EAR: An Energy-Aware Block Reallocation Framework for Energy Efficiency*

Woo Hyun Ahn

Department of Computer Science, Kwangwoon University
447-1 Wolgye-Dong, Nowon-Gu, Seoul, Korea
whahn@cs.kw.ac.kr

Abstract. File systems, which are embedded in low-power computer systems, have continuously improved energy saving features. However, the inevitable aging of the file system gives disks less chances to stay idle, thus increasing the energy consumption along with the decrease in performance. As a solution to the problem, we propose the energy-aware block reallocation framework (EAR), which is a software framework with a reallocation algorithm applicable to file systems. EAR dynamically reallocates fragmented data with the same relationship to disk locations where less disk accesses and seeks are achieved. The optimized file data layout improves the energy saving on aged disks.

Keywords: Embedded systems, energy saving, file systems, disks, data layout.

1 Introduction

Today, disks are increasingly adapted as primary storage systems in low-power embedded systems such as mobile devices. However, disks consume more power than other computer components due to their mechanical movements. Earlier researches [2][6] achieve the energy saving by stretching idle periods for disks. Less disk I/Os and seeks have important roles on keeping disks in the idle state as well as increasing performance of file systems[4][9]. Especially, optimizing data layout of small files (< 64 KB), the major file usages[3][8], has a large impact on disk I/Os and seeks.

Earlier file systems, FFS[5][7], C-FFS[3], EEFS[4], and DFFS[1], have focused on how to transfer large data at one disk I/O since the multiple data transfers can increase the disk burst ratio, thus granting disks chances to stay idle and thus consume less power. FFS attempts to cluster logically sequential blocks of a file on physically contiguous disk blocks. Multiple block transfers can be used to access the clustered blocks, thus reducing disk I/Os and improving the energy saving. FFS divides the disk into *cylinder groups*, each of which is a set of consecutive cylinders. A cylinder group co-locates related data in a directory. C-FFS allocates small files in a directory to contiguous disk locations, where they form a group whose size is limited by 64 KB. It attempts to cluster a created file into an existing group associated with the same directory. When a file of a group is read, C-FFS reads all files in the group at one time.

---

* The present Research has been conducted by the Research Grant of Kwangwoon University in 2006.

This multiple file transfers reduce disk I/Os, improving the energy saving. Hence, C-FFS is suitable to low-power computers. EEFS divides disk into fix-sized clusters, each of which stores a set of files with locality. A group with a large set of files spans several contiguous clusters. When a file of a group is accessed, all files of the group are prefetched at one time. This multiple file accesses increase the disk burstiness, improving the energy saving.

On heavily aged disks, however, FFS cannot cluster blocks of a created file due to the shortage of free clusters and thus the efficiency of energy saving is reduced. This file fragmentation is defined as *InterA-file Fragmentation(IAF)*. Also, if once related files are placed across cylinders, their accesses persistently incur disk seeks because they can be no longer reallocated. In C-FFS, the file system aging splits existing groups into smaller groups, thus decreasing the number of files transferred at a single disk I/O. This separation is defined as *IntEr-file Fragmentation(IEF)*. In EEFS, the file system aging incurs IEF in existing groups, draining free clusters needed to group related files. This causes the cleaning activity frequently executed to make rooms for free clusters. Unfortunately, the cleaning needs much CPU utilization as well as many disk accesses, increasing the energy consumption. Moreover, DFFS does not take the energy saving into consideration when reallocating fragmented data. Instead, it only focuses on improving small file fragmentation.

This paper proposes a new framework, called *Energy-Aware block Reallocation (EAR)* framework, which reallocates fragmented data in consideration of the energy saving on aged disks. EAR consists of the two components: first, reallocation algorithms that can be applied to the previous file systems for energy saving. The second is a filesystem-independent module that can be installed in operating systems. Related files with temporal locality are likely to be accessed and cached in memory together. The filesystem-independent module builds up the relation of the in-cache files in respective to relationship such as directory name and group. By using the relation information, EAR reallocates fragmented data that can be sources of large power consumption at contiguous disk locations. The improvement of file data layout increases a possibility that disk drives can keep idle for the longer periods.

2 Energy-Aware Block Reallocation Framework

EAR framework is a reallocation framework that is designed to reduce both IAF and IEF of small files (2 - 12 blocks). Related files are likely to be together in file cache due to temporal locality. The file cache exploits the write back that delays writes for some period of time, and periodically flushing the modified data to disk. Each time a dirty (or modified) block flushed to disk, EAR reallocates all fragmented data in a fix-sized *target region* of 64 KB, which starts from the disk location of the dirty block regardless of their modification, then eliminating IAF and IEF of the fragmented data.

The EAR mechanism, called *group write*, consists of the four phases: the decision of whether a target region is to be reallocated, the buffer gathering, the re-mapping, and disk write. EAR examines if a target region meets the three conditions. The first is if it has any data with either IAF or IEF, the second is if its data are in memory and the third is if it has two or more free blocks. If so, fragmented data of the target region

Fig. 1. EAR algorithm

are gathered into a buffer, which is called buffer gathering. Blocks addresses of the gathered data are re-mapped in order to eliminate their IAF and IEF. Finally, the re-mapped data are stored to the disk location indicated by the dirty block at one disk I/O.

Fig. 1 shows how blocks of a target region are gathered into a buffer and re-mapped during a group write. The process starts when a dirty block is stored to disk. The dirty block is first put at the head of the buffer. From the next of the dirty block to the last of the target region, EAR algorithm checks one by one if each block is a good candidate that meets conditions described below for a group write.

Any blocks of a target region may not be in memory under a buffer gathering. Two approaches can be taken to handle the uncached blocks; the first is to fetch the blocks into memory by using additional disk I/Os, but increases the time spent to do a group write. To avoid the cost, EAR algorithm uses the second approach in which the buffer gathering process stops when any uncached block is encountered. Then the reallocation is applied to the only gathered blocks because the uncached blocks can be lost on disk if some blocks being reallocated are overwritten to the disk locations of the blocks that were not fetched. After the stop, if free space still exists between the dirty block and the uncached block, it is used to reallocate fragmented data again.

EAR algorithm first checks if each disk block address is not only allocated, but also in memory since it reallocates the in-memory allocated block, called *target block*, to avoid additional disk I/Os, which is inserted to the tail of the buffer. EAR algorithm finds the file with the target block, then checking whether the logically next of the target block has physically contiguous disk address in reference to the target block. If not, the logically next block whose disk block may be outside of the target region is moved into the next of the target block. Then EAR algorithm re-assigns the two blocks newly physical block addresses in order to make them physically contiguous with each other. This process is progressed until the last block address of the target region is encountered. The process concentrates on eliminating IAF of files within the target region and thus decreasing disk I/Os needed to access fragmented files.

Some files gathered in a buffer may have the different relationships with the file with the dirty block. The mixture of the files with different relationships on disk

decreases the advantages of multiple data transfers. To solve the problem, EAR real-locates files irrelevant to the dirty block to the back of a target region, whereas it adjacently places files with the same relationship at the front.

There may be still some free blocks in a target region after the buffer gathering. By using the free blocks, EAR attempts to gather and re-map another fragmented data that have the same relationship, but are outside of the target region. The free blocks are preferentially filled up with a file with blocks that exist the inside and the outside of a target region. The purpose of this is to eliminate IAF of the files, which is one of sources that cause IEF of related files. In addition, if there are still free blocks in the target region, EAR algorithm scrapes up IEF-files that are outside of the target region into a larger group at the unit of relationship such as explicit group or directory.

Fig. 2. A relation group and its sub-relation groups

In Fig. 2, a *relation group*(RG) is implemented in memory to make the relationship of in-memory files with temporal locality at the relationship unit. It represents a logical set of files that are contained not only in the same relationship, but also in the file cache. When a block is accessed, the block and its file metadata are registered at the RG associated with its relationship. Examples of block and file metadata are a pointer to a buffer holding the block and the in-memory i-node respectively. The metadata are linked in the RG to make the relation of related files.

A set of consecutive disk locations where related files are usually co-located is defined as *disk region*, into which the file systems physically divide the disk. A cylinder of FFS, a group of C-FFS, and a cluster of EFFS are the examples of the disk region. The summary of a disk region is implemented as in-memory metadata, called *sub-relation group*(SRG) for the fast lookup of a disk region with files that are to be real-located. The summary includes the information of a disk region: the list of in-memory files out of files in the disk region, the total number of the in-memory files, the degree of IEF and IAF, etc. Its usage is explained in Section 3.

A SRG has two tables, *IAF table* and *indexing table*, to manage its in-memory files of 2 – 12 blocks. The IAF table is used to sort IAF-files of the SRG by sizes. A table entry indicated by an index points to a list of IAF-files with the same number of blocks as the index number. The IAF table is looked up to select fragmented files with specific sizes under the buffer gathering. The indexing table sorts files of the associated SRG by sizes regardless of their IAF states. A table entry indicated by an index links files with the same number of blocks as the index number. The indexing table is used to look up an in-memory file with a specific size in a disk region.

3 File Grouping Methodologies

EAR determines which fragmented data outside a target region are to be grouped into free space of the target region during a group write. The file grouping policies have different impacts on the efficiency of energy saving according to file system types.

In FFS, the effectiveness of multiple block transfers depends on how many blocks of a file are contiguously placed on disk. For the higher effectiveness, EAR first uses free blocks of a target region to reallocate as much of large files out of small files with IAF outside the target region. The files may be on either a cylinder with the target region or cylinders far away the target region. Out of the cylinders, EAR preferentially selects the small files of a cylinder that is at the longest distance from the current cylinder with the target region to reduce disk seek distances. This scheme reduces disk accesses and seeks that can be incurred by accessing the related files.

Fig. 3. File grouping of FFS. A file F_x is composed of blocks marked by F_x .

Fig. 4. File grouping of EEFS

A SRG manages metadata of related files on a cylinder in FFS while a RG does all in-memory files in a directory whose files are usually stored over several cylinders in Fig. 3(a). For free blocks of a target region, all SRGs of the RG are examined to find a cylinder that is at the longest distance from the current cylinder among cylinders with IAF-files like Fig. 3(b). For the found SRG, its IAF table is looked up to find an IAF-file of blocks that can be held in the free blocks. If there are not any files of the size, EAR again looks up the table entry indicated by the value subtracted by one from the current index number since EAR tries to scrape up related files scattered across outer cylinders into inner cylinders. This is repeated until the free space is filled up.

In EEFS, a group that is associated with a RG presents a set of files with locality while a cluster of a group is disk region associated with a SRG. When there are free blocks in a target region, EAR examines all SRGs of the group with the dirty block that triggers the group write. Among clusters with IEF, EAR selects a cluster with the largest amount of files, then reallocating its data to the free blocks. The cluster state is changed to empty so that EEFS can allocate new files. This sequence of operations is a cleaning activity achieved by a group write. Since clusters of large data becomes clean during group writes, EEFS does but clean clusters with small data which it takes less time to clean. Hence, EAR lifts a burden from the cluster cleaning activity.

The starting location of a target region should be aligned to the starting block address of a cluster with the dirty block, instead of the disk location of the dirty block. Moreover, the size of a target region is defined as a multiple size of a fix-sized cluster. In Fig. 4(a), the disk address of a dirty block may be at the center of a cluster instead of the front. If the related files that are gathered into a buffer are stored to the

Fig. 5. File grouping of C-FFS

starting location of the dirty block, they will be still placed across the two clusters, suffering from another IEF in Fig. 4(b). To solve this problem, the starting block of a target region is aligned to that of a cluster in Fig. 4(c).

In C-FFS, a directory name space is associated to a RG like FFS, whereas a group is considered to a disk region and thus associated with a SRG. In Fig. 5(a)(i), the file system aging may make related files of a directory split into small groups. When a group write is executed like Fig. 5(a)(ii), EAR finds SRGs corresponded to one or more groups with enough many files that can be migrated into the free blocks, then reallocating their files to the free blocks. However, if there are some free blocks that can only include one part of a group (i.e., G4), the group are not reallocated into the free space because the group is again split as separate group after the reallocation.

Small groups in a directory can be put on several cylinders like Fig. 5(b)(i). Accessing files of the groups can incur many disk seeks. To reduce disk seeks, EAR selects groups in a cylinder that is at the longest distance from the current cylinder with the target region, reallocating their files into the free space like Fig. 5(b)(ii). It is because clustering the groups of cylinders that are at the long distance from each other can minimize the distance of disk seeks that are caused by group accesses.

4 Experimental Evaluation

We used a low-cost hardware computer with a 700 MHz Pentium processor and 128 MB of memory to perform all experiments in the environment similar to embedded systems. EAR framework was developed as a module in the OpenBSD kernel version 2.8. Since it takes a long time to newly implement EEFS and C-FFS in the kernel, FFS implemented in the OpenBSD is used for our experiments. More parameters are shown in Table 1. A benchmark is made to recreate the non-adjacency of on-disk placements shown in the previous research[8]. On an empty disk, the benchmark executes sequences of file operations determined by the probability distribution in Table 2. When the disk became extremely full (disk utilization of 75%), the ratio of

Table 1. Testing system configuration

Disk		File System	
Disk space	6.4 GB	Size	2 GB
Rotation speed	5400rpm	Block size	4 KB
Cylinder	13328	Cylinder group	283
Average seek	9.5 ms		

Table 2. Ratio of file requests

Requests	Before 75%	After 75%
Create	10%	10%
Delete	5%	10%
Read	65%	60%
Write	20%	20%

create/delete is changed to make the disk heavily fragmented. File sizes of the requests are determined by the distribution observed on our file server in Fig. 6. For directory locality, the benchmark makes 50 sub-directories, each of which is selected by a Poisson distribution, then selecting a random number within the range of 100-5000 for the number of file requests that will be performed in a sub-directory.

Fig. 6. Distribution of file sizes

Fig. 7. Aggregate file data layouts

We used a *layout score*[8] to compare the efficiency of energy saving among file systems. Earlier researches[4][9] show that optimizing file data layout can improve the energy saving. File data layouts can be numerically presented as the layout score, which quantifies the degree of contiguous allocation of files. The layout score for an individual file is the ratio of the block physically contiguous with the previous block of the same file. For a file with a layout score of 1.00, all of its blocks are allocated contiguously. A file with a layout score of 0.00 has no contiguously allocated blocks. The aggregate layout score presents the average score of all files on one file system.

Fig. 8. Read performance and layout score

Fig. 7 shows the aggregate layout scores of EAR-FFS and FFS according to the number of file operations and file cache sizes. EAR-FFS represents FFS that cooperates with EAR framework, which uses 16 blocks as the size of a target region. EAR-FFS-64, 32 and 16 use 64 MB, 32 MB and 16 MB as the file cache size respectively. As file operations are executed for a long period of time, the difference in the layout score among the file systems increases. All EAR-FFSes outperform FFS in the layout score. Especially, the layout score of EAR-FFS-64 is larger than that of FFS by 14%.

The improvement in the file data layouts seriously decreases disk I/Os caused by related file accesses, giving disks more chances to keep in the idle state.

Fig. 8(a) and 8(b) respectively show the read performance of small files and the layout scores at the point (1) in Fig. 7, where the disks become heavily aged. For the measurement, we read files in each of the 50 directories sequentially. The read performance on disk that was aged by EAR-FFS-64 is much higher than that of FFS by 30%. The results allow us to make two points. First, disk I/Os are significantly reduced by the increase of multiple block transfers, which results from the improvement of EAR-FFS-64 in the layout score over FFS by 94%. Second, disk seeks go down because related files spread over long distant cylinders are reallocated into adjacent cylinders with free space that was made by the file system aging.

In Fig. 7 and 8, small file caches weaken the effectiveness of group writes. The decrease of the file cache sizes reduces the number of blocks that will be cached in memory, diminishing the number of blocks which a group write can gather into buffers for reallocation. The decrease of blocks that are to be reallocated gives EAR less chances to optimize the file data layout.

5 Conclusions

EAR optimizes file data layouts by using locality. The improvement of the file data layouts increases the efficiency of the energy saving for related file accesses. The measurements show that EAR improves the aggregate layout score, one of criteria for the efficiency of the energy saving as well as the read performance of small files.

References

1. Ahn, W.H., Park, D.: Mitigating Data Fragmentation for Small File Accesses. *IEICE transactions on information and systems*, Vol. E86-D, No. 6 (2003) 1126-1133
2. Helmbold, D.P., Long, D.D.E., Sherrod, B.: A Dynamic Disk Spin-down Technique for Mobile Computing. In *Proceedings of the 2nd ACM MOBICOM*, Rye NY (1996)
3. Ganger, G.R., Kasshoek, M.F.: Embedded Inodes and Explicit Grouping: Exploiting Disk Bandwidth for Small Files. In *Proceedings of the USENIX Technical Conference* (1997)
4. Li, D., Wang, J.: A Performance-oriented Energy Efficient File System. In *Proceedings of International Workshop on Storage Network Architecture and Parallel I/Os* (2004)
5. Mckusick, M., William, N.J., Leffler, S.J., Fabry, R.S.: A Fast File System for UNIX. *ACM Transactions on Computer Systems*, Vol. 2, Issues 3 (1983) 181- 197
6. Papathanasiou, A.E., Scott, M.L.: Energy Efficient Prefetching and Caching. In *Proceedings of the USENIX Technical Conference* (2004)
7. Seltzer, M., Smith, K.A., et. al.: File System Logging versus Clustering: a Performance Comparison. In *Proceedings of the USENIX Technical Conference* (1995)
8. Smith, K.A., Seltzer, M.: File System Aging – Increasing the Relevance of File System Benchmarks. In *Proceedings of the 1997 ACM SIGMETRICS*, Seattle WA (1997)
9. Zheng, F., et. al.: Considering the Energy Consumption of Mobile Storage Alternatives. In *Proceedings of the 11th IEEE/ACM MASCOTS*, Orlando Florida (2003)