Improving the Performance of Log-Structured File Systems with Adaptive Block Rearrangement

Mei-Ling Chiang
Department of Information Management
National Chi-Nan University, Puli, Taiwan, R.O.C.
joanna@ncnu.edu.tw

Jia-Shin Huang
Department of Information Management
National Chi-Nan University, Puli, Taiwan, R.O.C.
peterhuang1kimo@yahoo.com.tw

ABSTRACT
Log-Structured File System (LFS) is famous for its optimization for write performance. Because of its append-only nature, garbage collection is needed to reclaim the space occupied by the obsolete data. The cleaning overhead could significantly decrease the performance of file system. However, traditional cleaning policies do not consider the storage location where the valid data in the cleaned segments should be placed and rewritten to. In this paper, we propose a new method called *R-LFS* to dynamically reorganize data in disk to approximate the organ pipe heuristic that can place data in disk optimally. Basically, frequently accessed data are dynamically clustered and placed toward the center of disk, whereas less accessed data are moved and placed toward the edges of disk to reduce disk seek time. The essence of R-LFS is that R-LFS takes advantage of the chance of data reorganization during segment cleaning and data writing, no extra overhead is incurred for this data reorganization. Besides, because hot data and cold data are in nature separately clustered under R-LFS, cleaning overhead can be substantially reduced as well. Performance evaluation under both trace-driven simulation and practical implementation on NetBSD/LFS shows that R-LFS can effectively improve the performance of LFS.

Categories and Subject Descriptors
D.4.2 [**Operating Systems**]: Storage Management – *allocation/deallocation strategies, garbage collection.*

General Terms
Management, Performance, Design, Experimentation.

Keywords
Log-Structured File System, Garbage Collection, Data Rearrangement.

1. INTRODUCTION
Disk performance has become one of the major system bottlenecks of a computer system. Log-Structured File System (LFS) [8] improves write performance by writing all new data to disk in a sequential structure called the *log*. The log is divided into segments by writing segment data sequentially to disk, thus disk seek times can be significantly reduced and faster crash recovery is permitted. To ensure enough free space for writing new data, it requires a garbage collection process called *segment cleaner* to reclaim the storage space occupied by obsolete data. However, the performance of the cleaner greatly affects the system performance. Therefore, many researches [2,6,8,11-13] aim at improving cleaning performance.

To improve cleaning performance, several researches [4,11] show that separating hot data from cold data while performing garbage collection or writing data can significantly improve cleaning performance in log-based storage systems. This is because hot data have high possibility to be updated soon and the original space they occupy would become obsolete and invalid soon. Therefore, segments full of hot data would soon become garbage together. Cleaning these segments can be of low overhead. Among these researches, the *Dynamic dAta Clustering* (DAC) [3] technique is an efficient technique to dynamically cluster data according to their write access frequencies. It divides storage space into several regions. It moves hot data toward upper region during data writing and moves cold data toward lower region during garbage collection. In this way, data blocks with the similar write access frequencies would be effectively clustered in the same region. However, DAC is originally proposed for log-based flash memory storage systems and the disk characteristic is different from flash memory that has no seek time and no rotational latency at all in accessing stored data.

Researches [1,10] also show that clustering frequently referenced data and placing them near the center of disk can significantly reduce disk seek times. Especially, the *organ pipe* heuristic [5,14] can place data in disk optimally if data references are derived from an independent, random process with a known, fixed distribution. In this method, more frequently accessed data are placed toward the center of disk and less frequently accessed data are placed toward the edges of disk.

In this paper, we propose a new method called *Region-based LFS* (R-LFS) that combines the advantages of DAC and organ pipe heuristic methods to improve LFS performance. The R-LFS partitions disk space into several regions and the region organization in disk is an approximation to the disk layout of the organ pipe heuristic. The frequently accessed data clustered by DAC are placed toward the central region located at the center of disk. When a data block is updated, it will be promoted to the region near center of disk. When the cleaner selects a segment for garbage collection, the valid data in the selected segment will be demoted to the region toward edge of disk. In this way, hot data will be placed toward the central region and cold data will be placed toward the fringe region.

SAC'07, March 11-15, 2007, Seoul, Korea.

To evaluate the effectiveness of R-LFS on improving LFS performance, we have implemented R-LFS in the LFS of NetBSD. Experimental results show that cleaning overhead can be significantly reduced by 41.04-63.6%. Besides, a trace-driven simulation with real-world trace is also assisted to measure the performance improvement under various factors. Experimental results show that disk seek time and cleaning overhead can be greatly reduced as well.

2. USING R-LFS TO IMPROVE LFS PERFORMANCE

Though organ pipe heuristic can place data in disk optimally, the practical implementation is difficult since the access frequencies of data blocks that are used to place data in disk must be known in advance. Therefore, in practice, we can only implement a variation of organ pipe heuristic. However, the cost of reorganizing data in disk according to the organ pipe heuristic disk layout is expensive. So it is important to choose the right time and an efficient way to reorganize data in disk.

Our proposed Region-Based LFS (R-LFS) combines the advantages of DAC and organ pipe heuristic methods to improve LFS performance. The basic idea is to partition disk space into several regions and uses DAC technique to cluster frequently accessed data toward the central region located at the center of disk while cluster less accessed data toward the outer region located at the edges of disk. Therefore, the region organization of R-LFS in disk is an approximation to the disk layout of the organ pipe heuristic.

In the R-LFS disk layout, a disk is divided into several regions, and each region consists of several adjacent cylinders. The hot data is defined to be a block updated frequently, and the cold data is defined to be a block less frequently updated. In R-LFS, the hottest data would be placed in the hottest region located at the center disk, while the coldest data would be placed in the coldest region located at the edges of disk. For example, a R-LFS disk layout with three regions is shown in Figure 1. Suppose each region can accommodate M data blocks, then area A is the hottest region (i.e. region 3) which can store M hottest data blocks. Two areas B belonging to region 2 store M next hottest data blocks. Two areas C belonging to region 1 store M coldest data.

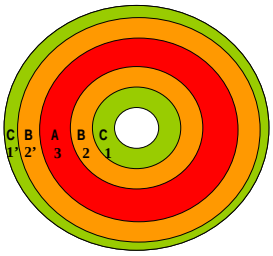
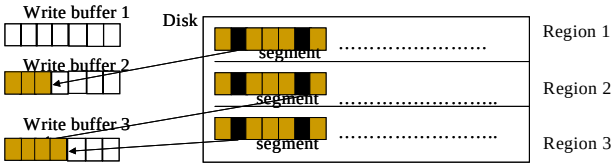
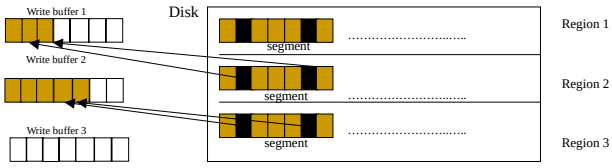


Figure 1. Disk region layout.

Reorganizing data in disk occurs at the time of data writing and garbage collection. Each region has a write buffer for accumulating dirty blocks. Figure 2(a) shows the updating process. When a data block is updated, the block will be promoted and accumulated in the write buffer of the upper region. When a write buffer becomes full, the data in it will be flushed into the region in disk. Figure 2(b) shows the cleaning process. When a segment is selected for cleaning during garbage collection, the valid data in it will be demoted and accumulated in the write buffer of the lower region. When a write buffer becomes full, it will be flushed into the region in disk.



(a) Update to upper region for updating process



(b) Copy valid blocks to lower region for cleaning

Figure 2. The updating process and cleaning process.

In comparison, when performing the cleaning, the original LFS does not consider the disk location where the valid data in the cleaned segment should be placed, nor does it consider the disk location where the data in write buffer should be flushed into. Whereas, R-LFS takes advantage of the chance of reorganizing disk data without paying extra overhead when performing the operations of data writing and garbage collection in LFS.

3. DESIGN AND IMPLEMENTATION OF R-LFS ON NETBSD/LFS

We have designed and implemented R-LFS technique in LFS on NetBSD [7]. Mainly two operations of LFS should be changed: modification of writing a log and modification of the cleaner.

3.1 Modification of Writing a Log

In R-LFS, a disk is partitioned into several regions and R-LFS separates hot data from cold data into different regions. Every flush operation should just flush dirty blocks in one region. Hot data should be placed into upper region located toward the center of disk, and cold data should be placed into lower region located toward the edges of disk.

Four modifications are needed when NetBSD/LFS needs to write a log to disk. First, every region needs a counter to record the number of locked buffers. Secondly, the lfs_check() and lfs_writerd() functions need to use those new counters to decide which region should be flushed. When the lfs_check() or lfs_writerd() functions detect that there are too many locked dirty buffers in one region, it will invoke the lfs_segwrite() function. Thirdly, the lfs_segwrite() then invokes the lfs_writevnodes() function to traverse the vnode list. If a vnode does not belong to the target region to be flushed, it is skipped. Finally, because dirty buffers should be written to upper region, so the address of the disk segment to be written should be changed.

3.2 Modification of the Cleaner

In the R-LFS, cleaning process will copy the valid data in the victim segments to lower region. Therefore, the cleaner should be modified for this change. Mainly four parts need modification. First, the amount of empty segments should be separately counted for each region. If the amount of empty segments is not enough in one region, it starts cleaning. Secondly, the cost-benefit policy is used to calculate the cost of every segment in this region, and then selects the segment with lowest cost to clean. Thirdly, the valid blocks of those selected segments and the corresponding region number of those segments are contained in the valid blocks array which is then sent to the kernel. Finally, the kernel uses these information and starts the clean operation.

4. SIMULATION STUDY

4.1 The Simulator and Traces

We have implemented a trace-driven simulator to simulate the read, write, and cleaning operations of a log-based file system. Each input trace record contains a block-level I/O request, including read or write, request length, block number, and time stamp. Table 1 summarizes the input and output parameters of our simulator. Table 2 shows all cleaning policies implemented in the simulator. "Cost-1" and "Greedy-1" mean R-LFS is not used, which represent the original LFS performance.

Table 1. Simulator parameters

Input	Output
1. Disk parameters 2. Block size, segment size 3. Cleaning policy (greedy, cost-benefit)[8] 4. Initial disk utilization 5. Trace (hplajw, snake) 6. The number of regions	1. The number of disk seek distance 2. The number of blocks copied during cleaning

Table 2. Cleaning policies

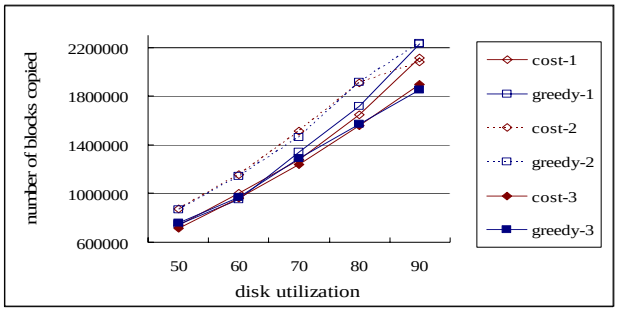
Cleaning policy	Description
Cost-1	Using cost-benefit policy
Cost-2	Using R-LFS (2 regions) and cost-benefit policy
Cost-3	Using R-LFS (3 regions) and cost-benefit policy
Greedy-1	Using greedy policy
Greedy-2	Using R-LFS (2 regions) and greedy policy
Greedy-3	Using R-LFS (3 regions) and greedy policy

Two real-world traces are used in our experiments. One is hplajw trace [9] which recorded two months of access behaviors in a personal workstation at HP Laboratories. The main jobs in this workstation were sending e-mail and editing text files. The system had two disks attached to it. The other is snake trace [9] which also recorded two months of access behaviors in a cluster server at UC Berkeley. The main use was for compilation and editing. The system had three disks attached to it.

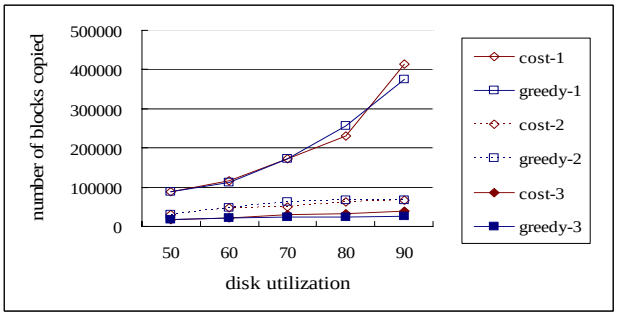
We evaluate the cleaning overhead in terms of the number of blocks copied during the cleaning process, which is equal to the amount of valid blocks in the selected segments that are moved and copied to other free storage space by the cleaner. The operation of copying blocks is the main overhead in executing garbage collection. The amount of disk seek distance is another metric used to measure the performance improvement.

4.2 Effect on Cleaning Overhead

When garbage collection occurs, the valid data in the cleaned segments should be read out and later written back to other empty disk segment. When the disk utilization increases, it needs to execute garbage collection more times, so the amount of blocks copied increases as well. Figures 3 and 4 show the performance comparison results of using different cleaning policies. The amount of blocks copied is significantly reduced by R-LFS. When R-LFS with three regions is used, the amount of blocks copied can be reduced by up to 90.48% for cost-benefit policy and 93.22% for greedy policy under hplajw trace, and 99.89% for cost-benefit policy and 99.9% for greedy policy under snake trace.

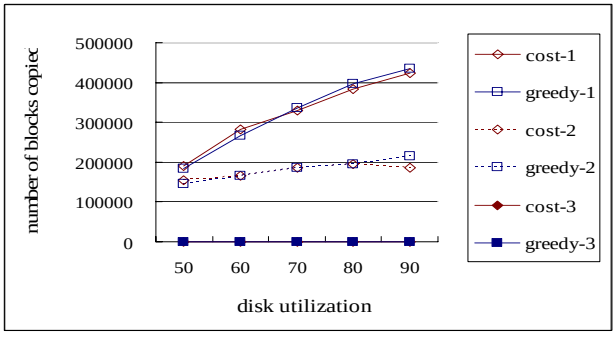


(a) Number of blocks copied (root disk)

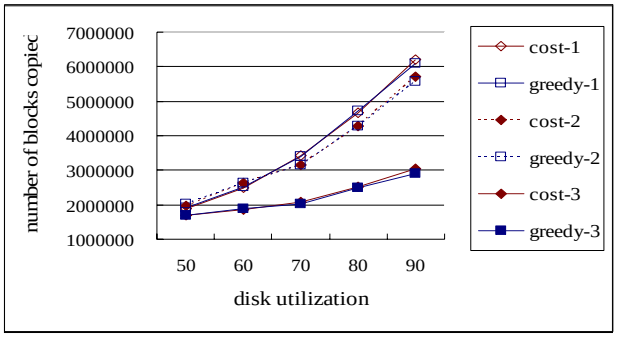


(b) Number of blocks copied (swap disk)

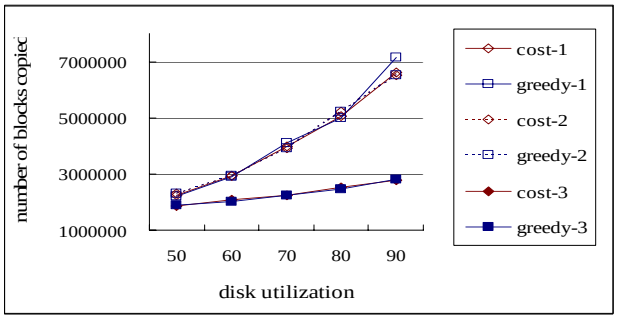
Figure 3. Cleaning overhead comparison (hplajw trace).



(a) Number of blocks copied (root disk)



(b) Number of blocks copied (/usr1 disk)

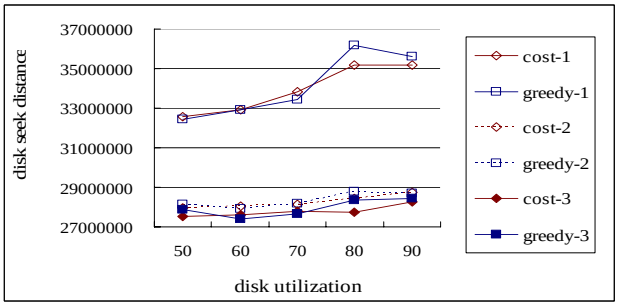


(c) Number of blocks copied (/usr2 disk)

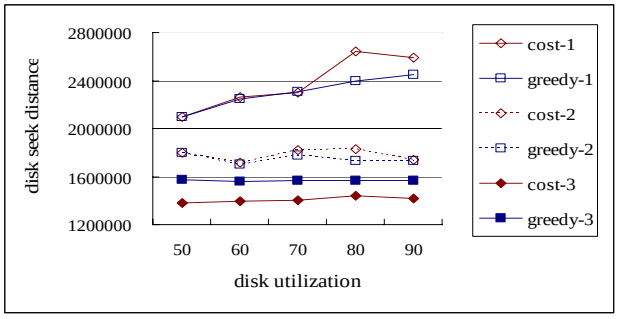
Figure 4. Cleaning overhead comparison (snake trace).

4.3 Effect on Disk Seek Time

This experiment evaluates the disk seek time for serving all the requests in the trace. Figures 5 and 6 show the performance comparison results of using different cleaning policies.



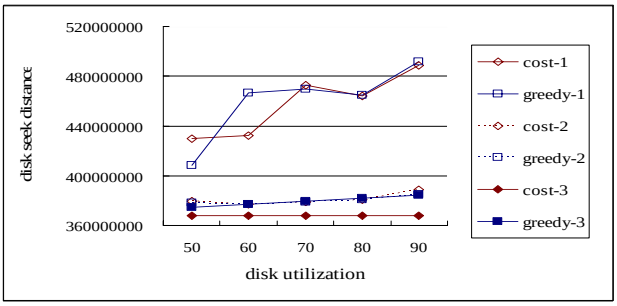
(a) Disk seek distance (root disk)



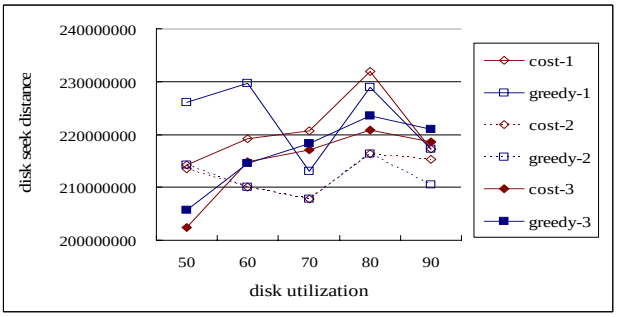
(b) Disk seek distance (swap disk)

Figure 5. Performance comparison (hplajw trace).

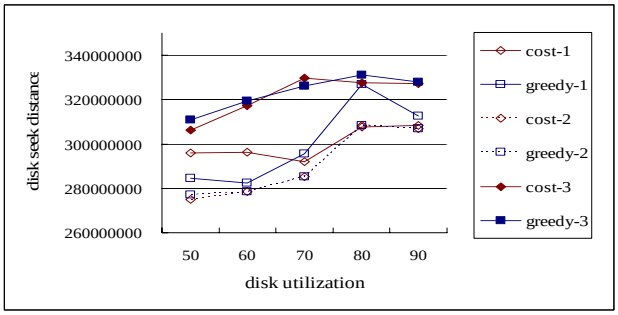
The results show that when R-LFS is used, the performance is greatly improved. The amount of disk seek distance is reduced by 14.07-45.45% for cost-benefit policy and 13.22-35.92% for greedy policy under hplajw trace, and up to 24.8% for cost-benefit policy and 21.74% for greedy policy under snake trace.



(a) Disk seek distance (root disk)



(b) Disk seek distance (/usr1 disk)



(c) Disk seek distance (/usr2 disk)

Figure 6. Performance comparison (snake trace).

5. PROTOTYPE EVALUATION

5.1 Experimental Environment

We have implemented a prototype on NetBSD 2.0 [7]. We modified NetBSD/LFS source codes to incorporate our proposed R-LFS technique. Table 3 shows our experimental platform.

Table 3. Experimental platform

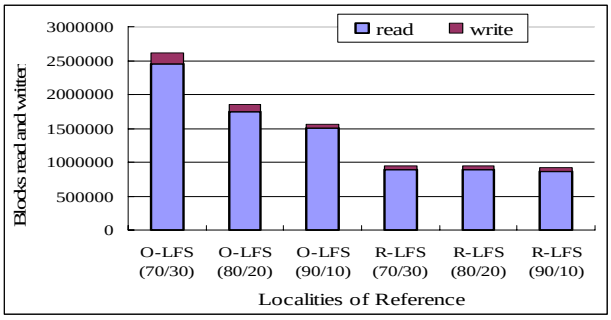
OS	NetBSD Ver. 2.0 LFS Ver. 2, Cleaner Ver. 1.5
Processor	Intel Pentium 4 2.0GHz
Memory	DDR400 512 MB
System Disk	80GB Seagate ST380011A
Test Disk	40GB Seagate ST340016A

We have created a simple benchmark to evaluate the performance of R-LFS. This benchmark first initialized the storage of test disk by creating 100 directories, and then measured the elapsed time for the following operations. It first created enough files under these directories to fill the disk to 85% of storage space and each file size is 100 KB. It then updated the initial data according to the designated locality of accesses. A total of 50,000 disk files (i.e. 5GB data) were updated. Finally, we recorded the information obtained from the cleaner, i.e. the number of blocks read and written.

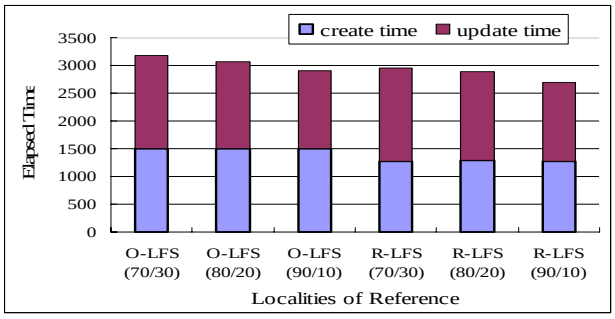
The workload for locality of reference was created based on the hot-and-cold workload used in the evaluation of Sprite LFS cleaning policies [8]. We used the notation "x/y" for locality of reference, in which hot data occupied y% of data, cold data and read-only data occupied half of (1-y)% each. Besides, x% of all accesses go to y% of the data while (1-x)% goes to the remaining half of (1-y)% of data.

5.2 Experimental Results

We compare the performance of R-LFS with the original LFS referred to in the experiment as O-LFS. Three regions are used in R-LFS. Figure 7(a) shows the results reported by the LFS cleaner when the benchmark was completed. By applying R-LFS, the cleaning overhead can be greatly reduced. The amount of blocks read and written are reduced by 41.04-63.6%. Figure 7(b) shows that though our prototype is not optimized, the elapsed time for running the benchmark still can be reduced by 5.77-7.27%.



(a) Cleaning overhead reported by cleaner



(b) Elapsed time comparison

Figure 7. Experimental results.

6. CONCLUSIONS

We have proposed a new method called R-LFS that dynamically reorganizes data in disk to build an efficient disk layout for successive data accesses to reduce disk seek times and cleaning overhead. Besides the simulation study, we have also implemented a prototype in LFS on NetBSD. Few modifications are needed for incorporating the R-LFS method.

Experimental results of the prototype show that the amount of blocks read and written during cleaning can be reduced by 41.04-63.6%. Besides, the amount of disk seek distance can be reduced by up to 45.45% and the amount of blocks copied can be reduced by up to 99.9% under simulation study. These results demonstrate that using R-LFS can effectively improve file system performance.

7. REFERENCES

[1] S. Akyurek and K. Salem, "Adaptive Block Rearrangement under UNIX," Software - Practice and Experience, 27(1), pp. 1-23, 1997.

[2] T. Blackwell, J. Harris, and M. Seltzer, "Heuristic Cleaning Algorithms in Log- Structured File Systems," Proceedings of the 1995 USENIX Technical Conference, pp. 277-288, 1995.

[3] M. L. Chiang, Paul C. H. Lee, and R. C. Chang, "Using Data Clustering to Improve Cleaning Performance for Flash Memory," Software Practice & Experience, Vol. 29, No.3, pp. 267-290, Mar. 1999.

[4] M. L. Chiang and R. C. Chang, "Cleaning Policies in Mobile Computers Using Flash Memory," Journal of Systems and Software, Vol. 48, No.3, pp. 213-231, Nov. 1999.

[5] D. D. Grossman and H. F. Silverman, "Placement of records on a secondary storage device to minimize access time," Journal of ACM, vol. 20, no.3, pp. 429-438, July 1973.

[6] J. N. Matthews, D. Roselli, A. M. Costello, R. Y. Wang, and T. E. Anderson, "Improving the Performance of Log-Structured File Systems with Adaptive Methods," Proceedings of the Sixteenth ACM Symposium on Operating System Principles, Saint Malo, France, Oct. 5-8, 1997.

[7] NetBSD Project, http://www.netbsd.org.

[8] M. Rosenblum and J. K. Ousterhout, "The Design and Implementation of a Log-Structured File System," ACM Transactions on Computer Systems, 10, (1), pp. 26-52, 1992.

[9] C. Ruemmler and J. Wilkes, "UNIX Disk Access Patterns," Proceedings of the 1993 Winter USENIX, pp. 405-420, 1993.

[10] P. Vongsathorn and S. D. Carson, "A System for Adaptive Disk Rearrangement," Software-Practice and Experience, 20, (3), pp. 225-242, 1990.

[11] J. Wang and Y. Hu, "WOLF - A Novel Reordering Write Buffer to Boost the Performance of Log-Structured File Systems," Proceedings of the FAST 2002 Conference on File and Storage Technologies, pp. 47-60, 2002.

[12] W. Wang, Y. Zhao, and R. Bunt, "HyLog: A High Performance Approach to Managing Disk Layout," Proceedings of the 3rd USENIX Conference on File and Storage Technologies, pp. 145-158, 2004.

[13] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan, "The HP AutoRAID Hierarchical Storage System," ACM Transactions on Computer Systems, 14, 1, pp. 108-136, 1996.

[14] C. K. Wong, "Minimizing Expected Head Movement in One-Dimensional and Two-Dimensional Mass Storage Systems," Computing Surveys, Vol.12, No.2, Jun. 1980.