Mining based file caching in Hybrid System

Sungwoo Hong Yohwan Kim Youjip Won
Department of Electronics and Computer Engineering
Hanyang University, Seoul, Korea
{*toggiya*|*yowhan82*|*yjwon*}*@ece.hanyang.ac.kr*

***Abstract*—In this work, we propose mining based file caching scheme for hybrid storage with NAND Flash and Hard disk drive. The objective of this research is to expedite the application launch latency in legacy Operating System. We apply data mining algorithm to extract the correlated file access pattern. In selecting the files to be loaded onto NAND flash, we incorporate the physical distance among the correlated file accesses so that the performance improvement can be maximized and apply greedy approach in selecting the files. When we incorporate the physical distance among the files on the disk, we can achieve upto 30% reduction on application lauch latency via relocating the selected files on the NAND Flash.**

***Keywords*-Hybrid Storage System, Mining, SSD, Hard Disk Drive**

I. Introduction

NAND Flash based Solid-state disk (SSDs) gradually takes over the place of hard disk drive (HDD) due to its advantage. It effectively resolves the technical issues which hard disk drive suffers from: fast read/write, high density, shock resistance, and low power consumption. In spite of many advantages of NAND flash device over the HDD, HDD still prevail NAND flash in terms of cost/byte, scale, and reliability.

Given the advantages of NAND flash and HDD, a number of works proposed to combine the two to form a hybrid storage system [9]. In hybrid storage system, non-volatile memory can be used as a temporary buffer. There has been many works which tried to improve I/O performance using SSD in hybrid storage system. Makatos et al. examined performance improvements using SSD as a cache in the I/O path in modern storage systems[8]. Their result showed that compressed caching scheme using SSD significantly improved in terms of CPU cycles for I/O performance of each workload. Although the proposed scheme improved I/O performance significantly, application launch time does not change significantly. Joo et al. proposed reducing application launch time by using flash memory [5]. They modeled the choice of the optimal pinned set as an integer linear programming (ILP) problem and reduced application launch times by 15% and 24% on average by pinning certain portion of blocks accessed by an application in flash memory. In the proposed scheme, however, accessed blocks on HDD may not match to the blocks requested by the application program because of schemes used in block device level, i.e. track buffering which is to improve I/O latency. Further, the computational overhead of solving ILP is expected to be significant.

Modern application accesses a number of files when it starts. They include images, bitmaps, shared objects, and etc. These files are usually small with a few tens of KByte size. Therefore, accessing these files incur significant seek operation. In this work, we aim at reducing the application launch time via caching the files which are accessed at the application launch on the NAND Flash based storage. However, the cost per byte is order of magnitude higher in NAND Flash storage than in hard disk drive. Therefore, it is not likely that we can cache all files involved in application launch at the NAND Flash storage. Kim et al. suggested updating the pinned set with more frequently accessed blocks instead of less frequently accessed blocks [6]. It is true that placing frequently accessed blocks in flash memory can reduce application launch time since it can reduce seek time and rotation delay in HDD. However, corresponding scheme does not consider loading timing of each request to those files. Application launch time varies widely subject to the characteristics of the application, storage system, access pattern, and caching strategy. When a set of files are accessed frequently in consecutive manner, via placing them in adjacent manner in the physical storage, requests of block I/O can be done with a single sequential scan and application launch time can improve.

We develop efficient file relocation algorithm which exploits NAND flash. The basic idea is to identify the correlated file accesses and to quantify the improvement of its caching and to select a set of files to cache at the NAND Flash based disk. We apply Data Mining algorithm which is used in similarity matching of text. Data mining is used to broad applications such as discovery of motifs and tandem repeats in DNA sequences, analysis of customer shopping sequences, scientific and medical processes, and etc [11]. Zaki proposed mining algorithm (SPADE) which can discover sequential patterns efficiently with limited number of database scans [12]. He applied SPADE to decompose the original problem into smaller sub-problems to enhance complexity and illustrated performance improvements with pre-processed data. Ayres et al. proposed depth-first search

[0]This work was supported by IT R&D program(Large Scale hyper-MLC SSD Technology Development, No. 10035202) of MKE/KEIT.

strategy which integrates a depth-first traversal of the search space with effective pruning mechanisms [3]. They improved SPADE using bitmap representation of the data for efficient counting. Pei. et al. proposed new sequential pattern mining method, PrefixSpan, which mines the complete set of patterns[10]. They also substantially reduces the size of projected databases when mining in large databases. Yan. et al. suggested using the concept of frequent closed subsequence to enhance previous sequential pattern mining algorithm by producing less number of sequences [11]. Li. et al. used data mining technique to discover correlated blocks in storage system [7]. They found blocks which are accessed frequently and subsequently more than certain times by analyzing accessed blocks and successfully improved average I/O response time by pre-fetching or rearranging them contiguously. We incorporate file size, file access frequency, and physical distance among the correlated file accesses in determining the files to be cached through mining scheme.

The contribution of our works is as follows. We derived launch sequences for real system workloads. The corresponding launch sequences are extracted from the requests of raw block I/O and irrelevant I/Os to the corresponding target application are separated. Then, we applied a notion of "file correlation" which can be used to detect the redundant movement incurred in HDD. To properly capture it, access patterns of files are analyzed when target application is launched. Mining scheme is used to extract *correlated files* from the access patterns. Files are defined as correlated when a CPU subsequently accesses files within short distance of time more than a threshold number. Through mining scheme, we can narrow down candidate files to relocate them in SSD. Finally, we successfully developed unified framework which reduces launch time by placing highly correlated files on SSD. The extracted correlated files are relocated on SSD considering its available capacity. Furthermore, by placing only subset of files, for example, 10% or 20% of total size to launch the application, significant amount of launch time is reduced compared with other schemes.

The remainder of the paper is organized as follows. Section II describes application launch cost in hybrid storage system. Section III analyzes file access sequence behavior. Mining correlated file accesses scheme is explained in Section IV. Section V carries the result of the performance evaluation. We conclude our work in Section VI.

II. Application Launch Cost in Hybrid Storage System

Hybrid storage system is combination of SSD and HDD. Hybrid storage system, which consists of different storage media, each has different time overhead. The time overhead to launch application can be divided into several categories: seek time, rotation time, and transfer time. Among these, the time except data transfer is called disk overhead. The total time overhead of ith file, f^i in HDD, f^i_{HDD} is defined as in Eq. 1.

$$f^i_{HDD} = f^i_{seek} + f^i_{rot} + f^i_{trans} \quad (1)$$

Eq. 2 illustrate time overhead in hybrid storage system.

$$t_{HY} = \sum_{f^i \in HDD} f^i_{HDD} + \sum_{f^j \in SSD} f^j_{SSD} \quad (2)$$

$\sum_{f^i \in HDD} f^i_{HDD}$ denotes total time overhead in HDD and $\sum_{f^j \in SSD} f^j_{SSD}$ denotes time overhead in SSD, respectively. Since the disk overhead in SSD is significantly lower than in HDD, it would be desirable to store every file in SSD that need to be loaded during the application launch so that the disk overhead can be minimized. However, SSD is very expensive than HDD and which limits the storage capacity. Hence, we propose efficient file selecting scheme to minimize the time overhead by relocating them in faster memory device, SSD. Using SSD, we can reduce access time overhead and transfer time overhead in HDD by reallocating frequently and continuously accessed files. For that, we first analyze file access patterns of target application.

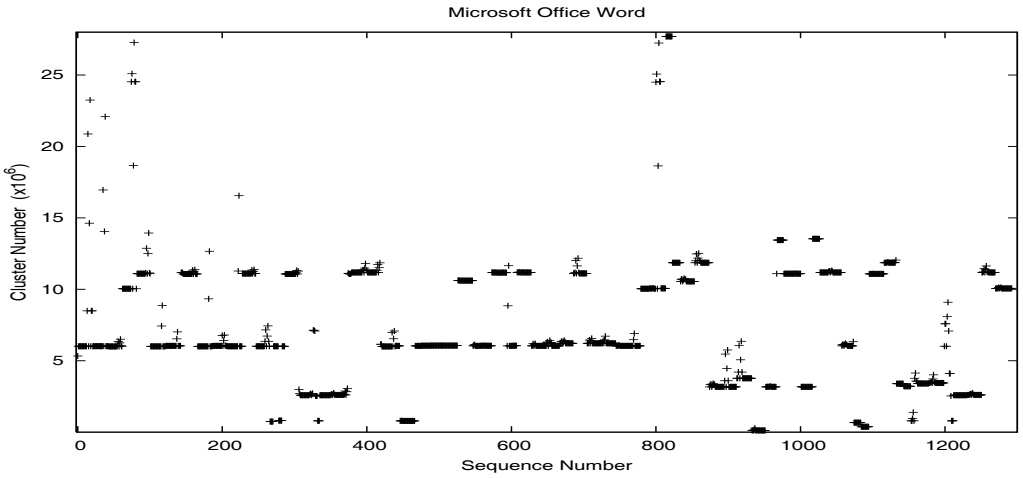
III. Analysis of File Access Pattern

When OS launches an application, the application access a number of files. The number of file accessed during the application launch is subject to the application. For each access to a file, a portion of or entire file is loaded onto memory. Corresponding files include font files, bitmaps, shared objects/libraries, images, and etc. Fig. 1 illustrates access behavior of Microsoft Office Word. Corresponding figure illustrates accessed cluster number of each access for Microsoft Office Word assuming that each accessed file is stored contiguously. Accessing of each file at a distance yields a sequence of random-access like behavior at the storage subsystem.

IV. Mining Correlated File Accesses

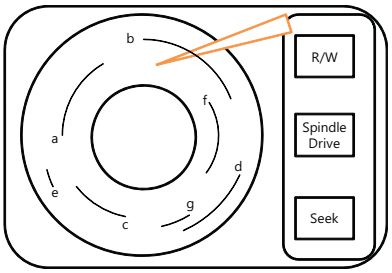
The objective of this work is to determine a set of files to place in a faster storage device (SSD).The key ingredient of this problem is to find right subset of files in file access sequence so that the improvement in application launch is maximized with minimum amount of NAND flash space. There can be many criteria in selecting files: frequency of accessed file, size of accessed file, and etc. We used mining scheme proposed in [7], [11] and modified to properly identify correlated files of each application. One of the advantages of the corresponding algorithm is its ability to obtain similar patterns, not only the identical ones.

The proposed mining scheme consists of three steps to find frequently and subsequently accessed files: mining frequent sequence, frequency and size based file prioritization, and moving gap based file sort. In step of mining frequent sequence, candidates of correlated sequences are extracted out of sequences of each application. In step of frequency and size based file prioritization, files are prioritized based



(a) Start Cluster Number Distribution

Figure 1: Access Sequence Behavior



(a) File Location on Disk

D={abced,abcef,abc,abgc,ac}

File	a	b	c	d	e	f	g
Access order	1	2	3	5	4		
	6	7	8		9	10	
	11	12	13				
	14	15	17				16
	18		19				

(b) Accessed Order

Figure 2: File Location on Disk and Corresponding Access Order

on its frequency and size value from extracted files. Finally, files are again sorted considering the head movement of HDD to minimize disk seek overhead in the process of moving gap based file sort.

A. Mining Closed Frequent Sequence using C-Miner Scheme

The mining scheme proposed in this work is based on a mining algorithm called *C-Miner* [7]. The main idea of C-Miner is to discover block correlations in storage systems. One can map a block to a file, and an access sequence to file access in the storage device. In C-Miner algorithm, they first preprocess access sequences by splitting them in equal size. In this work, we does not split access sequences into shorter sequences to prevent from possible loss of information by splitting. Once it has database of short sequences, corresponding algorithm mines the database and produces frequent subsequences. Corresponding process mainly consists of two stages: (1) generating a candidate set of frequent subsequences including frequent subsequences; and (2) pruning non-closed subsequences from the candidate set. Closed frequent subsequences is a subsequence whose frequency number is different from that of its super-sequences. For example, let there exists sequence database as (tuekd,uektf,tuhkg,ktiju,kjluz). Then, frequent subsequences can be (k:5, t:4, u:5, kt:4, ku:5, tu:4, ktu:4) where the number next to each subsequence denotes the frequency number. Then, the closed subsequences can be (ku:5, ktu:4).

The process of mining candidate set of frequent subsequences is explained as follows using simple example. Access sequence D is illustrated as D = (a b c e d a b c e f a b c a b g c a c). Sequence D accessed seven files a, b, c, d, e, f, g. The location of each file in D and corresponding access sequence is illustrated in Fig. 2. Access to file a occurs at position 1, 6, 11, 14, 18 where 1 denotes the first file access in file access sequence as illustrated in Fig. 2b. Access to file g occurs at position 16. File a and c are most frequently accessed (five times). File d and g are least frequently accessed (once). Let us define the notion of *minimum support*. *Minimum support* is the number of occurrence to be considered as frequent sequence. If we set minimum support as 4 in the example of Fig. 2b, then, there are three files which satisfy minimum support value of 4 in D: a, b, and c. Let D_x be a set of subsequence in D which

does not included in D_x. From D = (*a b c e d a b c e f a b c a b g c a c*), D_a can be denoted as (*bced*, *bcef*, *bc*, *bgc*, *c*). To extract frequent subsequences from D_a, similar approach can be continued. In D_a, *b* and *c* appeared four and five times, respectively. Then, D_{ab} can be obtained from D_a as (*ced*, *cef*, *c*,*gc*). Likewise, D_{ac} can be obtained from D_a as (ed, ef). In D_{ab}, *c* appeared four times. D_{abc} can be derived from D_{ab} as (*ed*, *ef*). Since there are no frequent accessed sequence in D_{ac} , frequent sequences from D_a are *abc* and *ab*. Frequent sequences from other subsequences and longer subsequences can be obtained similarly.

The process of achieving candidate correlated blocks are based on [7], [11]. In their work, considering the length of the achieved sequences of correlated blocks was enough to pre-fetch or rearrange them contiguously in HDD. However, in this work, it is necessary to consider the size of each file in the achieved frequent subsequence as well because we are relocating correlated files (not blocks or clusters which have same size) in the faster memory. Hence, it is necessary to prioritize (or sort) the achieved files after the process of mining frequent subsequence.

B. Frequency and Size based File Prioritization

Through the process illustrated in Section IV-A, several candidate of correlated files (sequences) are obtained. To prioritize achieved files of sequence through mining scheme, we consider the frequency and size of each sequences. The reduction time of seek is negatively proportional to the size of selected files and positively proportional to the corresponding frequency of files, $\frac{fr(f^i)}{\mathcal{Z}(f^i)}$ In case that the selected subsequence can not be relocated on non-volatile memory because of capacity availability, the next file with higher value of $\frac{fr(f^i)}{\mathcal{Z}(f^i)}$ which satisfies the capacity limit of non-volatile memory is selected.

C. Cluster Moving Gap based File Sort

After the process of selecting candidate files and prioritizing them based on the frequency and size, the time to move head in HDD is considered. By considering the sum of distances among files in each sequence, sequence with longer distance is selected first so that corresponding longer head movement can be reduced in SSD. For example, if the distance between f^{i-1} and f^i is longer than the one between f^{j-1} and f^j, f^i is selected first than f^j so that the longer distance of head movement can be saved.

In case that the selected subsequence can not be relocated on non-volatile memory because of capacity availability, the next file from the priority which satisfies the capacity limit of non-volatile memory is selected. The same progress is repeated depending on the availability of flash memory.

V. Performance Evaluation

A. Experimental Setup

For the performance evaluation of the proposed scheme, we used DiskSim disk simulator and NAND disk simulator (NANDSim), respectively [4], [1]. NAND flash memory is emulated by NANDSim included in the Linux Memory Technology Device package [1]. Using NANDSim, virtual NAND flash memory provides flash memory space using RAM. In our evaluation, NANDSim is modified to emulate the actual flash memory performance.

Various applications including Microsoft Office 2007 Word are used to evaluate the proposed scheme. Corresponding input files are acquired using Process Monitor [2]. For the performance evaluation, we measured seek time and transfer time to read/write corresponding cluster which are accessed during the application launch. We assume that if the two read/write I/O command is within 8 block distance on physical device, the corresponding I/O latency consists of one seek time and one transfer time instead of two seek time and two transfer time. This is because block device driver has several schemes which try to improve I/O command such as track buffering, and which reads every block on the same track with only one seek time. In case that the two read/write I/O command is apart from each other more than 8 block distance, two seek time and two transfer time are occurred. We also assume that relocating correlated files is done during the idle time and can predict when an application will be used by analyzing the memory utilization patterns of a user [5].

B. Variation of Launching time

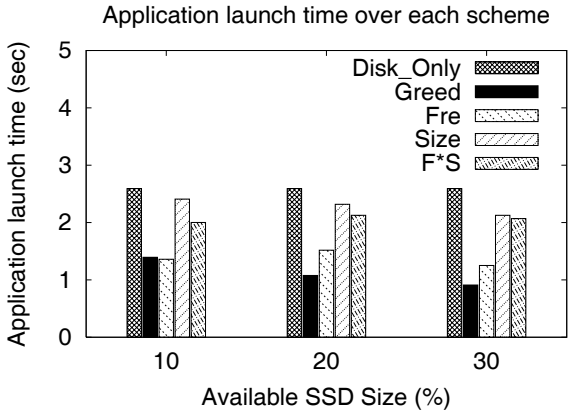


Figure 3: Simulation Result

Fig. 3 illustrates the launching time of Microsoft Office Word over different schemes. Available SSD size is set 10, 20, and 30 % of the total necessary size of application to see the effectiveness of the proposed scheme. The proposed scheme, which we refer as “Greed”, is compared with 4

other schemes: Disk Only (Disk_Only), Frequency based file selection (Fre), Size based file selection (Size), and the one, which we refer as "F*S", which relocates files with highest value of (file size * file access number) first on the non-volatile memory. As can be seen, "Disk_Only" is scheme which has no flash memory attached to it; hence, the launching time of corresponding scheme illustrates highest value compared with other schemes. Since "Disk_Only" does not have flash memory attached to it, the launching time of corresponding does not change. The application launch time of 4 other schemes decreases as the available SSD size increases although there is difference in terms of decreased size depending on the applied schemes.

"Greed" illustrates lowest launching time among all schemes. This is because the proposed scheme efficiently incorporates the correlation information of files in relocating them to flash memory. Generally "Fre" scheme illustrates shorter launch time than "Size" and "F*S" schemes. This is because considering frequency can reduce access to the HDD more than considering size. "F*S scheme" illustrates better performance than "Size", and which suggests that considering both file size and frequency can reduce launch time more than only considering file size.

Table I illustrates statistic results of application launch over other applications including Microsoft Office Word. Corresponding results has been averaged over ten times of experiments for each application. The available SSD size is set as 30% of total necessary size.

Table I: Statistical Results of Application Launch

	EXCEL	**MSN**	**WORD**
Disk_Only (sec)	1.88	3.08	2.59
Greed (sec)	0.38	1.35	0.91
Fre (sec)	0.91	1.69	1.25
Size (sec)	1.67	2.93	2.12
F*S (sec)	1.34	2.66	2.06

As can be seen in the table, "Greed" illustrates shortest application launch time among all applications. For example, "Greed" reduces 79%, 56%, and 64% of total application launch time compared with the case of single storage system. Compared with "Fre" scheme, which illustrates best performance compared with other schemes except "Greed" scheme, "Greed" scheme reduces 58%, 20%, and 27%, respectively. Hence, although there are a little difference in its effect among applications, launching time is significantly reduced when considering correlation among files than considering other factors such as frequency and size.

VI. Conclusion

In this paper, we propose new file caching scheme which relocates files on the flash memory in a hybrid disk to minimizes application launch time through mining scheme. Using mining scheme, we extracted correlated files from access flow when the target application is launched. Extracted correlated files are relocated on the flash so that the launch time is minimized. Proposed scheme is compared with four other schemes: disk-only scheme, frequency-based relocation scheme, size-based relocation scheme, and one which relocates files with highest value of (file size * file access number) first on the non-volatile memory. Experimental results show that the proposed scheme can reduce application launch time upto 50% by only using 10% of total required size.

References

[1] Nandsim linux mtd package[online] available: http://www.linux-mtd.infradead.org/.

[2] Process monitor [online] available: http://technet.microsoft.com/ko-kr/sysinternals/bb896645(en-us).aspx.

[3] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu. Sequential pattern mining using a bitmap representation. In *Proc. of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 429–435, New York, USA, July 2002.

[4] J.S. Bucy, J. Schindler, S.W. Schlosser, and G.R. Ganger. The DiskSim simulation environment version 4.0 reference manual. Technical report, Technical Report CMU-PDL-08-101, Carnegie Mellon University, 2008.

[5] Y. Joo, Y. Cho, K. Lee, and N. Chang. Improving application launch times with hybrid disks. In *Proc. of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, pages 373–382, Grenoble, France, Oct. 2009.

[6] Y. Kim, S. Lee, K. Zhang, and J. Kim. I/o performance optimization technique for hybrid hard disk-based mobile consumer devices. *Trans. on IEEE Consumer Electron*, 53(4):1469–1476, 2007.

[7] Z. Li, Z. Chen, S. M. Srinivasan, and Y. Zhou. C-miner: Mining block correlations in storage systems. In *Proc. of the 3rd USENIX Conference on File and Storage Technologies*, pages 173–186, San Francisco, CA, April 2004.

[8] T. Makatos, Y. Klonatos, M. Marazakis, M. D. Flouris, and A. Bilas. Using transparent compression to improve ssd-based i/o caches. In *Proc. of the 5th European conference on Computer systems*, pages 1–14, Paris, France, April 2010.

[9] R. Panabaker. Hybrid hard disk and readydrive technology: Improving performance and power for windows vista mobile pcs. In *Microsoft WinHEC 2006*, 2006.

[10] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proc. of the 3rd USENIX Conference on File and Storage Technologies*, pages 215–224, 2001.

[11] X. Yan, J. Han, and R. Afshar. Clospan: Mining closed sequential patterns in large datasets. In *Proc. of the Third SIAM International Conference on Data Mining*, pages 166–177, 2003.

[12] M.J. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1):31–60, 2001.