Design Issues of Shingled Write Disk for Database Table Implementation

Soror Sahri
Université Paris Descartes
Laboratoire d'Informatique Paris Descartes
45 Rue des Saints Pères
75270 Paris, France
Soror.Sahri@ParisDescartes.Fr

Thomas Schwarz, S.J.
Universidad Católica del Uruguay
Informática y Ciencias de la Computación
Av. 8 de Octubre 2738
11600 Montevideo, Uruguay
TSchwarz@Calprov.org

***Abstract*—To maintain the continuing growth of bit density in magnetic recording media, the disk industry will have to change technologies. Shingled write disks are expected to be the next generation of high capacity magnetic disks and already in prototype. Shingled write technology is not disruptive at the level of disk design and manufacturing, but as shingled writes prevent updates in place, the technology is disruptive at the level of usage. It is possible to design a disk device driver or disk firmware that allows a shingled write disk to be used as a drop in replacement for traditional disks. Database implementations however have traditionally bypassed the file system and accessed the disk directly in order to achieve better performance. We discuss here adaptation of B$^+$-trees and linear hash tables to shingled write disk to support indexed database tables and secondary indices. Our proposal is based on dividing the disk in low-capacity Random Access Zones (RAZ) and high capacity Log Access Zones (LAZ). The LAZ use the shingled disk effectively while RAZ places guard bands around each track in the zone in order to regain the capacity of in-place updates at the costs of loosing capacity.**

I. Introduction

Disk drive capacity has grown from 5MB for the IBM RAMAC 305 in 1956 to 1TB disks in 2013. To continue this growth rate, the current design of disks needs to change. The superparamagnetic effect for perpendicular recording stands at about 1 Tb/in^2 [50]. To overcome this boundary, one can change the medium or one can change the strength of the magnetic field. Solutions using both approaches are actively sought and the likely road map for the industry includes both, including Bit Patterned Media Recording (BPMR) and Heat (or thermally) Assisted Magnetic Recording (HAMR). Even before the introduction of these changes which will transform the design and manufacturing process of magnetic drives, we will most likely see the introduction of shingled write disks. Seagate is starting to ship shingled write disks in 2014. The write head in a shingled write disk has a stronger magnetic field that is assymetric, focused in one direction and diffused in the other. Writing with the assymetric write head destroys data in tracks parallel to the one being written but only in one direction. A shingled write disk overlaps the currently written track with the previous track, leaving only a relatively narrow strip of the previous write track untouched. While this remnant is only a fraction of the written track, it is still sufficiently wide to be read with current GMR heads. The overall results are tracks placed closer together. Since a stronger magnetic field is used, the data density inside a track is also higher. The combination of higher track and higher bit density in a track gives a density increase estimated conservatively to be at least 2.3 times [51]. Both BPMR and HAMR are expected to use shingled writing.

In comparison with the development and manufacturing impact of BPMR and HAMR, the current design and manufacturing process for shingled write disks does not change dramatically. The main burden of their introduction will be on the user of these devices. Since current disks allow writes at a random location, a direct drop-in replacement of a current disk by a shingled write disk is not possible. A possible solution could use a block translation layer that gives the view of a traditional disk, but reassigns dynamically logical block addresses to physical block addresses. This is similar to a Flash Translation Layer (FTL) used for flash based devices to mask the peculiarities of the read-erase-write cycle in flash and present an interface in which individual pages are read and over-written. More likely is a combination with Non-Volatile Random Access Memory (NVRAM) such as a flash-based solid state disk, possibly packaged as a single device, a "combo-disk". Using NVRAM as a large cache, only cold data needs to be stored on the shingled write disk.

Historically, database implementers have circumvented the file system in order to obtain the performance needed for transaction support. The work of Sears et al. shows that access to binary objects stored in a database is better if the object size is less than 256 KB and access for objects through the file system is better for objects larger than 1 MB [48]. There is no reason for us to assume that this difference is going to change. Databases just do not generate a "normal" load for which file systems are designed. The development of storage technology with larger RAM, the introduction of intermediate Non-Volatile Random Access Memory (NVRAM) such as flash and in the future Storage Class Memories (SCM) is pushing many database products towards using disks not at all or only as an archival medium. Often, the limiting factor for a database is not the storage capacity of a disk but its I/O bandwidth, which has not grown at the same rates as capacity.

doi:10.4304/jcp.9.10.2247-2257

We can say colloquially that a database needs actuators, not platters. As database applications and sizes widely differ, we assume that a small, but in absolute numbers large group of databases will still continue to use magnetic disk because of their attractive GB per dollar ratio. Many other will migrate towards in-memory and using NVRAM, especially SCM once they become available.

In this paper, we investigate the design changes necessary for databases wishing to use a shingled write drive with little or no support from NVRAM. We argue that databases as applications will still continue to use the shingled drive as a raw device and should use specialized data structures in order to do so. We propose an adaptation of the B$^+$-tree and of linear hashing to shingled writing. Of course, a database needs the disk for much more than just storing tables. Other needs such as the creation and management of temporary tables or logging are as efficient using shingled write disks as they are for current disks. In fact, since shingled write disks do not allow as much fragmentation, we can even expect that writing and reading large, temporary tables or writing and reading logs is more efficient in shingled write disks. Because of time and space constraints, we have to postpone an experimental evaluation of the proposed data structures for future work.

II. Shingled Write Technology

With the adoption of NAND flash technology for storage as Solid State Disks (SSD), the role of magnetic disks is slowly changing towards a more archival role as the medium of choice for bulk storage. However, for the near future, the bulk of stored information will be magnetically recorded on hard disks because of their high data density and their low cost. The disk drive industry is striving to maintain the high rate of annual increase in areal density of 30% and recently of 40% per year [16]. Further increase (from 500 Gb/in^2 to beyond 1Tb/in^2) will soon be limited by the super-paramagnetic 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. Chan et al. call this the *media trilemma* [8], following the practice at Seagate. Various approaches to overcome the media trilemma have been proposed; of these, shingled writing offers a solution that can be implemented without solving major technological obstacles. In any case, the current predictions for the evolution of magnetic drive technology presume shingled writing first and then in conjunction with other methods of increasing data density [50].

One possible approach to address the super-paramagnetic limit is to radically change the makeup of the magnetic layer, as is done in *Bit Patterned Media Recording* (BPMR) [43]. BPMR stores individual bits in lithographically defined "magnetic islands," essentially densely stippled protrusions. In addition to the challenge of manufacturing surfaces with such islands, writes need to be synchronized with the location of the islands.

A second approach to increasing density temporarily changes 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, [59]) or by heating the writing area with a laser (*Heat Assisted Magnetic Recording*— HAMR [6], [30], [46], or *Thermally Assisted Magnetic Recording* [38]). A temporary softening of the magnetic media allows the use of smaller magnetic fields, which in turn allows a smaller area to be magnetized as the softened region can be smaller than that affected otherwise by the magnetic field.

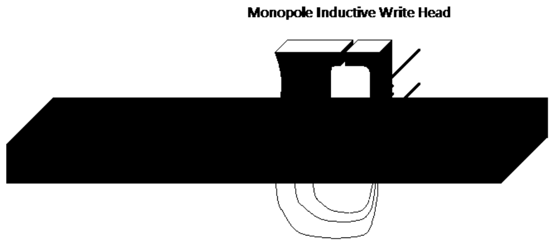

Fig. 1: Operation of perpendicular magnetic recording. In contrast to longitudinal recording, the magnetic field has to enter the recording track twice.

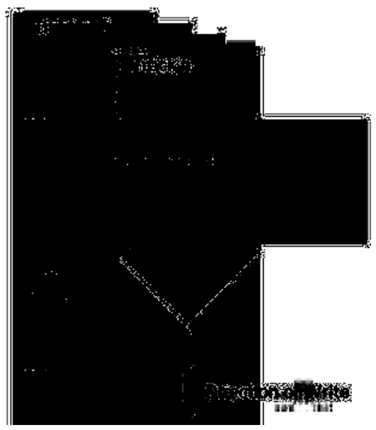

Fig. 2: Corner write head for shingled writes.

Both approaches offer significant construction and manufacturing challenges and require significant changes in current magnetic disk design.

While early hard drives used an electromagnet for writing and reading, separate read heads using Magneto-Resistance (MR) and Giant Magneto-Resistance (GMR) came into use starting in 1996 and 2000, respectively. Write heads need to control the magnetic field whose flux emanates from the head and needs to return to it without erasing previously written data. While perpendicular recording (Fig. 1) allows much more stable magnetization of the magnetic grains (and hence higher data density), the flux not only has to enter through the recording media in order to do its desired work, but also has to return back through the media to the head. In order to protect already stored data, the return flux needs to be sufficiently diffused, which limits the power that the magnetic field can have.

A weaker magnetic field allows a good focus of the flux and protects previously written data well, but lacks strength to magnetize permanently. A stronger magnetic field magnetizes permanently, but needs larger track widths to protect adjacent data. 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, but 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 [21]. Because of the larger pole, the strength of the write field can be increased, a more stable (but harder to write) magnetic medium used, which together allows a further reduction of the grain size. 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 with current technologies. Taken together, smaller grain size and increased track density result in an areal density increase by a factor of at least 2.5 [51] 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 [21].

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 is that in-place overwrites of data in a track destroy the data in subsequent tracks.

With or without shingled writing, we need to avoid erasing data on adjacent tracks when writing (*inter-track interference*, and this limits the density of tracks. *Two-Dimensional Magnetic Recording* (TDMR) [7], [26], [27], [8] turns this inter-track interference from an obstacle into an instrument. Using more sophisticated signal processing [27], [57] and write encoding, TDMR reads from several adjacent tracks and decodes the signal from the target track taking account of inter-track interference. 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. While shingled write disks already exist in prototype, 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 and colleagues recently proposed a possible transition path to incorporate these future technologies [50]. 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. This set of technologies could 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 [28], [29].

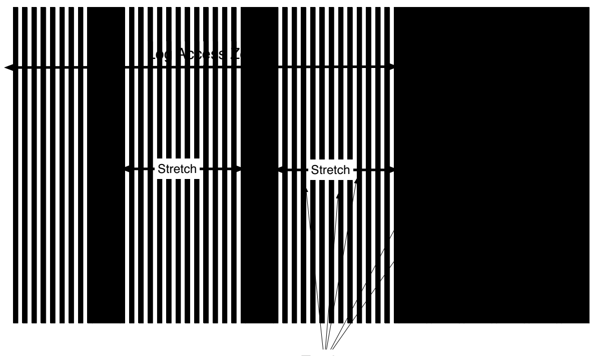

Fig. 3: Shingled write disk layout with Log Access Zone (LAZ) and Random Access Zone (RAZ).

Whatever the challenges involved in maintaining the road map, shingled write disks are about to ship. Other approaches (BPMR, HAMR, MAMR, TDMR) presume this technology and shingled writing appears to be feasible without major changes in technology. The influence of flash storage and in the future of Storage Class Memories (SCM) [29] will push magnetic recording towards a more archival role. As disks are a mass-manufactured commodity, we consider it unlikely to see continuous development of non-shingled write disks, once this technology has become adopted.

III. Related Work

A. *Using Shingled Write Disks*

The principal challenges for shingled write disks do not lie in the design and manufacturing of the drives themselves, but in their usage as replacements for standard hard drives. Because shingled write recording destroys information on tracks in one direction, traditional file systems cannot use shingled write disks without adjustment. A naive solution to the problem of destructive writes is a read-modify-write operation, which reads a portion of data from the disk, modifies parts or all of it, and then rewrites the whole portion back to the disk. Without several tracks without valid data, the portion that needs to be read could be a whole disk surface [5].

Kasiraj et al. propose organizing the disk into *bands*, where each band stores a single file such as a large multimedia file [24]. Bands are separated by a guard band 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. In general, assigning a single file to a band is neither necessary nor optimal.

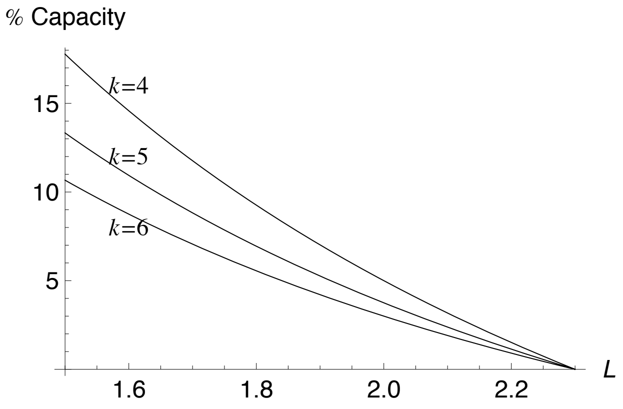

Fig. 4: Impact of introducing RAZ on the capacity gain L by shingled writing depending on the size k in tracks of the guard band.

To solve the problem of the read-modify-write in shingled writing, we can use some type of remapping between the addresses of physical blocks and the logical blocks that form the user interface. Cassuto et al. [5] propose two indirection methods for shingled write recording. The first method uses a cache located on disk and organized as a set-associative cache. A number of bands (Cassuto et al. call them native regions) map to a single band in the disk cache, which is itself a band, though possibly smaller. A write goes directly to the cache region. If the cache region becomes full, it is garbage-collected, which updates the bands mapped to that cache region with a read-modify-write. A read first seeks in the appropriate cache band and if that is unsuccessful, in the data band. Depending on the cache size ranging from 1% to 10%, Cassuto et al. observed a write overload of 1.5 - 9 and a total slowdown between 2 and 15. Read performance for sequential blocks also can suffer notably degradation.

By organizing bands as logs (a layout also proposed by Amer et al. [2], [3]) with a guard band between the beginning and the end of the log, Cassuto implements a second cache level between the set-associative disk cache and the data stored in fixed position on the shingled disk. At the cost of higher complexity, this design increases performance. The performance increase is based in great part by the lower cost of garbage collection of cache bands.

Amer et al. [2], [3] propose to adapt the design of log-structured file systems for shingled write disks. They introduce the notion of a Random Access Zone (Fig. 3) in addition to Log Access Zones (LAZ). A LAZ consists of bands that store a log, where head and tail are separated by an additional guard band. A RAZ consists of a single track band, or with other words, a single track with several guard tracks on both sides. This enables in-place change of blocks in the RAZ. Since the guard tracks protecting a single RAZ track cannot be used to store data, the use of RAZs diminishes the usable capacity of a shingled write disk. We define the capacity gain as the ratio of disk capacity with shingled writing and disk capacity without shingled writing and denote it by L. Tagawa and Williams project a capacity gain of $\alpha = 2.3$ [51]. Devoting a percentage for RAZ decreases L. Fig. 3 gives the relation between L and the percent of capacity devoted to RAZ for various width k of guard bands. It shows that reserving a small portion such as 1% of a platter surface to RAZ does not decrease the capacity gain by much. Of course, introducing bands and using logs within the bands introduces additional guard tracks that lowers the overall capacity gain further, (see below, Fig. 9).

B. Data Structures Tailored to Storage Architectures

Many data structures were developed for RAM only or for the classical memory hierarchy consisting of virtual memory and disk. The performance characteristics of other storage media frequently requires adjustments or even redesign of data structures. The same is true for shingled write disks.

Tapes consists of several parallel tracks on which blocks of data are written consecutively. They exhibit relatively large random access times compared to disk drives. Even though there is work on how to best use tape as a storage device (HPTFS [58]), it does not apply to the append only structure of shingled write disk, as metadata on tape can be updated in place. Write Once Read Many (WORM) devices such as optical disks did not allow changing any information written. Indirection is used heavily in the development of file systems and data structures [12], [13], [15], [36], [41], [42], [52]. Shingled write disks differ in offering the possibility to overwrite and reuse written space.

Flash memory can only be written in complete pages. A page needs to be erased before it can be written again. This operation clears all pages in an erase block consisting of typically 64 or 128 4KB-pages. Each erase-write operation is slightly destructive and frequently erased pages start to display high read bit-error rates and become eventually flagged as unusable. Most flash memory use a Flash Translation Layer (FTL) in order to achieve "wear leveling" (all pages have about the same number of writes) and to mask failed pages [14]. Additionally, writes are often considerably slower than read operations. A number of flash specific file systems exist (for example [10], [22], [34], [54]. Data structures for use with flash memory can be of two kinds, they can be "flash-aware" but used in conjunction with FTL, or they can be built on "raw" flash memory taking care of wear leveling themselves. For example, Wu et al. [56] uses a buffer structure in RAM in order to implement a B-tree. Kang et al. [23] propose the μ-tree, a variant of the B-tree, that allows any update (even if it should percolate to the root) by writing a single page, which contains a path from the root to the node changed. As information of several nodes has to fit in the same page, the height of a μ tree can be higher than that of the equivalent B+-tree, but the performance is not noticeably slower. Li et al. [33] propose the FD-tree that is a variant of the B-tree that stores the nodes below the root in continuous runs. We assume that the future will bring a large scale migration of databases to using flash and later storage class memory. Databases that use flash profit if their are being made "flash-aware" as is done with the Flash-based DBMS [31], with FlashDB [39], et cet.

A variety of groups has proposed to build databases using the principles of log-structured file systems [25], [40]. The great advantage of log-structured file systems [44], [45], [49] is the speed of writes as all writes are bundled and directed to the end of a log. The disadvantage is the cleaning necessary to reclaim the space occupied by stale data and the potential for ineffective in order reads (for example when scanning a table). Even a database that stores data in logs still uses a traditional disk with its capacity of in-place updates and existing implementations cannot simply be ported to shingled write disks. For example, the design of Graefe's write-optimized B-trees [17] stores data in logs but still uses in-place updates of index nodes. In this paper, we follow his strategy for write-optimized B-trees to design a B^+-tree for shingled write disks.

IV. Layout for Shingled Write Disks

Implementers of databases have traditionally shunned access to storage systems through the operating system and have preferred to use direct access for performance reasons [48]. The interface offered by most storage devices is primitive, modeling the disk as an array of Logical Block Addresses (LBA). Internally, an LBA is translated to a Physical Block Address (PBA). This translation also serves to mask faulty blocks, which are mapped to spare blocks on a track or possibly even to a spare track. We feel that the introduction of shingled writing presents a rare opportunity to change the system-disk interface. However, changes in a standard are difficult and cannot anticipate or support all usage patterns. The industry will aim at least for an efficient implementation of the current interface in order to facilitate a "drop-in" substitute, which could consist of a shingled drive coupled with a large flash-based cache. Clearly, any database implementation can profit from the performance and durability of NVRAM such as flash or SCM, whether it is provided in a "combo-disk" or separately. The limited number of erase-write cycles of flash might however limit the life-expectancy of a combo-disk.

While the standard interface between system and shingled write disk might be changed so that the system can be aware of the layout of tracks, we do have to assume this capacity in what follows. It becomes necessary to reconstruct the track layout in a disk to define RAZ and LAZ and define the minimal guard band between tail and head of a log in LAZ. Modern disk use zoning (zoned constant angular velocity) to vary the number of blocks in a track. A track in a zone closer to the center is shorter and cannot store as much data as a track in a zone on the outside. Even for traditional disks, it is possible to extract the layout from current disks through observation and timing of disk commands [1], [47], [55]. This task becomes considerably simpler for shingled write disks as we can make use of the destructive nature of a block write on the adjacent tracks in one direction. After extracting the layout of the tracks and zones, the shingled write disk is no longer a black box for the system but a *gray box*, allowing as often more efficient use of the disk [4].

Modern disks mask block faults. While most of these faults are caused by minute defects in the magnetic layer of the surface created during manufacturing, some faults can appear during the life time of a disk, for example through repeated head crashes. Masking is done by allocating from the start a certain number of spare blocks in each track and even a certain number of spare tracks evenly distributed throughout the disk. There is no reason to assume that this practice will not be continued for shingled write disks. If we start using a new shingled write disk, we reconstruct the layout including remapped tracks and blocks. If during the life-time of the disk, a block becomes faulty and is remapped to another block on the same track, then this remapping does not disturb our RAZ / LAZ layout. It can happen that a track runs out of spare blocks. In this case, the track needs to be moved to a nearby spare track. In a shingled write disk, this cannot be done by just copying the track to its new location, but involves moving data from all tracks in-between. While more involved than in traditional disk, this move does not disturb the partition of the disk in various RAZ, LAZ, or the separation of tail and head of a log. We conclude that masking new block errors does not create problems for our proposal.

V. B^+ Tree Structure for Shingled Write Disk

A database table supports a variety of operations such as scanning, fast access, whether by primary or secondary key, and often range queries by the primary key. The "work horse" for the implementation of database tables is the B-tree in its many variants such as the well-known B^+-tree [9]. A page pool in RAM contains recently accessed disk blocks. To allow transactional guarantees, we assume traditional write-ahead logging of content changes and structural changes in the tree. We also assume frequent checkpointing where dirty pages from the pool are written back to the shingled write disk.

Efficient implementations on shingled write disks can make sparing use of data units that can be updated in place, either in the RAZ, in the flash memory of a combo-disk, or in NVRAM belonging to the system. We assume here that our system has no NVRAM and uses only a shingled write disk for permanent storage. This is appropriate for some, but certainly not for all database applications.

A. Layout

A Random Access Zone (RAZ) consists of single track followed by a guard band of k tracks. We can overwrite its block in place without destroying any data because the destruction is limited to the tracks in the guard band and these do not store any data. A track contains 1MB or more of information, depending on the location of the track. Zoning allocates fewer blocks to an inside track than to an outside track, but maintains a constant bit density. We store the upper layers of the B-tree ("the index") in RAZ, while we store the lower part of the tree and at least all the leaf nodes in the Log Access Zone (LAZ) Figure5. We recall that a block update in LAZ consists in fetching the page into the page pool in RAM, updating it, and eventually writing it back at the end of the log in LAZ.

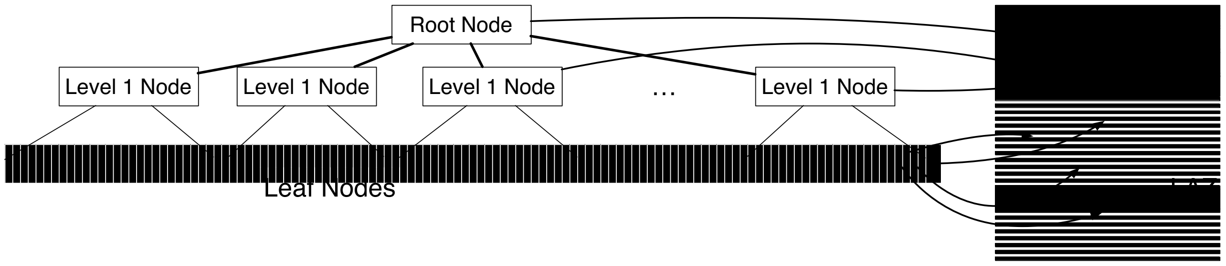

Fig. 5: Assignments of index nodes to RAZ and leaf nodes to the log in LAZ.

TABLE I: Page Utility for B-tree nodes on shingled write disk after [18]

Page Size	Records / page	Node utility	Access time	Utility / time	Access time	Utility / time	Access time	Utility / time
Configuration:			6000 rot 2 MB track		8000 rot 2.5 MB track		10000 rot 2.5 MB track	
4 KB	140	7	11.02 ms	0.647	9.76 ms	0.730	9.01 ms	0.791
16 KB	560	9	11.08 ms	0.823	9.80 ms	0.932	9.04 ms	1.010
64 KB	2240	11	11.31 ms	0.984	9.94 ms	1.120	9.15 ms	1.216
128 KB	4480	12	11.62 ms	1.044	10.12 ms	1.198	9.30 ms	1.304
256 KB	8960	13	12.20 ms	1.076	10.48 ms	1.253	9.58 ms	1.370
512 KB	17920	14	13.23 ms	1.068	11.13 ms	1.270	10.10 ms	1.399
1 MB	35840	15	14.81 ms	1.022	12.19 ms	1.241	10.95 ms	1.381
2 MB	76680	16	16.00 ms	1.008	13.38 ms	1.206	11.90 ms	1.355

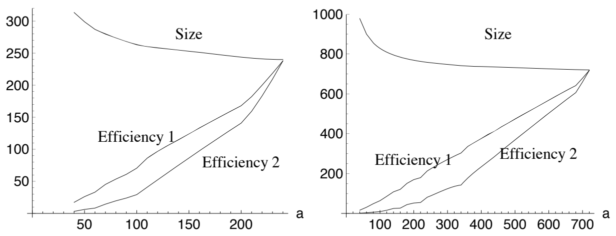

Fig. 6: Log size and read efficiencies after cleaning depending on cleaning size *a* for a log of 240 (left) and 720 (right) nodes.

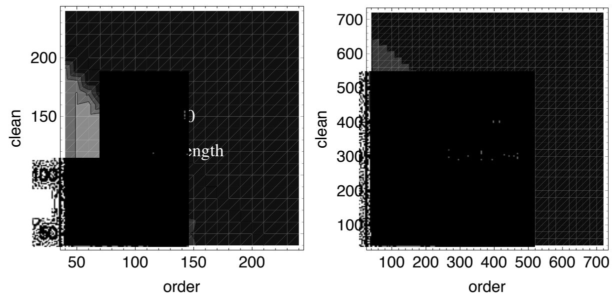

Fig. 7: Contour graph of log length for 240 and 720 nodes. (Black represents 240 (left) and 720 (right).)

B. Node Size

Graefe [18] has updated Gray's five minute rule [19], [20] in 2007. Gray and Graefe defined the utility of a B-tree node as the binary logarithm of the number of records in a node. They determine the optimal size of a B-tree index node by the ratio of utility over access time. As nodes become larger, the access time changes by an increase in the transfer time as the disk has to rotate longer under the head until the complete head is read. If we extrapolate the access times of current standard disks

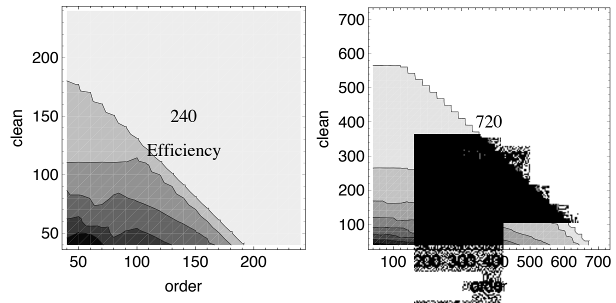

Fig. 8: Contour graph of efficiency for 240 and 720 nodes. (Black is low.)

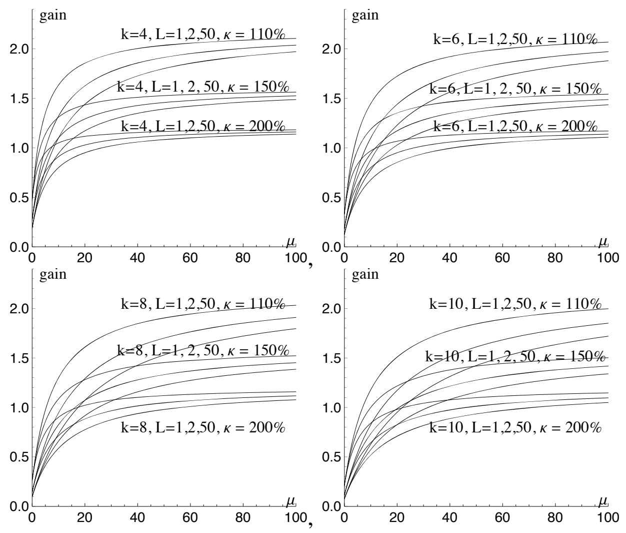

Fig. 9: Overall capacity gain using shingled writing.

to shingled write disks with higher density, the transfer time becomes shorter. Graefe concluded in 2007 that B-tree nodes of 256 KB are nearly optimal. We repeated his calculations and they show that the optimal size for shingled write disks is 256 KB, 512 KB, or 1 MB depending on our assumption on rotational latency (6000, 8000, and 10000 rotations/minute) and average track length (2 MB to 2.5 MB), Table I. A block size of 256 KB still remains close to optimal.

If we use Gray's and Graefe's assumption [19], [20] on the fan-out of B^+ trees, then we have 35 – 45 tracks worth of data per stretch of LAZ. The actual space needed is higher, as the log contains stale nodes. To regain space, a cleaning operation reads nodes from the head of the log and writes active nodes to the tail of the log. Reads and writes are of consecutive blocks and therefore very efficient compared to accessing random blocks. On average, a cleaning operation needs to recover as much or more space as is lost to updating. A large operation can write nodes in order and place consecutive nodes in adjoining blocks.

Storing nodes in logical order allows a logical read of nodes in a single read operation and enables read-ahead of blocks. While a log-structured table optimizes write operations, a series of updates will destroy the order in which active nodes are stored. To measure the efficiency of range queries, we introduce the notion of *scan efficiency*.

We count the number of times that the next node in logical order is physically located next to it in the track (Efficiency 1) and the number of times that the two following nodes in logical order are the two next nodes in the track (Efficiency 2). We simulated the behavior of the log with 240 and 720 nodes. We used two different types of operations, *(log) cleaning*, and *(log) ordering*. The first operation reads from the beginning of the log consecutively until it has encountered a certain value (the cleaning amount) of current nodes, which it then orders and writes at the end of the log. The second operation operates on a range of node numbers that walks through the set of nodes and when it reaches their end starts over at the beginning. All nodes are ordered according to their logical value and written to the end of the log. We introduced this operation because we observed that we needed to clean a large amount of nodes in order to obtain good read efficiency for scans (Fig. 6).

We simulated updating 40 nodes before starting an ordering and a cleaning operation with various amounts of nodes ordered or cleaned. Our results show that cleaning operations need to be moderately large to limit the storage overhead of the log (Fig. 6, 7). Ordering operations can have detrimental effects on the efficiency of scans and have at best the same effect as increasing the amount of nodes cleaned, thus they are a bad idea that should be abandoned in favor of increasing the cleaning amount (Fig. 8). The absolute size of the log is not important for either limiting the storage overhead of the log or for obtaining good read behaviour for scans. We calculate the storage overhead of our scheme. We assume an organization of L RAZ tracks with corresponding LAZ. If the node fan-out of the B^+-tree is μ, the LAZ contains a log with data filling μL tracks. Depending on the frequency of cleaning, the actual size of the log is larger by a factor of κ. Additionally, we have one guard band per RAZ and LAZ and two additional guard bands, one which separates the beginning and end of the log and one at the end of the assembly, each consisting of k tracks. This gives us a total storage need of $L+\mu\kappa L+(L+2)k$ for storing $L+\mu L$ worth of data. If we assume a storage capacity increase of 2.4 due to the introduction of shingled writing, we obtain the capacity gain g for B^+-trees as

$$\frac{2.4(L+1)\mu}{(1+\mu\kappa)L+(L+2)k}$$

We display the capacity gains in Fig. 9. We can see that L has little influence, the size of the guard band has some, but the greatest influence is that of efficient log cleaning resulting in smaller logs.

Taking our data together, it seems best to organize the B^+-table in ensembles consisting of one track of RAZ for lowest level index nodes and associated LAZ storing the node leafs in a LAZ. Higher levels of the index nodes should be stored in RAZ, but not necessarily next to a LAZ. The speed and extent of log cleaning offers a trade-off between better use of capacity and faster execution of range queries and scans on one hand and more frequent and involved cleaning operations on the other hand. The frequency and clustering of write operations will determine whether aggressive cleaning is possible.

C. Node Structure

We assume a standard B^+-tree structure, where the records are maintained in the leaf nodes. We also assume standard optimization techniques such as suffix and prefix compression to increase the number of entries in an index node. Many current variants of B^+ trees (such as B^{link}-trees [32] or Π-trees [37]) use sibling links between nodes of the same level. These links enable fast traversal of the table, but are also useful for key range locking [53]. Key range locking needs to find the next key, which might be in a neighboring node that needs to be accessed, a process known as "crawling".

If we try to maintain sibling links in an index or leaf node located in LAZ, then an update will ripple through all leaf nodes and index nodes, as moving one nodes forces link updates, which in turn force the nodes containing the links to be updated. We follow Graefe's lead [17] in using *fence keys*, that define the range of keys that can be inserted into the node in the future. A fence slightly increases the information that a node contains, but also eliminates the need for crawling at the lower levels of the tree. Fences also make prefix compression easier. On the downside, the lack of forward links creates the necessity to cache the next higher index node in order to find the next node, for example during a scan.

D. Page Updates

An update proceeds as usual by travelling from the root to the index node where the record should be inserted or which contains the pointer to the leaf node that needs to change. In Fig. 10 top, we show a three level B^+-tree with the upper two levels in RAZ. An update to a leaf node acquires a *latch* (a

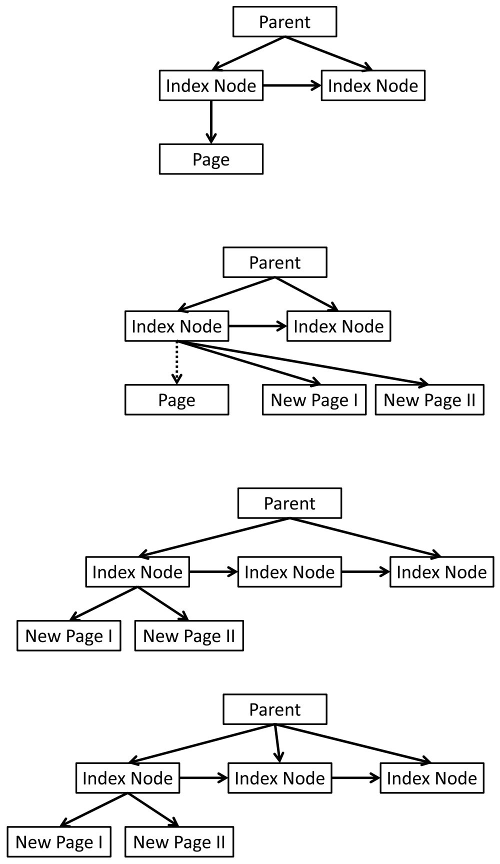

Fig. 10: Example of a split operation in the B$^+$-tree.

short-term low-cost lock whose holder guarantees the absence of deadlock) on the page and its parent index node. An update reads the page and writes the new page at the end of the log in LAZ. If there should be a crash at this moment, the update would be lost but for the write-ahead log. In our example (Fig. 10), we assume that the new page has to split and that we generate two different new pages. The next step is to change the parent index node in-place to point to the new page(s) (Second panel, Fig. 10). The index node itself might overflow because of the insertion of the additional entry. If this is the case, then the transaction creates an additional index node and sets the links in the second index node to point to the successor and the link in the old index node to point to the new one (Fig. 10, third panel). At this point, the transaction can free the latches and the tree is in a consistent state as all index nodes are reachable through the parent node.

Next, the transaction causes an update operation for the parent node to be scheduled. This update operation is independent of the transaction, that caused the change to the leaf node. The operation inserts the new link into the parent. If this leads to a split, then we proceed just as before with the lowest level index nodes (Fig. 10, bottom). This scheme based on the Π-trees of Lomet and Salzberg only updates small parts of the B$^+$-trees. In place of trying left- and right-rotation in order to avoid splits of index nodes, we can follow Lomet's example once more and run node consolidation procedures.

Assume now that the lowest layer of index nodes is stored in LAZ. In this case the index node referring to the leaf page cannot be updated in place. We therefore have to latch the parent in RAZ, the index node, and the leaf node. If we have to split the index node, then we write the two new index nodes in LAZ together with the two new leaf nodes. If we also need to split the parent, then we proceed with the parent as before, i.e. we use the sibling pointer in the parent to postpone a potential split of the parent's parent. The complexity of this operation is an additional argument against storing the index in LAZ.

Just like in a Π-tree, any operation involves at most one RAZ level in the B$^+$ tree. The operations are atomic and can succeed or fail independently of success or failure of another atomic transaction. Thus, only these actions need to be made recoverabe. Limiting the size of atomic transformations of the tree should result in improving concurrency.

E. Node Consolidation

Node consolidation is an option that tries to save space avoiding index nodes with low load by moving its entries to a left or a right neighbour and delete the under-utilized node. This can be done also at the level of leaf nodes. Necessary conditions for consolidation are that the neighbour shares the same parent node and that the neighbour can absorb the combined load. To maintain the atomicity of consolidation, the process needs to latch the node, its neighbour, and its parent. The latches are released only after the restructuring succeeds. Any operation that changes a node can discover an underutilized node and schedule a node consolidation.

F. Enabling Snapshot and Higher Read Concurrency

The ability to take a snapshot of a running system is important for debugging systems. For databases, it offers additional concurrency opportunities for transactions doing only reads. To take a snapshot, we make a virtual copy of the upper hierarchy of the B$^+$-tree that is stored in RAZ. All the index nodes in LAZ and all the leaf nodes, i.e. all nodes to which the index nodes in RAZ can point, are protected from overwrites because of the append-only nature of the log. Changes to the upper index nodes are relatively rare events. However, if a transaction needs to modify an upper index, for example because it is the parent of a leaf node that needs to split, we can maintain the snapshot by making a copy of that upper index node. We recall that in our scheme, splits that perculate above one level in RAZ are performed as an independent transaction and scheduled lazily. These tree maintenance operations can be scheduled to run after the snapshot is no longer needed, or they can run concurrently by safeguarding the node.

The same append-only log structure allows also additional types of transaction control for writes, as only the update to nodes in RAZ (or to their copies in the page pool) changes the state of the B^+-tree. The investigation of these opportunities has to be left for future work.

VI. Linear Hashing for Shingled Write Disk

Linear Hashing (LH) [35] is an extensible hashing scheme that offers single step access to a record given its key. LH calculates the page (the bucket in LH language) where the record resides using a variable hash function and a file state. In more detail, LH places all records in buckets numbered from 0 to $N-1$ with N being the number of buckets. The file state determines and is determined by N. It is written as (l,s) where $N=2^l+s$ and $l=\log_2(N)$ is the highest power of two fitting into N. LH uses a family of hash functions defined by $h_i(c)=c \pmod{2^i}$. Given a record with key c, the bucket $a(c)$ where the record resides is given by

$$a(c)=\begin{cases} h_l(c) & \text{if } s\le h_{l+1}c<2^l \\ h_{l+1}(c) & \text{else}\end{cases}$$

Bucket k with $s\le k<2^l$ have *bucket level l*, the remaining bucket level $l+1$. An LH-file adjusts to changes in the size of the file by *splitting* and *merging* buckets if the load factor (number of records divided by N) reaches thresholds. A split is always of the bucket s and a merger always merges buckets $N-1$ and $s-1$.

If an LH-file is stored in RAM, then the buckets tend to be small and the focus of implementation is on allowing concurrent addresses, as was done by the still up-to-date work of Ellis [11]. To store an LH file on disk, one should use a two layer implementation. A first layer of LH determines the disk block, a second layer of LH organizes the records in the disk block. For the lower layer, we use $c>>l$ in the first-level buckets with bucket level l and $c>>(l+1)$ for those with bucket level $l+1$, i.e. the key shifted by the file level as the key. A record access loads the bucket from disk into RAM (if it is not already in cached in RAM) and then uses the second level of LH to identify the record quickly in RAM. If we use Ellis' implementation, the first-level bucket contains the relative addresses of the buckets and each bucket is implemented as a short linear list [11]. As usual, access times is dominated by the time to fetch from disk. On a current disk, a first-level bucket is stored in a single block and updated in place. (It is possible that blocks overflow and we need to store parts of the block in an overflow bucket.)

An easy adaptation to shingled write disk is to store the LH-file completely in RAZ. At the risk of additional disk fetches or using more space in RAM for disk caching, we can pack the data in an LH-file more densely in LAZ. For this we need to introduce a block translation layer stored in RAM. As outlined before, we use a two-level implementation where the first level determines the disk bucket and the second level determines the location of the record within the disk bucket. When we update, merge, or split a block, we fetch the blocks involved, create the updated blocks and write them in a LAZ. We also maintain the addresses of the blocks in a RAM-dictionary. Thus, after calculating the bucket number, we consult the dictionary to find where the disk block is stored. We then fetch the block and use the lower level LH organization to access the record.

This implementation offers a trade-off between RAM usage and memory bandwidth usage. If we use various buckets to store the same first-level bucket, then the size of the dictionary is smaller, but we need to transfer larger units between disk and memory. Tuning the bucket size depends on local load and architecture, but can be approximated by comparing the costs of adding bandwidth and RAM. If the database shuts down orderly, we write the dictionary in a RAZ. In order to avoid writing the dictionary to RAZ after each bucket update, we maintain a table sequence number that is augmented with every action that changes the contents on disk. We then store the sequence data and bucket number and level at the beginning of each first-order bucket. As long as we know in which LAZ the LH-file is stored, we can then reconstruct the translation dictionary and the file state of the first level LH-structure by scanning the LAZ.

Our second design gives the same access time for shingled write disks as for current disks at the cost of maintaining a translation table in memory. If the table is only rarely accessed so that it should not be kept in memory, then our design adds one disk access to the dictionary on RAZ for each read and two disk accesses for each write (since we now have to update the dictionary in RAZ).

VII. Conclusions

While shingled write disks are only about to appear on the market (2013), their introduction will be disruptive for database table implementations. Just as other storage technologies introduced in the past, they require an adaptation of data structures if a new technology is going to be used at its fullest potential. We have shown how to adapt two of the most important data structures used to implement database tables — B^+-trees and linear hash files — without adding significantly to access times. To do so, we gave up some of the additional storage space gained by shingled writing. An evaluation of our proposed data structures through simulation needs to be postponed for future work. The use of a combination of NVRAM (such as flash or SCM) alleviates the design challenges, but needs to be investigated. Finally, the log structure imposed by shingled writing can be used for version based transaction control.

References

[1] M. Aboutabl, A. Agrawala, and J.-D. Decotignie, "Temporally determinate disk access (extended abstract): an experimental approach," in *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, ser. SIGMETRICS '98/PERFORMANCE '98, 1998, pp. 280–281.

[2] A. Amer, J. Holliday, D. Long, E. Miller, J. Paris, and T. Schwarz, "Data management and layout for shingled magnetic recording," *Magnetics, IEEE Transactions on*, vol. 47, no. 10, pp. 3691–3697, 2011.

[3] A. Amer, D. Long, E. Miller, J. Paris, and T. Schwarz, "Design issues for a shingled write disk system," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. IEEE, 2010, pp. 1–12.

[4] A. Arpaci-Dusseau, R. Arpaci-Dusseau, L. Bairavasundaram, T. Denehy, F. Popovici, V. Prabhakaran, and M. Sivathanu, "Semantically-smart disk systems: past, present, and future," *ACM SIGMETRICS Performance Evaluation Review*, vol. 33, no. 4, pp. 29–35, 2006.

[5] Y. Cassuto, M. Sanvido, C. Guyot, D. Hall, and Z. Bandic, "Indirection systems for shingled-recording disk drives," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. IEEE, 2010, pp. 1–14.

[6] 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.

[7] 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.

[8] K. Chan, R. Radhakrishnan, K. Eason, M. Elidrissi, J. Miles, B. Vasic, and A. Krishnan, "Channel models and detectors for two-dimensional magnetic recording," *Magnetics, IEEE Transactions on*, vol. 46, no. 3, pp. 804–811, 2010.

[9] D. Comer, "Ubiquitous B-tree," *ACM Comput. Surv.*, vol. 11, no. 2, pp. 121–137, Jun. 1979.

[10] H. Dai, M. Neufeld, and R. Han, "ELF: An efficient log-structured flash file system for micro sensor nodes," in *ACM Conference on Embedded Networked Sensor Systems*, 2004.

[11] C. Ellis, "Concurrency in linear hashing," *ACM Transactions on Database Systems (TODS)*, vol. 12, no. 2, pp. 195–217, 1987.

[12] R. Finlayson and D. Cheriton, "Log files: an extended file service exploiting write-once storage," in *SOSP '87: Proceedings of the eleventh ACM Symposium on Operating systems principles*, 1987, pp. 139–148.

[13] J. Gait, "The optical file cabinet," *Computer*, vol. 39, no. 1, pp. 2 – 9, June 1988.

[14] E. Gal and S. Toledo, "Algorithms and data structures for flash memories," *ACM Computing Surveys (CSUR)*, vol. 37, no. 2, pp. 138–163, 2005.

[15] S. Garfinkel, "A file system for write-once media," MIT Media Lab, Tech. Rep., October 1986.

[16] G. Gibson and G. Ganger, "Principles of operation for shingled disk devices," Tech. Rep. CMUPDL-11-107, Carnegie Mellon University, Tech. Rep., 2011.

[17] G. Graefe, "Write-optimized B-trees," in *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. VLDB Endowment, 2004, pp. 672–683.

[18] ——, "The five-minute rule twenty years later, and how flash memory changes the rules," in *Proceedings of the 3rd international workshop on Data management on new hardware*. ACM, 2007, p. 6.

[19] J. Gray and G. Graefe, "The five-minute rule ten years later, and other computer storage rules of thumb," *ACM Sigmod Record*, vol. 26, no. 4, pp. 63–68, 1997.

[20] J. Gray and F. Putzolu, "The 5 minute rule for trading memory for disc accesses and the 10 byte rule for trading memory for cpu time," in *ACM SIGMOD Record*, vol. 16, no. 3, 1987, pp. 395–398.

[21] 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, October 2009.

[22] W. Josephson, L. Bongo, K. Li, and D. Flynn, "DFS: A file system for virtualized flash storage," *ACM Transactions on Storage (TOS)*, vol. 6, no. 3, p. 14, 2010.

[23] D. Kang, D. Jung, J.-U. Kang, and J.-S. Kim, "μ-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.

[24] 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.

[25] J. Kohl, C. Staelin, and M. Stonebraker, "Highlight: Using a log-structured file system for tertiary storage management," in *Usenix Conference*, 1993.

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

[27] 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.

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

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

[30] 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.

[31] S.-W. Lee and B. Moon, "Design of flash-based DBMS: an in-page logging approach," in *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, 2007, pp. 55–66.

[32] P. Lehman *et al.*, "Efficient locking for concurrent operations on b-trees," *ACM Transactions on Database Systems (TODS)*, vol. 6, no. 4, pp. 650–670, 1981.

[33] 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.

[34] S. Lim and K. Park, "An efficient NAND flash file system for flash memory storage," *Computers, IEEE Transactions on*, vol. 55, no. 7, pp. 906–912, 2006.

[35] W. Litwin, "Linear hashing: A new tool for file and table addressing," in *Sixth International Conference on Very Large Data Bases, October*. IEEE Computer Society, 1980, pp. 212–223.

[36] D. Lomet and B. Salzberg, "Access methods for multiversion data," *ACM SIGMOD Record*, vol. 18, no. 2, pp. 315–324, 1989.

[37] ——, "Concurrency and recovery for index trees," *The VLDB Journal*, vol. 6, no. 3, pp. 224–240, Aug. 1997.

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

[39] 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.

[40] K. Norvag and K. Bratbergsengen, "Write optimized object-oriented database systems," in *Computer Science Society, 1997. Proceedings., XVII International Conference of the Chilean*. IEEE, 1997, pp. 164–173.

[41] S. Quinlan, "A cached WORM file system," *Software – Practice and Experience*, vol. 21, no. 12, December 1991.

[42] P. Rathmann, "Dynamic data structures on optical disks," in *Proceedings of the First International Conference on Data Engineering*, 1984, pp. 175–180.

[43] 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 and beyond," *IEEE Transactions on Magnetics*, vol. 42, no. 10, pp. 2255–2260, Oct. 2006.

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

[45] 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, October 1991.

[46] 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, October 2006.

[47] J. Schindler and G. R. Ganger, "Automated disk drive characterization (poster session)," *SIGMETRICS Perform. Eval. Rev.*, vol. 28, no. 1, pp. 112–113, Jun. 2000.

[48] R. Sears, C. van Ingen, and J. Gray, "To blob or not to blob: Large object storage in a database or a filesystem?" *Microsoft Research Technical Report MSR-TR-2006-45*, 2006.

[49] M. Selzer, K. Bostic, M. McKusick, and C. Staelin, "An implementation of a log-structured file system for UNIX," in *Winter Usenix Conference*, 1993.

[50] 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, October 2009.

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

[52] J. S. Vitter, "An efficient I/O interface for optical disks," *ACM Trans. Database Syst.*, vol. 10, no. 2, pp. 129–162, 1985.

[53] G. Weikum and G. Vossen, *Transactional information systems: theory, algorithms, and the practice of concurrency control and recovery*. Morgan Kaufmann Pub, 2002.

[54] D. Woodhouse, "JFFS: The journalling flash file system," in *Ottawa Linux Symposium*, vol. 2001, 2001.

[55] B. L. Worthington, G. R. Ganger, Y. N. Patt, and J. Wilkes, "On-line extraction of scsi disk drive parameters," in *Proceedings of the 1995 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, ser. SIGMETRICS '95/PERFORMANCE '95, 1995, pp. 146–156.

[56] C.-H. Wu, T.-W. Kuo, and L. P. Chang, "An efficient B-tree layer implementation for flash-memory storage systems," *ACM Trans. Embed. Comput. Syst.*, vol. 6, no. 3, p. 19, 2007.

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

[58] X. Zhang, D. Du, J. Hughes, and R. Kavuri, "HPTFS: A high performance tape file system," in *26th IEEE Symposium on Massive Storage Systems and Technology*, 2006.

[59] 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.