Improving Performance of Linux File System with Adaptive File Rearrangement

Kai-Yu Cheng, Chung-Wei Tsai, Mei-Ling Chiang
Department of Information Management, National Chi-Nan University, Puli, NanTou 545, Taiwan, Republic of China
henry0311@gmail.com, nightfly0205@gmail.com, joanna@ncnu.edu.tw

Abstract-*Though computer devices are innovated continuously with the evolution of the technology, hard disk is still the bottleneck of the system performance. In this paper, we propose an adaptive file rearrangement mechanism to reduce disk seek times in accessing files to improve file system performance. Basically, frequently accessed files will be dynamically migrated to the space located at the center of the hard disk during the manipulation of disk files. Besides, we exploit application provided data access behavior to create hot files in the central disk. Furthermore, for the cost-effective concern, we also propose a method to reduce the file migration overhead. If the original disk location of a file to be migrated is close enough to the central disk, the file will not be migrated.*

We have implemented the proposed adaptive file rearrangement mechanism by modifying the Ext3 file system of Linux kernel 2.6. Experimental results show that our modified Ext3 file system can effectively decrease disk access time and improve file system performance.

Keywords: File System, Seek Time, Cylinder

1. Introduction

Many computer devices are innovated continuously with the evolution of the technology. However, hard disk is still the bottleneck of the system performance. The data access time of the hard disk includes seek time, rotation latency, and transfer time. Among them, the seek time occupies most of the time in the access time. The seek time is the disk head movement time needed to read and write the sector in the target cylinder. When the data is stored discontinuously in the disk, the disk head needs to move more distances in order to access the data. Therefore, it will decrease the hard disk transfer rate and affect the system performance. Thus, lots of researches [1-6] focus on reducing the seek time in access files.

On the other hand, when a disk has been accessed for a long time, lots of data are inserted, deleted, and modified in the disk. The store of files in the disk may become vary chaotic and the way of data allocation will affect the system performance. There are researches exploring how to allocate data effectively, such as the organ pipe heuristic [5] and the adaptive block rearrangement [1]. Adaptive block rearrangement will put hot data blocks (i.e. frequently accessed data blocks) toward the central cylinders of the disk because hot data blocks have higher access frequencies. In this way, system can decrease the disk head movement distance and the seek time to improve the system performance.

Because the seek time for accessing files in disk will spend most of the time, therefore, in this research we will focus on how to reduce the seek time of the hard disk and how to store hot files in the central cylinders of the disk. By doing this, the disk head can move inside the central cylinders of the disk which can shorten the disk head movement distance and improve the access speed for accessing frequently accessed files. Basically, we will store the hot files into the central cylinders of the disk. The storage area of the central cylinders in the disk is named Reserved Area. When a file is accessed, the system will count the number of reading and writing times of this file. The system will check the storage space in the central cylinders of disk regularly and move the old articles according to LRU (Least Recently Used) strategy to ensure that there are enough free space for storing hot articles in the central cylinders of the disk. Besides, number of the access times of reading and writing a file is counted to determine whether an article becomes hot. The hot file will be moved to the central cylinders of the disk to improve the speed of accessing. However, if the storage location of this article in the disk is near the central cylinders of the disk, this article will not be moved to decrease the system overhead.

We take BBS system an example in this research because files will be read and written frequently in this system and the access frequencies of them are different. Besides, BBS system provides the information of hot boards or hot articles. Every board is a directory and every article is stored a file. The frequently accessed articles are stored in the ten hot boards. We modify the route of accessing files to dispose hot files in the central cylinders of the disk using the information provided by the application. We hope to improve the performance of file system by decreasing the movement of disk head to decrease the seek time.

We have implemented our work in Linux 2.6. We modify the EXT3 file system [7] and design a test program to imitate the characteristic of reading and writing of a BBS system. The result of experiments shows that files access in the modified Exts3 files system has effective performance improvement over the original Ext3 files system. Besides, if the original disk location of a file to be migrated is close enough to the central disk, the file will not be migrated. This method can further improve the file system performance.

2. Background and Related Work

Section 2.1 describes the optimization of disk layout. Section 2.2 introduces the adaptive block rearrangement. Section 2.3 describes restructuring based on seek time.

2.1 Organ Pipe Heuristic

According to the research of C. K. Wong, organ pipe heuristic [1,5] can be used to achieve the best allocation for files when the access frequencies of the files are different. Organ pipe heuristic puts the frequently accessed files to the cylinders at the center of the disk, and the next most frequently accessed data in either side of the center. On the other hand, the least frequently accessed files will be put in the border cylinders of the disk. The total distance of the disk head movement could be shortened effectively and the seek time will be decreased to improve the performance of file system.

2.2 Adaptive Block Rearrangement

Adaptive Block Rearrangement [1] was proposed by Akyurek and Salem to decrease seeks time effectively. This technique creates several cylinders at the center of disk to reserve some space to store the recently frequently access blocks. The system can analyze access frequencies and distinguish data blocks which dynamically copied from their original disk locations to a reserved space near the central of disk. When a data block is being accessed, the system will look up the block-remapping table [1] to see it is in the reserved space. This technique can decrease seek time effectively when the disk head mostly accesses the frequently accessed data blocks.

2.3 Dynamically Restructuring Disk Space

McDonald and Bunt [2] proposed a new method that can dynamically restructure the disk space and restructure a few parts of crucial files to decrease the seek time. The author provides a restructuring based on seek time method (seek time expansion factor, STEF) to select the candidate file. Experiments show that this method can effectively increase the disk performance by restructuring a few parts of crucial files.

3. System Design and Implementation

We first present the system architecture and overview. We then describe the implementation of our system that is based on the modification of the EXT3 file system.

3.1 Architecture Overview

We create a space from the center of the disk to be the Reserved Area (RA) which contains several cylinders in the center of the disk. And we modify the EXT3 file system of the Linux kernel. When the application reads/writes the file, the system will determine the file is a cold file or hot. If it is a hot file, it will be accessed in the Reserved Area. If it is a cold file, we will accumulate the frequency of accessing. Figure 1 shows system architecture. When the access frequency of the file is over the system set threshold, this file becomes hot. This file will be moved to the Reserved Area to decrease the disk head movement distance. Therefore we can reduce the seek time to improve the system performance.

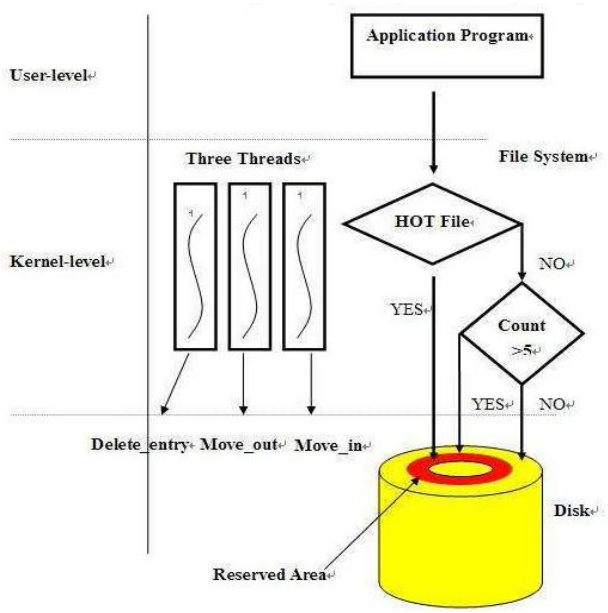

Figure 1. Systems architecture

Because moving a file to the Reserved Area will increase the overhead to the system, for this reason, we create a new *Move_in* kernel thread for managing how the file is moved to the Reserved Area. When a file is determined to become a hot file, the system will check whether the storage location of this file in disk is near the central cylinders. If the location of the file in disk is not near the central cylinders, it will be moved to the Reserved Area. If the location of the file in disk is near the central cylinders, it will be not moved.

Two other kernel threads are created. One is used to maintain the Reserved Area. When the storage space of the Reserved Area becomes full, some old files will be moved from the Reserved Area to outside of this Reserved Area. The other is used to maintain the total number of entries and storage space which are in the non-Reserved Area.

The whole system has several functions, such as managing files in the Reserved Area, managing files in the non-Reserved Area, reading/writing files, and deleting files.

3.2 Managing files in the Reserved Area

In order to decrease the movement distance of the disk head, hot files are put in the central cylinders of disk, because hot files have been accessed frequently. But first of all we need to know whether the file is in the Reserved Area. Thus, a table is maintained in RAM to keep track of all of the files in Reserved Area. The system must access hot files by the table called hot table which records what files are stored in Reserved Area. It could be determined that how many file entries can be stored in the hot table according to the number of *inodes* in disk from the Reserved Area.

Because the hot table must be stored in RAM when the system is booted, we modify Linux kernel source code to create the hot table in RAM by Linux kernel function *kmalloc()* in system booting procedure. Therefore, the system can access the file in the Reserved Area conveniently. When the system is accessing files in the Reserved Area, the entries in the hot table are ordered by LRU. That is the entry which has been accessed most recently will be at the head of linked list and the entry which has not been accessed for the longest time will be at the tail of linked list.

3.3 Managing files not in the Reserved Area

When the system determines the file is not a hot file, the file will be accessed on the disk through the original path. Besides, a table named cold table is allocated in RAM using the Linux kernel *kmalloc()* function during the system booting procedure to record file information about files which are not hot files. The fields of this table are like those of the hot table. Besides, we create a "count" field in the cold table and the value is increased with the counts of accessing files. If the count value is greater than the system set threshold, the file becomes hot and the system will determine whether the file should be moved into Reserved Area. So the file access frequency can be quickly searched from the cold table by searching the entry's "count" field.

3.4 Writing files in the disk

We modify the *sys_openat()* function of the Linux kernel 2.6. When system is writing the file by the function, the system will intercept the file in the function and determine whether the file is hot. If it is a hot file, the file will be written into the Reserved Area. We modify the original path name of this file to concatenate it with "/reserved" to be the near pathname. Therefore, the file can be written in the Reserved Area.

Some applications can provide information of hot files. When the system boots, they will load the log file into RAM. When the application writes a file, the system will determine whether it is a hot file by this hot file information. The file needs to be written in the Reserved Area if it is a hot file. Otherwise, it will be written in the original storage space outside the Reserved Area and be counted with the number of writing times.

If the file is to be written in the Reserved Area, it needs to add an entry in the hot table. The hot table is maintained in the least recently used (LRU) order. That is, the most recently used entry is at the head of the hot table and the least recently used entry is at the tail of the hot table. If the hot table already has the file's information, it means that the file is to be overwritten. The file's entry is updated directly and this entry will be moved to the head of the hot table. It means that the file is accessed most recently.

If the file is a cold file, it will be written in the disk using the original pathname. Then, an entry is added in the cold table which contains the entries for files not written in the Reserved Area and the "count" field of the entry is set to 1. If there is an entry for the file in the cold table, it means the file is to be overwritten. Then the number in the count field of this entry is increased by 1, and the entry is moved to the head in the cold table. When the number of the count field is greater than the system set threshold, the system will determine whether the cold file will become hot and wake up the *Move_in* kernel thread to move the file to Reserved Area. Figure 2 shows the diagram of writing a file in disk.

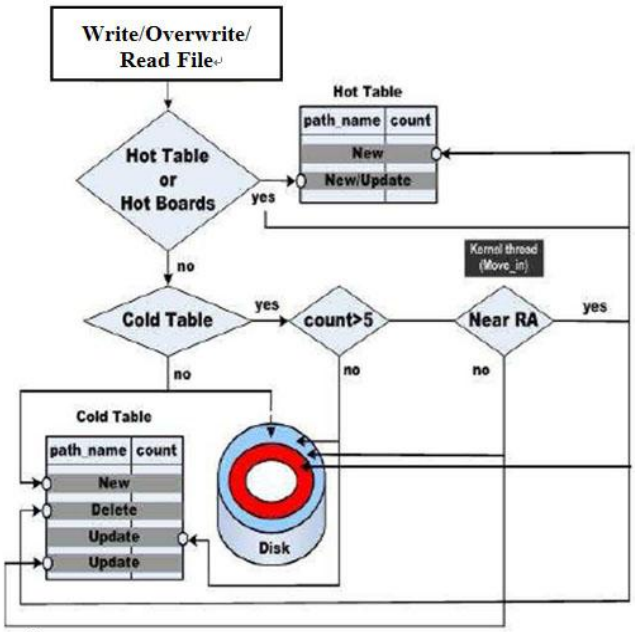

Figure 2. Writing/reading a file in/from the disk

3.5 Reading files from the disk

We modify the *sys_openat()* function in Linux kernel. When applications are reading files, the system will loop up the hot table whether the file is stored in the Reserved Area. If the file is in the Reserved Area, the system will concatenate the "/reserved" with the original pathname. According to this new pathname, the system will open and read the target file in the Reserved Area. The file's corresponding entry will be moved to the head of the hot table. It means the article has been read most recently.

If the file is not in the Reserved Area, the system will read it with the original pathname and update the number of the "count" field in the cold table. At the same time, the file's corresponding entry will be moved to the head of the cold table. It means that this file has been read most recently. When the number of the count field of a file is greater than the system set threshold, the system will determine that this file has become hot and wake up the kernel thread *Move_in* to move the file into the Reserved Area. Figure 6 shows the diagram of reading a file in disk.

3.6 Deleting files in the disk

Since all the hot files will be written into the Reserved Area in the disk, so when we want to delete a hot file, we can not use the original pathname to delete a hot file. We then modify *sys_unlinkat()* function in *namei.c* in Linux Kernel. When the system wants to delete a file, it will look up the hot table. If the entry of the file is not in the hot table, it means that the file is not stored in the Reserved Area. Otherwise, the file is stored in the Reserved Area. We modify the original pathname of the file which we want to delete. Therefore, we can delete the file in the Reserved Area. Meanwhile, we must delete the information about the file in the hot table.

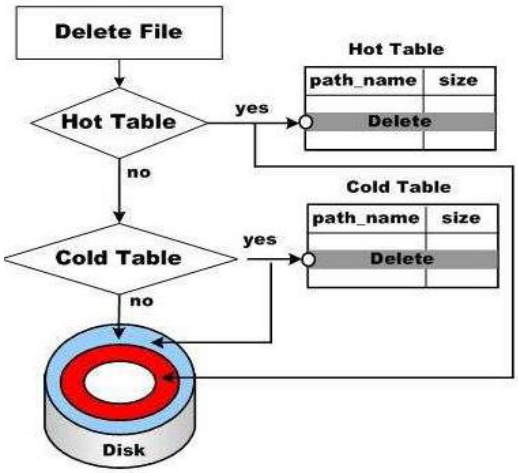

Figure 3. Deleting a file in the disk

When the system determines that the file is not in the Reserved Area, it then checks whether the entry of the file is in the cold table. If yes, this entry is deleted. Finally, the file is deleted using the original pathname. Figure 3 shows the diagram of deleting a file in the disk.

3.7 System maintenance

Because the capacity of the Reserved Area is limited, we must establish a mechanism to ensure there is enough available space of the Reserved Area for adding new hot files and move out the least recently used files in the Reserved Area. In order to ensure the hot files can be written in the Reserved Area, the system must check whether there is enough available space in the Reserved Area and whether there is enough number of entries available in the hot table before writing new files every time. The kernel thread (*Move_in*) is created and started during system initialization. It always sleeps to wait for being woken up by the kernel to move out the least recently used files until there are enough number of entries and spaces available in the Reserved Area.

We create a kernel thread (*Move_out*) which is in sleeping status to wait for being woken up by the kernel. When the hot file is in the Reserved Area, the system will check the number of entries in the hot table and the available space in the Reserved Area. If there is enough space, the file is written in the Reserved Area directly. Otherwise, the kernel thread (*Delete_entry*) will be woken up to move the least recently used file out of the Reserved Area until there is enough space and number of entries. Then, the thread will sleep again to wait for being woken up by the kernel at next time.

When the system determines that the file is not a hot file and writes it outside the Reserved Area, an entry about the information of this file is added in the cold table. To ensure there is enough number of entries available to add the new entry in the cold table, every time when a cold file is written in disk, a kernel thread (*Delete_entry*) would maintain the number of the entries available in cold table.

The kernel thread (*Delete_entry*) is created and started during system initialization and it sleeps immediately to wait for to being woken up by the system. The system will check that whether there is enough number of entries in the cold table when writing a cold file every time. If there is enough space, the file will be written immediately in the disk. If not, this thread will be woken up to delete the entry which has not been accessed for the longest time in the cold table by LRU strategy. Besides, because the cold file is written in the disk in the original pathname. So the entry of the file in the cold table only needs to be deleted.

Every time when a file is read or written, the system will check whether the file's entry is in the cold table. If not, a new entry is added into it. If it has been in the cold table, the same operations would be performed as in Section 3.3. If the file's location is near the center of disk, it is not moved. If not, the system will move the file into Reserved Area. Figure 4 shows moving a file outside Reserved Area.

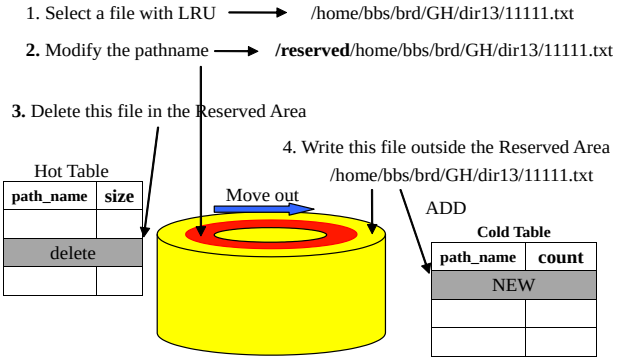

Figure 4. Moving a file outside he Reserved Area

4. Performance Evaluation

4.1 Experimental Environment

Table 1 shows our experimental environment. We partition the data disk which installs the BBS system into three partitions. The first partition is used to store articles of BBS boards. The second partition is used as the Reserved Area and stores the BBS TOP 10 Hot Boards. The third partition is used to install the MapleBBS-3.10-20 system. Besides, the Reserved Area is about 2.5% of the total hard disk storage space.

Table 1. Experimental Environment

CPU	Pentium 4 1.6GHz
Memory	256MB DDR-266
System Disk and Data Disk	Seagate BARRACUDA ATA IV 40G (Model ST240016A), Ultra ATA-100, 7200rpm, 2MB cache
Operating System	Fedora Core 5（kernel 2.6.18.6）
BBS	MapleBBS-3.10.20

To demonstrate that our modified EXT3 file system can effectively improve the system performance, we design a benchmark which will simulate the BBS system operation behaviors, such as: writing, deleting, and reading files. Finally, this benchmark will recode the elapsed time of the whole processes and compare it with the time that the same process spends on the original un-modified file system.

Our designed user-level benchmark can randomly perform wiring, reading, and deleting operations. We adjust their ratios in different experiments. We store ten directories in the Reserved Area as the TOP 10 Hot Boards. There will be 40 directories in each of them, and it is convenient for the files management. The non-Reserved Area which has 400 directories represents that BBS system has 400 boards, and each of them has 40 directories. The benchmark can decide the reading ratio of the hot files and will randomly write files into directories, each file is 4Kbytes. Beside, it will randomly read files or delete files from those directories. Initially, it will randomly write files into the hard disk which represents the situation that the hard disk has been used for reading and writing for a long time. The experiment will totally perform 204,800 times of files reading, writing, and deleting operations and record the elapsed of time the whole processes in the unit of microseconds.

4.2 Performance Comparison under Various Access Rates

We compare the performance of the original Linux EXT3 (O-Ext 3) file system with that of our modified EXT3 file system (M-Ext3). Our benchmark performs 204,800 times of files writing, reading, and deleting. The ratios of hot files and cold files written in the disk are 9:1. Besides, we adjust the rate of writing in the benchmark and dividedly write 10-30% files into the hard disk.

Besides, in order to test whether the long time access of the disk will affect the time needed to access files, the data disk is initially written randomly with file for 0-95% of disk space. Besides, because files might be cached in the RAM, which may affect the measured performance, the benchmark also executes “sync” command after finish all the read/write operations. In the following figures, the y-axis represents the total execution time of our user level benchmark and the x-axis refers to the disk utilization which represents the percentage of disk space occupied with file data.

Through our experiments above, even at the increasing ratio of the disk utilization, our modified files system still has significantly performance improvement over the original Ext3 file system. Figures 5 and 6 show our experiment results.

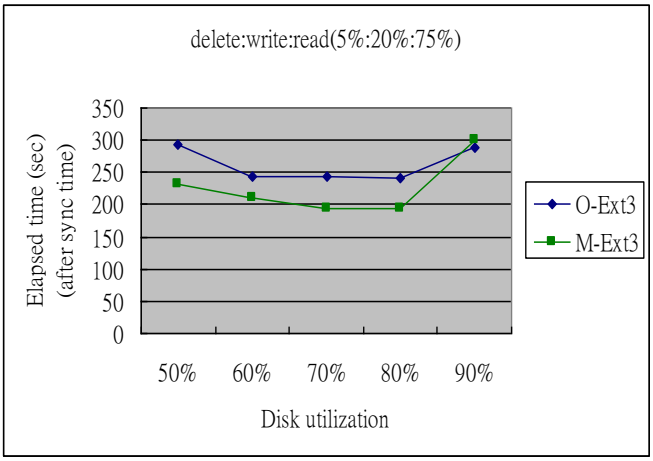

Figure 5. Experiment results (delete: 5%, write: 20%, read: 75%)

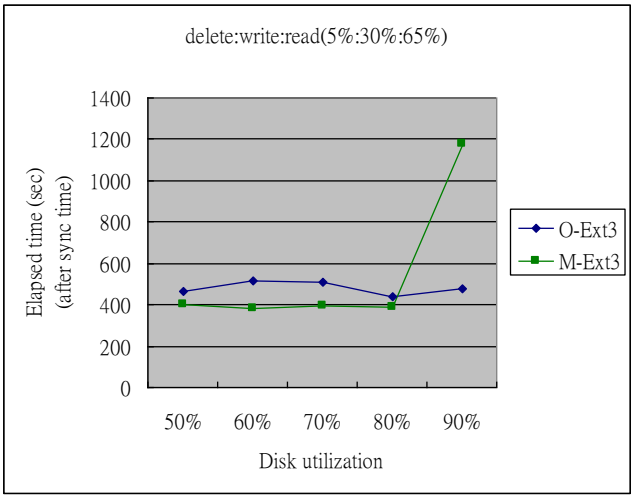

Figure 6. Experimental results (delete: 5%, write: 30%, read: 65%)

4.2.2 Analysis of Accessed Files

When hot files are accessed in the disk, they will be accessed in the Reserved Area by the modified Ext3 file system. We calculate how many files are accessed and the total size of files accessed in the Reserved Area.

Through above experiments, it shows that the original Linux files system (i.e. O-Ext3) stores files randomly and sparsely in different disk sectors and files are rarely stored in the Reserved Area so files are scattered in the disk. When the system needs to read the hot files, the disk head needs to move among different cylinders. Whereas, our modified file system will always keep the movement of the disk head in the Reserved Area in the central disk. By doing this, we can dramatically decrease the distance of the disk head movement to effectively decrease the seek time, and thus improve the system performance. Figures 7 and 8 show our comparing results.

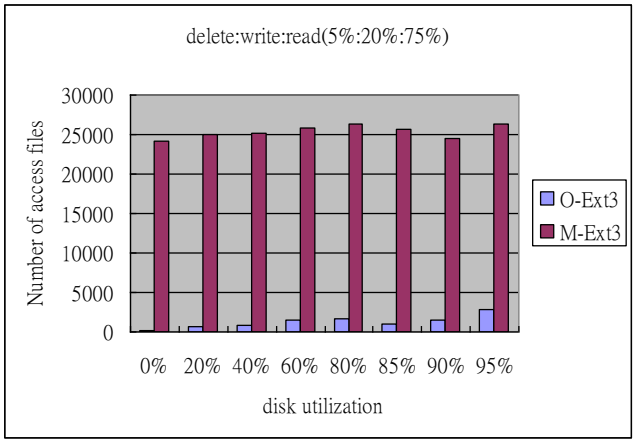

Figure 7. Comparing the number of access files

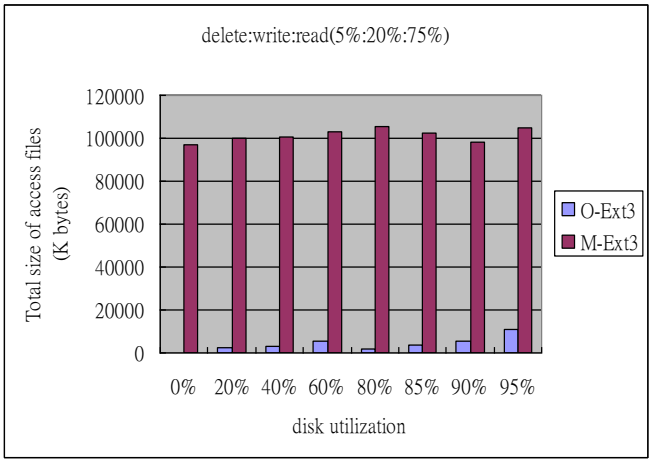

Figure 8. Comparing the total size of access files

5. Conclusion and Future Works

In our research, we store hot files (i.e. frequently accessed files) in the Reserved Area located at the center of the hard disk by modifying Linux Ext3 file system. Because the access frequencies of hot files are higher, the disk head will access files inside the Reserved Area when the system accesses hot files. The purpose is to decrease seek time for accessing hot file in the hard disk by shortening the disk head movement distance to improve the file system performance. Besides, we also propose a method to determine whether the cold file when it becomes hot should be moved to the Reserved Area. When the cold file is accessed frequently, the system will determine the file has become hot. The system will also determine whether the location of file in the hard disk is close to the Reserved Area. If the location of the data is closed to the Reserved Area, the data will not be moved, because the benefit is less than the system overhead of file moving.

We have implemented this system by modifying the Ext3 file system of Linux kernel 2.6 and created a performance benchmark to simulate the operation behaviors of the BBS system. The result of experiments shows that the modified Ext3 file system can effectively shorten the disk access time. Besides, when the cold file becomes hot, the file will be moved to the Reserved Area. If the location of this file is close to the Reserved Area, the system will determine whether the file needs to be moved into the Reserved Area. This method can further improve file system performance.

In the future, we hope to certainly research different applications in determining cold and hot files and the method of writing files to effectively increase the accessing performance of every application in Linux. This research is to simulate the BBS system to test the benchmark by ourselves. We hope to chronically monitor the BBS system to understand and record the behavior of reading and writing in the BBS system. Therefore, the access time in the hard disk can be accurately decreased to increase the performance of the file system.

References

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

[2] M. McDonald and R. Bunt, “Improving File System Performance by Dynamically Restructuring Disk Space,” Phoenix Conference on Computers and Communication, pp. 264-269, Mar. 1989.

[3] Wenguang Wang, Yanping Zhao, and Rich Bunt, “HyLog: A High Performance Approach to Managing Disk Layout”, 3rd USENIX Conference on File and Storage Technologies, pp. 144-158, March 2004.

[4] C. Ruemmler and J. Wilkes, “Disk Shuffling,” Technical Report HPL-91-156, Hewlett-Packard Laboratories, Palo Alto, CA, October 28, 1991.

[5] C. K. Wong, “Minimizing expected head movement in one-dimensional and two-dimensional mass storage systems”, ACM Comput. Surv. 12, 2, pp. 167-178, 1980.

[6] 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, 1973.

[7] Daniel P. Bovet and Marco Cesati, Understanding the LINUX Kernel, 3rd edition, O’Reilly, December 2006.