digital Corporation. 2019. Product brief: WD gold enterprise class SATA HDD. Retrieved from https://documents.westerndigital.com/content/dam/doc-library/en_us/assets/public/western-digital/product/internal-drives/wd-gold/product-brief-wd-gold-2579-810192.pdf.

[15] Western Digital Corporation. 2019. WD red NAS hard drives data sheet. Retrieved from http://products.wdc.com/library/SpecSheet/ENG/2879-800002.pdf.

[16] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: Amazon's highly available key-value store. *ACM SIGOPS Operating Systems Review* 41, 6 (2007), 205–220.

[17] Fan Deng, Qiang Cao, Shucheng Wang, Shuyang Liu, Jie Yao, Yuanyuan Dong, and Puyuan Yang. 2020. SeRW: Adaptively separating read and write upon SSDs of hybrid storage server in clouds. In *Proceedings of the 49th International Conference on Parallel Processing*. 76:1–76:11. DOI: https://doi.org/10.1145/3404397.3404437

[18] Diego Didona and Willy Zwaenepoel. 2019. Size-aware sharding for improving tail latencies in in-memory key-value stores. In *Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation*. 79–94.

[19] Samsung Electronics. 2017. Samsung SSD 960 EVO M.2 data sheet. Retrieved from https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/660p-series-brief.pdf.

[20] Nima Elyasi, Mohammad Arjomand, Anand Sivasubramaniam, Mahmut T. Kandemir, Chita R. Das, and Myoungsoo Jung. 2017. Exploiting intra-request slack to improve SSD performance. *ACM SIGARCH Computer Architecture News* 45, 1 (2017), 375–388.

[21] FUJITSU. 2007. MBC2073RC MBC2036RC hard disk drives product manual. 60–62. Retrieved from https://www.fujitsu.com/downloads/COMP/fel/support/disk/manuals/c141-e266-01en.pdf.

[22] Congming Gao, Liang Shi, Mengying Zhao, Chun Jason Xue, Kaijie Wu, and Edwin H.-M. Sha. 2014. Exploiting parallelism in I/O scheduling for access conflict minimization in flash-based solid state drives. In *Proceedings of the 2014 30th Symposium on Mass Storage Systems and Technologies (MSST'14)*. IEEE, 1–11.

[23] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google file system. *ACM SIGOPS Operating Systems Review* 37, 5 (2003), 29–43.

[24] Aayush Gupta, Youngjae Kim, and Bhuvan Urgaonkar. 2009. DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings.*ACM SIGPLAN Notices* 44, 3 (2009), 229–240. ACM.

[25] Qi Huang, Ken Birman, Robbert van Renesse, Wyatt Lloyd, Sanjeev Kumar, and Harry C. Li. 2013. An analysis of Facebook photo caching. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles*. ACM, 167–181.

[26] Congfeng Jiang, Guangjie Han, Jiangbin Lin, Gangyong Jia, Weisong Shi, and Jian Wan. 2019. Characteristics of co-allocated online services and batch jobs in internet data centers: A case study from Alibaba cloud. *IEEE Access* 7, 1 (2019), 22495–22508.

[27] Myoungsoo Jung, Wonil Choi, Shekhar Srikantaiah, Joonhyuk Yoo, and Mahmut T. Kandemir. 2014. HIOS: A host interface I/O scheduler for solid state disks. *ACM SIGARCH Computer Architecture News* 42, 3 (2014), 289–300.

[28] Jeong-Uk Kang, Jeeseok Hyun, Hyunjoo Maeng, and Sangyeun Cho. 2014. The multi-streamed solid-state drive. In *Proceedings of the 6th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'14)*.

[29] Taeho Kgil and Trevor Mudge. 2006. FlashCache: A NAND flash memory file cache for low power web servers. In *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*. ACM, 103–112.

[30] Jaeho Kim, Kwanghyun Lim, Youngdon Jung, Sungjin Lee, Changwoo Min, and Sam H. Noh. 2019. Alleviating garbage collection interference through spatial separation in all flash arrays. In *Proceedings of the 2019 USENIX Annual Technical Conference*. 799–812.

[31] Jaeho Kim, Kwanghyun Lim, Youngdon Jung, Sungjin Lee, Changwoo Min, and Sam H. Noh. 2019. Alleviating garbage collection interference through spatial separation in all flash arrays. In *Proceedings of the 2019 USENIX Annual Technical Conference*. Dahlia Malkhi and Dan Tsafrir (Eds.). USENIX Association, 799–812.

[32] Jaeho Kim, Yongseok Oh, Eunsam Kim, Jongmoo Choi, Donghee Lee, and Sam H. Noh. 2009. Disk schedulers for solid state drivers. In *Proceedings of the 7th ACM International Conference on Embedded Software*. ACM, 295–304.

[33] Shine Kim, Jonghyun Bae, Hakbeom Jang, Wenjing Jin, Jeonghun Gong, SeungYeon Lee, Tae Jun Ham, and Jae W. Lee. 2019. Practical erase suspension for modern low-latency SSDs. In *Proceedings of the 2019 USENIX Annual Technical Conference*. Dahlia Malkhi and Dan Tsafrir (Eds.). USENIX Association, 813–820.

[34] Anthony Kougkas, Hariharan Devarajan, and Xian-He Sun. 2018. Hermes: A heterogeneous-aware multi-tiered distributed I/O buffering system. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 219–230.

[35] Cheng Li, Philip Shilane, Fred Douglis, Hyong Shim, Stephen Smaldone, and Grant Wallace. 2014. c: A capacity-optimized {SSD} cache for primary storage. In *Proceedings of the 2014 USENIX Annual Technical Conference*. 501–512.

[36] Huiba Li, Yiming Zhang, Dongsheng Li, Zhiming Zhang, Shengyun Liu, Peng Huang, Zheng Qin, Kai Chen, and Yongqiang Xiong. 2019. Ursa: Hybrid block storage for cloud-scale virtual disks. In *Proceedings of the 14th EuroSys Conference 2019*. ACM, 15.

[37] Qixiao Liu and Zhibin Yu. 2018. The elasticity and plasticity in semi-containerized co-locating cloud workload: A view from Alibaba trace. In *Proceedings of the ACM Symposium on Cloud Computing*. ACM, 347–360.

[38] Shuyang Liu, Shucheng Wang, Qiang Cao, Ziyi Lu, Hong Jiang, Jie Yao, Yuanyuan Dong, and Puyuan Yang. 2019. Analysis of and optimization for write-dominated hybrid storage nodes in cloud. In *Proceedings of the ACM Symposium on Cloud Computing*. 403–415. DOI:https://doi.org/10.1145/3357223.3362705

[39] Bo Mao, Hong Jiang, Suzhen Wu, Lei Tian, Dan Feng, Jianxi Chen, and Lingfang Zeng. 2012. HPDA: A hybrid parity-based disk array for enhanced performance and reliability. *ACM Transactions on Storage* 8, 1 (2012), 4.

[40] Avantika Mathur, Mingming Cao, Suparna Bhattacharya, Andreas Dilger, Alex Tomas, and Laurent Vivier. 2007. The new ext4 filesystem: Current status and future plans. In *Proceedings of the Linux Symposium*. Vol. 2. 21–33.

[41] Changwoo Min, Kangnyeon Kim, Hyunjin Cho, Sang-Won Lee, and Young Ik Eom. 2012. SFS: Random write considered harmful in solid state drives. In *Proceedings of the 10th USENIX conference on File and Storage Technologies*. Vol. 12. 1–16.

[42] Subramanian Muralidhar, Wyatt Lloyd, Sabyasachi Roy, Cory Hill, Ernest Lin, Weiwen Liu, Satadru Pan, Shiva Shankar, Viswanath Sivakumar, Linpeng Tang, Sanjeev Kumar 2014. f4: Facebook's warm BLOB storage system. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation*. 383–398.

[43] Muthukumar Murugan and David H. C. Du. 2011. Rejuvenator: A static wear leveling algorithm for NAND flash memory with minimized overhead. In *Proceedings of the 2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST'11)*. IEEE, 1–12.

[44] J. Ou, J. Shu, Y. Lu, L. Yi, and W. Wang. 2014. EDM: An endurance-aware data migration scheme for load balancing in SSD storage clusters. In *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium*. 787–796. DOI: https://doi.org/10.1109/IPDPS.2014.86

[45] Mayur R Palankar, Adriana Iamnitchi, Matei Ripeanu, and Simson Garfinkel. 2008. Amazon S3 for science grids: A viable solution? In *Proceedings of the 2008 International Workshop on Data-aware Distributed Computing*. ACM, 55–64.

[46] Stan Park and Kai Shen. 2012. FIOS: A fair, efficient flash I/O scheduler. In *Proceedings of the 10th USENIX conference on File and Storage Technologies*. Vol. 12. 13–13.

[47] Raghu Ramakrishnan, Baskar Sridharan, John R. Douceur, Pavan Kasturi, Balaji Krishnamachari-Sampath, Karthick Krishnamoorthy, Peng Li, Mitica Manu, Spiro Michaylov, Rogério Ramos, Neil Sharman, Zee Xu, Youssef Barakat, Chris Douglas, Richard Draves, Shrikant S. Naidu, Shankar Shastry, Atul Sikaria, Simon Sun, Ramarathnam Venkatesan, 2017. Azure data lake store: A hyperscale distributed file service for big data analytics. In *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 51–63.

[48] Gokul Soundararajan, Vijayan Prabhakaran, Mahesh Balakrishnan, and Ted Wobber. 2010. Extending SSD lifetimes with disk-based write caches. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies*. Vol. 10. 101–114.

[49] Jens Suse. 2004. Linux block IO–present and future. In *Proceedings of the Ottawa Linux Symposium*.

[50] Arash Tavakkol, Juan Gómez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu. 2018. MQSim: A framework for enabling realistic studies of modern multi-queue SSD devices. In *Proceedings of the 16th USENIX Conference on File and Storage Technologies*. Nitin Agrawal and Raju Rangaswami (Eds.). USENIX Association, 49–66.

[51] Arash Tavakkol, Mohammad Sadrosadati, Saugata Ghose, Jeremie Kim, Yixin Luo, Yaohua Wang, Nika Mansouri Ghiasi, Lois Orosa, Juan Gómez-Luna, and Onur Mutlu. 2018. FLIN: Enabling fairness and enhancing performance in modern NVMe solid state drives. In *Proceedings of the 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA'18)*. IEEE, 397–410.

[52] Seagate Technology. 2016. Enhanced caching advantage–TurboBoost and advanced write caching. Retrieved from https://www.seagate.com/files/www-content/product-content/enterprise-performance-savvio-fam/enterprise-performance-15k-hdd/_cross-product/_shared/doc/enchanced-cache-advantage-tp691.1-1610us.pdf.

[53] Seagate Technology. 2018. Barracuda Pro Compute SATA HDD Data Sheet. Retrieved from https://www.seagate.com/www-content/datasheets/pdfs/barracuda-pro-14-tb-DS1901-9-1810US-en_US.pdf.

[54] Seagate Technology. 2019. Barracuda Compute SATA Product Manual. Retrieved from https://www.seagate.com/www-content/product-content/desktop-hdd-fam/en-us/docs/100799391e.pdf.

[55] Hua Wang, Ping Huang, Shuang He, Ke Zhou, Chunhua Li, and Xubin He. 2013. A novel I/O scheduler for SSD with improved performance and lifetime. In *Proceedings of the 2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST'55)*. IEEE, 1–5.

[56] Hui Wang and Peter Varman. 2014. Balancing fairness and efficiency in tiered storage systems with bottleneck-aware allocation. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies*. 229–242.

[57] Yeong-Jae Woo and Jin-Soo Kim. 2013. Diversifying wear index for MLC NAND flash memory to extend the lifetime of SSDs. In *Proceedings of the 11th ACM International Conference on Embedded Software*. IEEE Press, 6.

[58] Suzhen Wu, Weidong Zhu, Guixin Liu, Hong Jiang, and Bo Mao. 2018. GC-aware request steering with improved performance and reliability for SSD-based RAIDs. In *Proceedings of the 2018 IEEE International Parallel and Distributed Processing Symposium*. IEEE Computer Society, 296–305. DOI:https://doi.org/10.1109/IPDPS.2018.00039

[59] Erci Xu, Mai Zheng, Feng Qin, Yikang Xu, and Jiesheng Wu. 2019. Lessons and actions: What we learned from 10K SSD-related storage system failures. In *Proceedings of the 2019 USENIX Annual Technical Conference*. 961–976.

[60] Shiqin Yan, Huaicheng Li, Mingzhe Hao, Michael Hao Tong, Swaminathan Sundararaman, Andrew A Chien, and Haryadi S. Gunawi. 2017. Tiny-tail flash: Near-perfect elimination of garbage collection tail latencies in NAND SSDs. *ACM Transactions on Storage* 13, 3 (2017), 22.

[61] Puyuan Yang, Peiquan Jin, Shouhong Wan, and Lihua Yue. 2013. HB-storage: Optimizing SSDs with a HDD write buffer. In *Proceedings of the International Conference on Web-Age Information Management*. 28–39. DOI:https://doi.org/10.1007/978-3-642-39527-7_5

[62] 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 *Proceedings of the 11th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'19)*.

[63] Yang Yang, Qiang Cao, Hong Jiang, Li Yang, Jie Yao, Yuanyuan Dong, and Puyuan Yang. 2019. BFO: Batch-file operations on massive files for consistent performance improvement. In *Proceedings of the 35th International Conference on Massive Storage Systems and Technology (MSST'19)*.

[64] Young Jin Yu, Dong In Shin, Hyeonsang Eom, and Heon Young Yeom. 2010. NCQ vs. I/O scheduler: Preventing unexpected misbehaviors. *ACM Transactions on Storage* 6, 1 (2010), 2.

[65] Shengan Zheng, Morteza Hoseinzadeh, and Steven Swanson. 2019. Ziggurat: A tiered file system for non-volatile main memories and disks. In *Proceedings of the 17th USENIX Conference on File and Storage Technologies*. 207–219.

[66] Ke Zhou, Si Sun, Hua Wang, Ping Huang, Xubin He, Rui Lan, Wenyan Li, Wenjie Liu, and Tianming Yang. 2018. Demystifying cache policies for photo stores at scale: A Tencent case study. 284–294. DOI:https://doi.org/10.1145/3205289.3205299

Received October 2020; revised March 2021; accepted May 2021