JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 30, 1733-1754 (2014)

Mining-based File Caching in a Hybrid Storage System*

SEONGJIN LEE, YOUJIP WON AND SUNGWOO HONG
Department of Electrical and Computer Engineering
Hanyang University
Seongdong-gu, Seoul, 133-791 Korea
E-mail: {insight; yjwon}@hanyang.ac.kr; toggiya0701@naver.com

In this work, we propose a new mining-based file caching scheme for a hybrid storage disk system. In particular, we focus our efforts on reducing the latency of launching applications. The proposed scheme identifies correlated file accesses in a file access sequence via sequential pattern mining algorithm. Our scheme caches correlated files together to maximize the caching efficiency. The correlated files are extracted from the access patterns through the proposed mining scheme, which consists of three steps: frequent pattern based file extraction, cluster moving gap based file sort, and frequency and size based file prioritization. The extracted correlated files are relocated to an SSD during idle time. DiskSim and NANDSim are used to evaluate the proposed scheme, called Informed Mining. The proposed scheme is compared with a disk only scheme and five other mining based file relocation schemes: Mining based file relocation scheme (Miner), minimum distance based file relocation scheme (Min_Dist), frequency-based relocation scheme (Fre), size-based relocation scheme (Size), and one that relocates files with highest value of (file size * file access number) first to the SSD (Fr*Sz). From the simulation based experiment, launch time is reduced by about 50% using only 10% of sum of all file sizes accessed during a launch of an application.

Keywords: HDD, SSD, hybrid storage, pattern mining, application launch time

1. INTRODUCTION

A hard disk drive (HDD) is a storage device for both an enterprise scale storage system and the general purpose computing environment because of its capacity and price; however, it suffers greatly from its reputation for low bandwidth compared to recent technologies such as a solid state drive (SSD). There are at least three ways to increase the bandwidth of an HDD: (i) increase the rpm of the device, (ii) increase the bit density, and (iii) reduce the seek overhead. The problem in increasing the rpm of the device is that it has to solve two arising issues: first is power consumption and second is thermal temperature of the device [1]. As linear bit density of magnetic surface of an HDD increases, the bandwidth definitely improves; sustained bandwidth of an HDD has improved significantly with 10-15% CAGR [2]. Increase in linear bit density will have a subtle effect on mechanically locating right track, which leads to the third issue. Seek overhead may be one of dire reasons that HDD has hard time overcoming the current bandwidth. Although there has been significant improvement in reducing seek overhead of the device (improved with 5% CAGR [2]), the performance gap between the transfer rate and disk seek overhead is increasing due to different technology growth rates of the two technologies [3]. I/O scheduler and file system strive to make the best use of the

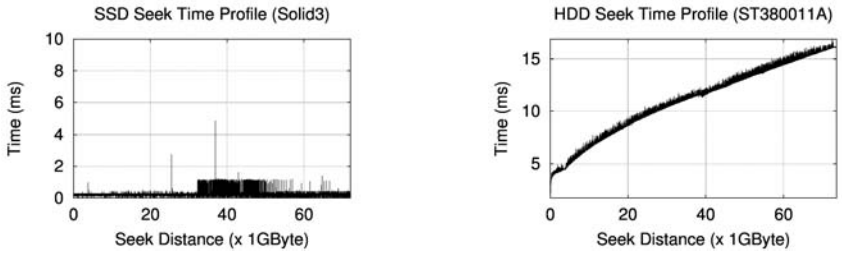
Received May 28, 2013; revised September 19, 2013; accepted December 18, 2013.
Communicated by Krishna M. Kavi.
* This research was supported by the MSIP (Ministry of Science, ICT&Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (NIPA-2014- H0301-14-1017) supervised by the NIPA(National IT Industry Promotion Agency).

hardware and to disguise its problems; however, since the problem lies in physical and mechanical reasons, one needs to search for different approaches from a different angle.

Since NAND Flash was introduced in 1989 [6], NAND Flash-based Solid State Drive (SSD) has earned limelight from various aspects, especially for its lower access latencies for random I/O requests, lower power consumption, and robustness for vibration and temperature. Table 1 summarizes pros and cons of the two devices: sequential and random read operations on SSDs are fast and sequential and random writes on HDDs are slow. It is because of the seek time characteristics of the devices, which are shown in Fig. 1. The seek time profile of the SSD shows almost constant seek latency compared to the HDD. Although SSDs have many favorable characteristics, some issues are still barriers to immediate replacement in the storage market. Three of the issues are price [7], endurance [8], and power budget [9]. The price of SSDs has fallen significantly since it entered the market. In fact, $ per GB became less than a dollar ([10], Table 1); however, it is a still long way to overwhelm the $ per GB of an HDD.

Table 1. HDD and SSD characteristics.

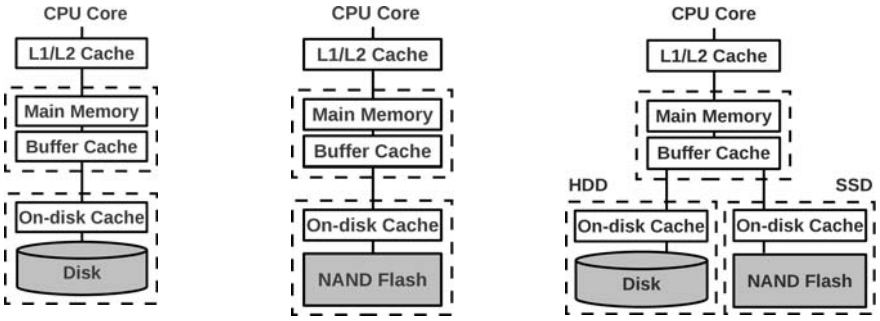
Features	HDD [4]	SSD [5]
Capacity (GB)	4,096	512
$/GB	0.05	0.87
Watts (W)	7.5	0.07
Seq. Read (MB/s)	146	540
Seq. Write (MB/s)	146	330
Rand. Read (IOPS)	37,376	97,000
Rand. Write (IOPS)	37,376	63,000
IOPS/$	178	141
IOPS/W	4,983	900,000



(a) SSD (OCZ Solid3) (b) HDD (Seagate Barracuda ST380011A)]

Fig. 1. Various disk models.

SSDs have limited endurance and can only be erased a certain number of times due to two intrinsic characteristics of the media. SSDs can only out-of-place update data. SSDs have different I/O interface than HDDs; in contrast to HDDs that have two I/O commands (Read/Write) with same I/O latency, SSDs act on three different I/O commands (Read/Write/Erase) with three different I/O latencies for each command. Despite the complex nature of an SSD, it shows better random I/O performance than an HDD; however, it must be noted that as the number of overwrites and erases of data increases, the number of garbage collection [11] also increases, which significantly affects the endurance of an SSD.



(a) HDD (b) SSD (c) Hybrid storage

Fig. 2. Various disk models.

Another significant difference between the two devices is power consumption. Table 1 shows active power consumption of an SSD device; the power consumption of the SSD is about one tenth of that of an HDD. However, it is important to note that managing power consumption is also a problem in an SSD. It is possible to improve the performance by increasing the number of channels that operate concurrently; however, the consequence of increasing the number of channels is that it increases the peak power consumption of the device [9]. Without proper power budget management, it may consume more power than an HDD.

Although it is tempting to exploit the strengths of the SSD within the limited cost boundaries, as Narayanan *et al.* [12] have pointed out, it may be too early to replace the HDD based storage systems with the SSD. Given these characteristics of SSDs and HDDs, a number of works proposed to combine the two to form a hybrid storage system. Fig. 2 illustrates different storage models including an HDD, an SSD, and a hybrid storage. There are mainly two approaches in organizing the hybrid storage system [13-16]. The first is to form a tiered storage system [16, 17]. The second is to use NAND Flash as a cache, *e.g.*, as read cache or as write buffer [18, 19]. In the second approach, NAND-based storage device, such as an SSD, is used to cache frequently used objects, such as files or blocks, to remove the seek time overhead in an HDD and to exploit the small I/O latency characteristics of the NAND Flash-based storage device. SSDs can read highly probable blocks beforehand to reduce the seek time overhead in HDDs. Elaborate object allocation policy is needed to efficiently reduce the seek overhead of an HDD by using flash memory.

Hybrid storage combines heterogeneous storage technologies such as HDDs and SSDs together and offers better I/O performance as well as cost efficient solutions in a storage system. Fig. 2 (c) illustrates brief structure of a hybrid storage model. A strategy in exploiting the hybrid storage is to use an SSD as a hard disk caching solution. For example, Turbo Memory [20] is a hard disk caching solution using flash memory, which processes random disk reads with better performance. Turbo Memory reduces total elapsed time by about two times using NAND-based disk cache compared to an HDD only case. There are some other works that focus on improving throughput and response time in hybrid storage systems [13-15, 18].

In this work, we aim at exploiting a hybrid storage system to reduce application launch latency using sequential pattern mining scheme [21, 22] to obtain highly corre-

lated files. Sequential pattern mining is used in a broad range of applications such as discovery of motives and tandem repeats in DNA sequences [23], analysis of customer shopping sequences [24], scientific [25] and medical processes [26], *etc.* The gist of sequential pattern mining is to find statistically relevant items in a large database with respect to order of the items, where the relevant items found do not need to be consecutive. In executing an application, *e.g.*, spreadsheet, word processor, web browser, *etc.*, the application accesses a number of small files, such as fonts, images, database tables, and a fraction of many other files from the storage device. Accessing a set of small files entails significant seek overhead caused by metadata accesses to open the files and data block accesses across the files. By caching a subset (or all) of these files into flash memory layer in the hybrid storage, we can reduce the application launch latency. To apply sequential pattern mining in extracting frequently accessed files while executing an application, we defined all files accessed during the execution of an application as a large database and used windows size of hundred files to define a transaction in the database.

The contributions of our work are in three folds. First, we successfully developed an elaborate algorithm, which properly identifies the access correlations in a file access sequence, and successfully extracted frequent sequential file access patterns from real system workloads. Second, we successfully minimized launch time of an application using only 10% of sum of all file sizes needed in launching the application as storage capacity of an SSD. As a result, we managed to reduce the average launch time of an application by 50% compared to when using an HDD. Lastly, we implemented a disk only scheme and five other selection algorithms to compare with the proposed algorithm: C-Miner, minimum distance (Min_Dist), maximum frequency (Fre), largest size (Size), and maximum frequency and size (Fr*Sz). We developed unified framework to verify efficiency of our scheme which places frequently accessed sequence of files on an SSD.

The remainder of the paper is organized as follows: Section 3 analyzes file access patterns. Three steps of mining-based file caching scheme is explained in section 4. Section 5 carries the result of the performance evaluation. Section 6 describes related work. We conclude our work in section 7.

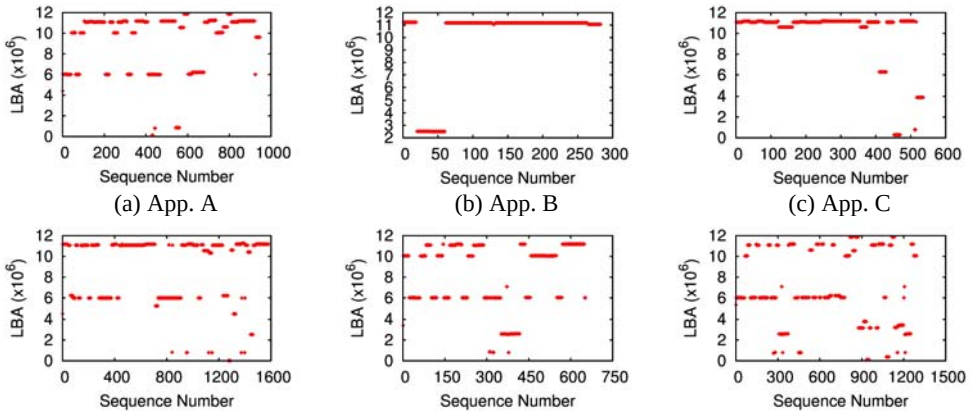
2. APPLICATION LAUNCH

2.1 I/O Characteristics in Launching an Application

Launching an application accompanies a sequence of read or write accesses to a number of files or a portion of a file, which include fonts, bitmaps, shared objects/libraries, images, *etc.* It is highly probable that the files in an aged file system are scattered over different parts of storage device, which increases the time to launch an application. In order to observe how the applications access the storage device during a launch time, we used Process Monitor [27] to capture the I/O trace. The Process Monitor captures information including the names of accessed files, location, size, access time, and the names of processes that issue the respective access requests. The applications we used to examine the file access patterns are Excel, Internet Explorer, Windows Media Player, MSN Messenger, PowerPoint, and Word processor, which are denoted as Application A, B, C, D, E, and F, respectively.

Table 2. Quantile statistics of size distribution of accessed files (A: excel, B: internet explorer, C: windows media player, D: MSN messenger, E: powerpoint, and F: word processor).

Stats	Min	Q1	AVG	Q3	MAX	SUM	Access	File	Trace
Unit	Byte	KByte	Kbyte	KByte	MByte	MByte	Count	Count	Duration
App. A	582	8	2066	653	17	230	946	46	3.4sec
App. B	803	5	746	624	11	20	283	13	17.4sec
App. C	154	8	632	486	5	39	1581	68	20.7sec
App. D	552	48	704	528	11	91	1168	50	6.4sec
App. E	582	8	2196	899	16	208	536	28	10.1sec
App. F	582	140	4248	7617	16	987	1293	57	12.5sec



(a) App. A (b) App. B (c) App. C

(d) App. D (e) App. E (f) App. F

Fig. 3. Spatial locality in application launch (A: Excel, B: internet explorer, C: windows media player, D: MSN messenger, E: powerpoint, and F: word processor).

Fig. 3 illustrates the spatial (LBA) access patterns in application launch. Y-axis in each figure shows the starting LBA of each disk access and X-axis represents sequence of I/Os generated by the application. There is a strong access locality in all access patterns of each application. Table 2 illustrates quantile statistics of the access frequency of individual files in each application launch. The number of accessed files during launch of the applications varies from 13 (Internet Explorer) to 68 (Windows Media Player). On average, 44 files are accessed. The length of the access sequences varies as well. Internet Explorer has the shortest access sequence length of 283, while Windows Media Player has the longest access sequence of 1,581. There is an average of 968 I/Os in launching an application. Although Internet Explorer exhibits the lowest file and access count, it has the second longest trace duration of 17.4 seconds. MSN Messenger, which has the second shortest trace with 6.4 seconds, accesses 50 files with a total access count of 1,168. It shows that trace duration does not necessarily depend on the number of files accessed or the total access count of files. Each file is accessed 22 times on average.

Fig. 4 shows frequency of the five most frequently accessed files of each application. In Internet Explorer and PowerPoint, the access frequency of the top five files account for more than 50% of the total access counts; in other applications, it is about 20%.

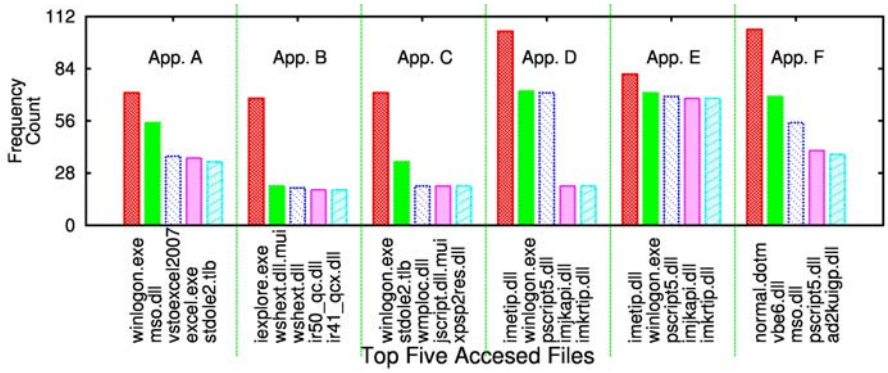
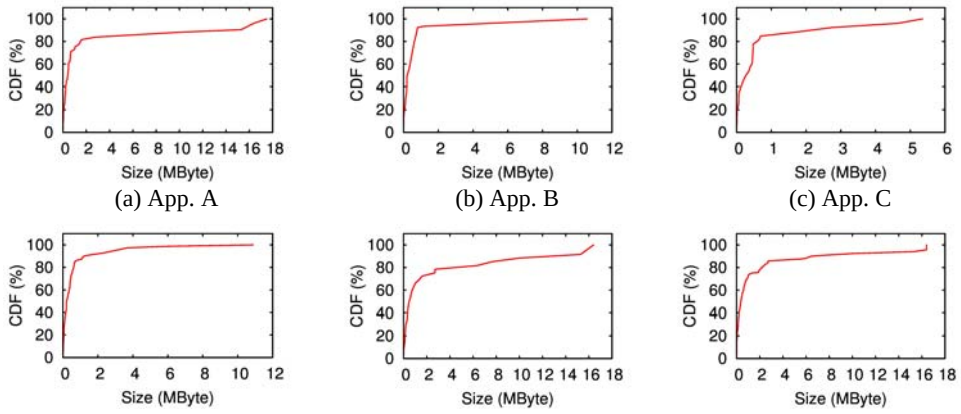


Fig. 4. Statistics of top five accessed files in application launch (A: excel, B: internet explorer, C: windows media player, D: MSN messenger, E: powerpoint, and F: word processor).



(a) App. A (b) App. B (c) App. C

(d) App. D (e) App. E (f) App. F

Fig. 5. Distribution of size of accessed files (A: excel, B: internet explorer, C: windows media player, D: MSN messenger, E: powerpoint, and F: word processor).

We observed that much of the access is targeted on dynamic link libraries. Fig. 5 illustrates CDF of the size of files, and Table 2 describes quantile statistics of the accessed files. Although the size distribution of the accessed files shows that 75% of the files have sizes less than 2MB on average, the five applications excluding Word processor access files less than 650KB. These access characteristics lead to an opportunity for caching frequently accessed files with limited cache size.

2.2 Cost Model for Application Launch

We limit the scope of the cost model for application launch to storage level and do not take into account the effects of host side cache because the purpose of this paper is to understand the performance of hybrid storage. Note that instead of illustrating a complete and detailed analytical cost model, we provide a simple cost model, and delegate measuring actual time consumption in devices to DiskSim [28] and NANDSim [29]. The

time overhead to launch an application can be divided into several elements. Time overhead of an HDD depends on seek, rotation, and transfer time, which are the three most time consuming elements. Each element is defined as t_{seek}, t_{rot}, and t_{trans}, respectively. Let $F = \{f_1, f_2, \ldots, f_i\}$ be set of all files within a system. Eq. (1) defines the total time spent in reading a file, f_n, where $f_n \in F$, from an HDD, $t_{HDD}(f_n)$, which is the sum of the three elements.

$$t_{HDD}(f_n) = t_{seek}(f_n) + t_{rot}(f_n) + t_{trans}(f_n) \quad (1)$$

Note that transfer time $t_{trans}(f_n)$ is proportional to the size of a file, denoted as $|f_n|$. In a hybrid storage system, an HDD is used as a main storage device that stores infrequently read data and an SSD as a cache for the HDD to relocate frequently accessed data. Let $HDD = \{h_1, h_2, \ldots, h_n\}$ and $SSD = \{s_1, s_2, \ldots, s_n\}$ be subset of F such that $HDD, SSD \subseteq F$. The overhead of accessing the files from an HDD and an SSD can be formulated as Eq. (2).

$$T_{HY} = \sum_{h_n \in HDD} t_{HDD}(h_n) + \sum_{s_n \in SSD} t_{SSD}(s_n) \quad (2)$$

T_{HY} in Eq. (2) denotes the total time overhead of accessing files h_n and s_n from the HDD and the SSD, respectively. $\sum_{f_n \in HDD} t_{HDD}(s_n)$ and $\sum_{f_m \in SSD} t_{SDD}(s_n)$ denote the total time to service the files in the HDD and the SSD, respectively. The key ingredient of increasing the performance of a hybrid storage is to determine the subset of files to relocate to an SSD from an HDD. Once a set of frequently accessed files are decided and relocated to an SSD, $\sum t_{HDD}(h_n)$ becomes smaller than $\sum t_{SDD}(s_n)$ which in turn reduces the overall overhead T_{HY}.

3. MINING CORRELATED FILE ACCESSES

3.1 Concept

The objective of this work is to determine the set of files that can be placed in a faster storage device to reduce launch time of an application. To achieve this goal, it is important to determine the right subset of files while minimizing space requirement in an SSD. There are a number of attributes that can be utilized or considered in determining the subset, such as access frequency of a file, file size, ratio between access frequency and file size, *etc*. However, each of these criteria captures only partial information of the files accessed during the launch of an application.

Although exploiting LBA distribution may increase the accuracy of the mining scheme, it is also important to consider the time to mine frequent patterns. In this paper, we use file as basic unit in finding frequent pattern to reduce mining overhead. We will briefly discuss other candidates of selection algorithms in section 3.6.

In this paper, we adapt pattern mining [21] to exploit access patterns and find correlated files within each application to minimize the launch time of the application. The sequential pattern mining analyzes the sequence of data by finding associations between different items in the data. For instance, if f_n is accessed by the host application, how likely the application will also access f_{n+1} in a given time window. One advantage of

C-Miner [21] is its ability to acquire similar patterns as well as identical patterns. Li *et al.* [21] used sequential pattern mining to find correlated blocks that can be used for prefetching and rearranging. They showed that C-Miner can successfully reduce the average I/O response time.

Before we proceed to a detailed description, we provide a brief description of proposed selection algorithm, which is called Informed Mining. The proposed mining scheme consists of three steps. In the first step, we adapt C-Miner algorithm and find a set of file access sequences that satisfies a certain frequency threshold. Second, we sort sequences in the order of sum of seek distance of files within a frequent sequence. We define seek distance of a file as an absolute difference between the start address of a file, f_n, and the end address of the next file, f_{n+1}. This step gives higher priority to the frequent sequences that have the same number of files in the sequence and also incurs larger overhead in seeking the HDD. In the last step in the algorithm, we divide access frequency of each file within the frequent sequence by the corresponding file size. The result of the computation gives us a unit measure that represents frequency per size.

3.2 Mining Frequent Subsequences Using C-Miner

The essence of Informed Mining in reducing the overhead of reading from an HDD is to relocate frequently accessed files to an SSD. Some files can be accessed more than once within a short time interval in the storage device. However, the accesses to the file may not be frequent enough to keep it in the buffer cache. The "frequent sequence" is a sequence of file accesses that is repeated more than a given threshold value. We use pattern mining to discover correlated block accesses in the storage system [21]. The algorithm preprocesses the access sequence by splitting the sequence in equal lengths. Once the pattern mining algorithm accumulates database with sequences, it mines the database and produces frequent subsequences. Corresponding process mainly consists of two stages: (1) generating a candidate set of frequent subsequences; and (2) pruning non-closed subsequences from the candidate set.

Here, we briefly describe the basic notions of pattern mining to formally describe the process of acquiring frequent sequences. More detailed description of the expressions is found in [21, 22]. Let $F = \{f_1, f_2, \ldots, f_n\}$ be a set of all files accessed during launch of an application. A sequence $\sigma = \langle s_1, s_2, \ldots, s_n \rangle$ is an ordered list and is a subset of F such that $\{s_n \subseteq F\}$. Given two sequences $\alpha = \langle a_1, a_2, \ldots, a_n \rangle$ and $\beta = \langle b_1, b_2, \ldots, b_n \rangle$, sub-sequence is defined as follows: Sequence α is a sub-sequence of another sequence β, if and only if there exist $i_1, i_2, \ldots, i_m$ such that $1 \leq i_1 \leq i_2 < \ldots < i_m \leq n$ and $a_1 \subseteq b_{i1}$, $a_2 \subseteq b_{i2}$, …, and $a_m \subseteq b_{im}$. If $\alpha \neq \beta$, then it is denoted as $\alpha \sqsubset \beta$. If files in sequence α are in the same order as sequence β, then we call β a super-sequence of α. A sequence database, $D = \{\sigma_1, \sigma_2, \ldots, \sigma_i\}$, is a set of sequences, and $|D|$ is the number of sequences in the database D. The support of a sequence α in D is the number of sequences of D which contains α as a subsequence. We define frequent sub-sequence as sequences with the support greater than or equal to *min_sup*, where *min_sup* is the minimum support threshold.

Example 1: Suppose a database D with $|D| = 5$ has the following set of sequences {*ab, acde, bcdf, abcd, abcf*}. Then, 2 is the support of *abc* and 3 is the support of *ac* in *D*. Suppose *min_sup* = 3, then *abc* is not a frequent sub-sequence but *ac* is a frequent sub-sequence. ❑

Closed sequence [21] is a sequence with no super-sequence with the same support, *i.e.*, α is closed sequence if and only if $\nexists\beta$ such that $\alpha \sqsubseteq \beta$ and support values of α and β are equal. A frequent closed sequence is defined as a sequence that is frequent and closed at the same time, *i.e.*, α is a closed sequence, and $\alpha \geq min\_sup$ and $\beta \geq min\_sup$.

Example 2: Suppose *min_sup* is 2 in the sequence database in Example 1. Then, there are ten frequent sequences in the database, which are, *ab*: 3, *abc*: 2, *ac*: 3, *acd*: 2, *ad*: 2, *bc*: 3, *bcd*: 2, *bf*: 2, *cd*: 3, and *cf*: 2 where the numbers next to the sequences are the frequency of occurrences. The set of frequent closed sequence in the database consists of nine sequences, which excludes *ad* from the frequent sequences. It is because $ad \in acd$ and support values of *ad* and *acd* are equal to 2, which does not fit in the definition of closed sequence. ❑

Since the files in an application launch are accessed in limited interval, we introduce the time constraint in pattern mining. *max_gap* defines maximum distance between the file accesses in order to be qualified as a frequent sub-sequence.

Example 3: Let *max_gap* be defined as 1 in the sequence database described in Example 1. Suppose we want to compute the support values for *ad* and *ae*, while taking *max_gap* into account; the support values for them are 1 and 0, respectively. When *max_gap* increases to 2, then the support values for the two sequences become 2 and 1, respectively. ❑

3.3 Incorporating Seek Overhead

In selecting the files to relocate to an SSD, we incorporate the frequency of the file access sequences (Step 1) as well as the total seek overhead in each file access sequence (Step 2). When the two file access sequences have the same frequency, it is more beneficial to choose the sequence with larger seek overhead. As described in Eq. (1), the time to fetch a file from a disk consists of the time to position the disk head to the target locations (seek and rotate) and the time to transfer the data (read or write); the dominant time overhead factor is seek and rotate time. Let S^* be the set of files pinned in the SSD after running the sequential pattern mining scheme, and T_{HY} is the time to load the files in an application launch. As described in Eq. (2), T_{HY} is the summation of time spent in loading the files from the HDD, t_{HDD}, and the SSD, t_{SSD}. We assume that garbage collection [11] or log block merge [30] does not happen in the SSD during the application launch because it is much less likely to write than to read files from the SSD. $t_{seek}(f_n)$ is a seek time involved in loading file f_n. If a file is not fragmented, it corresponds to the seek time from the last block of the preceding file access to the beginning of f_n. We define $t_{seek}(f_n)$ as a seek distance from the end of preceding access position to the start of f_n, that is $t_{seek}(f_n) = \left| f_n^{start} - f_{n-1}^{end} \right|$. Minimizing the disk seek time for a given set of files is NP-complete [31] and should be dealt as a separate work.

Example 4: Suppose we use *min_sup* of 3 from the earlier example. Then, there are four frequent closed sequences in the database: *ab*, *ac*, *bc*, and *cd*. Since all sequences have support of 3, we only need to reorder the sequences with respect to seek overhead. Assuming that seek overheads of the sequences are 100, 250, 200, and 300, respectively,

the order of sequences is as follows: *cd*, *ac*, *bc*, and *ab*. This order of sequences is passed onto the next step of the algorithm.

3.4 Incorporating File Size and Access Frequency

As the third step in the Informed Mining scheme, we exploit the size and the access frequency of files in a sequence to determine the files to locate to the SSD. We reorder the files in frequent closed sequence, γ_i, to give priority to files within the sequence to be placed in the SSD first. The purpose of the final step is to utilize the limited space by placing large files with high access frequency that cause long seek latency. Let $\alpha = \langle f_1, f_2, \ldots, f_n \rangle$ be a frequent closed sequence in a database D and the size and access frequency of a file, f_i, in α are denoted as $|f_i|$ and f_i^{FR}, respectively. Note that access frequency of a file, f_i, is the number of times the file, f_i, appears in the database D. Then, the ratio of the size and access frequency, C_i, is computed as $C_i = f_i^{FR}/|f_i|$. After the computation, the files in the sequence are sorted in the order of the largest ratio to the smallest. The sorted files are relocated in the order of the ratio as long as the total file size does not exceed the capacity of the storage.

Example 5: Assume that files *a*, *b*, *c*, and *d* from the earlier example have sizes of 2KB, 6KB, 4KB, and 6KB, respectively, and the size of the SSD is limited to 6KB. The access frequencies of files *a*, *b*, *c*, and *d* in the database *D* from Example 1 are 4, 4, 4, and 3, respectively. Then the ratios of file size and access frequency of each file are 2, 0.6, 1, and 0.5, respectively. The size of first sequence, *cd*, is 10KB and it does not fit in the SSD. Using the size and access frequency ratio, we chose *c* to relocate to the SSD because it has higher ratio. ❑

3.5 Overhead of Computing Frequent Subsequences

We measured the time overhead of running the algorithm, which includes mining for frequent sequences (Step 1) and sorting the results in terms of seek distance (Step 2) and the ratio of file size and file frequency (Step 3). Table 3 shows the time required to run the Informed Mining scheme and to relocate the files to the SSD; it is measured in the system described in Table 6. Note that running of the Informed Mining scheme is performed in idle time so as not to interfere with system performance. Acquiring fre quent sequences and relocating them to the SSD take about 2.3 sec and 1.2 sec on average, respectively. Given that this is one time cost and can be executed in background, we carefully argue that the overhead of running the Informed Mining scheme is acceptable.

Table 3. Time overhead in mining frequent sequences and relocating them to the SSD. (A: excel, B: internet explorer, C: windows media player, D: MSN messenger, E: powerpoint, and F: word processor).

Overhead	App. A	App. B	App. C	App. D	App. E	App. F
Mining (msec)	656	156	7188	859	1765	3094
Relocation (msec)	642	328	3436	719	937	1093

Table 4. File access sequence and their LBA, size, and access frequency.

	f_1	f_2	f_3	f_4	f_5	f_6
LBA	1000	7600	10200	4082	18200	1800
Size	4KB	16KB	8KB	32KB	24KB	32KB
Frequency	6	100	30	57	5	3

3.6 Selection Algorithms

In this section, we provide an example of how different selection algorithms work. Note that description and examples of the informed mining are shown in sections from 3.2 to 3.4. For example, let us consider a file access sequence $\rho = \langle f_1, f_2, f_3, f_4, f_5, f_6 \rangle$. Note that for Min_Dist and Fre, if I/O requests access portions of a file at different starting LBAs, we regard them as different files in set F as described in section 3.2. The size, disk location, and the access frequency of the files are summarized in Table 4. For the sake of illustration, we assume that an LBA denotes a sector (512Byte) and an SSD size is 64KB. We compared application launch time of the Informed Mining with a baseline Disk Only model and four other relocation schemes. Disk_Only scheme is a baseline that shows application launch latency in an HDD only system to provide comparison to various relocation schemes. The four relocation schemes are Miner, Min_Dist, Fre, and Fr*Sz.

Miner scheme mines for correlated file access sequences within access window of 100 with C-Miner algorithm. Section 3.2 illustrates how Miner works. Upon retrieving mined correlated file access sequences, most frequently accessed sequences are relocated to an SSD until the SSD is full.

Min_Dist scheme exploits head movement involved in fetching files. As we have defined earlier, at the start of section 3.3, seek distance of a file is defined as an absolute difference between the start address of f_n and the end address of f_{n-1}. Since there is no preceding file in f_1, we use its starting address as seek distance. In Min_Dist scheme, we do not incorporate the access frequency of each file but only consider the seek overhead in determining the files to load to an SSD. In Min_Dist, the file with the longest seek distance is relocated to an SSD first. Min_Dist algorithm recalculates the seek distance between the consecutive file accesses after a file is relocated to an SSD. Let us provide an example to help the understanding. Fig. 6 illustrates a sequence of disk accesses and the distance between the consecutive disk accesses. Table 4 describes the location of each file. Here is an example of computing the seek distance for f_2. The access f_2 starts at 7600. Since we assumed that LBA denotes a sector address, the end address of f_1 is 1008. Seek distance of f_2, denoted as (2) in Fig. 6, is 6592. Similarly, the distances of (1), (3), (4), (5), and (6) are 1000, 2568, 6134, 14054, and 16048, respectively. Since (6) is the longest, Min_Dist scheme relocates file f_6 to the SSD. After relocating (5), which is the file with the second longest seek distance, Min_Dist stops relocating files to the SSD because it cannot relocate the file with the third longest seek distance, which is (4).

Fre scheme sorts the files according to frequency of file access count, and relocates files with the highest access frequency to an SSD until the storage capacity is full. When we

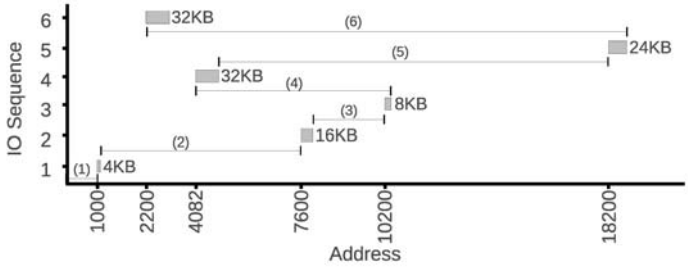


Fig. 6. Layout of files and their I/O sequence, address, and size.

sort the files in the order of frequency, from the highest to lowest, the order is as follows: f_2, f_4, f_3, f_1, f_5, f_6. Fre scheme relocates files f_2, f_4, f_3 and f_1, to the SSD; the four files sum up to 60KB.

Size scheme relocates files based on their size. The priority goes to the large sized files. The rationale behind this approach is that the latency improvement becomes more significant as one migrates larger files from an HDD to an SSD due to the performance difference between the HDD and the SSD. From the sequence ρ described previously, files sorted in descending order by size are as follows: f_4, f_6, f_5, f_2, f_3, f_1. Note that when files with the same size are in the list, the file with the lower address has higher priority to be relocated to the SSD. Size scheme relocates files f_4 and f_6 to the SSD; the two files sum up to 64KB.

Fr*Sz scheme takes into account both frequency and size of accessed files to relocate to an SSD. In Fr*Sz scheme, the size and the access frequency of a file are multiplied and used as a metric that decides the priority. Files are relocated to the SSD in the order of the size of the multiplication result, largest to smallest. The results of multiplication in this criteria are as follows: f_1: 24, f_2: 1600, f_3: 240, f_4: 1824, f_5: 120, and f_6: 96. Fr*Sz scheme relocates files f_4, f_2, and f_3, to the SSD; the three files sum up to 56KB.

Informed Mining is a mixture of three aforementioned schemes: Miner, Min_Dist, and Size scheme. First, it retrieves correlated file access sequences using C-Miner algorithm. Second, we compute the sum of seek distance of each frequent sequences and reorder them such that a sequence with larger sum of seek distance of files has higher priority to be relocated to an SSD. Then, Informed Mining divides access frequency by size of each file within a correlated access sequence to determine which files in a frequent sequence are going to be relocated to an SSD first.

4. PERFORMANCE EVALUATION

4.1 Experiment Setup

We studied the effect of the Informed Mining under six different applications (Microsoft Excel, Internet Explorer, Windows Media Player, MSN Messenger, PowerPoint, and Word). We collected the file access traces in launching these applications using

Process Monitor [27] in Windows XP. The Process Monitor captures object name, Read/ Write command, access time, path of an object, offset within an object, and the size of the object. Access sequence of an application may vary widely depending on buffer and page cache on host computer. Although we use static analysis with captured I/O trace of a system, we assume that trace of running system can be captured. One way to have file information along with LBA and length pair is to place filter driver between application layer and file system or between file system and device driver. In order to minimize the effect of host side cache, we flushed the buffer cache before acquiring the trace.

Experiment environment is described in Table 5. We built a hybrid storage using DiskSim [28] and NANDSim [29]. We used DiskSim 4.0 simulation environment and Quantum Atlas III (9.1GB, 7200RPM) disk model. The NANDSim is configured to model 1GB K9K8G08U0A NAND Flash memory [32] with 2KB page, 128KB block, and program time of 200 μsec. Note that we limit the NAND flash memory size as 10%, 20%, and 30% of total memory required for the application launch to see the effectiveness of the proposed scheme. For example, if an application requires twelve files which sum up to 36MB, then we set the size of flash memory to be 3.6MB, 7.2MB, and 10.8 MB. Table 2 illustrates basic statistics of accessed files in launch of the applications.

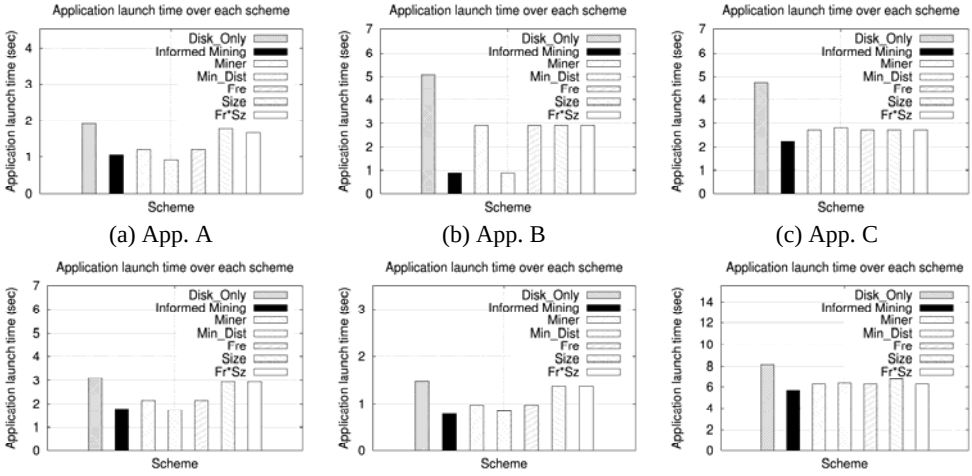
Table 5. Environment.

Parts	Specification
CPU	Intel Pentium 4 3.0GHz
RAM	2GB
Disk	ST3320620A 320GB, 7200 RPM
OS	Linux Fedora Core 7 Kernel 2.6.21
Simulation	DiskSim [28] and NANDSim [29]
DiskSim	Quantum Atlas III (9.1GB, 7200RPM)
NANDSim	K9K8G08U0A NAND Flash memory [32]
Application	Microsoft Excel, Internet Explorer, Windows Media Player, MSN Messenger, PowerPoint, and Word

4.2 Effect of Caching Strategy

The Informed Mining is compared with six other schemes: Disk Only scheme (Disk_Only), C-Miner based file selection scheme (Miner), minimum distance based file selection scheme (Min_Dist), frequency based file selection scheme (Fre), size based file selection scheme (Size), and Fr*Sz scheme which relocates files with the highest value of (file size * file access count) first on the flash layer. The difference between Miner and Fre is that Miner relocates files that are accessed frequently and simultaneously within a short time interval, while Fre relocates files that are accessed frequently during launch time of an application. Min_Dist is a scheme that considers distances between the files in the disk and relocates the files with the longest distances. Size scheme relocates big files first to an SSD. Brief description of each scheme is illustrated in section 3.6.

Fig. 7 illustrates the launch time of each application under different schemes. Disk_ Only scheme showed the longest launch time. The Informed Mining had the shortest launch time in all applications except for App. A and App. D. In App. A and App. D, Min_Dist performed better than Informed Mining by about 15% and 4%, respectively.



(a) App. A (b) App. B (c) App. C

(d) App. D (e) App. E (f) App. F

Fig. 7. Application Launch Time Variation under Each Scheme with Minimum SSD Availability (A: Excel, B: Internet Explorer, C: Windows Media Player, D: MSN Messenger, E: PowerPoint, and F: Word processor).

This is because the proposed scheme efficiently utilizes the frequency and the head distance between files simultaneously when relocating files to an SSD. Depending on the application, Miner and Min_Dist illustrate different efficiency in reducing launch time of an application. In App A and App D, performance advantage of mining-based file caching is marginal. This is because in these launch sequences, most of the launch latency is due to seek time overhead. It is practically impossible to accurately compute the seek time overhead of a given I/O access due to the complex sector geometry of modern hard disk drive [33]. When disk head movement overhead constitutes a dominant fraction of launch latency, it is possible that Miner algorithm may not be able to find an optimal solution. Miner scheme illustrates superior performance to Min_Dist scheme in Windows Media Player and Word Processor. In Windows Media Player and Microsoft Word, the movement of head in the HDD has less effect on launch time of the application. Hence, Min_Dist yields the worst performance.

Miner scheme and Fre scheme produce similar performances because both schemes relocate files based on access count of files. In general, Fre scheme exhibits better launch time than Size and Fr*Sz schemes because frequency based relocation scheme successfully reduces unnecessary access to the hard disk drive. Hence, the launching time (or seek time in hard disk drive) can be reduced compared to Size and Fr*Sz. Relocating large sized files to the SSD may ease the overhead of accessing large files from the HDD; however, the result speaks otherwise. In all of the applications, Size and Fr*Sz schemes compete for the last place in the measurements. Three things should be noted. First, three quarters of files are less than 900KB in most of the applications (Table 2.). Second, the largest file in each application consumes an average of 54% of available capacity of the SSD. Third, top five frequently accessed files are mostly small files. From these findings, we can assume that size oriented relocation schemes have a high probability of moving infrequently accessed files to the SSD.

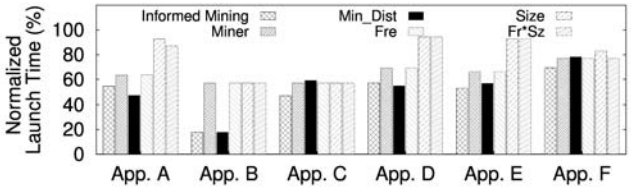
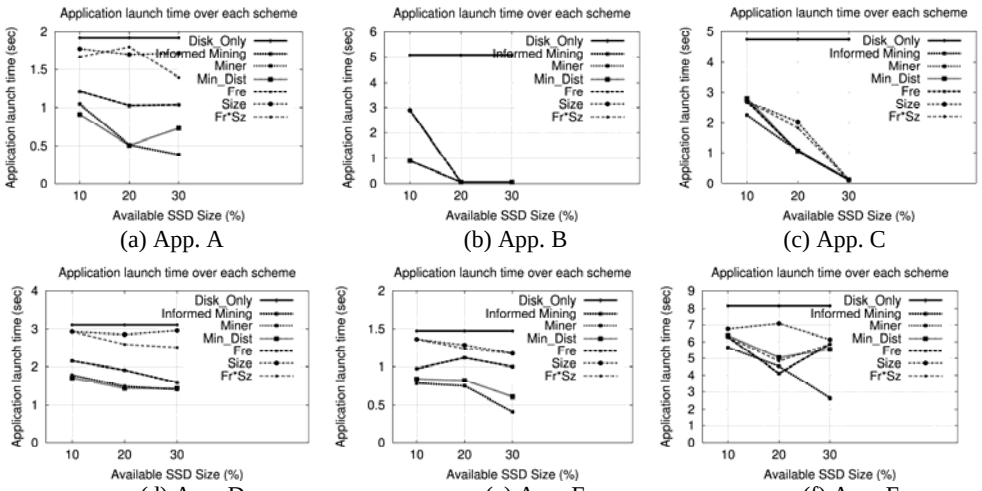


Fig. 8. Normalized application launch time (A: excel, B: internet explorer, C: windows media player, D: MSN messenger, E: powerpoint, and F: word processor)

Fig. 8 compares six different relocation schemes we have implemented, which are normalized with respect to launch time of an application in Disk Only environment. It shows the amount of time reduced by applying the scheme using only 10% of sum of all file sizes accessed in launch of an application. From the result of Min_Dist scheme, we can deduce that many frequently accessed files are far apart from each other. On average, application launch time is reduced by 46% using Min_Dist and 50% using the Informed Mining. It shows that launching time is significantly reduced when the information on the file size, frequency, and distance of files are properly exploited.

4.3 Effect of Cache Size

A rule of thumb is to have a fast cache device with abundant storage capacity for slower HDD to aid in reducing the launch time of an application; however, we limit the size of the SSD to understand the effect of size constraints during an application launch. We varied the percentage of available storage capacity to be 10%, 20%, and 30% of the sum of file sizes used in launch of an application. We measured time required to launch an application under different relocation schemes: Disk Only, Miner, Min_Dist, Fre, Size, Fr*Sz, and Informed Mining. Six applications we have used are Word, Excel, Power-Point, Internet Explorer, Windows Media Player, and MSN Messenger. Fig. 9 illustrates the effect of changing the available storage capacity of the SSD.



(a) App. A (b) App. B (c) App. C

(d) App. D (e) App. E (f) App. F

Fig. 9. Effect of cache size under each scheme (A: excel, B: internet explorer, C: windows media player, D: MSN messenger, E: powerpoint, and F: word processor).

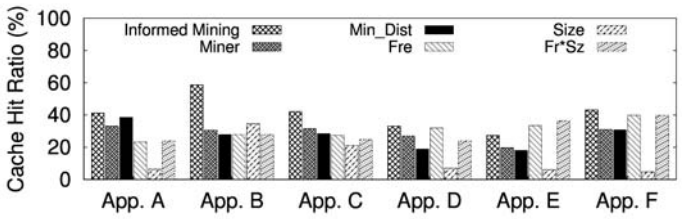


Fig. 10. Cache hit ratio (A: excel, B: internet explorer, C: windows media player, D: MSN messenger, E: powerpoint, and F: word processor).

Observe that as available capacity of the SSD increases, the launch time of an application decreases. 30% of storage is sufficient in relocating all the essential files to the SSD in Internet Explorer and Windows Media Player because the result shows that six different schemes all converge on one point. It can be justified by consulting quantile statistics of size distribution of accessed files in Table 2. For example, the sum of files in launching Windows Media Player is 39MB and the available space as cache for the HDD is about 11.7MB (30% of sum of all files). Note that Size and Fr*Sz schemes show longer launch time compared to the other schemes, which means that small files are more frequently used in the launching process. CDF in Fig. 5 (c) shows that 80% of all files are less than 500KB. Naïve calculation gives us about 24 files that can be stored in the SSD, which is more than one third of files accessed during launch of the application. Thus, we can deduce that most of frequently accessed sequences of files are relocated to the SSD. Overall, Informed Mining exhibits the shortest launch time in all applications. Since the Informed Mining combines merits of all other schemes, it is by far the most efficient relocation scheme among the six schemes we have implemented. It successfully reduces the launch time by selecting frequent sequence of files and relocates them to the SSD. Cache hit ratio of using different schemes are illustrated in Fig. 10. The result shows average cache hit ratio of using 10%, 20%, and 30% of sum of all files required to launch an application as total capacity of NANDSim. Informed mining yields the best average cache hit ratio with 41%. Fre, Miner, Fr*Sz, and Min_Dist are the next in line with 31%, 30%, 29%, and 27%, respectively. Size based relocation scheme came in last with 13%.

We compared the cache hit ratio of the Informed Mining with three other schemes suggested in other papers. Chen *et al.* [34] introduces Hystor which manages SSDs and HDDs as one single block device where the SSDs store blocks that result in long latencies and filesystem metadata. Hong *et al.* [35] exploits SLC/MLC as disk cache in a non-volatile cache (NVC) embedded HDD; they split the cache area into two regions, SLC and MLC, as first NVC and second NVC, respectively. All writes are first moved to SLC region; cold data from SLC region is evicted first to MLC and later on to the HDD.

Table 6 summarizes the configuration of aforementioned hybrid storages. All of the papers in Table 6 uses sizes larger than the proposed scheme, yet achieve similar average hit ratio as the proposed scheme. Although it is not fair to compare the hit ratio of different workloads, it seems acceptable to argue that limited storage size of our scheme does not fall behind the cache hit ratio of other schemes. Table 6 shows application launch time of different workloads while setting the size of the SSD as 10%, 20%, and

Table 6. Comparison of different hybrid storage schemes (10%$^+$: 10% of sum of all file sizes used in launching an application is used as size of the SSD.)

Schemes	Environment	HDD	SSD	Workload	Hit Ratio
Hystor [34]	Real System	73GB	32GB	Postmark, Email, TPC-H	40%
NVC [35]	Simulator	80GB	1GB	Bonnie++, Desktop, Financial 1, 2	42%
Informed Mining	Disksim, NANDSim	9.1GB	10%$^+$	Excel, Internet Explorer, WMP, Messenger, PPT, Word	41%

30% of sum of all files used in application launch. The result shows that average of 43% of application launch time is dropped by increasing the storage size to 30% from 10%; we can expect the same in cache hit ratio because the application launch time and the cache hit ratio are closely related metrics. Although our result only depicts the result on a desktop environment, the same can be prospected in the server environment.

5. RELATED WORK

Cost is one of the factors that should be considered in installing and managing a storage system. Narayanan *et al.* [12] analyzed the cost of replacing storage devices to SSDs, as well as creating a heterogeneous system that uses an SSD as an intermediate layer in a traditional storage system. They showed that although an SSD is an energy-efficient and better performing alternative for a storage system, the hardware cost of the device holds back a complete replacement of the HDD. As a means to reduce the hardware cost while enjoying the performance of SSDs, Kim *et al.* [16] introduced HybridStore, which finds the most cost effective storage configuration and manages to reduce the average response time by 71% compared to an HDD-based system.

Introducing an SSD or NAND Flash memory as an intermediate tier in the system allows increasing the I/O performance of a system. Kgil *et al.* [17] used flash memory as a disk cache which is split into read and write region. To provide a solution to wear-level issue of the flash memory in the system, they kept account of erase counts of each block and used LRU based replacement. Koltsidas *et al.* [36] introduced an on-line optimal placement algorithm which places database workloads based on workload characteristics; they placed a read-intensive workload on the flash memory and write-intensive workload on the HDD. To reduce overall I/O cost of heterogeneous storage system, they also presented a buffer replacement policy, which considers the probability of a page being accessed in the near future and the cost of evicting the page from the memory. Soundararajan *et al.* [37] took a different approach in improving the I/O performance of a system; they used an HDD as a log-structured buffer cache for an SSD, which reduced the average I/O latency by 56% and extended the life of the SSD by a factor of two in the system.

Another reason to use an SSD in a storage system would be to resolve energy consumption issue. Chen *et al.* [38] introduced SmartSaver that exploits flash memory as a buffer for caching and prefetching data from the disk; trace-driven simulation of the design showed that 41% of energy is saved by storing frequently reused data on the flash memory. In an effort to reduce energy consumption in an enterprise storage system, Lee *et al.* [39] used an SSD based cache for RAID system; their approach copies read data to the SSD in anticipation of servicing the data in the near future and buffers all the writes

on the SSD. Using cello99 and SPC traces, they reduced energy consumption by maximum of 14%. Bisson [15] presented an I/O scheduling algorithm exploiting the SSD in the system to solve the bottleneck problem in the HDD that has a very long I/O write latency; write latency was reduced up to 70% by using the SSD as a cache compared to the system without the SSD. Makatos *et al.* [40] presented cache system called FlaZ that uses online compression to reduce the size of the data and store compressed data in the SSD. Kim *et al.* kept track of access frequency and age of an LBA group to allocate most frequently accessed LBA groups to nonvolatile memory [18]. They kept Most-Frequently-Updated LBA groups in DRAM cache and later demoted aged LBA groups to nonvolatile memory. Hsieh *et al.* [41] proposed flash memory cache to reduce energy consumption and read response time; trace-driven simulation showed that energy can be reduced by 20% and response time by 67%.

Some researchers have focused their work on supporting ensemble of storage device through a file system. Garrison *et al.* [42] implemented Umbrella file system which provides flexibility in directing workloads with specific access characteristics to a device that can best exploit them. Josephson *et al.* [43] created a file system that exploits virtualized flash storage layer in FusionIO's ioDrive; use of virtualized flash storage layer in the system allows hiding the latency of bulk erasures and managing wear leveling.

Determining the data that should be moved to faster SSD in a hybrid storage configuration is an issue. Pritchett *et al.* [44] exploited skewed block popularity distribution in SieveStore. They observed that less than one percent of popular blocks account for 14% to 53% of all accesses each day and the 99% of all blocks are visited 10 or fewer times. With their findings, they devised a selective cache allocation scheme to manage access counts of popular blocks and store them in an SSD. Chen *et al.* [34] proposed Hystor which integrates high speed SSD with low price HDD. In their heterogeneous storage, the SSD has two roles. First role is to store important and highly valued data such as metadata. Identifying high-cost blocks are based on access count of blocks and pages within the block. Second role is to act as a write back buffer to improve the write performance of the system. Payer *et al.* [45] produced a prototype of a hybrid storage device called Combo Drive which attempts to optimize the storage through file system level strategies. Their approach gives priority to executable files, program libraries, randomly accessed files, and frequently accessed files; these files are copied to an SSD.

Although existing schemes improve performance in terms of I/O response time and power consumption, the improvement on application launch time is marginal. The closely related work on minimizing launch time of an application using an SSD is proposed by Joo *et al.* [14, 46]. Prefetching algorithm proposed in Joo *et al.* [14] utilizes system profilers such as blktrace and strace to identify common I/O sequences and their file names in the I/O trace; their approach reduced launch time by 28% compared to an HDD only system.

To find frequently and simultaneously accessed files, Li *et al.* [21] applied pattern matching algorithm, which is computationally much less expensive than the one used in [22]. Yan *et al.* [22] proposed to exploit block access correlations for storage caching, prefetching, data layout, and disk scheduling. While this approach exhibits significant improvement, assigning the blocks to different storage devices requires another management layer in the I/O stack, which can be an additional overhead from performance and compatibility's point of view.

6. CONCLUSION

Recent advent of SSDs has greatly increased storage performance; however, the price of the media is yet too high to replace HDDs entirely. In this paper, as a remedy to the issue, we propose to use a hybrid storage to meet price and performance criteria. In an effort to minimize the launch time of an application and to show the direct impact of using the hybrid storage, we propose to use an SSD as a cache in the hybrid storage. In order to achieve the goal, we used pattern mining to find frequent sequences and extracted correlated files acquired from launching of applications. After acquiring frequent sequences from the proposed scheme, we relocated the corresponding sequences to an SSD to reduce the launch time of an application. We compared the proposed scheme called Informed Mining with five other schemes: Mining based file relocation scheme, minimum distance of files based relocation scheme, frequency based relocation scheme, size based relocation scheme, and frequency times file size based relocation scheme. Our experiment shows that the Informed Mining reduces an average of 50% of launch time of an application, using only 10% of the sum of all file sizes accessed during a launch of an application.

7. Acknowledgement

This research was supported by the MSIP(Ministry of Science, ICT&Future Planning), Korea, under the ITRC(Information Technology Research Center) support program (NIPA-2014-H0301-14-1017) supervised by the NIPA(National IT Industry Promotion Agency)

REFERENCES

1. S. Gurumurthi and A. Sivasubramaniam, "Thermal issues in disk drive design: Challenges and possible solutions," *Transactions on Storage*, Vol. 2, 2006, pp. 41-73.
2. J.-J. Maleval, "HDD technology trends," http://www.storagenewslettercom/rubriques/hard-disk-drives/hdd-technology-trends-ibm/, 2011.
3. W. W. Hsu and A. J. Smith, "The performance impact of I/O optimizations and disk improvements," *IBM Journal of Research and Development*, Vol. 48, 2004, pp. 255-289.
4. Seagage, "Desktop HDD ST4000DM000 specification," http://www.seagate.com/internal-hard-drives/desktop-hard-drives/desktop-hdd, 2012.
5. Samsung, "512gb 2.5-inch SSD 840 pro series," http://www.samsung.com/us/computer/memory-storage/MZ-7PD512BW-specs, 2012.
6. M. Momodomi, Y. Itoh, R. Shirota, Y. Iwata, R. Nakayama, R. Kirisawa, *et al.*, "An experimental 4-Mbit CMOS EEPROM with a NAND-structured cell," *IEEE Journal of Solid-State Circuits*, Vol. 24, 1989, pp. 1238-1243.
7. V. Kasavajhala, "Solid state drive vs. hard disk drive price and performance study," ed., Dell Technical White Paper, Dell Power Vault Storage Systems, 2011.
8. S. Boboila and P. Desnoyers, "Write endurance in flash drives: measurements and analysis," in *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, 2010, pp. 9:1-9:14.
9. B. Yoo, Y. Won, J. Choi, S. Yoon, S. Cho, and S. Kang, "SSD characterization: from energy consumption's perspective," in *Proceedings of the 3rd USENIX Con-*

ference on Hot Topics in Storage and File Systems, 2011, pp. 3:1-3:5.
10. L. Ramirez, "High-capacity SSDs finally match the per-GB prices of smaller SSDs," http://dealnews.com/features/High-Capacity-SSDs-Finally-Match-the-per-GB-Prices-of-Smaller-SSDs/622014.html, 2012.
11. W. Bux and I. Iliadis, "Performance of greedy garbage collection in flash-based solid-state drives," *Performance Evaluation*, Vol. 67, 2010, pp. 1172-1186.
12. D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron, "Migrating server storage to SSDs: analysis of tradeoffs," in *Proceedings of the 4th ACM European Conference on Computer Systems*, 2009, pp. 145-158.
13. R. Panabaker, "Hybrid hard disk and readydrive™ technology: Improving performance and power for windows vista mobile PCs," in *Proceedings of WinHEC*, 2006.
14. Y. Joo, J. Ryu, S. Park, and K. G. Shin, "FAST: quick application launch on solid-state drives," in *Proceedings of the 9th USENIX Conference on File and Storage Technologies*, 2011, pp. 19:1-19:15.
15. T. Bisson and S. A. Brandt, "Reducing hybrid disk write latency with flash-backed I/O requests," in *Proceedings of the 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2007, pp. 402-409.
16. Y. Kim, A. Gupta, B. Urgaonkar, P. Berman, and S. Anand, "HybridStore: A cost-efficient, high-performance storage system combining SSDs and HDDs," in *Proceedings of IEEE 19th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2011, pp. 227-236.
17. T. Kgil, D. Roberts, and T. Mudge, "Improving NAND flash based disk caches," in *Proceedings of the 35th International Symposium on Computer Architecture*, 2008, pp. 327-338.
18. Y.-J. Kim, S.-J. Lee, Z. Kangwon, and J. Kim, "I/O performance optimization techniques for hybrid hard disk-based mobile consumer devices," *IEEE Transactions on Consumer Electronics*, Vol. 53, 2007, pp. 1469-1476.
19. U. Maheshwari, "Designing storage systems with flash," in *Proceedings of the 4th USENIX Workshop on Hot Topics in Storage and File Systems*, 2012.
20. J. Matthews, S. Trika, D. Hensgen, R. Coulson, and K. Grimsrud, "Intel® turbo memory: Nonvolatile disk caches in the storage hierarchy of mainstream computer systems," *Transactions on Storage*, Vol. 4, 2008, pp. 1-24.
21. Z. Li, Z. Chen, S. M. Srinivasan, and Y. Zhou, "C-Miner: mining block correlations in storage systems," in *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, 2004, pp. 13:1-13:14.
22. X. Yan, J. Han, and R. Afshar, "CloSpan: Mining closed sequential patterns in large datasets," in *Proceedings of the 3rd SIAM International Conference on Data Mining*, 2003, pp. 166-177.
23. P.-Y. Wong, T.-M. Chan, M.-H. Wong, and K.-S. Leung, "Efficient algorithm for mining correlated protein-DNA binding cores," in *Database Systems for Advanced Applications*, Vol. 7238, S.-G. Lee, Z. Peng, X. Zhou, Y.-S. Moon, R. Unland, and J. Yoo, eds., Springer Berlin, Heidelberg, 2012, pp. 470-481.
24. T. Nakahara and K. Yada, "Analyzing consumers' shopping behavior using RFID data and pattern mining," *Advances in Data Analysis and Classification*, Vol. 6, 2012, pp. 355-365.

25. K. Kaneiwa and Y. Kudo, "A sequential pattern mining algorithm using rough set theory," *International Journal of Approximate Reasoning*, Vol. 52, 2011, pp. 881-893.
26. J. Reps, J. M. Garibaldi, U. Aickelin, D. Soria, J. E. Gibson, and R. B. Hubbard, "Discovering sequential patterns in a UK general practice database," in *Proceedings of IEEE-EMBS International Conference on Biomedical and Health Informatics*, 2012, pp. 960-963.
27. M. Russinovich and B. Cogswell, *Process Monitor*, ed., 2012.
28. J. S. Bucy, J. Schindler, S. W. Schlosser, and G. R. Ganger, "The disksim simulation environment version 4.0 reference manual (cmu-pdl-08-101)," *Parallel Data Laboratory*, 2008, p. 26.
29. "Nandsim linux mtd package," http://www.linux-mtd.infradead.org/.
30. S. Lee, D. Shin, Y.-J. Kim, and J. Kim, "LAST: locality-aware sector translation for NAND flash memory-based storage systems," *SIGOPS Operating Systems Review*, Vol. 42, 2008, pp. 36-42.
31. E. L. Lawler, J. K. Lenstra, A. R. Kan, and D. B. Shmoys, *The Traveling Salesman Problem: a Guided Tour of Combinatorial Optimization*, Vol. 3, Wiley Chichester, 1985.
32. Samsung, "1g×8 bit/2g×8 bit/4g×8 bit NAND flash memory (K9K8G08U0A)," 2006.
33. J. Gim and Y. Won, "Relieving the burden of track switch in modern hard disk drives," *Multimedia Systems*, Vol. 17, 2011, pp. 219-235.
34. F. Chen, D. A. Koufaty, and X. Zhang, "Hystor: making the best use of solid state drives in high performance storage systems," in *Proceedings of International Conference on Supercomputing*, 2011, pp. 22-32.
35. S. Hong and D. Shin, "NAND flash-based disk cache using SLC/MLC combined flash memory," in *Proceedings of International Workshop on Storage Network Architecture and Parallel I/Os*, 2010, pp. 21-30.
36. I. Koltsidas and S. D. Viglas, "Flashing up the storage layer," in *Proceedings of VLDB Endowment*, Vol. 1, 2008, pp. 514-525.
37. G. Soundararajan, V. Prabhakaran, M. Balakrishnan, and T. Wobber, "Extending SSD lifetimes with disk-based write caches," in *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, 2010, pp. 8:1-8:14.
38. F. Chen, S. Jiang, and X. Zhang, "SmartSaver: Turning flash drive into a disk energy saver for mobile computers," in *Proceedings of International Symposium on Low Power Electronics and Design*, 2006, pp. 412-417.
39. H. J. Lee, K. H. Lee, and S. H. Noh, "Augmenting RAID with an SSD for energy relief," in *Proceedings of Conference on Power Aware Computing and Systems*, 2008, pp. 12:1-12:5.
40. T. Makatos, Y. Klonatos, M. Marazakis, M. D. Flouris, and A. Bilas, "Using transparent compression to improve SSD-based I/O caches," in *Proceedings of the 5th European Conference on Computer Systems*, 2010, pp. 1-14.
41. J.-W. Hsieh, T.-W. Kuo, P.-L. Wu, and Y.-C. Huang, "Energy-efficient and performance-enhanced disks using flash-memory cache," in *Proceedings of International Symposium on Low Power Electronics and Design*, 2007, pp. 334-339.
42. J. A. Garrison and A. L. N. Reddy, "Umbrella file system: Storage management across heterogeneous devices," *Transactions Storage*, Vol. 5, 2009, pp. 1-24.
43. W. K. Josephson, L. A. Bongo, K. Li, and D. Flynn, "DFS: A file system for virtu-

alized flash storage," *Transactions Storage*, Vol. 6, 2010, pp. 1-25.
44. T. Pritchett and M. Thottethodi, "SieveStore: a highly-selective, ensemble-level disk cache for cost-performance," in *Proceedings of the 37th Annual International Symposium on Computer Achitecture*, 2010, pp. 163-174.
45. H. Payer, M. Sanvido, Z. Z. Bandic, and C. M. Kirsch, "Combo drive: Optimizing cost and performance in a heterogeneous storage device," in *Proceedings of the 1st Workshop on Integrating Solid-state Memory into the Storage Hierarchy*, 2009, pp. 1-8.
46. Y. Joo, Y. Cho, K. Lee, and N. Chang, "Improving application launch times with hybrid disks," in *Proceedings of the 7th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis*, 2009, pp. 373-382.

Seongjin Lee is a Ph.D. student at Embedded Software Systems Laboratory at Division of Electrical and Computer Engineering, Hanyang University. He received his BS and MS degrees in Department of Electronics and Computer Engineering, Hanyang University, Seoul, Korea in 2006 and 2008, respectively. He is pursuing his Ph.D. in File System and Storage Analysis. His research interests include performance measurements, analysis, modeling, characterization and classification.

Youjip Won is a Professor at Division of Electrical and Computer Engineering, Hanyang University, Seoul, Korea. He is leading Embedded Software System Lab. He received his BS and MS degrees in Department of Computer Science, Seoul National University, Seoul, Korea in 1990 and 1992, respectively. He received his Ph.D. in Computer Science from University of Minnesota in 1997. Before joining Hanyang University in 1999, he worked at Intel Corp. as a Server Performance Analyst. His research interests include network traffic modeling, analysis and characterization, multimedia system and networking, file and storage subsystem, and lower power storage system. In 2006, Multimedia File System project funded by Samsung Electronics was awarded "Best Academy-Industry Collaboration Practice in Samsung Electronics". In 2007, he was awarded "National Research Lab" grant which is highly selective and prestigious governmental grant.

Sungwoo Hong received the B.S. and Ph.D. in Electronics and Computer Engineering from Hanyang University, Seoul, Korea, in 2005 and 2010, respectively. From 2010 to 2011, he was a Postdoctoral Researcher with the Department of Electronics and Computer Engineering, Hanyang University. He majored in networks, operating systems, and multimedia communications. His current research interests include multimedia networking, wireless resource control mechanisms, and network security.