Idler : I/O Workload Controlling for Better Responsiveness on Host-Aware Shingled Magnetic Recording Drives

Baoquan Zhang, Ming-Hong Yang, Xuchao Xie, and David H.C. Du, *Fellow, IEEE*

Abstract—Host-Aware/Drive-Managed Shingled Magnetic Recording (SMR) drives can accept non-sequential writes using a buffer called media cache. Data in the media cache will be migrated to its designated location by a cleaning process if the buffer is full (blocking cleaning) or the drive is idle (idle cleaning). However, blocking cleanings can severely extend the I/O response time. Therefore, it is crucial to fully understand the cleaning process and find ways of mitigating the caused performance degradation. In this article we further evaluate the cleaning process and propose a potential remedy scheme called *Idler* on Host-Aware SMR drives. Idler adaptively induces idle cleanings based on dynamic workload characteristics and media cache usages to reduce the severity of blocking cleanings. Our evaluations show that in the workloads with a small non-sequential write ratio (about 10 percent), Idler can reduce the tail response time and the workload finish time by 56–88 and 10–23 percent, respectively, compared with those without such control. With the help of an external write buffer on an SSD, the tail response time of SMR drives with Idler can be closer to that of conventional disk drives.

Index Terms—Storage systems, shingled magnetic recordings, tail response time, workload characterizations

1 Introduction

Shingled Magnetic Recording (SMR) is a technology to increase the areal density of spinning drives by overlapping data tracks [1]. In an SMR drive, overlapped data tracks are grouped into zones, and each zone has a write pointer to indicate the current write location. An update is considered a non-sequential write if it does not begin at the write pointer in a zone. A non-sequential write may destroy the existing content in its adjacent tracks if it writes directly in its location. To avoid impacting the overlapped data, Host-Managed SMR (HM-SMR) drives prohibit non-sequential writes. That means existing applications need to be modified if we intend to use HM-SMR drives or a log-structured file needs to be used to support applications.

Device-Managed SMR (DM-SMR) and Host-Aware SMR (HA-SMR) drives implement a Shingled Translation Layer (STL) to handle non-sequential updates. STL includes an on-device persistent cache called Media Cache and an Extent Mapping Table. A non-sequential write will be buffered in the media cache first, and later migrated to its designated location by a cleaning process. Data can be either stored at the media cache or its designated location. Any data access needs to check with the mapping table to identify its current location.

• *B. Zhang, M.-H. Yang, and D.H.C. Du are with the Department of Computer Science, University of Minnesota–Twin Cities, Minneapolis, MN 55455. E-mail: {zhan4281, yang5445}@umn.edu, du@cs.umn.edu.*
• *X. Xie is with the College of Computer, National University of Defense Technology, Changsha 410073, China. E-mail: xiexuchao@nudt.edu.cn.*

Manuscript received 5 Apr. 2019; revised 15 Nov. 2019; accepted 8 Jan. 2020. Date of publication 15 Jan. 2020; date of current version 8 May 2020.
(Corresponding author: Baoquan Zhang.)
Recommended for acceptance by H. Jiang.
Digital Object Identifier no. 10.1109/TC.2020.2966194

The cleaning process can be triggered by either an idle duration (i.e., a duration without any requests) called idle cleaning, or by depleted STL resources (e.g., media cache free space or free entry slots in the mapping table) called blocking cleaning. HM-SMR drives can only be used in workloads without any non-sequential writes like data archives and backups or with the support of a log-structured file. DM-SMR and HA-SMR drives can be deployed in many more scenarios since they both can handle non-sequential writes with the help of STL [2].

In the modern storage infrastructure, the tail response time of a system becomes significant. Applications require a low and predictable tail I/O response time [3], [4], [5], [6]. The 99.9th percentile (P99.9) response time is used as a useful statistical metric to evaluate the tail response time [4], [5], [6], [7], [8], [9]. However, HA/DM-SMR drives suffer from a long tail response time caused by the blocking cleaning. Evaluations [2], [10] show that during a blocking cleaning, an SMR drive spends up to seconds to serve a single I/O request.

Although workloads has burstiness, an SMR drive cannot rely on idle cleaning only to clean the media cache. An idle cleaning requires a relatively long idle duration to trigger and will be interrupted by any newly arrived requests. Therefore, the data cleaning efficiency of idle cleaning is degraded and blocking cleaning is still triggered. Moreover, the beginning time and the lasting duration of a blocking cleaning are unpredictable. The unpredictable performance with a long tail response time limits the usage of SMR drives to sequential workloads. Therefore, in this paper, we further evaluate the cleaning process of HA-SMR drives and explore ways of mitigating the caused performance degradation to extend the application scenarios of SMR drives.

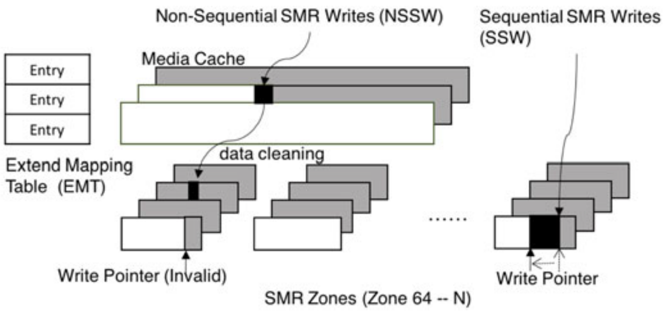


Fig. 1. Internal structure of HA-SMR drives.

We choose HA-SMR drives to evaluate since HA-SMR drives provide interfaces with Zoned Block Command (ZBC) [11] to check the current states of its internal structures. The internal information is helpful for our design and evaluations. By conducting evaluations, we find an idle cleaning triggered by a minimal idle duration has limited impacts on the drive performance since it stops cleaning once new requests are issued. While a blocking cleaning considerably degrades the drive performance and increases the I/O response time significantly.

Inspired by the different influences on the drive performance from idle cleanings and blocking cleanings, we propose a scheme called *Idler* to decrease the long tail response time of an HA-SMR drive caused by the blocking cleaning. Idler considers both the resource utilization of the media cache and the current workload characteristics to induce idle durations adaptively. With the induced idle duration, data in the media cache can be cleaned in a controllable way by idle cleanings without increasing the I/O response time significantly. It can also be implemented in the firmware of DM-SMR drives. We evaluate Idler with real-world workloads. Our evaluations show that Idler can avoid blocking cleanings in workloads with a non-sequential write ratio of 10 percent. The tail response time and the workload finish time are reduced by 56–88 and 10–23 percent respectively compared with those without such control. With the help of an external write buffer on an SSD, the tail response time of Idler can be further reduced. However, the performance of Idler is still worse than that of a conventional disk drive.

In the rest parts of the paper, Section 2 evaluates the data cleaning process. Section 3 discusses the performance improvements of artificially triggered idle cleanings. Then we present the proposed Idler in Section 4.2. We evaluate Idler in Section 5. Section 6 discusses the existing work. Section 7 offers conclusions and the future work.

2 Media Cache Cleaning and Evaluation

We evaluate the media cache cleaning using HA-SMR drives whose internal structure is shown in Fig. 1. It includes STL and SMR zones. An SMR zone is a collection of overlapped data tracks [11]. STL consists of a media cache and a mapping table. The buffered data in the media cache are log-structured since the media cache also consists of shingled tracks. Each shingled zone has a writing position called a write pointer. Any write beginning with the position pointed by the write pointer is considered as a *Sequential SMR Write* and can be issued directly to its designated location. The position of the write pointer will also be updated accordingly after a sequential write. A write with its targeted address not beginning at the write pointer is called a *Non-Sequential SMR Write*. A non-sequential write will be re-directed to media cache by STL. Its designated zone will be changed from a sequential state to a non-sequential state. The write pointer is no longer valid in a non-sequential zone. All the following write requests to a non-sequential zone will be buffered in the media cache [2].

STL utilizes a cleaning process to migrate the buffered data from the media cache to its designated locations in a Read-Modify-Write fashion [10]. A zone will be switched back to a sequential state after being cleaned. The cleaning process will be triggered once one of the resources (either free space in the media cache or free entry slots in the mapping table) is depleted [10].

In the rest of this section, we investigate how both the blocking cleaning and the idle cleaning affect the drive performance. We use *fio* to benchmark a raw HA-SMR drive (Seagate ST8000AS0022, 8TB Capacity) with direct I/O. In this drive, the size of the media cache is 28GB, and the maximum number of mapping entries is 182,000.

Fig. 2 shows the influences on the I/O response time by idle and blocking cleanings respectively. In this experiment, we use a single I/O thread. Since an ongoing cleaning increases the I/O response time and decreases the count of non-sequential zones, we use the change of the I/O response time (right y-axis) and the count of non-sequential zones (C_{NSZ}, left y-axis) to indicate when the cleaning starts. C_{NSZ} can be checked by Zoned Block Commands [11].

Fig. 2a shows the impact of idle cleanings on the I/O response time. We issue a set of non-sequential SMR writes (NSSWs) and then idle for a duration. Once we detect an idle cleaning starts, i.e., the number of non-sequential zone decreases at around 100s on x-axis, we issue more requests including non-sequential SMR writes, sequential SMR writes (SSWs), and reads. The I/O response time of the following requests is not increased. It indicates an idle cleaning has

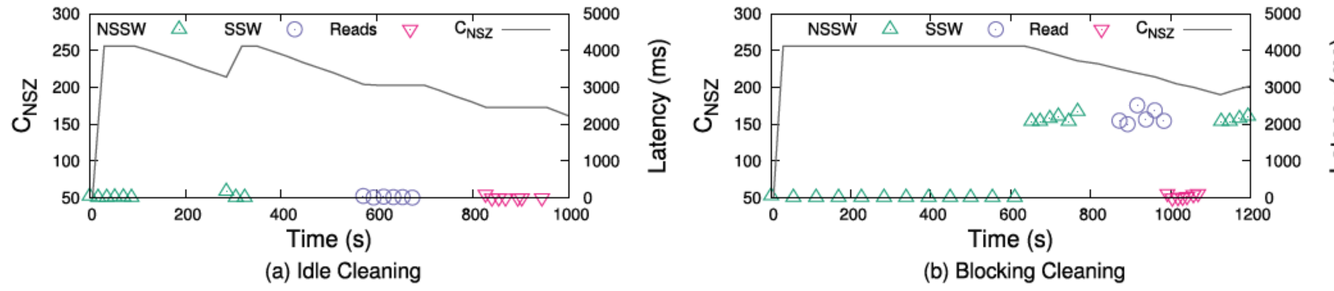


Fig. 2. Service latencies of an HA-SMR drive.

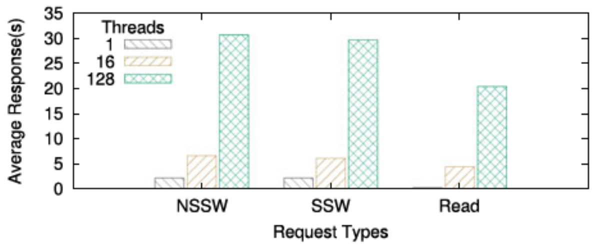


Fig. 3. The response time with IO dependencies.

TABLE 1
Time Durations to Trigger an Idle Cleaning

IO size	Idle Time	Non-sequential Zone
4 KB	240 ms	8, 8, 8
	245 ms	8, 8, 8
	250 ms	7, 6, 6
	260 ms	0, 0, 0

limited impacts on the I/O response time since it will stop if any I/O requests are issued. For the blocking cleaning (Fig. 2b), we issue a set of NSSWs to trigger a blocking cleaning. At about 650s, a blocking cleaning is triggered since the I/O response time is increased. Then we issue more SSWs, Reads, and NSSWs. The response time of the following requests is increased due to blocking cleaning. The response time of all requests after a blocking cleaning triggered is longer than that in Fig. 2a.

I/O dependencies influence the I/O response time since an I/O request may wait in the I/O queue before submitted to the device. When there is only 1 I/O thread, i.e., most of the I/O requests are dependent, subsequent requests may need the results of the previous requests. The waiting time in the I/O queue will be short because a new I/O request is only generated after the last request finishes. However, the drive may have multiple I/O threads. I/O requests from various applications are independent and submitted to the I/O queue concurrently. An SMR drive has only one disk head to serve I/O requests one by one. The I/O wait time of concurrent I/O requests will be increased since a request has to wait until the drive finishes all of the previous requests. If a blocking cleaning is ongoing on an SMR drive, the I/O wait time of a later request will be increased significantly due to the enlarged I/O response time of the previous requests. Therefore, in the next experiment, we study the influences from a blocking cleaning on the I/O response time under different I/O dependencies.

Fig. 3 shows the average I/O response time under different I/O dependencies when a blocking cleaning is ongoing. We emulate the I/O dependencies by issuing requests with varying numbers of threads. We assume only the I/O requests in the same thread are dependent. In this experiment, we first trigger a blocking cleaning by issuing enough non-sequential writes. Then we start different numbers of threads to issue I/O requests including non-sequential SMR writes (NSSW), sequential SMR writes and reads. From the results, we find that when the number of threads increases, the I/O response time are enlarged more severely. The reason is that with more independent threads, more I/O requests submitted to I/O queue concurrently. With the increased response time of previous requests, the wait time of the later I/O requests are largely increased. As a result, the total I/O response time will be increased to an extremely large value with multiple independent threads due to the enlarged I/O wait time.

An idle cleaning utilizes the idle duration to clean the media cache. It is critical to find its minimum trigger time. In this experiment, we choose 8 zones and issue one non-sequential write to each with a given idle time between two subsequent requests. After 8 non-sequential writes, we check the number of non-sequential zones right away. We repeat the experiment three times for the same idle duration.

Table 1 shows the number of non-sequential zones. It should be 8 if an idle cleaning is not triggered by this idle time. We only show the results of 4 KB IO size here even though we perform the experiments with 4 KB, 64 KB, 256 KB, 512 KB, and 1 MB I/O sizes since they have similar results. When the idle time reaches 250 ms, the number of non-sequential zones is less than 8 in all tests indicating an idle cleaning is triggered by a 250 ms idle time. Note that it takes some time to clean even one zone. The real minimum triggering time may be a bit shorter than 250 ms.

3 Improving Drive Performance With Artificially Triggered Idle Cleanings (AT-IC)

3.1 AT-IC

A blocking cleaning can largely increase the I/O response time. Although an idle cleaning has less impact on drive performance, it needs some minimal trigger time and stops with any newly arrived I/O requests. Idle cleanings may work well in a workload with burst I/O requests and relatively long idle time between bursts. However, in a workload with a short idle time smaller than minimal triggered time, idle cleanings might not even be triggered.

Inspired by the different performance influences from idle cleanings and blocking cleanings, we explore a potential concept of reducing the large tail response time caused by blocking cleanings using Artificially Trigger Idle Cleanings (AT-IC). AT-IC mitigates the performance degradation caused by a blocking cleaning with a way of invoking idle cleanings. AT-IC artificially creates idle durations when executing a workload. The I/O requests issued during an induced idle duration are stalled and served in the next execution duration. With these artificially created idle durations, idle cleanings are invoked to conduct cleaning and will not be interrupted for a fixed idle duration. With the invoked idle cleanings, a drive can clean the data in the media cache in a controllable manner which does not significantly enlarge the I/O response time.

The benefits of AT-IC is highly related to idle configurations and the characteristics of a workload. For a given workload, two configuration parameters, idle ratio and idle length, have the most significant impacts. The idle ratio means the proportion of total idle time in the duration of executing the whole workload. The idle length means the duration of each idle period. A higher idle ratio means we create longer idle time such that an idle cleaning can clean more buffered data. However, it also means that the total execution time will be longer. If the idle ratio is fixed, an

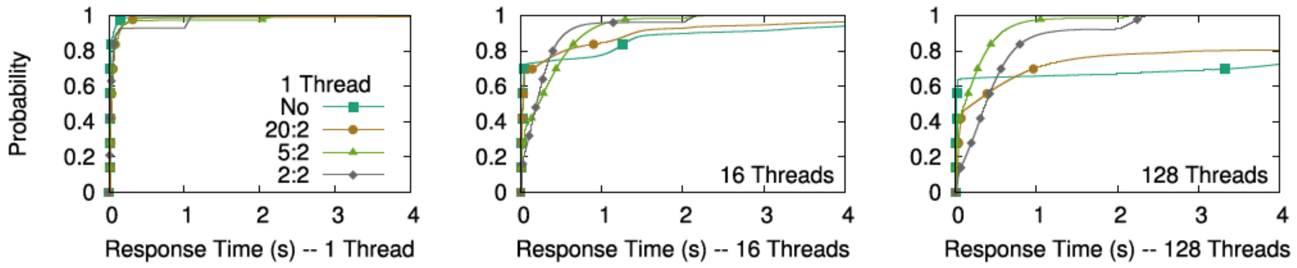


Fig. 4. CDF of the response time with different *idle ratios*.

idle cleaning can be triggered frequently with a smaller idle length or less frequently with a larger idle duration. Since I/O requests are delayed during an idle duration, with a shorter and frequent idle length, I/O requests will have a shorter response time. However, triggering idle cleanings more frequently with a smaller idle duration is inefficient since an idle cleaning needs some minimal idle duration, 250 ms based on our previous evaluations, to trigger.

Besides, characteristics of a workload including request arrival speeds, non-sequential write ratios, I/O request sizes, and the number of non-sequential zones may also influence the performance of AT-IC. The request arrival speed represents the intensity of a workload. It determines the number of requests delayed during an idle duration. A higher arrival speed makes AT-IC delays more I/O requests during a given idle duration leading to a larger I/O response time. The non-sequential write ratio represents the consuming speed of media cache resources. A higher non-sequential write ratio means we may fill up the media cache more frequently. The I/O request size may also influence the time duration to fill up the media cache. With a larger I/O request size, the media cache is filled up in a shorter duration, and blocking cleaning may be triggered by the depleted free space. While with a smaller I/O request size and a lower non-sequential write ratio, the media cache can serve for a longer time before a blocking cleaning is triggered. However, a high non-sequential write ratio and a smaller I/O request size may trigger a blocking cleaning due to the depleted mapping entries. At last, a lower number of non-sequential zones represents a shorter time to clean the media cache by either blocking cleanings or idle cleanings.

AT-IC can reduce the performance degradation caused by blocking cleanings. It makes the I/O response time more predictable and provides much better performance than that without such control. However, all of the above factors influence the effects of AT-IC. Therefore in the next subsection, we validate the performance improvement achieved by AT-IC and discuss the impacts from the idle configuration and workload characteristics.

3.2 AT-IC Validation

We implement AT-IC as a block I/O replay engine with libaio [12]. AT-IC issues block I/O requests respecting the time intervals between I/O requests. Beside, AT-IC can also periodically add a user-defined idle duration.

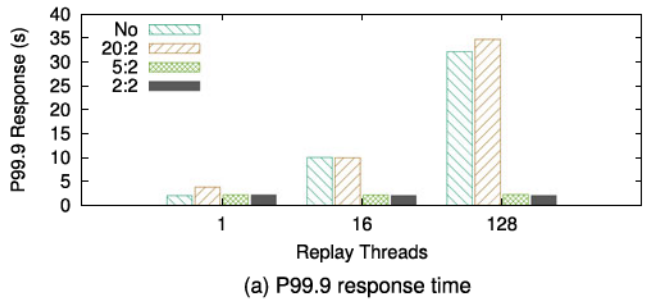
3.2.1 *Idle Configurations*

Since SMR drives are generally deployed with workloads of low non-sequential write ratios, we first evaluate AT-IC with workloads of low ratios of non-sequential writes. We vary the two parameters in the idle configuration: idle ratios and idle lengths. We synthesize traces consisting of I/O requests of 1 MB size. Here we use a large I/O size since a larger request size fills up the media cache sooner. And it will negatively influence the performance of AT-IC. However, it has less impact on the total cleaning time since it is the distance between the update offset and the write pointer position that dominates the cleaning time [2]. The request intervals are evenly distributed in the range from 10 ms to 100 ms. 10 percent of the requests are non-sequential writes with a total of 35 GB non-sequential writing size that spread over 1,024 zones. Another 10 percent of requests are sequential writes, and the remaining 80 percent of requests are reads. To study the conditions of different I/O dependencies, we partition the same trace into different numbers of subtraces (1, 16, 128). The I/O requests will be distributed evenly among subtraces based on their timestamps. Finally, we replay every subtrace with a separate thread. Only the requests within the same thread are dependent.

Idle Ratio. In this experiment, we intend to discover the influence of various idle ratios on the performance of AT-IC. AT-IC replays a workload with different idle ratios while keeping the same idle length. Here we set the idle length to 2s since it is close to the largest service latency when a blocking cleaning is ongoing on a drive. Fig. 4 shows the Cumulative Distribution Function (CDF) probability of the I/O response time with different idle ratios. The line marked with "No" means we replay the trace with an idle ratio of 0 percent. The others marked with the $Execution : Idle$ like $< 20:2 >$ means the drive serves requests for 20s and then idle for 2s and the idle ratio is about 2/22=9.09 percent. We measure I/O response time with different idle ratios under different numbers of I/O threads (1, 16, 128). We also display the tail (P99.9) response time and the workload finish time in Fig. 5.

With only 1 thread (Fig. 4), all requests are dependent. The distribution of the response time of AT-IC is similar with that of "No". The reason is that the idle length, 2s, is close to the largest response time when a blocking cleaning is ongoing. The current idle length blocks the subsequent request for a time duration same with blocking cleanings. Better results can be achieved with a shorter idle length which will be discussed later. However, the P99.9 response time with 1 thread in Fig. 5a shows that with a too-small idle ratio, .e.g $20:2$, AT-IC even increases the tail response time since it can not avoid blocking cleanings and introduces extra delay by the idle duration.

When the number of I/O threads increases (Column 2 and 3 in Fig. 4), i.e., more independent requests are issued concurrently, AT-IC can achieve a better tail response time



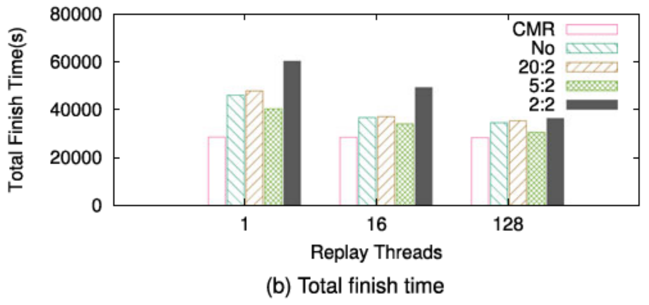


Fig. 5. P99.9 response time and total finish time of different *idle ratios*.

than no controls. The P99.9 response time of AT-IC with $< 5:2 >$ and $< 2:2 >$ is reduced by over 90 percent with 16 and 128 threads respectively compared to the results of no controls since blocking cleanings are not triggered when using AT-IC. Even though the concurrent requests wait in the I/O queue during an idle duration, they are satisfied in a short time after the idle duration. While if a blocking cleaning is triggered, the drive spends up to seconds to serve a request. The I/O response time will be increased to a huge value due to the extended waiting time, as we discussed in Section 2. The results of $< 20:2 >$ shows that with a too-small idle ratio, AT-IC only has limited benefits since it does not clean enough data. As a result, BC is still triggered.

The median (P50) response time of AT-IC is larger than that of the no control in all cases. The reason is that with fixed configurations, AT-IC is triggered from the beginning of the workload although there are still plenty available resources in the media cache. For no control, the response time will be only enlarged during a blocking cleaning. The number of requests with enlarged response time is smaller than that of AT-ICs.

Fig. 5b shows workload completion time. The results of $< 20:2 >$ indicate that with a too-small idle ratio, AT-IC can not finish the workload earlier. While as the idle ratio increasing ($< 5:2 >$), the workload completion time can be reduced by 10–15 percent compared to those without any controls. The reason is that AT-IC avoids the large response time increased by the blocking cleaning. However, if the idle ratio is too large, e.g., $2:2$, the total completion time will be enlarged due to the excessive idle duration, even though the blocking cleaning can be avoided.

Idle Length. In this experiment, we discuss the performance influences from different idle lengths while keeping the same idle ratio (about 28.75 percent). Fig. 6 shows the distribution of the response time. Fig. 7 displays the P99.9 response time and the workload finish time. Comparing the results of $< 2.5:1 >$ and $< 5:2 >$, we find that by shortening the idle length, the tail response time of AT-IC can be further reduced. However, if the idle length is too short, AT-IC may have no effects. The distribution of the response time of $< 1.25:0.5 >$ is close to that of no control. Since an idle cleaning needs about 250 ms to trigger, almost half of the idle time is wasted without cleaning data. The total finish time of $< 1.25:0.5 >$ is also larger than those of $< 2.5:1 >$ and $< 5:2 >$ as shown in Fig. 7b.

3.2.2 Workload Characteristics

We have investigated the following factors of a workload including request arrival rates (Arrival Interval), ratios of non-sequential SMR writes (NSSW ratio), counts of non-sequential zones (NSZ) with I/O sizes of 1 MB and 8 KB respectively. We only show the results of 1 MB I/O size in Table 2. The evaluations with 8 KB I/O size have similar results since the cleaning efficiency of the media cache is determined by the distance between the update offset and the write pointer location [2]. Since we have discussed the detailed results in Section 3.2.1, in this evaluation, we demonstrate the tail (P99.9) response time and the completion time of the workload (Comp. Time). In Table 2, we synthesize workloads with 35 GB total NSSW which is enough to fill up the media cache (28 GB). For each workload, we execute the workload with AT-IC ($< Execution : Idle > = < 5:2 >$) using independent 16 threads. We compare the P99.9 response time and workload finish time of AT-IC with those of no control (No).

Request Arrival Speed. We use the time interval between requests (Arrival Interval) in Table 2 to represent the request arrival rate. In this experiment, the non-sequential writes spread over 1,024 zones. We set the NSSW ratio to 10 percent (10 percent sequential writes and 80 percent read) and vary the Arrival Interval among 30 ms, 50 ms, and 100 ms. A smaller Arrival Interval means a higher request arrival rate, i.e., the workload is more intense.

In the results of different Arrival Intervals, both of the tail response time and completion time for no control become larger for the smaller Arrival Intervals. In a more intense

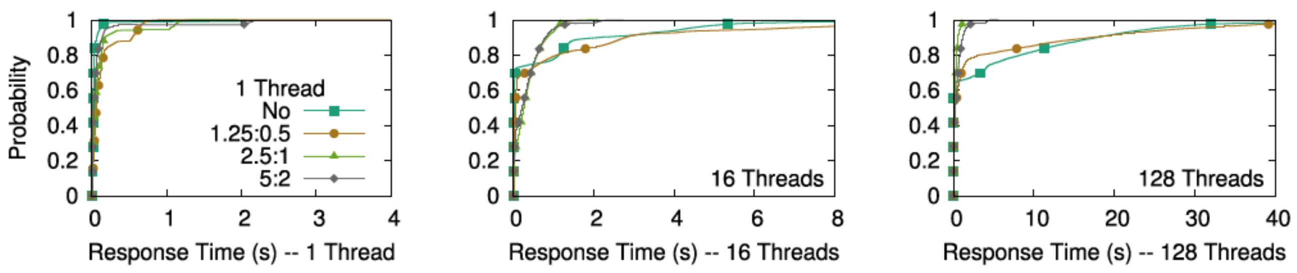
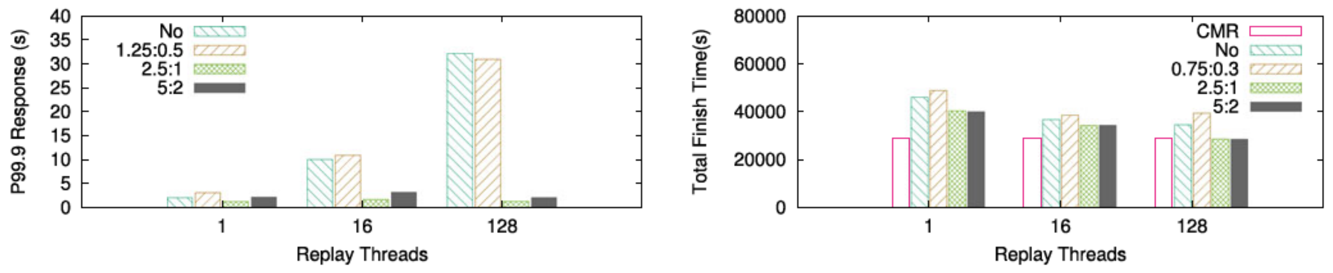


Fig. 6. CDF of the response time with different *idle lengths*.



(a) P99.9 response time (b) Total finish time

Fig. 7. P99.9 response time and total finish time of different *idle lengths*.

workload, e.g., 20 ms request arrival interval, the filling speed of media cache is faster than the cleaning speed of AT-IC. Therefore, AT-IC can not avoid blocking cleaning.

AT-IC can achieve better performance with larger Arrival Intervals. When the Arrival Interval becomes 50 ms, AT-IC can reduce the tail I/O response time and the total completion time by 72.2 and 8.1 percent respectively. When the Arrival Interval is 100 ms, AT-IC still reduces the P99.9 response time of no control by 47.1 percent. Therefore, we argue that workload intensity (Arrival Interval) can significantly influence the effect of AT-IC. AT-IC works better in light-weight workloads. With a very intensive workload, AT-IC may need to be reconfigured for a longer idle duration or a higher idle ratio to clean the media cache more efficiently.

NSSW Ratio. In the "NSSW Ratio" of Table 2, we generate workloads with different NSSW ratios (10, 20, and 40 percent). We keep the same ratios of SSW (10 percent) and Read (80, 70, and 50 percent) with 100 ms request arrival intervals. NSSW spread among 1,024 zones. In the results without control, a higher NSSW ratio results in a worse drive performance since more write requests are blocked during a BC. The service latency of a write is increased more than that of a read based on our previous evaluations (Fig. 2b). The blocked I/O requests accumulate a larger I/O response time with more non-sequential writes.

By comparing the results of no control to that of AT-IC, we find that with 10 and 20 percent NSSW ratios, AT-IC can reduce the tail response time by 50.9 and 55.3 percent respectively. The workload completion time of AT-IC is also closer to that of a CMR drive comparing with the completion time without controls. However, with a large NSSW ratio like 40 percent, AT-IC with the current configuration cannot achieve enough benefits since it can not avoid BCs. Therefore, we confirm that NSSW ratio significantly influences the AT-IC performance. AT-IC may need to be configured with a longer idle duration for workloads with higher NSSW ratios to achieve higher cleaning efficiency.

Counts of Non-Sequential Zones. In the "NSZ Count" in Table 2, we keep the workloads with the same NSSW ratio (10 percent), and request arrival speed (100 ms). However, we spread the non-sequential writes among different numbers of zones (8, 1,024, and 2,048). When the NSZ count is 8, a blocking cleaning can be finished in a short time. Therefore, the drive does not have exceptionally long I/O response times. When the non-sequential writes spread among more zones, a higher I/O response time is produced because a blocking cleaning will last longer. The drive will suffer a bad performance for a longer duration. I/O Requests thereby accumulate a longer I/O response time.

By comparing the results of AT-IC to those without control, we find that AT-IC is able to reduce the I/O response time when the requests are distributed among more zones. When the count of non-sequential zones is very small, e.g., 8, the P99.9 response time of AT-IC is more than twice of that without controls since the response time is enlarged by the induced idle duration. Therefore, for workloads with a small count of non-sequential zones, AT-IC may not be helpful. A blocking cleaning can finish in a very short time without influencing the drive performance significantly.

Conclusions. We have validated that AT-IC can significantly reduce the tail response time of HA-SMR drives. Both idle configuration and workload characteristics significantly influence the effects of AT-IC. Carefully configuring the idle length and idle ratio is required if AT-IC wants to achieve excellent performance benefits. However, the characteristics in a real workload will dynamically change from time to time. They are harder to handle by AT-IC with a fixed idle configuration. Besides, as we discussed, AT-IC with fixed configurations has a long median (P50) response time, indicating AT-IC blocks more requests than no controls. The reason is that AT-IC does not consider the current utilization of the media cache and starts idle duration from the beginning of the workloads. Therefore, in the next section, we will discuss how to improve the drive performance by using artificially triggered idle cleanings with adaptive idle configurations considering both the utilization of the media cache and workload characteristics.

TABLE 2
Workload Characteristics (1 MB I/O Size)

Value	P99.9 (s)		Comp. Time (s)	
Arrival Interval				
	No	AT-IC	No	AT-IC
20ms	12.99	14.37	7617	8832
50ms	9.22	2.56	24559	22546
100ms	4.32	2.12	40375	38861
NSSW Ratio				
	No	AT-IC	No	AT-IC
10%	4.32	2.12	40375	38861
20%	6.78	3.03	22112	19356
40%	12.32	10.22	11018	11128
NSZ Count				
	No	AT-IC	No	AT-IC
8	1.1	2.23	36037	38066
1024	4.32	2.12	40375	38861
2048	4.66	2.57	41003	39230

4 Idler: Artificially Triggering Cleanings With Adaptive Idle Durations

4.1 Basic Principle

Idler can trigger idle cleaning using adaptive idle durations based on the current media cache resource utilization as well as real-time workload characteristics. Typically, Idler needs to decide when to trigger an idle duration and how long the idle duration should be. Based on the previous evaluations in Section 3.2.2, Idler considers the following parameters: $\boldsymbol{U}$ is the media cache resource utilization including free space and mapping slots. We set two thresholds, U_{low} and U_{high} ($0 < U_{low} < U_{high} < 100\%$). $\boldsymbol{T_r}$ is the average time interval between consecutive requests. $\boldsymbol{T_{lat}}$ is the average request latency. $\boldsymbol{V_m}$ and $\boldsymbol{V_c}$ represent the consuming and cleaning speeds of media cache resources respectively. $\boldsymbol{L}$ is the length of the duration to trigger idle cleaning.

These parameters are continuously monitored, and the conditions are periodically evaluated. Both media cache space utilization and the usage of mapping entry slots are monitored and evaluated with two thresholds U_{low} and U_{high}. Since blocking cleanings can be triggered either by depleted space or mapping entry slots, U will consider the resource, space or mapping entry slots, closer to be depleted to represent the media cache utilization. When $U < U_{low}$, Idler will not trigger an idle duration since a good amount of resources are still available in media cache. When $U_{low} < U < U_{high}$, we start to trigger an idle duration. The idle duration length L is decided based on Equation (1). Since media cache still has a certain amount of resources, Idler will not trigger a long idle duration to avoid introducing a large I/O response time. We calculate the average latency of the requests T_{lat} and compare T_{lat} with the average time interval between requests T_r. If $T_r \leq T_{lat}$, it means the requests are issued relatively intensively. In this case, Idler will not trigger an idle duration ($L = 0$) since any idle duration may block a large number of requests. When $T_r > T_{lat}$, there are some time intervals between requests. In this condition, Idler triggers the idle duration with a minimal length to limit the maximal I/O response time. We set the minimum length to 0.3s here to make sure that idle cleanings will be triggered.

$$L = \begin{cases} 0, & T_r \leq T_{lat} \\ 0.3s, & T_r > T_{lat} \end{cases}. \tag{1}$$

When $U > U_{high}$, Idler has to clean the media cache more aggressively. In this condition, L is determined by Equation (2). If the workload is not intensive ($T_r > T_{lat}$), Idler will trigger an idle duration with a maximal length to achieve a higher cleaning efficiency. We set the maximal length to 2s which is equal to the largest service latency when a blocking cleaning is triggered to limit the maximal I/O response time. If $T_r \leq T_{lat}$, it means the current workload is intensive. A large number of requests will be delayed during an idle duration. To clean the media cache more efficiently and minimize the I/O response time, Idler will further compare the consuming speed V_m to the cleaning speed V_c of the media cache resource. If $V_m > V_c$, it means U will keep increasing. Idler has to increase L to achieve better cleaning efficiency. When $V_m \geq V_c$, Idler can decrease U with the current L. Therefore, Idler reduce the L to reduce the I/O response time further.

$$L = \begin{cases} L - 0.1s, & T_r \leq T_{lat} \ \&\ V_m < V_c \\ L + 0.1s, & T_r \leq T_{lat} \ \&\ V_m > V_c \\ 2s, & T_r > T_{lat} \end{cases}. \tag{2}$$

4.2 Idler Implementation

Media Cache Resource Monitoring. Monitoring media cache resources is challenging. There are no available interfaces to check the media cache resource utilization directly. The only useful interface is the *report_zone* interface in libzbc [11], [13]. It can check the states of zones (sequential or non-sequential) and the write pointer location of each zone. The *report_zone* is an intrusive command incurring a relatively large performance overhead. Therefore, it cannot be called frequently. However, we do not need the precise values of these parameters.

To reduce the performance overhead, we only estimate the media cache resource utilization U instead of getting an accurate value. Idler maintains a ghost media cache which records the total amount of consumed resources (both space and mapping entries). It also remembers the media cache resources used by each zone and the write pointer location of each zone. For non-sequential writes directed to media cache, Idler keeps updating U. For sequential writes, Idler will also update the location of write pointers. Since data is buffered in media cache with a log structure [10], repetitive updates to the same address will also be counted.

We periodically update the information in the ghost media cache by checking the current zone states using *report_zone* (every 10 minutes). By comparing the set of non-sequential zones in ghost media cache with that from the drive media cache, we identify the cleaned zones and the cleaning speed (V_c) during the last period. U can be updated by eliminating the resources used by the cleaned zones.

Workload Characterization. Idler calculates request intervals T_r, the average latency of requests T_{lat} and the consuming speed of the media cache resource V_m during the execution duration. After an idle duration, both of the blocked requests and newly issued requests will be served during the execution duration. Since the blocked requests are issued in an intensive way without following their original time intervals, the calculations of T_r and V_m are only performed after we finish all of the delayed requests. For the un-delayed requests, Idler samples 10 consecutive requests to calculate T_r, T_{lat} and V_m. T_r and T_{lat} will be counted for each new request, while V_m will be counted for only non-sequential writes.

Idle Frequency. The idle frequency is determined by both idle duration and execution duration. In Idler, the execution duration is not fixed. The length of an execution duration is long enough to serve all blocked requests from the previous idle duration plus some number of new requests such that enough new requests can be sampled to determine the next idle duration. If Idler decides not to trigger an idle duration, a new execution duration will start right away.

4.3 Integrating Idler With a Write Buffer

In many scenarios, storage systems may include a high-speed write buffer to speed up the writes. The difference between an external write buffer and the on-device media cache is that writing to the external buffer will not interrupt the data cleaning on SMR drives. Although writes to and reads from an write buffer may create idle durations, the triggered idle cleaning may not clean enough data in media cache since uncontrolled buffer evictions and buffer-missed reads will interrupt the idle cleanings. Idler can also be helpful when it is integrated with a write buffer to control the page evictions and buffer-missed reads. Besides, this write buffer also provides opportunities to reduce the I/O response time introduced by the triggered idle durations since it can temporarily hold writing requests during the idle duration. The data temporarily buffered at the write buffer will eventually migrate to SMR drives.

Designing an efficient write buffer for SMR drives is not the primary contribution of this paper. Existing studies have discussed multiple designs to write buffers for SMR drives [2], [14], [15], [16]. Our goal here is to study how Idler designed for SMR drives can work together with a write buffer in SSD to further improve the performance of SMR drives. To simplify our discussion, we make the following assumptions for a write buffer.

The write buffer is a page-based write buffer on SSD. Here we set the page size to 4 KB. A read request will check the buffer first and then go to SMR drives if they cannot be satisfied from the buffer. We assume a simple eviction policy–the Zone-based Least Recently Used (ZLRU). ZLRU remembers the access time for all of the zones who have data buffered. The victim zone will be selected by LRU. All of the buffered data from the same zone will be evicted together. We have two data paths when evicting data from the write buffer to SMR drives. If we have more than 400 pages (i.e., 1.6 MB) to be cleaned for a zone, we write the data back to its designated zone directly by read-merge-writes. Otherwise, we evict the data to the media cache. We choose 400 pages as a threshold since we assume the average service latency for a 4 KB write is about 10 ms, and it takes about 4–6s to read a zone out, reset the write pointer and write data back.

The write buffer consists of a free page pool and a page map. We load the map into memory and persist it in a reserved region on SSD. The write buffer allocates new pages from the free page pool to buffer new writes. For SMR drives, a victim zone list is maintained based on ZLRU. If not enough free pages are available, page evictions will be triggered. The reclaimed pages will be returned to the free page pool.

We integrate Idler with a write buffer in the following way. During an idle duration, the write buffer will temporarily hold both non-sequential writes and sequential writes if it has free space. Reads will not be delayed if they can be read from the buffer. At the meantime, we set up a wait queue to receive requests during an idle duration under two conditions. First, if the buffer is full and the data to be written are not in the buffer, the writes will be put into the wait queue. Second, reads will also be put into the queue if they cannot be satisfied from the buffer. In the next execution duration, we will issue the delayed reads and SSWs in the wait queue to SMR drives. The delayed NSSWs will be buffered in the write buffer after zone evictions.

The basic principle discussed in Section 4.1 is still applied. That is, both workload characteristics and media cache resource utilization will be considered when deciding the length of an idle duration. We still use the ghost media cache, as discussed in Section 4.2, to estimate the media cache resource utilization. However, instead of increasing media cache resource utilization for each non-sequential writes, we update the used media cache resource information during the eviction. If the data is evicted, we increase the media cache resource utilization accordingly in the ghost media cache. We still periodically (every 10 minutes) check the zone states by *report_zone* interface and adjust media cache resource utilization(U) and the cleaning speed (V_c).

We characterize the workload during the execution duration. The average request interval and the average request latency are counted only for the buffer-missed requests (including non-sequential writes, sequential writes, and reads) since buffer-missed requests will probably be delayed during an idle duration. When calculating the consuming speed of the media cache resource V_m, we only count the non-sequential writes that do not have a hit in the buffer. The non-sequential writes will not increase the media cache utilization if they can be served from the buffer.

TABLE 3
Characteristics of Traces

trace	write counts & sizes	zones	write%
proj_1	2,496,935 & 25.576 GB	1,475	10.56%
prn_1	2,769,610 & 30.785 GB	1,222	24.66%
usr_1	3,857,714 & 56.127 GB	1,886	8.52%
usr_2	1,994,612 & 26.469 GB	926	18.87%
proj_2	3,624,878 & 168.686 GB	1,474	12.39%

5 Idler Evaluation

We implement Idler as a block I/O replay engine and evaluate it by executing real-world traces [17]. To emulate the different I/O dependencies, we still partition the traces into different numbers of sub-traces and replay with separate threads. Again the requests in the same partitions are dependent. Precisely finding I/O dependencies from block traces is hard without extra information. HFReplayer [18] can approximately estimate I/O dependencies in block traces based on the latency information. Therefore, we use a similar algorithm when partitioning a trace.

When executing the traces, we respect the request intervals between the requests to emulate their characteristics. We run the traces on an HA-SMR drive using No Control (regular operation without idle durations), AT-IC, and Idler respectively. After we try multiple times, we configure AT-IC with $< 5 : 2 >$ since it provides a better result. For Idler, we set $< U_{low}, U_{high} >$ to $< 0.1, 0.8 >$ to allow the media cache to buffer enough data before an idle cleanings is triggered. And after U_{high} is reached, the media cache still has available resources. We also execute the traces on a Conventional Magnetic Recording (CMR) drive and compare the workload completion time.

The characteristics of the workloads are shown in Table 3 including the counts of writes, the total write sizes, counts of non-sequential zones, and the write ratio. The traces are captured from scenarios of user home directories (usr_1, usr_2), project directories (proj_1, proj_2) and print servers

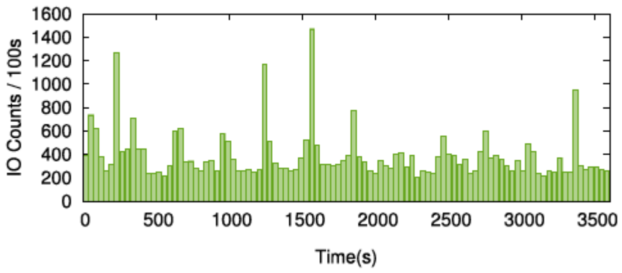


Fig. 8. The variation of the request intensity in proj_1.

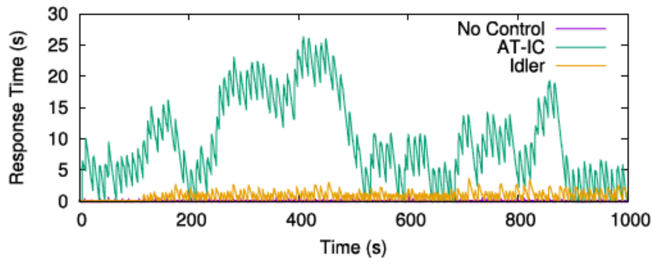


Fig. 9. Response time of proj_1 *before BC starts* with 128 threads.

(prn_1) [19]. I/O response time is significant for those scenarios. Traces are captured on a CMR drive. Almost all of the writes are non-sequential. Blocking cleanings are triggered by depleting the mapping entry slots, although the writing sizes of traces are not significantly larger than the size of media cache. During a workload, requests are not issued with a unified rate. Fig. 8 uses the trace proj_1 as an example to show the variation of I/O counts for every 100 seconds within an hour. The intensity of the workload varies a lot with time. We partition the trace into 1, 16 and 128 sub-traces to emulate different conditions of I/O dependencies.

Fig. 9 shows the response time of three designs in non-cleaning condition (the first 1,000s) of the workload under 128 I/O threads. During the first 1,000s, No Control has the shortest response time since there are no artificially triggered idle durations delaying the subsequent requests. However, with a fixed configuration, the response time of AT-IC varies a lot. When the workload is I/O intensive, the predefined execution time may not be able to finish all of the blocked requests from the previous idle duration. Thus, AT-IC accumulates a long response time. The proposed Idler can adjust its idle length and execution length dynamically. When the workload is intensive, Idler will keep the idle duration short if the media cache still has a lot of available resources. Even with a longer idle duration, Idler can make sure that the next execution duration will finish all of the blocked requests from the previous idle duration. Therefore, Idler can keep the response time in a small value (mostly under a second) even though they are still longer than those of No Control when no blocking cleaning is triggered.

Fig. 10 shows the CDF plot of the response time. We also compare the P99.9 response time and workload finish time in Fig. 11. The results of no control indicates that the blocking cleaning is triggered although the workload may have some burstiness. The idle cleanings triggered by the natural idle duration cannot clean the media cache efficiently. Idler is able to provide better response time than AT-IC and no control since Idler only triggers minimal required idle durations to avoid blocking cleanings with limited performance overheads. Under a single thread, Idler reduces the P99.9 response time and workload finish time by 56 and 23 percent compared with those of without a control. The improvement is more significant if having more independent I/O requests for Idler. The P99.9 response time is reduced by 83.2 and 88.8 percent with Idler compared to the results without control under 16 and 128 threads respectively. However, the tail response time of Idler is still not comparable with that of a CMR drive, even though the total completing time of the workload with Idler is closer, about 8–10 percent longer, to that of a CMR drive. Results in Fig. 11b indicate Idler can finish the workload in a shorter time. Since we issue the requests according to their

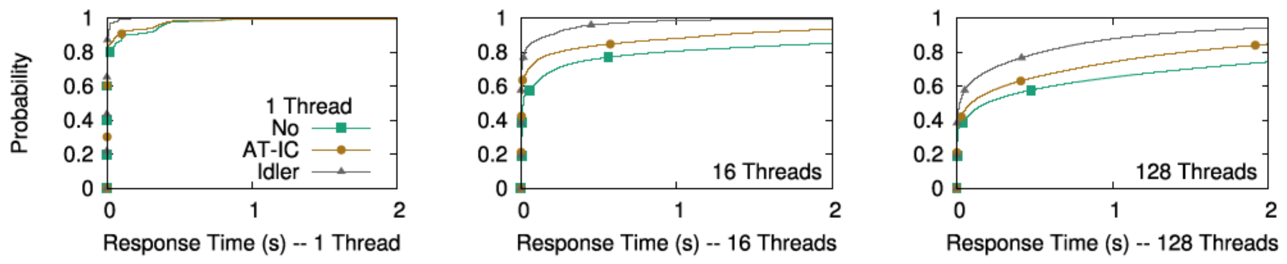
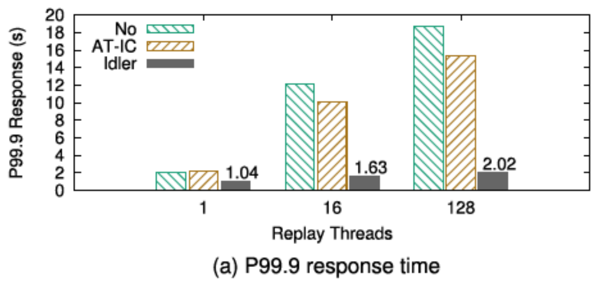


Fig. 10. CDF of the response time with *proj_1*.



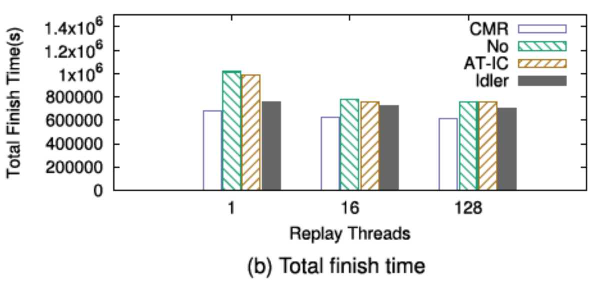


Fig. 11. P99.9 response time and total finish time of *proj_1*.

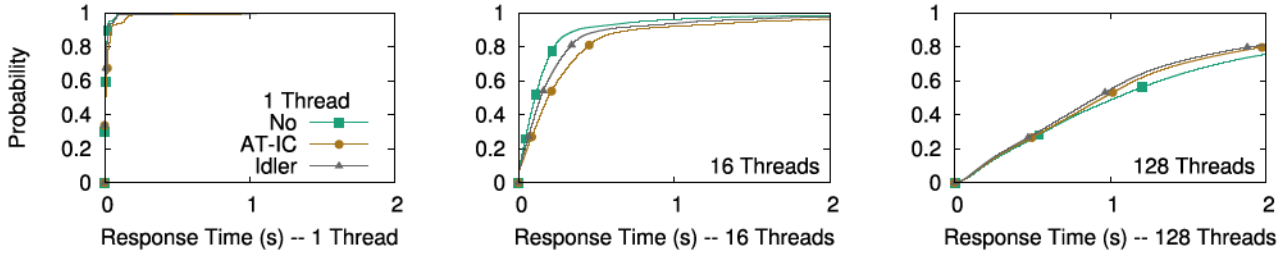


Fig. 12. CDF of the response time with *prn_1*.

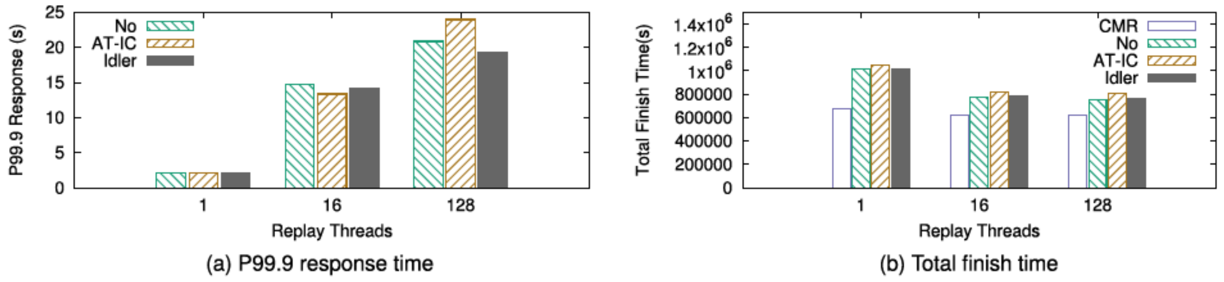


Fig. 13. P99.9 response time and total finish time of *prn_1*.

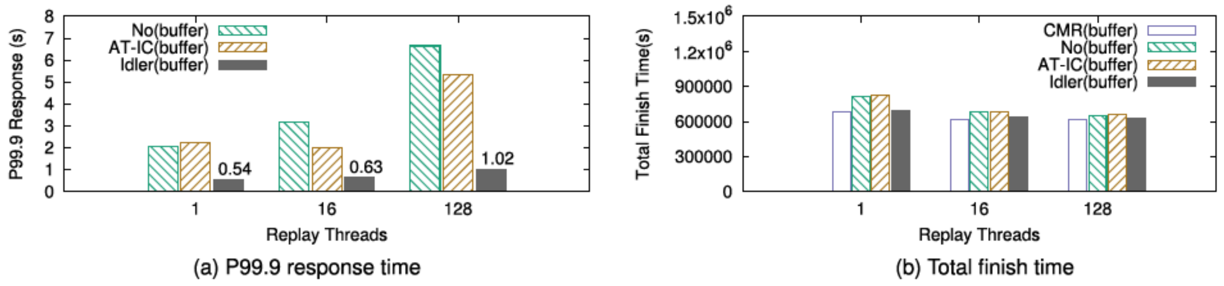


Fig. 14. P99.9 response time and total finish time of *proj_1* with a buffer.

timestamp when replay the trace and no blocking cleaning is triggered, the total finish time of Idler under different numbers of I/O threads is close to each other.

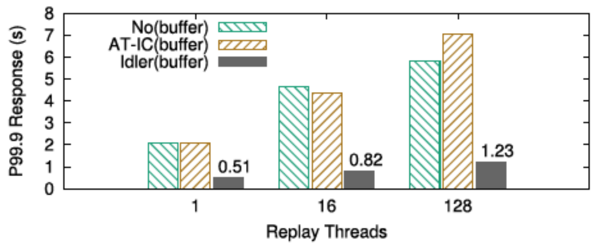
Idler needs extra memory and computing resources for buffering the delayed requests during an idle duration and updating the ghost cache information. Therefore, we measure the memory and CPU overheads of no control, AT-IC, and Idler while replaying the trace proj_1 with different numbers of threads. The maximum memory consumption is similar under different I/O threads which are about 3.9 MB (no control), 4.2 MB (AT-IC) and 5.1 MB (Idler). Idler does not introduce a significant memory overhead since it keeps the idle duration small and will not buffer too many requests. However, Idler needs some memory space to store the ghost cache information. Idler does not introduce a significant CPU overhead either since updating ghost cache information for each request only modifies several variables and *report_zone* is called infrequently. The differences in CPU utilization among no control, AT-IC, and Idler under different I/O threads are less than 10 percent.

However, the results of Figs. 12 and 13 shows that when the workload has a higher ratio of non-sequential writes (24.66 percent), both AT-IC and Idler can not help to reduce the response time. The distributions of the response time of AT-IC and Idler are similar to those of no controls. Both of the P99.9 response time and workload completion time are close to those of no controls. Therefore, we recommend that Idler can be deployed in the workloads with low ratios of non-sequential writes under 10 percent.

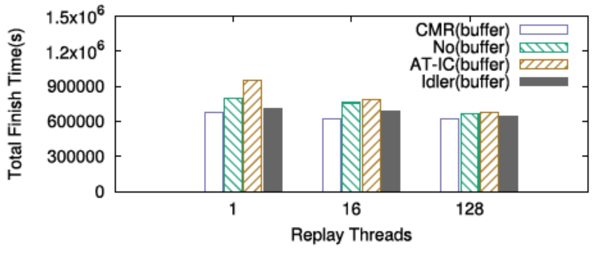
We further evaluate Idler with three more traces (usr_1, usr_2, proj_2) with 16 I/O threads. The characteristics of those workloads are shown in Table 3. Fig. 16 shows the tail response time of these three traces. The results indicate that Idler can significantly reduce the tail response time in the workloads with a small non-sequential write ratio (under 10 percent) compared with the results with no control.

Idler can reduce the response time compared to that without any control in workloads with low non-sequential write ratios. However, the response time of Idler can still be large. The P99.9 response time can be up to seconds depending on the configurations of maximal idle length. Besides, for workloads with high non-sequential write ratios, Idler can not avoid the blocking cleaning. Therefore, we further integrate the Idler with a write buffer on SSD. The buffer can absorb the non-sequential writes and temporarily hold the requests during an idle duration. To demonstrate the effectiveness, we only use a small buffer (64 MB). The performance results will have no significant differences if we deploy the buffer in host memory. When executing workloads on the CMR drive with a write buffer, we use regular LRU instead of ZLRU evictions.

Figs. 14 and 15 shows the P99.9 response time and the total finish time of Idler with a write buffer. For "No" and



(a) P99.9 response time



(b) Total finish time

Fig. 15. P99.9 response time and total finish time of *prn_1* with a buffer.

AT-IC, a write buffer cannot reduce the tail I/O response time since blocking cleaning is triggered. The page evictions and buffer-missed reads are still delayed. The results of Idler indicate that with the help of a small write buffer, the P99.9 response time can be further reduced compared to those without a write buffer in Fig. 11a. When the ratio of non-sequential writes is small (Fig. 14), the P99.9 response time can be shorter than a second. In the workloads with 10 percent ratio of non-sequential writes, the P99.9 response time is reduced by 71.3–87.7 percent. Meanwhile, Fig. 15 shows that with a write buffer, Idler can reduce the P99.9 response time by 70–85.2 percent in a workload with a higher ratio of non-sequential writes. The reason is that the write buffer absorbs a good amount of non-sequential traffics, Idler only triggers short idle durations.

6 Related Work

Most of the existing researches including file systems [20], [21], key-value stores [22], [23] and other storage systems [24], [25], [26], [27] target at HM-SMR drives. However, applications or systems have to be modified significantly or even re-designed since HM-SMR drives do not accept non-sequential workloads. Other studies also intend to help HM-SMR drive handle non-sequential workloads [28].

For DM/HA-SMR drives, existing studies focus on performance modeling and evaluations of the drives. Skylight [10] and Wu *et al*. [2] conduct evaluations on DM- and HA-SMR drives respectively to find the internal structure, cleaning policies, etc. Mhalagi develops a simulator to simulate the drive performance [29]. Shafaei also accurately models the performance of DM-SMR drives [30]. Aghayev involves ext4 in DM-SMR drives [31], [32].

Besides, there are other designs for HA-SMR drives to avoid the long response time. H-Buffer [2] and VPC [3] intend to improve the performance of HA-SMR drives by utilizing a log-structured buffer/cache on shingled zones in additions to the Media Cache. SMRC is another design using an SSD as the upper-layer cache for SMR drives [16]. Idler can also be integrated with them.

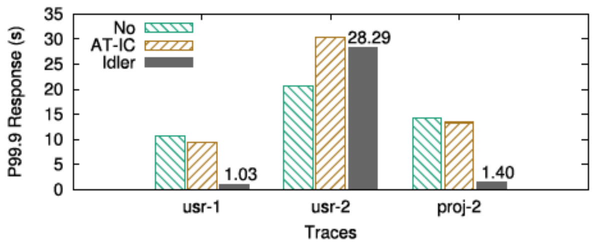


Fig. 16. P99.9 response time of different traces.

7 Conclusion and Future Work

In this paper, we evaluate the cleaning process of HA-SMR drives. We identify how data cleaning affects the drive performance. We propose a scheme called Idler. Our evaluations show that Idler can avoid the extremely long and unpredictable I/O response time in workloads with an NSSW ratio smaller than 10 percent. In the future, we would like to deploy Idler with some applications like database and key-value store to verify the performance improvement.

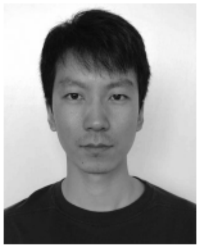
Acknowledgments

This work was partially supported by NSF I/UCRC Center for Research in Intelligent Storage and the following NSF awards 1439662, 1525617, 1536447, 1708886, 1763008, and 1812537.

References

[1] T. Feldman and G. Gibson, “Shingled magnetic recording areal density increase requires new data management,” *USENIX; login: Mag.*, vol. 38, no. 3, pp. 22–30, 2013.
[2] F. Wu, M.-C. Yang, Z. Fan, B. Zhang, X. Ge, and D. H. Du, “Evaluating host aware SMR drives,” in *Proc. 8th USENIX Workshop Hot Top. Storage File Syst.*, 2016, pp. 31–35.
[3] M.-C. Yang, Y.-H. Chang, F. Wu, T.-W. Kuo, and D. H. Du, “On improving the write responsiveness for host-aware SMR drives,” *IEEE Trans. Comput.*, vol. 68, no. 1, pp. 111–124, Jan. 2019.
[4] J. Li, N. K. Sharma, D. R. Ports, and S. D. Gribble, “Tales of the tail: Hardware, OS, and application-level sources of tail latency,” in *Proc. ACM Symp. Cloud Comput.*, 2014, pp. 1–14.
[5] M. E. Haque *et al.*, “Few-to-many: Incremental parallelism for reducing tail latency in interactive services,” *ACM SIGPLAN Notices*, vol. 50, pp. 161–175, 2015.
[6] M. Hao, G. Soundararajan, D. R. Kenchammana-Hosekote, A. A. Chien, and H. S. Gunawi, “The tail at store: A revelation from millions of hours of disk and SSD deployments,” in *Proc. 14th USENIX Conf. File Storage Technol.*, 2016, pp. 263–276.
[7] N. Li, H. Jiang, D. Feng, and Z. Shi, “PSLO: Enforcing the Xth percentile latency and throughput SLOs for consolidated VM storage,” in *Proc. 11th Eur. Conf. Comput. Syst.*, 2016, Art. no. 28.
[8] S. Kim, Y. He, S.-W. Hwang, S. Elnikety, and S. Choi, “Delayed-dynamic-selective (DDS) prediction for reducing extreme tail latency in web search,” in *Proc. 8th ACM Int. Conf. Web Search Data Mining*, 2015, pp. 7–16.
[9] T. Zhu, A. Tumanov, M. A. Kozuch, M. Harchol-Balter, and G. R. Ganger, “PriorityMeister: Tail latency QoS for shared networked storage,” in *Proc. ACM Symp. Cloud Comput.*, 2014, pp. 1–14.
[10] A. Aghayev, M. Shafaei, and P. Desnoyers, “Skylight-A window on shingled disk operation,” *ACM Trans. Storage*, vol. 11, no. 4, 2015, Art. no. 16.
[11] T. T. Committee, “Information technology—zoned block commands (ZBC),” 2014. [Online]. Available: http://www.t10.org/ftp/zbcr01.pdf
[12] Linux, “Kernel asynchronous I/O (AIO) support for Linux,” 2004. [Online]. Available: http://lse.sourceforge.net/io/aio.html

[13] A. M. Damien Le Moal, "ZBC device manipulation library," 2015. [Online]. Available: https://github.com/hgst/libzbc
[14] C.-I. Lin, D. Park, W. He, and D. H. Du, "H-SWD: Incorporating hot data identification into shingled write disks," in *Proc. IEEE 20th Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst.*, 2012, pp. 321–330.
[15] C. Wang, D. Wang, Y. Chai, and D. Sun, "Larger cheaper but faster: SSD-SMR hybrid storage boosted by a new SMR-oriented cache framework," in *Proc. 33rd Int. Conf. Massive Storage Syst. Technol.*, 2017.
[16] X. Xie, L. Xiao, X. Ge, and Q. Li, "SMRC: An endurable SSD cache for host-aware shingled magnetic recording drives," *IEEE Access*, vol. 6, pp. 20 916–20 928, 2018.
[17] M. R. Cambridge, "MSR Cambridge traces," 2008. [Online]. Available: http://iotta.snia.org/traces/388
[18] A. Haghdoost, W. He, J. Fredin, and D. H. Du, "On the accuracy and scalability of intensive I/O workload replay," in *Proc. 15th USENIX Conf. File Storage Technol.*, 2017, pp. 315–328.
[19] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," *ACM Trans. Storage*, vol. 4, no. 3, 2008, Art. no. 10.
[20] A. Palmer, "SMRFFS-EXT4," 2015. [Online]. Available: https://github.com/Seagate/SMR_FS-EXT4
[21] C. Jin, W.-Y. Xi, Z.-Y. Ching, F. Huo, and C.-T. Lim, "HiSMRfs: A high performance file system for shingled storage array," in *Proc. 30th Symp. Mass Storage Syst. Technol.*, 2014, pp. 1–6.
[22] R. Pitchumani, J. Hughes, and E. L. Miller, "SMRDB: Key-value data store for shingled magnetic recording disks," in *Proc. 8th ACM Int. Syst. Storage Conf.*, 2015, Art. no. 18.
[23] T. Yao *et al.*,"GearDB: A GC-free key-value store on HM-SMR drives with gear compaction," in *Proc. 17th USENIX Conf. File Storage Technol.*, 2019, pp. 159–171.
[24] W. He and D. H. Du, "SMaRT: An approach to shingled magnetic recording translation," in *Proc. 15th USENIX Conf. File Storage Technol.*, 2017, pp. 121–134.
[25] C.-F. Wu, M.-C. Yang, and Y.-H. Chang, "Improving runtime performance of deduplication system with host-managed SMR storage drives," in *Proc. 55th Annu. Des. Autom. Conf.*, 2018, Art. no. 57.
[26] D. Park, Z. Fan, Y. J. Nam, andD. H. Du,"A lookahead read cache: Improving read performance for deduplication backup storage," *J. Comput. Sci. Technol.*, vol. 32, no. 1, pp. 26–40, 2017.
[27] P. Macko *et al.*, "SMORE: A cold data object store for SMR drives (extended version)," in *Proc. 34th Symp. Mass Storage Syst. Technol.*, 2017.
[28] A. Manzanares, N. Watkins, C. Guyot, D. Le Moal, C. Maltzahn, and Z. Bandic, "ZEA, A data management approach for SMR," in *Proc. 8th USENIX Workshop Hot Top. Storage File Syst.*, 2016, pp. 26–30.
[29] S. Mhalagi, L. Duan, and P. Rad, "Designing and evaluating hybrid storage for high performance cloud computing," in *Proc. Annu. IEEE Int. Syst. Conf.*, 2018, pp. 1–8.
[30] M. Shafaei, M. H. Hajkazemi, P. Desnoyers, and A. Aghayev," Modeling SMR drive performance," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 44, no. 1, pp. 389–390, 2016.
[31] A. Aghayev, Y. Theodore, G. Gibson, and P. Desnoyers, "Evolving ext4 for shingled disks," in *Proc. 15th USENIX Conf. File Storage Technol.*, 2017, pp. 105–120.
[32] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, and L. Vivier, "The new ext4 filesystem: Current status and future plans," in *Proc. Linux Symp.*, 2007, vol. 2, pp. 21–33.



Baoquan Zhang is working toward the PhD degree in computer science at the University of Minnesota Twin Cities (UMN), Minneapolis, Minnesota advised by professor David H.C. Du. Currently, He is working on Memory/Storage Systems, Key-Value Stores, RAID Systems and so on. As a research assistant (June 2013—April 2015), He worked in Cloud Storage and Distributed Systems at Research Institute of Information Technology, Tsinghua University, Beijing, China.

Ming-Hong Yang received the BS and MS degrees in computer science from National Tsing Hua University, Taiwan, in 2008 and 2010, respectively. He is currently working toward the PhD degree in computer science at the University of Minnesota, Twin Cities, Minneapolis, Minnesota. His research interest includes system design for emerging storage devices and technologies such as SMR drives and storage class memories, caching policy design, protocol design for wireless networks, and software-defined networks.

Xuechao Xie received the PhD degree in computer science from the National University of Defense Technology (NUDT), China, in 2018. Currently he is an assistant professor with the College of Computer, NUDT, China. His current research interests include high performance computing, file and storage systems with non-volatile memory, solid-state drives, and shingled magnetic recording drives.

David H. C. Du is currently the Qwest chair professor in computer science and engineering at the University of Minnesota Twin Cities, Minneapolis, Minnesota and the director of the NSF Center for Research in Intelligent Storage (CRIS). His current research interest includes intelligent and large storage systems. He serves on editorial boards of several international journals. He was a program director (IPA) at National Science Foundation (NSF) CISE/CNS Division from 2006 to 2008. He has served as conference chair, program committee chair, and general chair for several major conferences in database, security and parallel processing, He is fellow of the IEEE (since 1998).