Design Issues for a Shingled Write Disk System

Ahmed Amer
Santa Clara University
Santa Clara, CA
a.amer@acm.org

Darrell D.E. Long
University of California
Santa Cruz, CA
darrell@cs.ucsc.edu

Ethan L. Miller
University of California
Santa Cruz, CA
elm@cs.ucsc.edu

Jehan-François Pâris
University of Houston
Houston, TX
paris@cs.uh.edu

Thomas Schwarz, S.J.
Universidad Católica del Uruguay
Montevideo, Uruguay
tschwarz@calprov.org

Abstract

If the data density of magnetic disks is to continue its current 30–50% annual growth, new recording techniques are required. Among the actively considered options, shingled writing is currently the most attractive one because it is the easiest to implement at the device level. Shingled write recording trades the inconvenience of the inability to update in-place for a much higher data density by a using a different write technique that overlaps the currently written track with the previous track. Random reads are still possible on such devices, but writes must be done largely sequentially.

In this paper, we discuss possible changes to disk-based data structures that the adoption of shingled writing will require. We first explore disk structures that are optimized for large sequential writes with little or no sequential writing, even of metadata structures, while providing acceptable read performance. We also examine the usefulness of non-volatile RAM and the benefits of object-based interfaces in the context of shingled disks. Finally, through the analysis of recent device traces, we demonstrate the surprising stability of written device blocks, with general purpose workloads showing that more than 93% of device blocks remain unchanged over a day, and that for more specialized workloads less than 0.5% of a shingled-write disk's capacity would be needed to hold randomly updated blocks.

1. Introduction

Disk drive capacity has grown by nearly six orders of magnitude since their introduction over fifty years ago. This growth has been fueled by advances in many areas, including recording technology, manufacturing, and materials. However, current disk drives store 400 GB/in^2, and are rapidly approaching the density limit imposed by the super-paramagnetic effect for perpendicular recording, estimated to be about 1 Tb/in^2 [29]. As a result, new approaches are needed to ensure that disk density continues to improve. Once such approach is *shingled writing*, which can achieve much higher data density with minimal changes to disk hardware and recording technology, albeit at the cost of eliminating the ability to perform random writes. This paper discusses approaches to effectively utilizing shingled disks, both as "drop-in" replacements for standard disks and in storage systems optimized to leverage shingled disks' unique access characteristics.

The elegant solution offered by shingled disks [6], [11], [30] is to use a write head with a stronger, but asymmetric, magnetic field. This approach is made possible by the fact that writes require a much stronger magnetic field than do reads. Shingled writing leverages this property by overlapping the currently written track with the previous track, leaving only a relatively small strip of the previous write track untouched. While this remnant is a fraction of the feasible write size, it is still sufficiently large to be read with current GMR read heads. As a result, shingled writing can place tracks closer together, and data density within a track can also be increased, giving a conservative estimate of density increase of about $2.3\times$ [30]. While a Shingled Write Disk (SWD) still allows for traditional random access reads, writes must be done sequentially because a single track write destroys the next k tracks, where k is typically 4–8. This radically changes the way in which the system must interact with the SWD. In addition to revised management of the SWD, effective use of non-volatile RAM (NVRAM) such as flash or newer storage class memories can further overcome any potential architectural limitations of shingled writing. With the recent availability of large-scale solid state disks, magnetic disks may well become relegated to primarily archival storage roles. SWDs would be particularly well-suited to the sequential-write, random-read access patterns that are likely in such scenarios. Nonetheless, in order to increase the chances of becoming economically viable and gaining widespread adoption, SWDs must be able to meet the traditional expectations of a random-access persistent storage device.

In the remainder of the paper, we first present an overview of the technology behind shingled writing, describing its advantages and limitations. Next, we describe the various options for data layout in shingled write disks. In particular, we consider an alternative to using NVRAM for storing data that needs to be updated in-place as well as the capacity losses introduced by *banding*—skipping tracks to allow for random writes. The subsequent sections consider some of the reasons to prefer particular options. Our paper can only lay the groundwork for some of the experimental work that needs to be done. We therefore describe the results from our analysis of traces drawn from personal laptops in general use and running specific applications. Our analysis aims to better understand whether shingled write drives are likely to encounter favorable workloads. Of particular interest is our finding that changes to device blocks are very heavily concentrated in hot zones, and that most blocks are written only once over extended observation periods.

2. Background

While shingling is a new approach to improving data density on disks, it builds on existing technology, and disks that use it must, at least initially, fit into roles filled by traditional disks. In this section, we describe approaches that disk designers are using to improve density, and the roles that disks using such technologies must fill.

The dramatic improvements in disk data density will eventually be limited by the superparamagnetic effect, which creates a trade-off between the media signal-to-noise ratio, the writeability of the media by a narrow track head, and the thermal stability of the media; Sann *et al.* call this the *media trilemma* [28]. Various approaches to this problem have been proposed; of these, shingled writing offers perhaps the most elegant and currently realizable solution, but there are other technologies that may also increase disk density.

One possible approach to address the superparamagnetic limit is to radically change the makeup of the magnetic layer, as is done in *Bit Patterned Media Recording* (BPMR) [24]. BPMR stores individual bits in lithographically defined “magnetic islands.” This approach faces several design problems, however. The most obvious is the challenge of manufacturing the surfaces themselves to have such islands. An additional non-trivial challenge, however, is the requirement that writes must be synchronized with the locations of the islands.

A second approach to increasing density involves temporarily changing the receptivity of a standard magnetic layer by “softening” the magnetic material, making it easier to magnetize. This can be done with microwaves (*Microwave Assisted Magnetic Recording*—MAMR, [34]) or by heating the writing area with a laser (*Heat Assisted Magnetic Recording*—HAMR [3], [16], [27], or *Thermally Assisted Magnetic Recording* [18]). Making the magnetic media easier to magnetize allows the use of a smaller magnetic field, and can also allow smaller areas to be magnetized, since lasers and microwaves can be focused on small areas to restrict the “softening” to a smaller region than that covered by the magnetic field.

Unfortunately, both of these approaches offer significant challenges in construction and mechanical design, and differ significantly from existing magnetic disk designs. Because of the difficulty in manufacturing such devices, disk designers have been pursuing other approaches to increase data density.

In contrast to these techniques which require radical changes to the structure of the underlying media, shingled writing builds directly upon existing magnetic recording technologies. A fundamental problem in magnetic recording is the control of the magnetic field whose flux emanates from the write head and must return to it without erasing previously written data. While perpendicular recording allows much more stable magnetization of the magnetic grains, it complicates engineering the magnetic field for writing because the flux has to enter through the recording media in order to do its desired work, but also has to return back through it to the head. In order to protect already stored data, the return flux needs to be sufficiently diffused, limiting the power that the

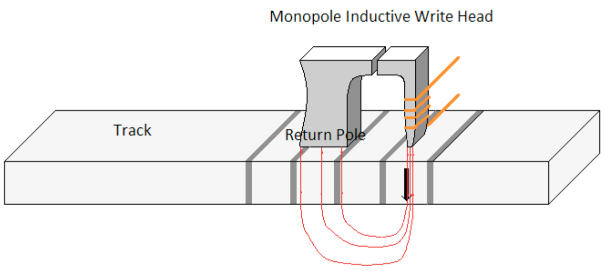


Figure 1. Operation of perpendicular magnetic recording. In contrast to longitudinal recording, the magnetic field has to enter the recording track twice.

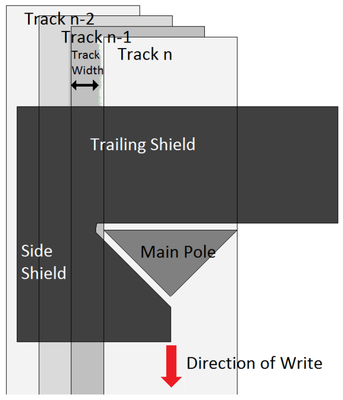


Figure 2. Corner write head for shingled writes.

magnetic field can have, as shown in Figure 1.

Shingled writing addresses this problem by allowing data in subsequent, but not prior, tracks to be destroyed during writes. Shingled writing uses a write head that generates an asymmetric, wider, and much stronger field that fringes in one lateral direction, but is shielded in the other direction. Figure 2 shows a larger head writing to track n, as used by Greaves *et al.* in their simulations [7]. Because of the larger pole, the strength of the write field can be increased, allowing a further reduction of the grain size because the technique can use a more stable medium. The sharp corner-edge field brings a narrower erase band towards the previous track, enabling an increase in the track density. Shingled writing overlaps tracks written sequentially, leaving effectively narrower tracks where the once-wider leading track has been partially overwritten. Reading from the narrower remaining tracks is straightforward using currently-available read heads. Taken together, the smaller grain size and increased track density result in an areal density increase by a factor of at least 2.5 [30] and possibly higher (3–5) according to our industry sources. Greaves *et al.* modeled shingled writing and found a maximum density of 3 Tb/in^2 [7], a nominal increase of $3\times$ over the superparamagnetic limit of 1 Tb/in^2.

Without shingled writing, avoiding interference with, or erasure of, adjacent tracks limits the maximum storage density of a device. *Two-Dimensional Magnetic Recording* (TDMR) [4], [12], [13], [28] can be used in addition to shingled writing to place tracks even closer together by changing inter-track interference from an obstacle to an instrument. TDMR reads from several adjacent tracks and uses inter-track interference to decode the signal from the target track. It uses more sophisticated signal processing [13], [33] and write encodings. In a traditional disk architecture with a single read head, reading a single sector with TDMR involves reading the sectors on adjacent tracks, requiring additional disk rotations. To avoid this problem, TDMR disks could use multiple read heads on the same slider, thus restoring traditional read service times. TDMR presupposes shingled writing, but shingled writing can be used without TDMR. In our view, the viability of Shingled Write Recording (SWR) depends mainly on integrating SWDs into current storage and computer systems, whereas much research and development is still needed to assess the viability of TDMR.

Shingled writing can be used alone or in conjunction with other new magnetic recording technologies. Shiroishi *et al.* recently proposed a possible transition path to incorporate these future technologies [29]. While perpendicular magnetic recording (PMR) reaches densities of up to 1 Tb/in^2, the next generation of technologies might use BPMR in conjunction with HAMR or MAMR and Shingled Write Recording, with a transitional use of Discrete Track Recording (DTR) as a predecessor to BPMR to reach 5 Tb/in^2. With TDMR in the mix, they see the possibility of densities of 10 Tb/in^2. The Information Storage Industry Consortium targets this density for 2015, enabling 7 TB and more in a single 2.5” disk at a cost of about \$3/TB [14], [15].

While BPMR, HAMR, and MAMR offer daunting challenges at the level of device engineering, the bulk of the challenges and opportunity for shingled writing lie at the systems architecture level. The major, but significant, functional difference of shingled writing from current disk drives is that in-place overwrites of data in a track destroy the data in subsequent tracks.

3. Integration of SWDs into Systems

SWDs are an ideal replacement for tape for backup and archival data, thanks to the largely-sequential nature of writes to these devices. Moreover, the ability of SWDs to be read randomly makes them more attractive than tapes for archival and backup. However, to be most useful and to provide a mass market, SWDs must remain capable of serving as primary random-access storage devices. Kasiraj *et al.* propose organizing the disk into *bands*, where each band stores a single file such as a large multimedia file [11]. Bands are separated by a gap of k tracks, so that a write to the last track in a band does not destroy the data in the first track of the subsequent band. While we agree that some type of banding is necessary to store the bulk of the data in a SWD, there are many ways of using banding to manage data in a SWD and other design issues to consider. The first consideration regarding banding might be whether to assign individual files or objects to dedicated bands. This is neither necessary nor optimal, since any volume of collocated data unlikely to experience updates can be grouped into a single band. Second, random updates can be supported at the block level through augmentation with NVRAM, or through the dedication of single tracks or smaller bands to random updates. NVRAM, even in small sizes, can be used to store a system log or buffer information allowing us to write in bulk. Similar functions can be taken over by areas on the SWD consisting of bands containing a single cylinder, making that region functionally identical to a traditional disk track with respect to random block updates. Third, any file system for a storage device that serves (most) write requests by appending to already written data shows similarity to log-structured file systems (LFS) [25], [26], so the opportunity to manage the SWD at the file or object level should not be ignored.

We can organize the possibilities for the SWD architecture along at least three axes:

1) We can use each single band to store a single LFS segment or we can have data in a band organized as a circular log.
2) We may potentially separate or combine metadata with user data in the same band, or we could opt for an intermediate strategy such as focusing on subparts of the metadata. For instance, we can maintain access time (atime) in a separate log, but store the rest of the metadata with the user data to which it refers.
3) We can have a single user data log (possibly with a separate single metadata log), made up of segments stored in bands or we can have many different logs. For example, we can introduce a hierarchy of x levels, where cleaning moves live data from the log at level n to the log at level $n + 1$. Logs at different levels can thereby be afforded different cleaning strategies. A different use of multiple logs is to separate data from different sources in order to colocate related data, and avoid artificial interleaving.

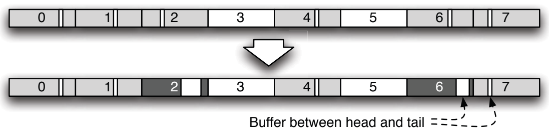
We also see two basic strategies for using SWD in current computer systems. First, we can mask the operational differences of a SWD. This can be achieved solely by adapting the layout of the device, or through a combination with NVRAM—flash, phase change memory (PCM), or other storage class memories (SCM)—as proposed by Gibson and Polte [6]. Second, we can use a stand-alone SWD, possibly with relatively little added NVRAM, and with a specialized file system or object store implementation. The first approach results in a drop-in replacement for current disks and uses a standard block interface. Opting for an object store interface allows more flexibility in the data layout and block management of an SWD because the increased semantic knowledge would implicitly include an awareness of which blocks were unused for data storage. However, a higher-level object-based interface also results in reduced flexibility of SWD deployment and usage since it would no longer be a simple disk drop-in replacement.

4. Data Layout of Shingled Write Disks

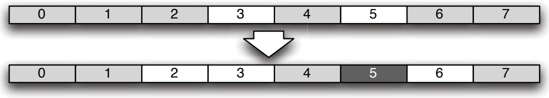
A SWD has to store the bulk of its data in bands, a collection of b contiguous tracks in which data can only be appended. Nevertheless, the disk can reserve a substantial amount of storage space for data that must be updated in-place; even reserving several gigabytes of such space would consume less than 1% of the total available disk capacity. In addition to such an area, a SWD can optionally be supplemented with some NVRAM. Currently, this would likely be flash, but other storage class memories (SCMs) such as phase change memory (PCM) might be viable in the future. Because of these factors, the size and layout of bands on disk and the mapping of data structures to bands and NVRAM are critical design decisions for SWDs.

4.1. Design of Bands

Bands consist of contiguous tracks on the same surface. At first glance, it seems attractive to incorporate parallel tracks on all surfaces (*i.e.*, cylinders) into bands. However, a switch to another track in the same cylinder takes longer than a seek to an adjacent track: thermal differences within the disk casing may prevent different heads from hovering over the same track in a cylinder. To switch surfaces, the servo mechanism



(a) Bands filled with circular logs. Bands 2 and 6 are cleaned, moving their free space buffers to the right and potentially increasing the size of the buffers by not copying "dead" blocks from tail to head.



(b) Bands treated as segments. Bands 2 and 6 are cleaned into Band 5, allowing them to be freed.

Figure 3. Two basic layout options for shingled write disks.

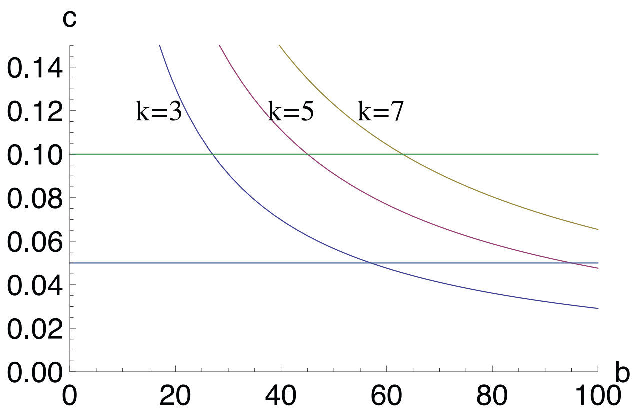


Figure 4. Proportional capacity loss c when using bands of b tracks for gaps consisting of $k = 3, 5, 7$ tracks.

must first wait for several blocks of servo information to pass by to ascertain its position before it can start moving the head to its desired position exactly over the desired track. In contrast, a seek to an adjacent track starts out from a known position. In both cases, there is a settling time to ensure that the head remains solidly over the track and is not oscillating over it. Because of this difference in switching times, and contrary to traditional wisdom regarding colocation within a cylinder, bands are better constructed from contiguous tracks on the same surface.

At the end of each band is a buffer of k or more tracks, where k is the number of tracks that are overwritten by a shingled write. Given these restrictions, Figure 3 shows the two basic layouts that SWDs may use. One option is to keep a circular log within each band and reclaim freed space by moving live data at the tail of the log to the head of the log and then considering the cleansed part free. A second option is to clean bands "atomically"; in this approach, we either write complete bands or bands or append to them until they are filled. In the first case, the circular log would need an additional buffer between its head and tail of at least k tracks. These options are discussed in greater detail in Section 4.3.

In a layout that uses bands without circular logs, the actual capacity is $b/(b+k)$ of the nominal capacity and the proportion of the capacity loss is $c = 1-b/(b+k)$. We show this proportional capacity loss c for various values of k in Figure 4. For instance, if we assume $k = 5$ and want to limit the capacity loss to 10%, then each band must contain at least 45 tracks ($b > 45$). Track capacity on current drives averages less than 1 MB, but since perpendicular magnetic recording still supports further improvements to density, we use a value of 1.5 MB, though tracks in the interior of a disk surface typically have less capacity than tracks on the outside. Given our assumptions, a band would have on average 67.5 MB, which is much more substantial than the original LFS research assumed [25], but not out of place for a modern LFS. If we try to make bands of a uniform size, 100 MB per band would still seem reasonable. As object size increases over the years and as even a small amount of NVRAM allows us the luxury of writing only full bands, concerns about the large size of bands will abate.

4.2. Combining Overwrites and Appends

Uses of a Shingled Write Disk (SWD) as a primary storage device in a system can benefit greatly from a storage area that allows in-place updates. This can be provided by augmentation with NVRAM, but we first consider providing this through a *Random Access Zone* (RAZ) in the disk itself. The RAZ consists of single tracks, each of which is followed by $k-1$ empty tracks so that the track can be overwritten without affecting other data on the disk.

4.2.1. Hardware Support for In-Place Updates. While it may be technically possible to use a different track pitch in various parts of a disk, SWDs use a much larger write head that fringes laterally in one direction, thus forcing us to use a track pitch much larger than that of a comparable "normal" disk. The resulting track density is not much higher than leaving k tracks unused following each used track in the RAZ. Moreover, integrating dynamically changing track pitch into an SWD complicates the task of mapping bad sectors and requires large changes to the servo system microcode. Thus, we believe that dynamic track pitch changes are not economically feasible. While we could lay out static regions with different track pitch to enable in-place updates, this decision would have to be made at

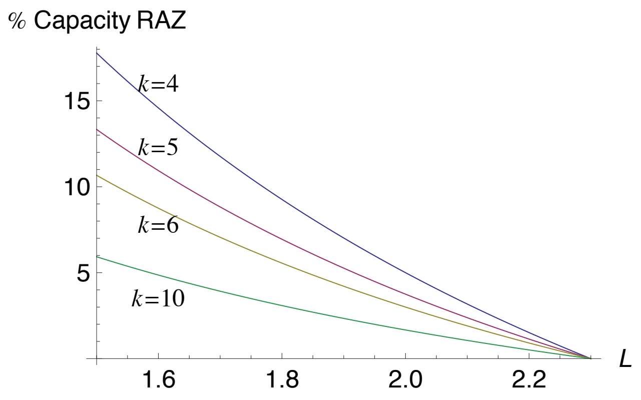


Figure 5. Proportion of Capacity used in RAZ to obtain a L-fold capacity gain for the shingled write device. We assume $\alpha = 2.3$ and $k = 4, 5, 6, 10$ top to bottom

manufacturing time and cannot be tailored to the needs of users. Instead, we favor the use of random access zones with "blank" tracks following them.

4.2.2. System Software Support for In-Place Updates. We define a random access zone by leaving k tracks unused between each two tracks in the RAZ. The rest of the disk, which we call the *Log Access Zone* (LAZ), is still organized in bands. We can think of the RAZ as consisting of bands with only a single track. Alternatively, we can think of RAZ bands as being thicker, namely $k + 1$ times thicker than LAZ tracks, and that the last track in a LAZ has the same width as the RAZ. Figure 6 shows a simple layout with two LAZ bands followed by a RAZ of 8 tracks. In reality, the number of tracks would be order of magnitudes higher. We can intersperse RAZ with LAZ, allowing us to, for example, place metadata in a RAZ close to the data in the nearby LAZ. Additionally, using cylinders instead of tracks for forming RAZ might prove advantageous.

We now calculate the impact of dividing a shingled write device into a single RAZ and a single LAZ (though in fact both may well be divided up and judiciously distributed over the disk). Call α the capacity gain of a shingled write device with LAZ only. For example, Tagawa and Williams [30] project a capacity gain of $\alpha = 2.3$, *i.e.*, using only a LAZ the capacity of a shingled write disk is 2.3 times higher than with "normal" writes. We devote a certain part of the surface, denoted by α, to RAZ. Since RAZ uses buffer space, the data carrying capacity of the disk, called L, is now smaller. These parameters are related by the formula

$$r \cdot \frac{\alpha}{k} + (1 - r) \cdot \alpha = L$$

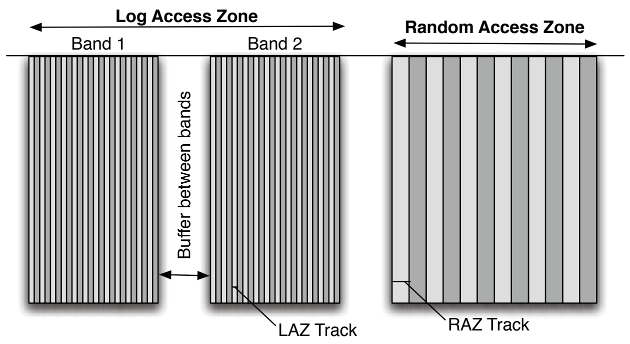


Figure 6. Schematics of Track Layout on a Shingled Write Device with a LAZ and a RAZ. The LAZ consists of two stretches of 24 tracks separated by an inter-stretch buffer and the RAZ is made up of eight tracks.

that is equivalent to

$$r = \frac{k(\alpha - L)}{(k - 1)\alpha}.$$

Given a target gain L, we can calculate the proportion of disk area r and the resulting capacity that can be devoted to RAZ We depict the results in Figure 5, which relates the percentage of data capacity devoted to RAZ based on the capacity gain L. We set the maximum capacity gain (a single LAZ) to 2.3, the value suggested by Tagawa and Williams [30]. Since k depends on the head design, we give several alternatives for its value. For example, choosing $k = 5$, a capacity gain L of 2 allows us to devote 16% of the surface area to RAZ, but the RAZ then only has 3.75% of the total storage capacity of the device. Increasing the gain to 2.2 lowers these numbers to 5.4% and 1.13% respectively. As $L \rightarrow \alpha$ these proportions approach zero, and the effects of higher values of k become less pronounced, though overall capacity drops sharply.

While the area we can devote to RAZ is proportionally small, it is quite substantial in absolute capacity as 1% of 1 TB is 10 GB. However, storing all metadata in RAZ is impossible unless files are, by current standards, exceedingly large, or unless individual blocks are large. Examples of such large files might be high resolution multimedia files or the archive files used by the Internet Archive [2]. A file system only storing full backups of machines could also use this strategy, but we doubt that such highly-specialized file systems will be the driving force behind widespread adoption.

We can break up the set of tracks making up RAZ in whatever form we want and place the tracks on the disk without any loss of capacity. If we use RAZ to store metadata or directories, we can place it close

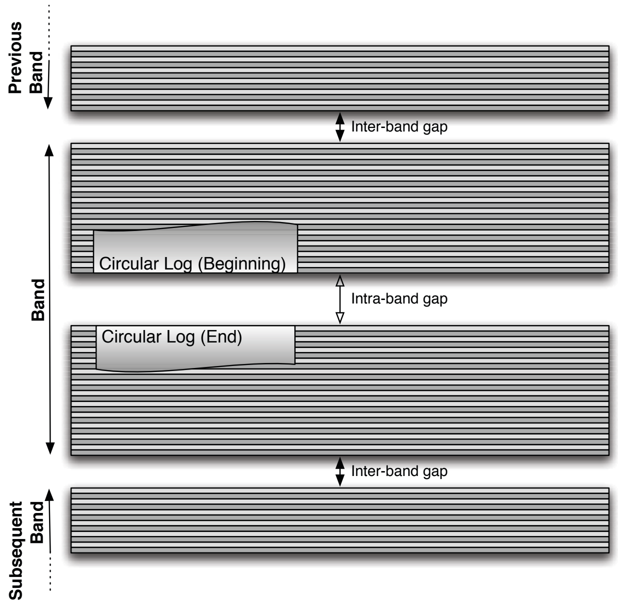


Figure 7. Layout in a Band with Circular Log

to the LAZ areas where the files themselves are being stored, in a manner quite similar to Berkeley FFS [19].

4.3. Design Space for SWD Block Layout

We can now list the design issues for block layout on shingled write devices. We have already described two methods for allowing in-place updates to some data: NVRAM and RAZ. We now discuss the tradeoffs involved in determining the best band usage, the best number of logs, and the reclamation of space.

4.3.1. Band Usage. The bulk of data in a SWD is stored in bands. A simple, elegant solution only writes complete bands that each contain a segment of a LFS. This presupposes buffering in NVRAM, but has very effective writes.

A second possibility only appends to bands. Compared to the previous solution, writes are less efficient, but both possibilities utilize the band completely.

A third possibility stores a circular log in each band. Presumably, the size of the band would be at least doubled compared to a design that cleans complete bands atomically. To prevent writes to the head from destroying data in the tail, an additional k track gap (the *intra-band gap*) between the head and the tail is (usually) necessary. Figure 7 gives the layout. To recover freed space, a cleaning operation moves live data from the tail to the head, recovering the freed space by not copying it to the head. Strictly speaking, if the log happens to start within k tracks of the beginning track, then the intra-band gap is not necessary. In some sense, this design goes back from Rosenblum's version of LFS [25] to the first proposal by Ousterhout and Douglis [23], which did not break up the log into segments.

A final possibility would use flexible band sizes. In the absence of special workloads, neighboring bands could be joined to store large objects more efficiently by using the normally unusable *inter-band* gap between bands. We can manage the bands using the "buddy principle" but if objects do not endure, would have to manage fragmentation. We do not consider this layout suitable for a general purpose SWD, but included it for completeness.

4.3.2. Number of Logs. While a mainly write-by-append device is likely to use a LFS, there is nothing that prevents it from having more than a single log. One reason for the separation is the difference in longevity between metadata and data. A particularly egregious example is the Access time (A-time) that changes whenever a file or directory is read. Some file systems only approximate A-times for simpler maintenance.

A reason to separate data into different logs is to allocate files for more efficient read access later. An example where this capacity would be vital would be a user who downloads several movies at the same time at much smaller individual download speeds than replay requires. The SWD that intersperses all movie objects because it writes them into the same log should have no problem digesting the offered input, but since the movies are interspersed, might not be able to retrieve a single movie fast enough for replay. In this scenario, the read of a single movie essentially reads all movies internally and then selects one of them for transfer into the main memory of the system.

A final reasons for multiple logs is the creation of a hierarchy of logs for cleaning. In most workloads, data that has not been deleted for some time is much more likely to remain undeleted in the future. Our log hierarchy exploits this well-known phenomenon of hot and cold data. If we clean a log, we copy the live data not to the head of the same log, but to the head of a higher level log. We expect that data in a higher level log is stable and that higher level logs therefore do not have to be cleaned at the same frequency. Only the highest level log is cleaned in the usual manner. A hierarchy of logs does not set an upper limit on the number of times that a long-living stored object is copied, but it should help to keep this number low, and result in increased write efficiency.

When using a hierarchy of logs, we can also make use of disk geography. Inner tracks on the disk surface contain less data, and objects stored there take slightly longer to access. The innermost cylinders would be best used for storing the coldest log in our hierarchy.

A draw-back of having different logs is the need to move to different locations for writes. In contrast, a single log never has to reposition the head for writes. As writes become the dominant part of the load, the performance difference of one against several / many logs may become more pronounced, and should be considered.

4.3.3. Cleaning. The introduction of a hierarchy of logs is already one decision to be taken for the implementation of the cleaning operation. A second decision concerns the amount of cleaning to be done. At one extreme, we can clean complete bands only. The alternative is a pro-rated cleaning operation that cleans partial bands. Ultimately, the amount and rate of cleaning required will depend heavily on how rapidly data is updated on the device, and how well we can identify data that is more likely to be updated.

5. Further Design Considerations

To realize the benefits of using shingled writing involves both the management of the device and the nature of the workload such systems will experience. We discuss workload properties in Section 6, but will now focus on possible system design characteristics. Specifically, we discuss the possibility and potential of combining NVRAM, the special aspects of engineering a "drop-in" block-level storage system based on a shingled write device, data structures, and the problems and opportunities of an object-level interface.

5.1. The Best Use of NVRAM

A SWD can benefit from the appropriate use of NVRAM technologies, whether it be used to replace or to supplement the random-update functionality supported by the layout, *e.g.,* in the form of the RAZ. NVRAM is currently almost exclusively flash memory. Several efforts are under way to replace flash memory and we can expect other SCM technologies such as PCM to become more cost effective and more efficient. Even NVRAM implemented in NAND flash has different performance, reliability, number of sustainable write-erase cycles, and price. Currently (December 2009), appropriate NVRAM costs about 20 to 25 times more per GB than disks, but this ratio continues to fall. We can assume that the introduction of shingled writing alone will not significantly increase the price of disks. While the density of data due to shingled writing at least doubles, the density of RAZ is $1/(1+k)$ of that of the raw SWD. If we assume $k = 3$, we can conservatively expect the density of a RAZ in a SWD to be half that of a comparable disk. This should still give a clear price advantage of about 10 to a disk that employs a RAZ. Once newer storage class memories make it to market, they will exceed the performance and reliability of flash at a reduced price. The widely superior performance of flash (or SCM) together with the lower costs of RAZ will offer interesting, workload dependent trade-offs between RAZ and NVRAM.

We now consider how much NVRAM might be useful. First, we can use NVRAM to buffer incoming data in order to always write full bands. Since bands are small (they should not exceed 100 MB), this capacity requires little additional NVRAM, but using NVRAM only for the purpose of buffering writes requires attention to the limited number of write-erase cycles of current flash memory.

In the Write Anywhere File System (WAFL) [8] NVRAM is used to maintain a log of all file system activity. This use of NVRAM allows us to do away with the need to add metadata to each physical write. Incidentially, the eponymous placement strategy of WAFL shows that colocating related data is not essential for good read performance. A further advantage of logging system activities is the capacity of fast recovery from system crashes.

We can use NVRAM to store recently created objects. In most storage workloads, the longevity of a block of data or of whole objects is not a memoryless property. Rather, a block of data that survives for a given period (such as an hour or a day) is disproportionally more likely to survive the next period. Using NVRAM to store a few hours or days worth of incoming data likely reduces the amount of non-live data in a SWD considerably and with it the necessity for cleaning. If we assume that we write first to NVRAM, then decisions on the placement of data in the SWD are not taken at ingress time. This opens up possibilities of using a placement strategy without sacrifying write efficiency.

Finally, we can use NVRAM to store all or some metadata. Metadata tends to have a higher amount of activity in all categories (reads, writes, updates) per GB stored making it an ideal candidate for special treatment.

Calculating an exact amount of NVRAM useful for SWD devices is impractical, but a "back of the envelope" calculation yields a few 100 MB in order to write complete bands, maybe 1 – 2% of the total storage capacity in order to buffer blocks and objects on ingress, a few GB to maintain a file system log, which totals maybe 30 GB for a 2TB SWD. The exact values would be workload dependent, but the lower the rate and percentage of updated blocks, the more that can be achieved with smaller amounts of NVRAM.

5.2. A SWD-based Block Level Storage System

For shingled write recording to find widespread adoption, a SWD needs to be able to function within existing systems without major changes to other system components. Economic reasons make it unlikely that different magnetic recording technologies will be used only for specialized components suited for certain workloads. This means that SWD need to function as a "drop-in" replacement for current hard drives, even though that market is potentially shrinking or shifting due to the advent of Solid State Disks. One useful change would be to augment the basic device interface with a command that tells the device that the system considers a given block to be free (rather than just eventually overwriting the block) Such a command is also very useful to designers of solid state drives or flash devices in general. The knowledge can be used in a SWD to clean more efficiently as the data need not be preserved. The same command allows a flash based device to more efficiently prepare an erase block for erasure, as only live pages in an erase block would need to be copied.

A principle problem for a general block-based file system using a SWD device is contiguity, as most file systems spend effort on placing related information in contiguous Logical Block Addresses (LBA). In the append-only write mode of SWD, these efforts will fall short leading potential degradation. Throughput to a selection of completely random blocks is about three orders of (decadic) magnitude slower than that to contiguous blocks in a track. While a heavily edited object might not see such a heavy access degradation, we can certainly imagine a workload such as very large database tables with frequent edits of small records where the performance loss could become visible. A similar case might be the intermingling of objects created simultaneously and over time such as the concurrent downloads of movies or other large files. Predictive data structures are able to separate these objects so that they can be placed contiguously at least after a first round of cleaning. Using a large NVRAM could contribute to a solution of this problem. While it acts as a type of cache, hit rate on reads is the wrong metric, as large objects can be accessed faster if they are stored on magnetic disk rather than on flash. In fact, we prefer to cache objects likely to be soon edited.

5.3. Data Structures for a Block Device

To our knowledge, no research exists on data structures with properties suited to a banded SWD. For instance, the work on cache-oblivious data structures (*e.g.*, [1]) uses random access devices for all members of the storage hierarchy. Such research exists for optical WORM (*e.g.*, [31]) and flash memory (*e.g.*, BFTL, [32], $B+$-tree(ST), [21], FD-trees [17], μ-trees, [10], LSM trees, [22]), and some of its ideas might prove useful for SWD.

A device driver for a block structured storage device made up of NVRAM and SWD needs to be able to quickly translate Virtual (Logical) Block Addresses (VBA) to Logical Block Addresses on the SWD. We assume that the NVRAM comes with its own driver that, in the case of flash provides, wear leveling. We then need to implement a VBA to LBA translation table. The size of the table depends on the block size S, the capacity C, and the length of an entry, which needs to accomodate the total number of blocks, *i.e.*, is C/S. For a block size of 4KB and a capacity of 2TB, the number of entries is 2^{29}. If we change the block size to $1/2$KB, then the number of entries is 2^{32}.. This number just allows us to store each entry in 4B, but probably any design needs to accomodate additional device capabilities. Thus, if each entry is a generous 8B (to also accomodate flags), the size of the translation table is 4GB for 4KB blocks and 32GB for $1/2$KB blocks. The latter should probably not be stored completely in NVRAM for cost reasons.

The simplest way to implement such a translation table is through a B+-tree type structure. However, unlike a B+-tree, the structure need not be dynamic and can have a fixed number of levels. As there is no direct overwriting of data in the NVRAM nor the SWD, a change to a leaf involves changing all nodes from the leaf to the root, *i.e.* we have a "wandering" tree [9]. If we store the top level(s) always in NVRAM, where appending to a single page is possible, we can use Easton's construction [5] of updatable nodes. A node is stored in complete pages. If we delete an entry in a node, we append a delete record at the end of the node, and if we want to update an entry, we append the new entry to the node. Thus, entries within a node are not ordered. To find an entry, the complete node needs to be read and evaluated, which makes this data structure only efficient for NVRAM. (Reading a complete band of 35 tracks in SWD takes 35 or more rotations, or about 140 msec with a rotation speed of 15000 rotations per minute.) Alternatives or supplements use a transaction log as proposed by Nath, [21], or by Hunter, [9], or add a tree in NVRAM that contains recently changed entries. These auxiliary structures can be mirrored in RAM for fast access.

In order to allow cleaning, we need to select auspicious bands for cleaning. This also necessitates keeping track of blocks stored in SWD and now overwitten. In particular, we need to be able to easily tell which data in a SWD band is live and which is not. A simple method is to associate each of the 10^4 to 10^5 bands (of about 100 MB each) with a band descriptor maintaining the status of all the blocks in the band in a bitmap. This map takes about 1GB of space (or 1/4096 of the storage capacity of the drive using 512B blocks). Efficient implementation of the bitmap makes use of the expected locality of changes to the bitmap by maintaining a log or a write cache so that we need rarely change an individual entry in flash or SWD.

5.4. An Object-level Interface

An object store built with an SWD simplifies several design issues. In general, object stores can leverage the knowledge they have about which blocks are related for better allocation of storage locations. First, objects are deleted as a whole, which simplifies cleaning as it prevents already-freed data to be copied and as it gives better information on which bands are good candidates for cleaning. Second, it is much easier to use several logs to store separate objects that arrive at the same time separately. This capacity is critical when read access needs to be optimized. In the case of large objects which fill up whole bands, whole bands are freed all at once. Third, a storage object comes with a well defined, though flexible set of metadata including security information, which enables the designer of the interface between the file system and the SWD to decide whether to store all or some metadata separately from the storage object.

If the goal of 7 TB disks by 2015 is achieved, the usual file system interface for the user might become difficult to use. If SWDs serve several computing systems as secondary storage, an object store with its clearer solution to security, sharing, and authentication could prove to be the simpler design.

6. Experimental Results

The efficacy of a SWD as a replacement for a general purpose disk, and the need for more or less NVRAM or RAZ capacity, is heavily dependent on the workload encountered by the disk. Specifically, the rate at which individual blocks are updated is of particular interest. If relatively few blocks are updated, this would imply a lesser need for RAZ and NVRAM space to render a SWD usable in place of a conventional disk.

For the workloads we have evaluated, our results indicate that blocks are indeed rarely updated. This in turn suggests that a very limited use of RAZ or NVRAM would make SWD a successful replacement for current disks. The modest 1 to 3% capacity overheads of a RAZ we discussed above may be sufficient to mask the majority of block updates. While some workloads would seem perfectly suited to a SWD, particularly workloads with minimal updates to previously written data such as archival workloads, we evaluated workloads typical of general purpose personal usage of a disk, as well as specialized workloads drawn from a system being used to edit video, and a third system dedicated to managing a music library. We present results from all but the last experiment, which was found to have negligible block update events over a period of almost a month. This was not surprising as the update of a block is the re-writing of its contents, and we do not consider the initial writing of data to a block as an update event. This meant that the regular addition of media files to the library did not incur any updates beyond the negligible updates to the library metadata. Our general purpose and video editing workloads showed noticeable block update behavior, but nonetheless this remained restricted to a very small percentage of all blocks and supports the general usefulness of a SWD device.

6.1. Workload Description

The main general-purpose trace sets evaluated were drawn from laptop and personal computers running Mac OSX, and were gathered from November to December, 2007. The results presented here are from a Mac laptop running Mac OSX 10.4 with the filesystem formatted for HFS+ with journaling enabled. The workload is typical for personal workstations or laptops and consists mainly of web browsing, file editing, and code compilation. It also features personal media management including video, image, and audio libraries. While there were momentary trace interruptions due to full system reboots, including one due to a major software update, these were infrequent and brief enough as to be negligible. Any changes in device behavior due to the major software update would be typical for such occurrences. The traces include block update behavior generated by both user file manipulation and system file updates. Our data only reflects device-level requests, but not requests satisfied by a cache, and is therefore reflective of what would be experienced by the individual disk device. The requests

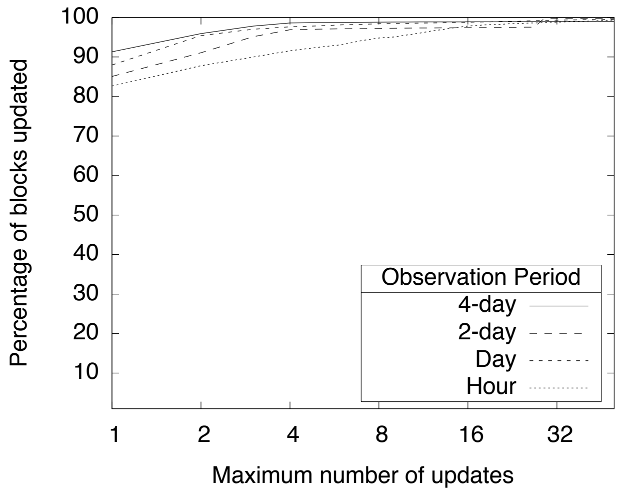


Figure 8. The effect of time on the distribution of block update rates.

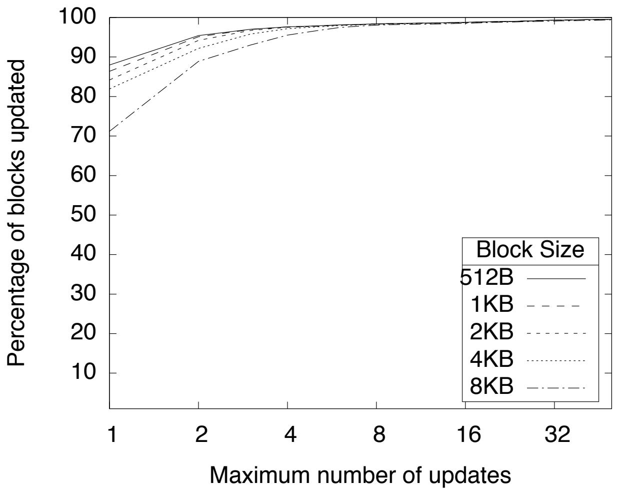


Figure 9. The effect of selected block size on update rates.

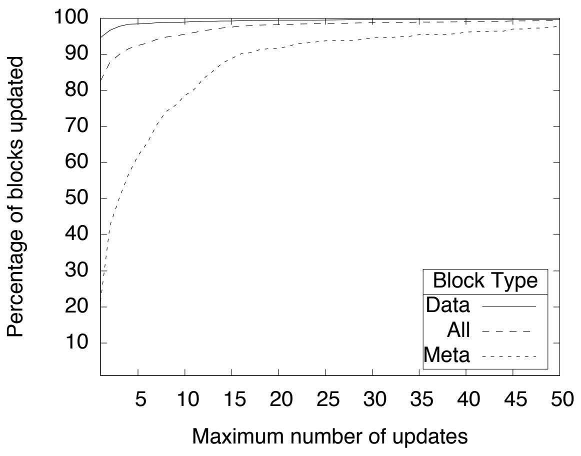
are in the simple form of block reads and writes. They also include requests resulting from paging activity. The "video editing" workload was gathered in January of 2010, on a system running Mac OSX 10.5.8, and also using a filesystem formatted for HFS+ with journaling enabled. The workload was gathered over an approximately three hour period during which two videos were edited and transcoded, each composed of multiple video files edited to form the final videos. The workload also included the transcoding of an hour-long video file. All block updates were recorded, and were not limited to those generated by the movie editing suite.

6.2. Results

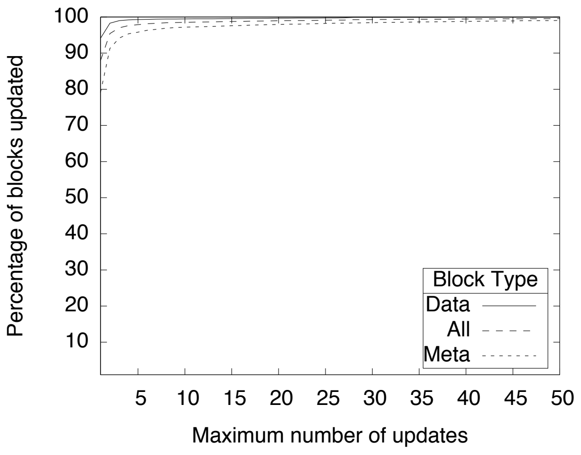
The number of times that a given block is updated grows with the observation period. Interestingly enough, we found that a decreasing percentage of written blocks were written multiple times. In other words, we observe a very small percentage of hot blocks being rewritten, whereas a growing percentage of written blocks is written only once. For example, just under 94% of all accessed disk blocks have undergone four or fewer updates. Figure 8 gives our result. The x-axis gives the maximum number of updates and the y-axis gives the percentage of blocks updated. We give four curves, one for an observation period of an hour, a day, two days, and finally four days. We see that more than 85% of all disk blocks written were never updated within the hour and 93% of all disk blocks written were never updated within a day. This trend continues as the observation period lengthens. This suggests that the reclamation rate of data stored in a LAZ is very low. It also suggests that if data is stored in NVRAM or a RAZ until it reaches a certain age (*e.g.*, an hour or a day), then the vast majority of multiple updates to already written disk blocks are masked.

In Figure 9, we see the impact of varying the tracked block size on the percentage of updated blocks for a one-day trace. This was done by replaying the trace, and tracking a different block size for each run. We formed larger blocks by combining adjacent sectors and counted an update to any constituent sector as an update to the larger block. With larger blocks, it is increasingly likely that a larger percentage of blocks will be updated, requiring relocation, reclamation or invalidation. What makes larger block sizes desirable is the ability to reduce overheads imposed by any data structures used in a data layout scheme. While the negative impact of larger block sizes is noticeable, it is important to note that it quickly becomes negligible for any blocks updated more than 2 to 4 times. While multiple updates, however infrequent, may still drive the need to reclaim, relocate, or invalidate blocks upon update, these numbers do support the argument that blocks demonstrating infrequent updates can quickly be identified as stable. Such a quick classification can better inform the relocation of data from RAZ or NVRAM to stable, shingled, tracks.

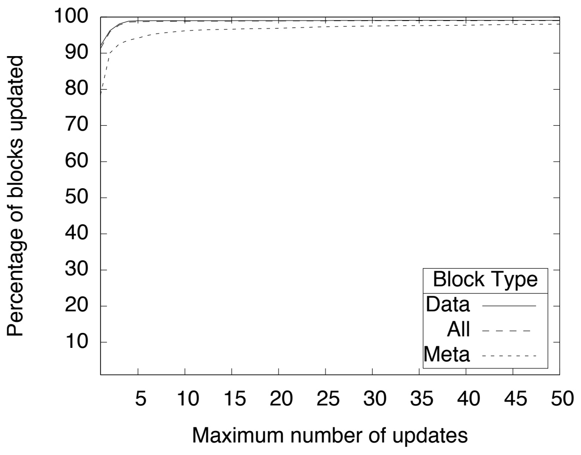
Figure 10 shows the impact of differentiating disk blocks that hold filesystem metadata from those that hold user data. This distinction showed the most marked difference in the percentage of stable disk blocks. Specifically, blocks containing metadata are consistently more likely to endure multiple updates than data blocks. The difference is significantly more pronounced for shorter observation periods, which



(a) One Hour



(b) One Day



(c) Four Days

Figure 10. The impact of block-type separation on update rates, as observed for different periods.

supports the potential for a small amount of NVRAM, or RAZ, to absorb the bulk of updates that occur over

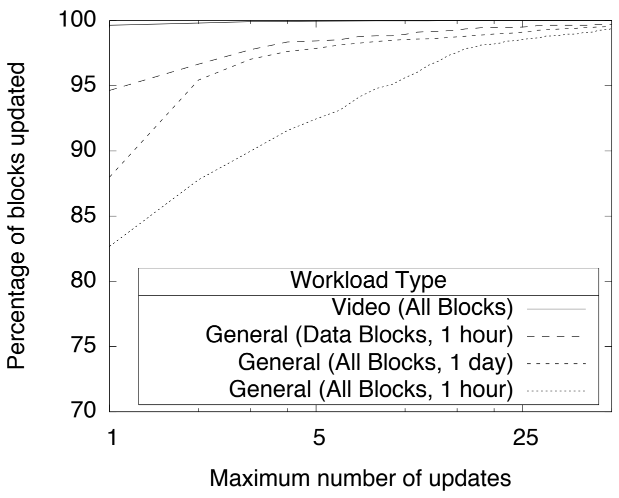


Figure 11. The impact of favorable workloads. In this instance, the video editing workload demonstrates much greater block stability than general purpose usage. This is true even when we observe the general purpose workload for a longer period, or limit our focus to blocks holding user data.

such shorter time periods, particularly if the blocks selected for caching are metadata blocks.

We can exploit this result in one of two basic ways: implementing shingled layout optimizations at the filesystem or object level; or attempting to automatically classify block types based on their behavior. Object and file-level approaches that can separate metadata blocks from data blocks [20] have demonstrated the benefits of such semantic knowledge on improving the utilization of hybrid storage systems. A similar result can be achieved through the automated classification of device blocks, and is a topic we plan to address in future work. In either case, it would then become feasible to assign metadata and data blocks to different regions of the disk, to the appropriate log segment or circular buffer, or to the appropriate block on disk or in NVRAM. Interestingly, while the difference in short-term behavior is more ponounced, there remains a pronounced difference over the long term. Metadata blocks remain more likely to experience multiple and regular updates, whereas data blocks tend to persist as originally written.

In Figure 11 we see the dramatic effect of a specialized application on the observed device workload. The figure shows the percentage of blocks updated for both the general purpose workload and the more specialized video editing workload. The general-purpose workload was in no way modified to accommodate our SWD model, was drawn from computers acting as their users' primary systems, and yet demonstrated an encouragingly low rate of block updates. The percentage of updated blocks drops as the observation period grows, but the effect of focusing solely on blocks holding user data ("Data Blocks" in the Figure) is even more pronounced. When we consider the "Video" editing workload, we see a dramatically reduced percentage of updated blocks. This demonstrates the additional benefits to be gained by pairing a SWD with a complementary application. While suitable for general purpose use, it would appear that video editing, and media library applications are practically ideal. In this instance we see less than 0.4% of blocks experiencing content updates during the entire editing session (approximately three hours of heavy use). As we mentioned above, further experiments with an audio library produced results that simply offered no notable block update behavior over the entire trace collection period. This was due to the fact that problematic block updates are only those that would result in a change of previously written data, and in that environment, such events were almost nonexistent.

Our workload analysis allows us to reach the following conclusions for the general purpose device workloads considered:

1) Keeping track of updates to block allows us to identify hot blocks. The volume of hot blocks is small enough to allow us to efficiently allocate them to a RAZ or NVRAM of minimal capacity.
2) Tracking larger block sizes diminishes accuracy in the determination of hot blocks, but the effect is negligible. We can further reduce layout metadata by indexing larger blocks within bands and zones.
3) A file system that allows the device to distinguish metadata from user data can gain from the ready and efficient identification of hot blocks. This suggests the benefits of object-stores based on SWDs, the automated classification of block types, or the sharing of block type information with the block device driver.

7. Future Work and Conclusions

The aim of this article was the exploration of the design space for systems software support of SWDs. While SWDs still need to be designed and prototyped, enough is known about the technology to develop designs and test against workloads. As always, selecting good workloads is difficult, as the workload of tomorrow's devices is going to be different from today's devices, but this is a common and not insurmountable obstacle. Thus one of our future goals is the simulation of many of the ideas proposed in this paper and testing them against existing and newer workload traces. A primary tasks in assessing the viability of shingled writing will be the implementation of a block-level device driver using a simulated SWD. Another avenue for future work is developing and testing the automated classification of stable blocks, and testing heuristics for deciding when best to move device blocks from a RAZ to a LAZ. The objective of these efforts would be to improve the prospects of SWDs being viable as drop-in device replacements.

Acknowledgment

We would like to thank John Palmer of IBM Research, Almaden for helpful and enlightening discussions. Thomas Schwarz was supported by a grant of the Stiles Family Foundation. Ethan Miller and Darrell Long were supported by the National Science Foundation under grants CNS-0917396 (part of the American Recovery and Reinvestment Act of 2009 [Public Law 111-5]) and IIP-0934401, and by the industrial sponsors of the Center for Research in Intelligent Storage (CRIS) and the Storage Systems Research Center (SSRC).

References

[1] M. A. Bender, E. D. Demaine, and M. Farach-Colton, "Cache-oblivious B-trees," *SIAM Journal on Computing*, vol. 35, no. 2, pp. 341–358, 2005.

[2] M. Burner and B. Kahle, "Arc file format," available at http://archive.org/web/researcher/ArcFileFormat.php.

[3] W. A. Challenger, C. Peng, A. Itagi, D. Karns, Y. Peng, X. Yang, X. Zhu, N. Gokemeijer, Y. Hsia, G. Yu, R. E. Rottmayer, M. Seigler, and E. C. Gage, "The road to HAMR," in *Asia-Pacific Magnetic Recording Conference (APMCR '09)*, 2009.

[4] K. S. Chan, J. Miles, E. Hwang, B. V. K. Vijayakumar, J. G. Zhu, W. C. Lin, and R. Negi, "TDMR platform simulations and experiments," accepted by IEEE Transactions on Magnetics, 2009.

[5] M. Easton, "Key-sequence data sets on indelible storage," *IBM Journal of Research and Development*, vol. 30, no. 3, May 1986.

[6] G. Gibson and M. Polte, "Directions for shingled-write and two-dimensional magnetic recording system architectures: Synergies with solid-state disks," Carnegie Mellon University Parallel Data Lab, Tech. Rep., May 2009, CMU-PDL-09-014.

[7] S. Greaves, Y. Kanai, and H. Muraoka, "Shingled recording for 2–3 Tbit/in^2," *IEEE Transactions on Magnetics*, vol. 45, no. 10, pp. 3823–3829, Oct. 2009.

[8] D. Hitz, J. Lau, and M. Malcolm, “File system design for an NFS file server appliance,” in *Winter 1994 USENIX Conference*, 1994.

[9] A. Hunter, “A brief introduction to the design of UBIFS,” available at http://www.linux-mtd.infradead.org/doc/ubifs_whitepaper.pdf.

[10] D. Kang, D. Jung, J.-U. Kang, and J.-S. Kim, “Mu-tree: an ordered index structure for NAND flash memory,” in *EMSOFT ’07: Proceedings of the 7th ACM & IEEE International Conference on Embedded Software*, 2007, pp. 144–153.

[11] P. Kasiraj, R. New, J. de Souza, and M. Williams, “System and method for writing data to dedicated bands of a hard disk drive,” United States Patent 7490212.

[12] A. R. Krishnan, R. Radhakrishnan, and B. Vasic, “LDPC decoding strategies for two-dimensional magnetic recording,” in *IEEE Global Communications Conference*, 2009.

[13] A. R. Krishnan, R. Radhakrishnan, B. Vasic, A. Kavcik, W. Ryan, and F. Erden, “Two-dimensional magnetic recording: Read channel modeling and detection,” in *IEEE International Magnetics Conference*, May 2009.

[14] M. H. Kryder and C. S. Kim, “After hard drives – what comes next?” *IEEE Transactions on Magnetics*, vol. 45, no. 10, pp. 3406–3413, Oct. 2009.

[15] M. Kryder, “After hard drives – what comes next?” in *Proceedings of the IEEE International Magnetics Conference*, May 2009.

[16] M. Kryder, E. Gage, T. McDaniel, W. Challener, R. Rottmayer, G. Ju, Y.-T. Hsia, and M. Erden, “Heat assisted magnetic recording,” *Proceedings of the IEEE*, vol. 96, no. 11, pp. 1810–1835, Nov. 2008.

[17] Y. Li, B. He, Q. Luo, and K. Yi, “Tree indexing on flash disks,” in *ICDE ’09: Proceedings of the 2009 IEEE International Conference on Data Engineering*, 2009, pp. 1303–1306.

[18] K. Matsumoto, A. Inomata, and S. Hasegawa, “Thermally assisted magnetic recording,” *Fujitsu Scientific and Technical Journal*, vol. 42, no. 1, pp. 158–167, Jan. 2006.

[19] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry, “A fast file system for unix,” *ACM Transactions on Computer Systems*, vol. 2, no. 3, pp. 181–197, 1984.

[20] E. L. Miller, S. A. Brandt, and D. D. E. Long, “HeRMES: High-performance reliable MRAM-enabled storage,” May 2001, pp. 83–87.

[21] S. Nath and A. Kansal, “FlashDB: dynamic self-tuning database for nand flash,” in *IPSN ’07: Proceedings of the 6th international conference on Information processing in sensor networks*, 2007, pp. 410–419.

[22] P. O’Neil, E. Cheng, D. Gawlick, and E. O’Neil, “The log-structured merge tree (LSM-tree),” *Acta Informatica*, vol. 33, no. 4, pp. 351–385, 1996.

[23] J. Ousterhout and F. Douglis, “Beating the I/O bottleneck: a case for log-structured file systems,” *SIGOPS Operating Systems Review*, vol. 23, no. 1, pp. 11–28, 1989.

[24] H. Richter, A. Dobin, O. Heinonen, K. Gao, R. Veerdonk, R. Lynch, J. Xue, D. Weller, P. Asselin, M. Erden, and R. Brockie, “Recording on bit-patterned media at densities of 1 Tb/in^2 and beyond,” *IEEE Transactions on Magnetics*, vol. 42, no. 10, pp. 2255–2260, Oct. 2006.

[25] M. Rosenblum, “The design and implementation of a log-structured file system,” Ph.D. dissertation, UC Berkeley, 1992.

[26] M. Rosenblum and J. Ousterhout, “The design and implementation of a log-structured file system,” *Operating Systems Review*, vol. 25, no. 5, pp. 1–15, Oct. 1991.

[27] R. E. Rottmeyer, S. Batra, D. Buechel, W. A. Challener, J. Hohlfeld, Y. Kubota, L. Li, B. Lu, C. Mihalcea, K. Mountfiled, K. Pelhos, P. Chubing, T. Rausch, M. A. Seigler, D. Weller, and Y. Xiaomin, “Heat-assisted magnetic recording,” *IEEE Transactions on Magnetics*, vol. 42, no. 10, pp. 2417–2421, Oct. 2006.

[28] C. K. Sann, R. Radhakrishnan, K. Eason, R. M. Elidrissi, J. Miles, B. Vasic, and A. R. Krishnan, “Channel models and detectors for two-dimensional magnetic recording (TDMR),” submitted, http://www.ece.arizona.edu/˜vasiclab/Projects/DataStorage/Papers/TDMRKheong.pdf.

[29] Y. Shiroishi, K. Fukuda, I. Tagawa, S. Takenoiri, H. Tanaka, and N. Yoshikawa, “Future options for HDD storage,” *IEEE Transactions on Magnetics*, vol. 45, no. 10, Oct. 2009.

[30] I. Tagawa and M. Williams, “High density data-storage using shingle-write,” in *Proceedings of the IEEE International Magnetics Conference*, 2009.

[31] J. S. Vitter, “An efficient I/O interface for optical disks,” *ACM Transactions on Database Systems*, vol. 10, no. 2, pp. 129–162, 1985.

[32] C.-H. Wu, T.-W. Kuo, and L. P. Chang, “An efficient B-tree layer implementation for flash-memory storage systems,” *ACM Transactions in Embedded Computer Systems*, vol. 6, no. 3, p. 19, 2007.

[33] Y. Wu, J. O’Sullivan, N. Singla, and R. Indeck, “Iterative detection and decoding for separable two-dimensional intersymbol interference,” *IEEE Transactions on Magnetics*, vol. 39, no. 4, pp. 2115–2120, July 2003.

[34] J.-G. Zhu, X. Zhu, and Y. Tang, “Microwave assisted magnetic recording,” *IEEE Transactions on Magnetics*, vol. 44, no. 1, pp. 125–131, Jan. 2008.