HyLog: A High Performance Approach to Managing Disk Layout

Wenguang Wang Yanping Zhao Rick Bunt
Department of Computer Science
University of Saskatchewan
Saskatoon, SK S7N 5A9, Canada
{wang, zhao, bunt}@cs.usask.ca

Abstract

Our objective is to improve disk I/O performance in multi-disk systems supporting multiple concurrent users, such as file servers, database servers, and email servers. In such systems, many disk reads are absorbed by large in-memory buffers, and so disk writes comprise a large portion of the disk I/O traffic. LFS (Log-structured File System) has the potential to achieve superior write performance by accumulating small writes into large blocks and writing them to new places, rather than overwriting on top of their old copies (called Overwrite). Although it is commonly believed that the high segment cleaning overhead of LFS makes it a poor choice for workloads with random updates, in this paper we find that because of the fast improvement of disk technologies, LFS significantly outperforms Overwrite in a wide range of system configurations and workloads (including the random update workload) under modern and future disks.

LFS performs worse than Overwrite, however, when the disk space utilization is very high due to the high cleaning cost. In this paper, we propose a new approach, the Hybrid Log-structured (HyLog) disk layout, to overcome this problem. HyLog uses a log-structured approach for hot pages to achieve high write performance, and Overwrite for cold pages to reduce the cleaning cost. We compare the performance of HyLog to that of Overwrite, LFS and WOLF (the latest improvement on LFS) under various system configurations and workloads. Our results show that, in most cases, Hylog performs comparably to the best of the other three approaches.

1 Introduction

Disk I/O performance is crucial to the performance of many computer systems. With large in-memory buffers, most disk reads can be resolved in memory [18]. As a result, in write-intensive systems, such as a DBMS running OLTP (On-Line Transaction Processing) applications, email servers, and file servers, write requests make up a large portion of the total disk traffic [3, 22]. Since these writes are usually small, most of the disk I/O time is seek time and rotational latency, resulting in less than 10% of the disk maximum bandwidth being utilized [19, 21].

LFS (Log-structured File System) [18, 19] is a disk layout that can achieve superior write performance by writing data to new places in large chunks rather than *overwriting* on top of their old copies. But LFS has to perform segment cleaning to reclaim large contiguous free space for further writes. Previous studies found that this cleaning overhead significantly degrades system performance when the disk space utilization is higher than 50% on a 1991 disk under OLTP workloads [21]. Disk technology has improved dramatically since these studies were published. Using 1991's DEC RZ26 and today's Cheetah X15 36LP as examples, the disk positioning time has decreased from 15ms to 5.6ms (2.7x improvement), while the disk bandwidth has increased from 2.3MB/s to 61MB/s (27x improvement). The disk bandwidth improved 10 times more than the positioning time for these two disks, and this trend is likely to continue [6]. Since LFS was designed to utilize the disk bandwidth, this trend favors the performance of LFS. Whether the cleaning cost of LFS is still prohibitively high under modern and future disks is an unaddressed issue.

In this paper, we use a simple cost model to study the performance of LFS and Overwrite (i.e., the traditional approach that new data are overwritten on top of old copies). Our results show that although the cleaning overhead was expensive for old disks and still accounts for a large amount of disk traffic in modern and future disks, the overall performance of LFS is significantly better than that of Overwrite. LFS loses its advantage to Overwrite only under very high disk space utilization. For example, LFS performs better than Overwrite under uniform random update workload (a pathological workload for LFS that causes the most cleaning cost) when the

disk space utilization is lower than 97% on a year 2003 disk because of its cleaning overhead, assuming 8KB page size and 1MB segment size.

Because of the skewness in page access distribution in real systems [8], most writes are to a small portion of data pages (called *hot* pages), while the other pages (called *cold* pages) are updated infrequently. In LFS, hot pages rarely need to be cleaned because their current copies on the disk are often invalidated by further writes to these pages before the space they occupy is reclaimed by the cleaner. Therefore, most of the cleaning cost comes from cold pages, while most of the high write performance comes from accumulating the writes to hot pages.

We propose a new disk layout called *HyLog* (Hybrid Log-structured). The basic idea underlying HyLog was first mentioned in the conclusions of [12]: "it is not impossible to envision an LFS in which some segments are managed using in-place updating". To our knowledge, no further analyses or experiments have been conducted. HyLog uses a log-structured layout for hot pages to achieve high write performance, and overwrite for cold pages to reduce the cleaning cost. We evaluate the performance of HyLog, Overwrite, LFS, and WOLF (the latest LFS variant [26]) under RAID-0 and RAID-5 disk arrays with concurrent users, a wide range of disk space utilization, and a number of benchmarks and real workloads. We also speculate a disk model five years into the future and study the performance of these disk layouts when using this future disk. Our results show that the performance of HyLog is the most robust among the disk layouts we considered. In most cases, HyLog achieves performance comparable to or better than the best of Overwrite, LFS, and WOLF.

2 Related Work

Many approaches have been proposed to improve disk write performance. NVRAM (non-volatile RAM) is used in many storage systems to cache bursts of writes. Since NVRAM is constrained in size due to its high price, Disk Caching Disk [9] employs a log disk to substitute for NVRAM and achieves similar write performance. The problem with these two approaches is that they achieve high write performance only in systems with many idle periods. Virtual Log is an approach to improving small disk write performance [27] even in systems with no idle periods. But it requires detailed knowledge of the disk layout and the location of the disk head at any moment, which might be difficult to obtain from modern disks [10].

LFS was designed to optimize the write performance of file systems [19]. In LFS, the disk is divided into large fixed-size chunks called *segments*. Writes to data pages are accumulated in memory and written out to free segments. At the same time, the old copies of these pages are invalidated, leaving free space in the segments where they reside. From time to time, *segment cleaning* must be conducted to reclaim the free space in these partially filled segments so that free segments are always available for future writes.

A previous study [21] found that LFS has high cleaning overhead under OLTP-like workloads, where small random writes make up a large portion of the disk I/O requests. Many algorithms have been proposed to reduce the cleaning cost of LFS [2, 14, 16, 19]. But the cleaning cost is still high in systems with high disk space utilization and little idle time. *Freeblock scheduling* [13] has the potential to conduct cleaning without affecting foreground response times, even in a never-idle system. This algorithm relies on detailed knowledge of internal disk activities in order to make correct scheduling decisions, which is difficult for modern disks [10]. PROFS [25] attempts to improve the performance of LFS by placing hot data in the faster zones of the disk and cold data in the slower zones during the cleaning, but this approach does not address the high cleaning cost of LFS. Write Anywhere File Layout (WAFL) [7] and Log-Structured Array [15] use LFS and NVRAM to manage disk layouts. WAFL also maintains multiple snapshots of the file system. Although NVRAM eliminates writes for keeping the metadata integrity and improves write performance, the high cleaning cost of LFS is not addressed.

WOLF [26] is the most recently proposed approach to reducing LFS cleaning overhead. WOLF separates hot and cold pages when they are written to the disk. It usually writes two segments of data to the disk at one time. Pages are sorted based on their update frequencies before being inserted into the segment buffers. The rationale is that the segments containing pages with higher update frequencies will soon become low-utilized since the pages in them are likely to be updated again in a short time, thus reducing the cleaning overhead. This approach works well only when about half of the pages are hot and half are cold, so that they can be written into separate segments. In other cases, WOLF might have little advantage over LFS.

3 Disk Layout Write Cost Model

The extensive use of client and server caching on read traffic makes write performance an important factor in many systems [18]. In fact, write traffic was found to exceed read traffic on some recent file sys-

tem [3] and OLTP workloads [22]. For the purposes of modeling, we assume that the read performance is not affected by different disk layouts, and seek to model the write cost of these disk layouts.

3.1 Assumptions and Definitions

We use a simple disk model with seek time, rotational latency, and transfer bandwidth. The positioning time T_{pos} is the sum of the average seek time and the average rotational latency, i.e., the time for the disk to rotate half a rotation. The transfer bandwidth B is the average sustained bandwidth at which the disk can read/write data. We assume the read bandwidth is the same as the write bandwidth.

We assume that data are stored on the disk in fixed-size pages. The size of each page is P bytes. In LFS, the disk is separated into fixed-size segments, each of which has S pages. The time to read or write a page is T_{pg} and the time to read or write a segment is T_{seg}. We have $T_{pg} = T_{pos} + P/B$ and $T_{seg} = T_{pos} + SP/B$.

3.2 Modelling LFS and Overwrite

3.2.1 Segment I/O Efficiency

One design objective of LFS is to achieve better write performance than Overwrite. This is achieved by writing data in units of segments instead of pages. The *segment I/O efficiency* η represents the saving of disk I/O time for writing one segment over writing S pages of the segment individually. η is defined as

$$\eta = \frac{ST_{pg}}{T_{seg}} = \frac{S(P + T_{pos}B)}{SP + T_{pos}B}. \quad (1)$$

The higher the η, the better the performance of LFS, if other factors are not changed. η is a monolithically increasing function of the segment size S. η is also a monolithically increasing function of $T_{pos}B$, named as the *disk performance product*, which represents the amount of data the disk can transfer during the time to position the disk head. We list the parameters of three high end SCSI disks of different years in Table 1 and show their segment I/O efficiency in Figure 1. Modern disks have much larger η than old disks, implying LFS performs much better on modern disks than on old disks.

When a disk has multiple pending requests from several users, a disk scheduling algorithm is often used to reorder the requests so that the average disk positioning time can be reduced. η decreases with an increase in number of users as a result.

3.2.2 Space Utilization

The disk space utilization is an important factor affecting the performance of LFS [19, 21]. The *disk space utilization* u_d represents the proportion of the disk space occupied by user data. The space utilization of the segments that are selected for cleaning is called the *cleaning space utilization* u, which is the space utilization of the segment with the most free space. Therefore, $u \leq u_d$. More specifically, their relation is given in [15] as

$$u_d = (u - 1)/\ln u. \quad (2)$$

Figure 2 shows that the simulation results match this formula well.

Table 1: Disk Parameters

Brand Name	Year	Positioning Time (ms)	Bandwidth (MB/s)
Cheetah X15 36LP	2003	5.6	61.0
Quantum atlas10k	1999	8.6	20.4
DEC RZ26	1991	15.0	2.3

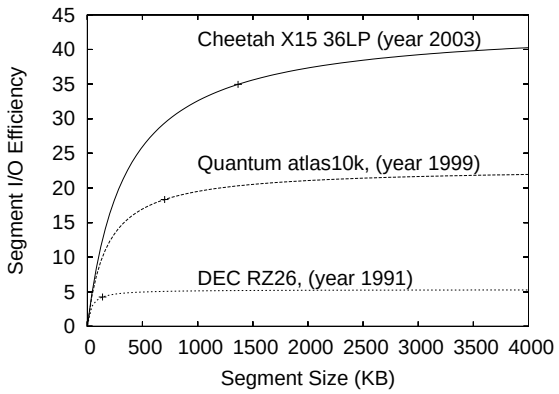


Figure 1: Segment I/O Efficiency of Different Disks. [Page size is 8KB. The small crosses indicate the segment size used for each disk in this paper.]

3.2.3 The Write Cost Model

In Overwrite, each write takes T_{pg} time. Thus the write cost of Overwrite C_{ow} is

$$C_{ow} = T_{pg}$$

To model the write cost of LFS, the segment cleaning overhead must be considered. There are two segment cleaning methods: *cleaning* [19] and *hole-plugging* [14]. We call these variants of LFS *LFS-cleaning* and *LFS-plugging*, respectively.

In LFS-cleaning, some candidate segments for cleaning are first selected, and then these segments

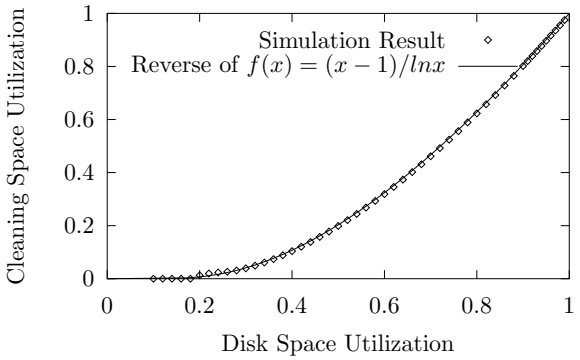


Figure 2: Disk Space and Cleaning Space Utilization.

are read into memory, and their alive pages are written out in segments, after which the free space in these segments is reclaimed. After 1 segment is read, u segment space is written and $1-u$ segment space is freed. Therefore $\frac{1+u}{1-u}$ segment I/O operations are required to free 1 segment space. For the system to be balanced, whenever a segment of user data is written to the disk, a segment of free space is reclaimed by cleaning. Thus LFS requires $1 + \frac{1+u}{1-u} = \frac{2}{1-u}$ segment I/O operations to write one segment of user data. The average time required to write one page in LFS is defined as the write cost $C_{lfscleaning}$.

$$C_{lfscleaning} = \frac{T_{seg}}{S} \cdot \frac{2}{1-u}.$$

In LFS-plugging, some candidate segments are read into memory, and the alive pages of these candidate segments are written out to holes found in other segments so that the space occupied by these candidate segments becomes free. To reclaim one segment of free space, 1 segment read and uS page writes are needed. Therefore, the write cost of LFS-plugging $C_{lfsplugging}$ is defined as the average time required to write one page.

$$C_{lfsplugging} = \frac{1}{S} \cdot (2T_{seg} + uST_{pg}).$$

3.2.4 Performance Comparisons

The performance of these disk layouts can be compared by comparing their write costs. To simplify the write costs, we define the *scaled write cost* by scaling all write costs by $\frac{S}{T_{seg}}$.

$$C'_{ow} = \frac{S}{T_{seg}} C_{ow} = \eta \qquad (3)$$

$$C'_{lfscleaning} = \frac{S}{T_{seg}} C_{lfscleaning} = \frac{2}{1-u} \qquad (4)$$

$$C'_{lfsplugging} = \frac{S}{T_{seg}} C_{lfsplugging} = 2 + u\eta \qquad (5)$$

Note that $C'_{lfscleaning}$ is the same as the traditional write cost of LFS [19]. In [19], the write cost of Overwrite was defined as the reciprocal of the utilized disk bandwidth (i.e., $\frac{T_{pos}B+P}{P}$), which ignores the effect of segment size. Segment size is important to the performance of LFS [14] and is taken into account by C'_{ow}. Figure 3 shows the scaled write cost of the disks listed in Table 1. The relationship between LFS-cleaning and LFS-plugging is consistent with previous studies [14]. Overwrite, LFS-cleaning and LFS-plugging always cross at the same point when $u = 1 - 2/\eta$. Since faster disks have larger η, this cross point happens at higher disk space utilization (e.g., $u = 94\%$ or $u_d = 97\%$ for a year 2003 disk), which means the performance advantage of LFS over Overwrite increases as disk technologies improve. Figure 3 indicates that LFS outperforms Overwrite under such workloads when the cleaning space utilization is below 94% under modern disks. LFS should perform better than what is shown in this figure under more realistic workloads since the cleaning space utilization is lower than that of a uniform random update workload [14]. Therefore, under modern and future disk technologies, the importance of cleaning cost of LFS is much less important than the common belief derived from studies with 10-year-old disks [21].

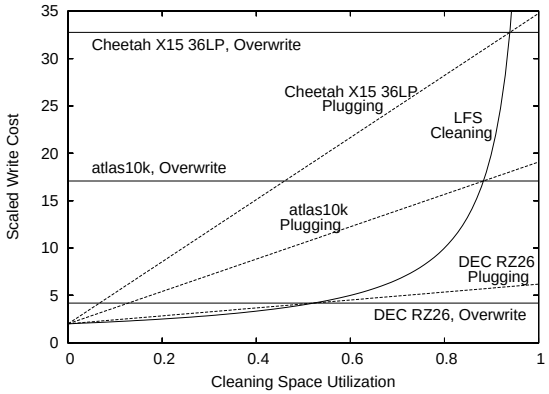


Figure 3: Write Costs of Different Layouts. [Smaller values indicate better performance. The segment size for Cheetah X15 36LP is 1MB, for atlas10k is 512KB, and for DEC RZ26 is 128KB. The selection of segment sizes are discussed in Section 3.3. The write cost of LFS-cleaning for all disks overlaps.]

3.3 Segment Size of LFS

Simulation studies in [14] showed that the segment size is important to the performance of LFS. By experimenting on different disks, the following rules-of-thumb were found [14]:

1. The optimal segment sizes are different for different disks. Only the disk performance product (product of positioning time and transfer bandwidth) matters.
2. Larger segments are required for faster disks. The optimal segment size can be approximated by 4 times the disk performance product.

Equation (1) shows that $T_{pos}B$ is the only disk characteristic that affects η, which is consistent with the first rule-of-thumb. The scaled write costs (Equations (3), (4) and (5)) indicate that the higher the η and the lower the u, and the more advantage LFS can achieve over Overwrite. Figure 1 shows that the larger the segment, the higher the η. However, on one hand, the increase of η is slower with larger segment sizes, while on the other hand, the cleaning space utilization becomes higher with larger segments [14]. Therefore, there is an optimal segment size to achieve the best performance. From Equation (1), we have

$$\lim_{S\to\infty} \eta = \frac{T_{pos}B + P}{P}.$$

Assume that we want to select a segment size so that α proportion of this limit is achieved ($0 < \alpha < 1$). Then

$$\frac{S(T_{pos}B + P)}{T_{pos}B + SP} = \alpha\frac{T_{pos}B + P}{P}.$$

Thus we have

$$S \cdot P = \frac{\alpha}{1-\alpha}T_{pos}B,$$

where $S \cdot P$ is equal to the segment size. If $\alpha = 80\%$, we have

$$S \cdot P = 4T_{pos}B, \tag{6}$$

which is consistent with the second rule-of-thumb. These preferred segment sizes are marked by small crosses in Figure 1. The crosses are close to the "knee" of the curve. In this paper, we use this formula to calculate the segment size and then round it to the closest size in powers of two.

3.4 Multiple Users and RAID

Many systems use disk arrays and have multiple concurrent users. We use N_d to represent the number of disks and N_u to represent the number of users. We assume that users send out requests without think time. When RAID is used, all disks are viewed as one large logical disk. We assume the stripe size is S pages. In RAID-0, the segment size of the logical disk is N_dS; in RAID-5, the segment size of the logical disk is $(N_d-1)S$, because one disk worth of space is used to store parity data. This organization allows segment I/O to utilize all available disk bandwidth and eliminates the write penalty in RAID-5.

3.5 The HyLog Model and Performance Potential

Figure 3 indicates that a small reduction in disk space utilization can significantly reduce segment cleaning cost and improve the performance of LFS. Because of the skewness in page access distribution [8], most writes are to a small portion of hot pages. If only these hot pages are managed by LFS while cold pages are managed by Overwrite, we can dedicate all free space to the hot pages, since Overwrite does not need extra free space. The resulting space utilization for the hot pages would be lower, which implies higher performance for the hot pages. Therefore, the overall performance could exceed both LFS and Overwrite. We call this approach the HyLog layout.

We first give the write cost model of HyLog and then analyze its performance potential.

In HyLog, the disk is divided into fixed-size segments, similar to LFS. A segment is a hot segment (containing hot pages and free pages), a cold segment (containing cold pages and free pages), or a free segment (containing only free pages). The hot segments and free segments form the *hot partition*, and the cold segments form the *cold partition*.

Since LFS-plugging performs worse than LFS-cleaning under low space utilization and worse than Overwrite under high space utilization, including LFS-plugging in HyLog does not bring performance benefit. Therefore, we do not consider LFS-plugging when modeling HyLog. Assume the proportion of hot pages is h ($0 < h < 1$) and the proportion of writes to the hot pages (called *hot writes*) is w ($0 < w < 1$). We use u'_d and u' to represent the disk space utilization and cleaning space utilization of the hot partition, respectively. If all free space is in the hot partition, we have

$$u'_d = \frac{uh}{1-u+uh}. \tag{7}$$

u' can be calculated from u'_d based on Equation (2). The scaled write cost of HyLog C'_{hylog} is

$$\begin{aligned} C'_{hylog} &= (1-w)C'_{ow} + wC'_{lfs} \\ &= (1-w)\eta + \frac{2w}{1-u'}. \end{aligned} \tag{8}$$

When h is 0 and 1, the cost of HyLog degrades to Overwrite and LFS, respectively. The proportion of hot writes w is a function of h, which is the CDF of the write frequencies.

For uniformly distributed random access, $w = h$. It was found the CDF of page update frequency in production database workloads follows the $Hill(f_{max}, k, n)$ distribution $Hill(105, 0.528, 0.546)$ [8], which is defined by $f(x) = f_{max} \cdot x^n/(k + x^n)$. Note that these distributions are for page updates before being filtered by the buffer cache. When write through is used (such as in an NFS server), these distributions can also describe the page writes to disks. When write back is used (such as in a database server), the page writes to disks are less skewed (closer to the uniform distribution).

Another distribution commonly used to represent the skewness of data accesses is defined by Knuth [11, p. 400]: $p_i = \frac{i^\theta - (i-1)^\theta}{N^\theta}$, where $i = 1 \ldots N$ and $0 \le \theta \le 1$. When $\theta = 1$, this is the uniform distribution. When $\theta = \frac{\log 0.80}{\log 0.20} = 0.1386$, this is the "80-20" rule where 80% references go to 20% pages. We call this distribution $Knuth(a, b)$, where $\theta = \frac{\log 0.01a}{\log 0.01b}$. Figure 4(a) shows the CDF of the above distributions with different parameters.

Figures 4(b) and 4(c) show the scaled write cost of HyLog under these distributions. Equation (2) is used to convert between the disk space utilization and cleaning space utilization. Since this equation works only for uniform random workloads, the results shown in Figure 4(b) and 4(c) are conservative for skewed distributions. With the right number of hot pages, HyLog outperforms both Overwrite and LFS. The higher the skewness of the distribution, the fewer hot pages are required and the more benefit can be achieved. In other words, HyLog has greater performance potential than LFS and Overwrite under high disk space utilization. When the disk space utilization is low, HyLog has limited benefit over LFS.

4 The Design of HyLog

4.1 Design Assumptions

We assume the disk layouts under study (Overwrite, LFS, WOLF, and HyLog) are at the storage level rather than the file system level. Therefore, the allocation and deallocation of data are not known. We assume NVRAM is used by these disk layouts so that small synchronous writes caused by metadata operations are not necessary. Therefore, we omit the metadata operations and focus on the impact of cleaning overhead of LFS. This omission greatly simplified the design and implementation of the disk layout simulator. Since LFS performs much better than Overwrite on metadata operations [19, 21], the omission of metadata operations makes our results for LFS, WOLF, and HyLog conservative compared to Overwrite.

These assumptions, however, do not mean that HyLog can only be used at the storage level with NVRAM. When technologies such as Soft-updates [4] and journaling [28] are applied to HyLog, it could be used at the file system level as well.

WOLF [26] reduces the segment cleaning cost of LFS by sorting the pages to be written based on their update frequencies and writing to multiple segments at a time. This idea can be easily applied to HyLog to further reduce its cleaning cost, but, to isolate the benefit realized from the design of HyLog and from the idea of WOLF, this optimization is not performed in this paper.

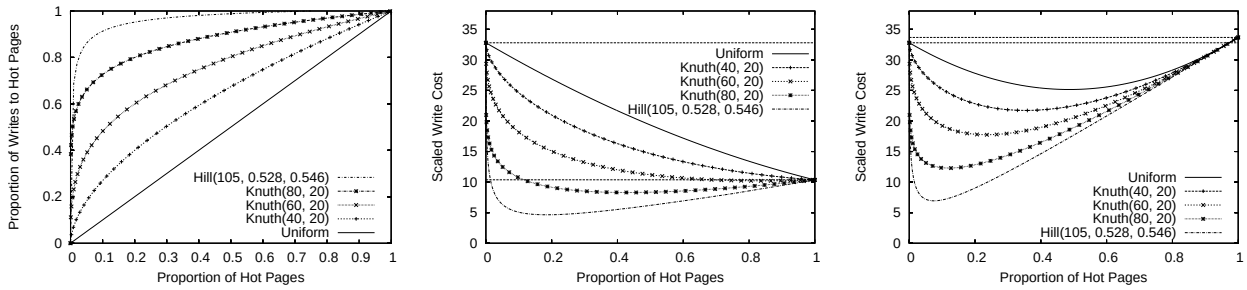
4.2 Separating Algorithm

Before a page is written to the disk, HyLog runs a *separating algorithm* to determine if this page is hot. If it is, the write is delayed and the page is stored temporarily in an in-memory segment buffer. Otherwise, it is immediately overwritten to its original place on the disk. When hot pages fill up the segment buffer, they are written out to a free disk segment, freeing the disk space occupied by their old copies.

As time goes on, some hot pages may become cold. These pages are written to the cold partition rather than to their current locations in the hot partition to avoid extra cleaning overhead. As some cold pages become hot and are written to the hot partition, free space may appear in the cold partition. To reclaim this free space more effectively, HyLog uses an adaptive cleaning algorithm to select segments with the highest cleaning benefit from both hot and cold partitions.

Accurately separating hot pages from cold pages is the key to the design of HyLog, as shown in Figure 4. The basic idea of the separating algorithm is simple. First, the write frequencies of recently updated pages are collected. These write frequencies are used to get the relationship between w and h. Then Equation (8) is used to calculate C'_{hylog} for all h. The hot page proportion h with the lowest C'_{hylog} is used as the expected hot page proportion.

Accurately measuring η is important for HyLog to make correct decisions. The service time of page I/O and segment I/O of each request is collected at the disk level. The average of the most recent 10,000 requests is used to compute η. Since a segment I/O always keeps all disks busy, while one page I/O only keeps one disk busy, page I/O is less efficient in disk arrays. If the proportion of the disk idle time is



(a) Cumulative Distribution Functions (b) Disk Space Utilization 90% (c) Disk Space Utilization 97%

Figure 4: Performance Potential of HyLog.

[The two horizontal lines in Figure 4(b) and 4(c) represent the write cost of Overwrite and LFS, respectively. η is 32.8, representing Cheetah X15 36LP with 1MB segment size and 8KB page size. $Knuth(a, b)$ means a% of the references go to b% of the pages.]

P_{idle}, η is adjusted to $\eta/(1 - P_{idle})$.

The write frequencies of all disk pages are collected in real time. A frequency counter is associated with each page. This counter is initialized to 0, and reset to 0 after every *measurement interval*. Whenever a page is written to the disk, its frequency counter is incremented. At the end of each measurement interval, all frequency counters are sorted in a descending order and stored in an array, which is used to calculate hot writes given the hot page proportion. The separating algorithm is invoked every measurement interval. After the expected hot page proportion is obtained, a page separating threshold can be determined so that all pages with write frequencies no less than the threshold are considered hot pages.

Preliminary experiments were conducted to study the sensitivity of system performance to the value of the measurement interval. When the measurement interval is smaller than 20 minutes, the throughput is not sensitive to the measurement interval. However, the throughput starts dropping with larger measurement intervals. Since the separating algorithm is invoked every measurement interval, 20 minutes is used as the measurement interval to reduce the separating algorithm overhead.

4.3 Segment Cleaning Algorithm

We adapted HyLog's segment cleaning algorithm from the adaptive cleaning algorithm [14], which dynamically selects between cost-age cleaning and hole-plugging based on their write cost.

In Hylog, the cleaner is invoked whenever the number of free segments is below a threshold (set to 10). In every cleaning pass, the cleaner processes up to 20MB of data. It first calculates the cost-benefit values of the following four possible cleaning choices: (1) cost-age in the hot partition, (2) hole-plugging in the hot partition, (3) cost-age in the cold partition, and (4) hole-plugging in the cold partition. It then performs cleaning using the scheme with the lowest cost-benefit value.

5 Methodology

5.1 The Simulator, Verification, and Validation

We used trace-driven simulations to compare the throughput of different disk layouts. Our simulator consists of a disk component, a disk layout component, and a buffer pool component. We ported the disk component from DiskSim 2.0 [5]. The disk layout component simulates disk layouts for Overwrite, LFS, WOLF, and HyLog. The implementation of LFS is based on the description in [14, 19] and the source code of the Sprite operating system [23]. The implementation of WOLF is based on the description in [26]. The buffer pool component uses the LRU algorithm. The three components communicate through an event-driven mechanism. Overwrite, LFS and WOLF are implemented as special cases of HyLog. By considering all pages as cold, we get Overwrite, and, by treating all pages as hot, we get LFS and WOLF. Therefore, the only difference between these disk layouts is the page separating algorithm. This guarantees the fairness of the performance comparison. We validated the simulator carefully in this subsection to improve the credibility of our performance comparisons.

In order to verify the disk layout component, a simple disk layout simulator called *TinySim* was developed independently. TinySim simulates LFS and WOLF, and supports single user and single disk. TinySim and the disk layout component were

run under uniformly distributed random update and hot-cold (10% of the pages are referenced 90% of the time) synthetic workloads, respectively. The overall write cost, which is the key performance metric of LFS and WOLF [14, 26], was obtained from both simulators. In most cases, the differences between the results of the two simulators were within 5%.

After verification, the cleaning algorithms in the disk layout component were validated against those discussed in [14]. Figure 5 shows the overall write costs of the cost-age, hole-plugging, and adaptive cleaning algorithms under a uniformly distributed random update workload. These cleaning algorithms show trends very similar to those in Figure 6 of [14].

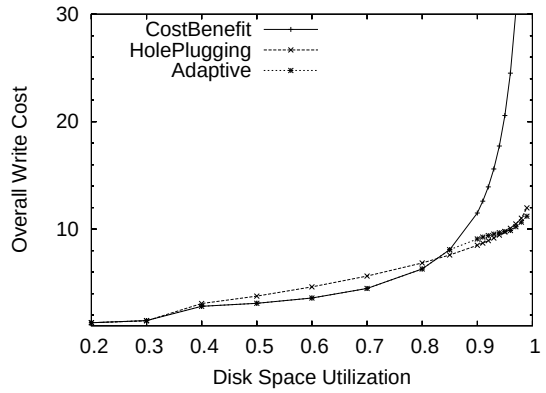


Figure 5: Validation of Overall Write Cost of LFS. [The workload is uniform random update. Page size is 8KB, segment size is 256KB, $T_{pos} = 15ms$, $B = 5MB/s$, and cache size is 128MB.]

We further validated our implementations of Overwrite and LFS by comparing their performance with that of FFS (uses overwrite) and BSD LFS published in [21] under the TPC-B benchmark. As in [14], we used the uniformly distributed random update workload to simulate the TPC-B benchmark. Since the DSP 3105 disk used in [21] is not available in DiskSim, a similar disk, the DEC RZ26, was used in the validation experiments. Table 2 lists the specifications of the two disks. The DEC RZ26 has 3% slower average seek time and slightly higher transfer rate because it has one more sector per track than the DSP 3105.

Table 3 shows the throughput of Overwrite and LFS obtained from our simulator and that in [21]. Although the reported throughput of LFS with cleaning in [21] was 27.0, it has been argued [17] that 34.4 should be a more reasonable value. Therefore, 34.4 is used here when calculating the difference.

Table 2: Disk Comparison for Simulator Validation

Parameters	DSP 3105	DEC RZ26
RPM	5400	5400
Sectors/Track	57	58
Cylinders	2568	2599
Platters	14	14
Track Buffer	256KB	285KB
Avg. Seek Time	9.5ms	9.8ms
Transfer Rate	2.3MB/s	2.3MB/s

The 4.8% lower throughput we observed for Overwrite in our experiments may be due to the 3.2% slower seek time of the DEC RZ26. The LFS implementation used in [21] can achieve only 1.7MB/s write throughput, 26% slower than the maximum hardware bandwidth, because of "missing a rotation between every 64 KB transfer". Since the number of segment reads is equal to the number of segment writes (for every segment read, there is always u segment write for cleaning and $1-u$ segment write for new data), this slowdown of segment write should cause 13% performance difference, which matches the difference in Table 3. Since the differences in all results are within a reasonable range, we believe that our implementations of Overwrite and LFS are valid.

Table 3: Throughput Validation. $u_d = 50\%$.

Layout	Previous[21]	Ours	Diff.
FFS/Overwrite	27.0	25.7	-4.8%
LFS w/o cleaning	41.0	43.3	5.6%
LFS w cleaning	**27.0** (**34.4**)	39.0	13.4%

5.2 The Workloads

We used three traces in our experiments: TPC-C™, Financial, and Campus. Their characteristics are summarized in Table 4.

The TPC-C benchmark is a widely accepted benchmark for testing the performance of database systems running OLTP workloads developed by the Transaction Processing performance Council (TPC) [24]. The TPC-C trace contains all read and write requests to the buffer pool when running the benchmark on IBM® DB2® 7.2 on Windows NT® Server 4.0. The scale of the TPC-C benchmark is expressed in terms of the number of warehouses represented. The database used in this study contains 50 warehouses. The TPC-C trace was collected using a tracing package ported from [8].

The Financial trace [22] was published by the Storage Performance Council. It is a disk I/O trace

Table 4: Trace Characteristics

	TPC-C	Financial	Campus
Data size(MB)	5088	10645	9416
Page size(KB)	4	4	8
#reads($\times 10^6$)	176.27	0.97	21.05
#writes($\times 10^6$)	31.17	3.41	7.64
Logical reads/writes	5.66	0.28	2.76
Physical reads/writes	1.37	0.13	2.56

of an OLTP application running at a large financial institution. Since the trace was collected at the I/O controller level, many reads have already been filtered out by the in-memory buffer. This trace contains I/O requests to 23 containers. In our experiments, the requests to the three largest containers were ignored to reduce the resources required by the simulator. Since these three containers have the fewest requests relative to their sizes, this omission has little impact on our results.

The Campus trace is a one-day trace taken from the NFS trace collected at Harvard University campus in October 2001 [3]. This trace is dominated by email activities. It contains reads, writes and directory operations to the NFS server. Reads and writes make up 85% of the requests. Since the sizes of the directories are unknown from the trace, it is difficult to replay the directory operations in the simulator, but because we assume NVRAM is used, these directory operations do not cause expensive synchronized writes. So their impact on performance is small. We discard the directory operations and use only the reads and writes.

5.3 Experimental Setup

Since our interest is the performance of various disk layouts on busy systems, we configured the simulator as a closed system without think time, i.e., the next trace record is issued as soon as the processing of the previous one finishes. Using this method, we were able to use traces with a small number of users to represent the workloads imposed on a system by many more users with think time. For example, the workload generated by 30 users without think time with 1.28 seconds average response time is equivalent to that generated by about 500 users with 21 seconds[1] average think time between requests. The details of this deduction are in the appendix.

We used the Quantum atlas10k 1999 disk model, the latest disk model provided by DiskSim. Its specifications are given in Table 5. Write-back caching is disabled to protect data loss from power failure. The disk scheduling algorithm is SCAN based on logical page numbers.

[1]The weighted average think time plus keying time defined in Clause 5.2.5.7 of TPC-C benchmark version 5.0

Table 5: Disk Specifications

Parameters	Atlas10k (Year 1999)	Year 2003 Disk	Year 2008 Disk
RPM	10025	15000	24000
Sectors/Track	229-334	476	967
Cylinders	10042	10042	10042
Platters	6	8	8
Size(GB)	9.1	18	36
Seek Time(ms)	5.6	3.6	2.0
Bandwidth(MB/s)	20.4	61	198

To study the performance of disk layouts on today's and future disks, we also designed models for a high-end disk of year 2003 and a high-end disk of the sort we might imagine to appear in year 2008. Looking back over 15 years history of disk technology evolution, we made the following assumptions: every 5 years, transfer rate increases by 242% [6], average seek time decreases by 76% [20], and RPM (Rotations Per Minute) increases by 61% [1]. We also assumed that all cylinders have the same number of tracks, the number of platters is 8, and the disk size is 18GB for the year 2003 disk and 36GB for the year 2008 disk. The specifications of these two disks calculated on the basis of these assumptions are given in the two rightmost columns of Table 5. The seek time distribution data were created by linearly scaling the seek time distribution of the atlas10k disk defined in DiskSim.

We used RAID-0 and RAID-5 as multi-disk models. The stripe size for both RAID-0 and RAID-5 is computed based on Equation (6) and then rounded to the closest powers of two. For RAID-0 arrays with n disks, the segment size is $n \times StripeSize$. For RAID-5 arrays, the segment size is $(n-1) \times StripeSize$, since $1/n$ of the total disk space is dedicated to parity data.

In order to vary the disk space utilization, only part of the disk is accessed, independent of the actual size of the disk. For example, if the data size is 6GB and the disk space utilization is 60%, the total disk space required is $\frac{6GB}{60\%} = 10GB$. If there are 5 disks, the first 2GB of each disk is used. Since the disk layout approaches do not handle page allocation and deallocation, all data are stored on the former part of the disk initially. As a result, the seek time (particularly for Overwrite) is very short,

which makes η smaller. Thus this data layout makes the performance results for LFS, WOLF, and HyLog conservative compared to Overwrite than in real workloads where data are often placed far apart. For LFS, WOLF, and HyLog, the data will eventually spread across the whole disk as data are written, which is considered as the warmup period. Only the performance data collected after the warmup period is considered.

The performance metric used in this paper is throughput, defined as the number of I/O requests finished per second.

Table 6 summarizes the parameters and values used in our experiments. Since these parameters can be easily controlled in the TPC-C trace, this trace is used to study the impacts of various parameters on throughput. When evaluating the throughput of RAID-5, we compare a 9-disk RAID-5 array with an 8-disk RAID-0 array so that they have the same segment size.

Table 6: Experimental Parameters

Configuration	Range	Default
Disk layout	Overwrite, LFS, WOLF, HyLog	—
Number of users	1–30	20
Number of disks	1–15	4
Disk utilization	0.5–0.98	—
Disk type	atlas10k, year 2008 disk, year 2003 disk	atlas10k
Disk array type	RAID-0, RAID-5	RAID-0
Workload	TPC-C, Financial, Campus	TPC-C
Buffer pool size	—	400MB

6 Simulation Results

6.1 Validation of the Cost Model

Since the cost model was developed for uniform random update workload, we use results for the same workload to validate the cost model. In particular, we use previous results for TPC-B (reported in [17, 21]), a random update workload, to verify the throughput of LFS and Overwrite. Since the write cost is the average time required to write a page and a transaction requires a page read and a page write, the throughput X is computed as

$$X = \frac{1}{T_{pg} + C + T_{cpu}},$$

where C is the write cost of the disk layout, and T_{cpu} is the CPU overhead for processing each page, which is $0.9ms$ for Overwrite and $1.8ms$ for LFS [21]. The results in Table 7 show that the model matches the measurement results well.

Table 7: Cost Model Validation

Layout	Previous	Model	Difference
Overwrite	27.0 [21]	28.6	6.0%
LFS-cleaning	34.4 [17]	37.3	8.4%

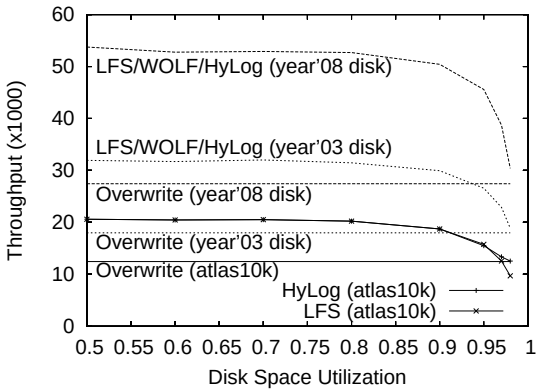
6.2 Impact of Various Factors

Disk Space Utilization and Disk Type

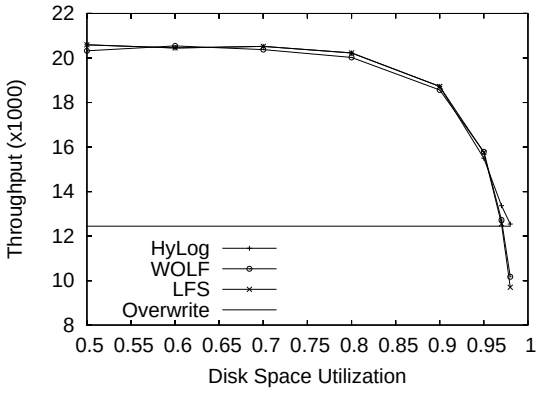
Figure 6(a) shows the throughput of different layouts under various disk space utilization and different disks. Since the throughput of LFS, WOLF, and HyLog almost overlaps for the year 2003 and year 2008 disks, only one line is shown for each of these disks. The throughput of all layouts improves with faster disks. The throughput of Overwrite is not affected by the disk space utilization, while the throughput of other layout approaches decreases when the space utilization is high. The faster the disk, the more LFS and WOLF can tolerate the high space utilization because faster disks have higher η as shown in Figure 1. Figure 6(b) gives a closer look at the throughput of the atlas10k disk. The throughput of WOLF overlaps that of LFS for most configurations and outperforms LFS by 5% when the disk space utilization u_d is very high (98%). The throughput of HyLog overlaps that of LFS when $u_d \leq 95\%$. This is because HyLog considers all pages as hot based on its cost model Equation (8) (see Figure 4(b)). The throughput of HyLog is comparable with Overwrite when the disk space utilization is higher. HyLog outperforms Overwrite by 7.4% when the disk space utilization is 97%.

To provide some insights into the performance that LFS and HyLog show above, we give further analysis of two example points: LFS running on atlas10k with 95% disk space utilization and HyLog running on atlas10k with 98% disk space utilization.

In the LFS example, the measured cleaning space utilization is $u = 88.4\%$. This is lower than the 90.2% computed from Equation (2) because of the skewness of accesses in TPC-C. Therefore, to write one segment of new data, $\frac{1+u}{1-u} = 16.3$ segment I/Os need to be performed for cleaning. So the cleaning traffic contributes 94.2% to the total segment I/O traffic. The measured T_{pg} and T_{seg} from simulation is 5.6ms and 27.3ms, respectively. Therefore, η = 26.3. The measured proportion of disk idle time is



(a) All Disks



(b) Atlas10k Disk

Figure 6: The Impact of Disk Space Utilization on System Throughput.

[The number of users is 20, the number of disks is 4, the trace is TPC-C, and the buffer pool size is 400MB.]

30%, so η should be adjusted to $\eta/(1-30\%) = 37.6$. Based on the scaled write cost model,

$$C'_{ow}/C'_{lfscleaning} = \eta(1-u)/2 = 2.2,$$

which means the write throughput of LFS is 2.2 times of the write throughput of Overwrite. Since the write traffic contributes 42% to the total traffic after filtering by the buffer cache (Table 4), LFS outperforms Overwrite by 30%, which is close to the simulation result of 27%.

In the HyLog example, the hot page proportion selected by the page separating algorithm during the run is 35-45%. We use the data collected at the first measurement interval after warmup as the example. The proportion of hot pages is 42.2%, and the proportion of hot writes is 58.2%. The measured cleaning space utilization is 93.4%, which is lower than that in LFS for the same configuration (96.2%). The proportion of disk idle time is 22.5%, the measured T_{pg} and T_{seg} are 5.8ms and 27.2ms, respectively, and the adjusted η is $\frac{T_{pg}S}{T_{seg}(1-P_{idle})} = 35.2$. Therefore, the write cost model indicates that the write throughput of the hot partition is 16% higher than Overwrite. Thus the overall weighted write throughput is 9% higher than Overwrite. Taking the read traffic into account, the throughput of HyLog is 1.036 that of Overwrite, which is close to the simulation result of 1.008. The write throughput of LFS computed from the cost model under 98% disk space utilization is 66.9% of Overwrite, and the overall throughput of LFS including read and write traffic is 82.8% of Overwrite, which is close to the simulation result of 78.0%.

Figure 7 shows how well the separating algorithm works. The hot page proportion found by the separating algorithm (35-45%) is close to optimal, and the achieved throughput is 96.4% of the maximum possible throughput.

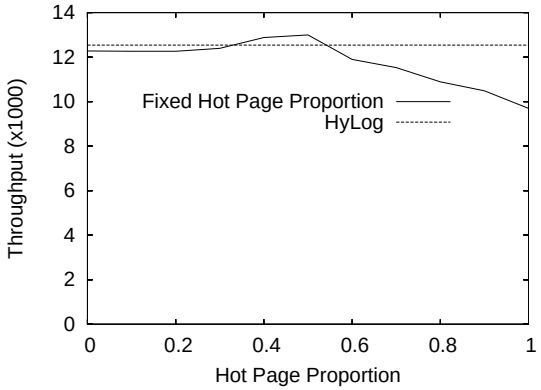


Figure 7: Sensitivity of Separating Criteria.

[The number of users is 20, the number of disks is 4, the trace is TPC-C, the disk space utilization is 98%, and the buffer pool size is 400MB.]

Impact of Number of Users and Number of Disks

Figure 8 shows the throughput normalized to Overwrite under different numbers of users and disks. WOLF is not shown since it almost overlaps with LFS. Two trends can be observed in the relative throughput of LFS, WOLF, and HyLog: (1) it drops with more users; (2) it drops with more disks.

With more users, the average disk seek time is reduced because of the disk scheduling algorithm, which reduces η. The disk idle time in Overwrite is also reduced with more users. Therefore, the first trend happens in both low disk space utilization

(Figure 8(a)) and high disk space utilization (Figure 8(b) and 8(c)).

With more disks, the segment size is larger, thus the cleaning cost is higher [14], which reduces the benefit of the log-structured layout. This happens only when cleaning cost plays an important role, which is true when the disk space utilization is high. Therefore, the second trend is apparent only when the disk space utilization is high (Figure 8(b)).

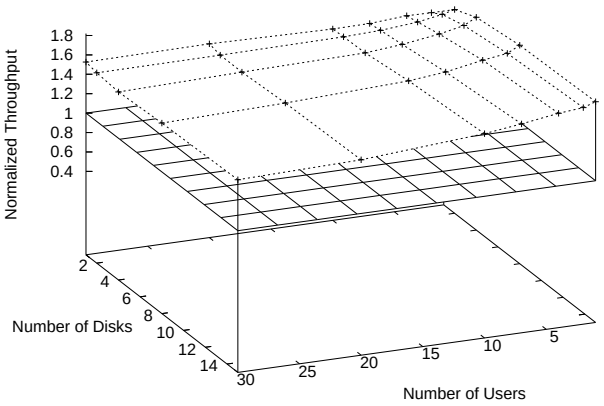
In Figure 8(b) and 8(c), the throughput of HyLog overlaps with that of LFS when LFS outperforms Overwrite, and HyLog becomes comparable with Overwrite when Overwrite outperforms LFS. HyLog incorrectly follows LFS when there are 4 users and 15 disks, because at this configuration, a very small error in the estimation of η can cause HyLog to make the wrong decision, while HyLog can tolerate some error in the estimation of η in other configurations.

Impact of Disk Array Type

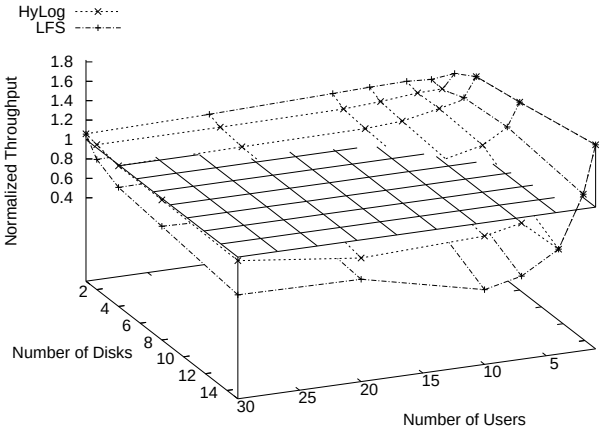
Figure 9 shows the throughput of the four disk layouts (Overwrite, LFS, WOLF, and HyLog) on RAID-0 and RAID-5. For Overwrite, the throughput on RAID-5 is about 50% of that on RAID-0. This performance degradation is caused by the slower page update of RAID-5. For LFS and WOLF, the use of RAID-5 increases throughput by 6.5-10%, because the segment I/O performance is not affected by RAID-5, while the one more disk in RAID-5 increases the page read throughput. When the disk space utilization is high, the throughput of HyLog on RAID-0 is comparable with Overwrite. The throughput of HyLog on RAID-5 is comparable with LFS because the slower page I/O in RAID-5 makes η higher. Thus most pages are considered to be hot pages.

6.3 Results for the Other Workloads

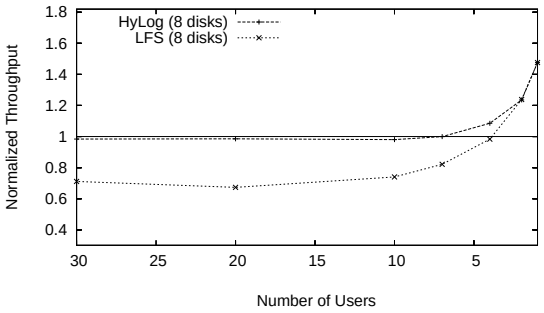
Figure 10 shows the throughput of the four disk layouts using the Financial and Campus traces. The throughput is normalized relative to that of Overwrite. For both traces, the performance advantage of LFS, WOLF, and HyLog is much higher than that observed with the TPC-C trace. This difference is attributed to two facts. First, the distribution of data updates in the Financial and Campus traces is more skewed than it is in the TPC-C trace, leading to less cleaning cost. Second, the proportion of writes in these two traces is much higher than in the TPC-C trace, since many reads have already been filtered out by client-side buffers (in the Campus trace) or in-memory buffers (in the Financial trace). Since the Financial trace is more skewed than the Campus trace and the writes in the Financial trace have higher proportion than in the Campus trace, the advantage of log-structured layouts in the Financial trace is higher than in the Campus trace. The performance results under other configurations



(a) Disk Space Utilization 90%



(b) Disk Space Utilization 98%



(c) Disk Space Utilization 98%

Figure 8: Impact of Number of Users and Number of Disks.
[Disk is atlas10k, trace is TPC-C. In Figure 8(a), the throughput curves of LFS and HyLog almost overlap, thus only the throughput of LFS is drawn. Figure 8(c) shows the hidden data points of Figure 8(b).]

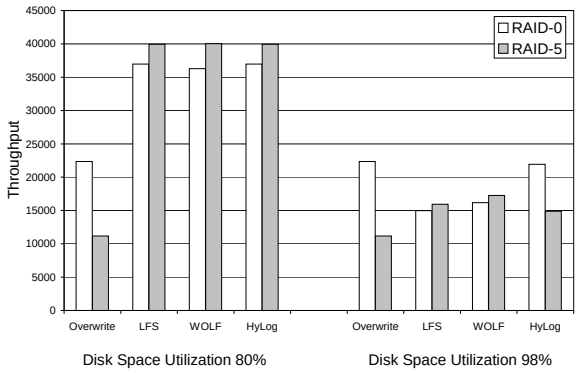


Figure 9: Throughput under RAID-0 and RAID-5 arrays.

[The RAID-0 array contains 8 disks, the RAID-5 array contains 9 disks, the disk is the atlas10k, the trace is TPC-C, the number of users is 20, the segment size is 512KB per disk, and the buffer pool size is 400MB.]

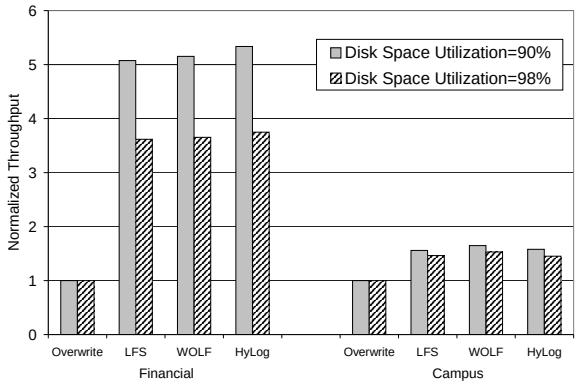


Figure 10: Throughput of LFS, WOLF, and HyLog normalized relative to Overwrite for the Financial and Campus traces.

[The number of disks is 1, the segment size is 512KB, and the buffer pool size is 400MB.]

show similar trends.

7 Conclusions and Future Work

In this paper we study the write performance of the Overwrite and LFS disk layouts. A write cost model was developed to compare the performance of these disk layouts. Contrary to the common belief that its high cleaning cost disadvantages LFS, we found that because of the advancement of disk technologies, the performance of LFS is significantly better than Overwrite even under the most pathological workload for LFS (random update), unless the disk space utilization is very high.

Since LFS still performs worse than Overwrite under certain conditions such as high disk space utilization, we propose a new disk layout model called HyLog. HyLog uses a log-structured approach for hot pages to achieve high write performance and overwrite for cold pages to reduce the cleaning cost. The page separating algorithm of HyLog is based on the write cost model and can separate hot pages from cold pages dynamically. Our results on a wide range of system and workload configurations show that HyLog performs comparably to the best of Overwrite, LFS, and WOLF in most configurations.

As future work, we want to study the read performance of LFS and HyLog and the impact of disk technology on them.

Acknowledgements

We would like to thank our shepherd Edward Lee and the anonymous reviewers for their insightful comments, which greatly improved the paper. We also thank Professor Derek Eager, Professor Dwight Makaroff, and Greg Oster for their feedback. Our system administrators, Greg Oster, Brian Gallaway, Cary Bernath, and Dave Bocking, provided the enormous computing resources for this study, and Daniel Ellard provided the Campus trace. This work was supported by IBM's Centre for Advanced Studies (CAS) and the Natural Sciences and Engineering Research Council of Canada (NSERC).

Appendix

The average think time plus keying time defined by the TPC-C benchmark is 21 seconds. The simulation results indicate that the system with 30 users without think time has a response time of 1.28 seconds if one disk is present in the system. Assuming that the number of users with think time in the system is n, the average arrival rate of users is $\frac{n}{21+1.28} = \frac{n}{22.28}$. From Little's Law, we have: $30 = \frac{n}{22.28} \times 1.28$. Therefore, $n = 522 \approx 500$, which indicates that the workload generated by 30 users without think time presents equivalent workload to that generated by about 500 users with 21 seconds think time between requests.

References

[1] D. Anderson, J. Dykes, and E. Riedel. More than an interface — SCSI vs. ATA. In *Proc. FAST'03*, pages 245–257, San Francisco, CA, March 2003.

[2] T. Blackwell, J. Harris, and M. Seltzer. Heuristic cleaning algorithms in log-structured file systems. In *Proc. USENIX ATC'95*, pages 277–288, New Orleans, LA, January 1995.

[3] D. Ellard, J. Ledlie, P. Malkani, and M. Seltzer.

Passive NFS tracing of email and research workloads. In *Proc. FAST'03*, pages 203–216, San Francisco, CA, March 2003.

[4] G. Ganger, M. McKusick, C. Soules, and Y. Patt. Soft updates: a solution to the metadata update problem in file systems. *ACM Trans. on Computer Systems*, 18(2):127–153, 2000.

[5] G. Ganger, B. Worthington, and Y. Patt. *The DiskSim Simulation Environment Version 2.0 Reference Manual*, December 1999.

[6] J. Gray and P. Shenoy. Rules of thumb in data engineering. In *Proc. 16th Intl. Conf. on Data Engineering*, pages 3–12, San Diego, CA, February 2000.

[7] D. Hitz, J. Lau, and M. Malcolm. File system design for an NFS file server appliance. In *Proc. USENIX Winter ATC'94*, pages 235–246, San Francisco, CA, January 1994.

[8] W. Hsu, A. Smith, and H. Young. Characteristics of production database workloads and the TPC benchmarks. *IBM Systems Journal*, 40(3):781–802, 2001.

[9] Y. Hu and Q. Yang. DCD – disk caching disk: A new approach for boosting I/O performance. In *Proc. Intl. Symp. on Computer Architecture*, pages 169–178, Philadelphia, PA, May 1996.

[10] L. Huang and T. Chiueh. Experiences in building a software-based SATF disk scheduler. Technical Report ECSL-TR81, State University of New York, Stony Brook, March 2000. revised in July, 2001.

[11] D. Knuth. *The Art of Computer Programming – Sorting and Searching*, volume 3. Addison-Wesley, second edition, 1998.

[12] D. Lomet. The case for log structuring in database systems. In *Proc. Intl. Workshop on High Performance Transaction Systems*, September 1995.

[13] C. Lumb, J. Schindler, and G. Ganger. Freeblock scheduling outside of disk firmware. In *Proc. FAST'02*, pages 275–288, Monterey, CA, January 2002.

[14] J. Matthews, D. Roselli, A. Costello, R. Wang, and T. Anderson. Improving the performance of log-structured file systems with adaptive methods. In *Proc. SOSP'97*, pages 238–251, Saint-Malo, France, October 1997.

[15] J. Menon. A performance comparison of RAID-5 and log-structured arrays. In *IEEE Symp. on High-Performance Distributed Computing*, pages 167–178, Charlottesville, VI, August 1995.

[16] J. Menon and L. Stockmeyer. An age-threshold algorithm for garbage collection in log-structured arrays and file systems. IBM Research Report RJ 10120, IBM Research Division, San Jose, CA, 1998.

[17] J. Ousterhout. The second critique of Seltzer's LFS measurements. `http://www.eecs.harvard.edu/~margo/usenix.195/ouster_critique2.html`.

[18] J. Ousterhout and F. Douglis. Beating the I/O bottleneck: A case for log-structured file systems. *Operating Systems Review*, 23(1):11–28, January 1989.

[19] M. Rosenblum and J. Ousterhout. The design and implementation of a log-structured file system. *ACM Trans. on Computer Systems*, 10(1):26–52, 1992.

[20] J. Schindler, J. Griffin, C. Lumb, and G. Ganger. Track-aligned extents: Matching access patterns to disk drive characteristics. In *Proc. FAST'02*, pages 259–274, Monterey, CA, January 2002.

[21] M. Seltzer, K. Smith, H. Balakrishnan, J. Chang, S. McMains, and V. Padmanabhan. File system logging versus clustering: A performance comparison. In *Proc. USENIX ATC'95*, pages 249–264, New Orleans, LA, January 1995.

[22] Storage Performance Council I/O traces. `http://traces.cs.umass.edu/storage/Financial1.spc`.

[23] The Sprite operating system. `http://www.cs.berkeley.edu/projects/sprite/sprite.html`.

[24] Transaction Processing Performance Council. `http://www.tpc.org/`.

[25] J. Wang and Y. Hu. PROFS–performance-oriented data reorganization for log-structured file systems on multi-zone disks. In *Proc. MASCOTS'01*, pages 285–292, Cincinnati, OH, August 2001.

[26] J. Wang and Y. Hu. WOLF – a novel reordering write buffer to boost the performance of log-structured file systems. In *Proc. FAST'02*, pages 46–60, Monterey, CA, January 2002.

[27] R. Wang, T. Anderson, and D. Patterson. Virtual log based file systems for a programmable disk. In *Proc. OSDI'99*, pages 29–43, New Orleans, LA, February 1999.

[28] Z. Zhang and K. Ghose. yFS: A journaling file system design for handling large data sets with reduced seeking. In *Proc. FAST'03*, pages 59–72, San Francisco, CA, March 2003.