Device-aware cache replacement algorithm for heterogeneous mobile storage devices

Young-Jin Kim and Jihong Kim

School of Computer Science & Engineering, Seoul National University,
San 56-1 Shillim-dong, Kwanak-gu, Seoul, KOREA, 151-742
{youngjk,jihong}@davinci.snu.ac.kr

Abstract. Hard disks, most prevalent mass-storage devices, have high power consumption and high response time for random I/O requests. Recent remarkable technology improvement of flash memory has made it a rising secondary storage device but flash memory still has high cost per bit. Usage of heterogeneous storage devices such as a pair of a hard disk and a flash memory can provide reasonable cost, relatively acceptable response time, and low-power consumption. In this paper, we propose a novel buffer cache replacement algorithm which targets a mobile computing system with a heterogeneous storage pair of a hard disk and a flash memory. The algorithm partitions the cache per each device and adjusts the size of each partition based on the performance indices of the devices, and manages each partition according to workload patterns. Simulations show that the proposed algorithm yields a hit rate up to two times higher than LRU on the typical mobile traces according to the cache size and achieves also better system I/O response time and energy consumption.

Key words: Heterogeneous storage, mobile systems, device-aware, workload-aware, cache replacement

1 Introduction

As the mobile and ubiquitous computing technology progresses, end-users tend to want that they can use high-performance and high I/O load applications such as games and MPEG players. In the last decade, the innovational development of processors, memories, network devices, and secondary storage devices has enabled this. These days mobile computing systems with high-capacity storage devices are popular, such as PDAs, PMPs, and MP3 players. Since hard disk drives are widely adopted for mobile computing platforms, the demand for hard disk drives with a small form-factor ($2.5''$ or less), embedded in or connected to such systems, is also incrementally rising [1]. Concurrently, due to recent remarkable technology improvement of flash memory, it appears a rising secondary storage device.

However, despite attractive low cost per bit, hard disks are big power consumers and have poor performance for random I/O requests. Flash memory still

has relatively high cost per bit. For example, NAND flash memory is said to be at least several times more expensive than disk drives with the same capacities [2]. Therefore, complementary storage architectures or techniques have been emerging. Several researchers have proposed combined storage techniques with a flash memory and a hard disk, which can be categorized into two: 1) using a flash memory as a non-volatile cache [3,4,5,6,7,8,9]; 2) using a flash memory as a secondary storage device [10]. Specially, [10] studied the potential of heterogeneous secondary storage by employing these two devices together with data concentration techniques. The heterogeneous storage solution in this work is expected to yield more energy saving in that it employs a flash memory as a secondary storage device directly and can maintain a larger set of energy-hungry blocks altogether on it compared with other work. However, the authors did not investigate performance improvement or load balancing problems deeply.

In case of using heterogeneous devices generally, file systems require caching algorithms that take into account the different miss penalties across file blocks depending on which devices they belong to. But, the most commonly used cache replacement algorithm, LRU is not aware of such different cost and treats all cache blocks as if they have the same replacement costs. In the web cache communities, there have been abundant research results on cost-aware cache replacement algorithms, which consider different file size, network latency during re-fetch due to a cache miss, file access frequency, etc. Recent web cache algorithms may be based on or enhance the *GreedyDual-Size* algorithm [11], which incorporates locality with miss penalty and file size concerns, generalizing the LRU algorithm. In disk-based storage systems, [12] studied storage-aware cache management algorithms using different costs on heterogeneous disks. This work maintained one partition per each disk and adjusted partition sizes based on the time spent per each disk (they call this *wait time*) over a period of device requests, and controlled the blocks within each partition similarly to the GreedyDual-Size algorithm. But, the authors did not take into account minutely the case of there being a number of sequential accesses, which may be problematic in their algorithms. This is because if a considerable number of sequential accesses are requested to a disk its wait time can be lengthened and the corresponding partition size will increase filling this partition with less valuable blocks. Consequently, in the worst case this algorithm may fail in obtaining good load balance.

In this paper, we build a novel cache replacement algorithm to overcome such limit, which targets mobile storage systems exploiting a pair of a hard disk and a flash memory as secondary storage. Our algorithm intends to enhance the system performance through both device-aware and workload-aware load balancing. For the former we use cache miss counts and access time per device and for the latter we have our algorithm manage the cache in the direction of exploiting the fast sequential performance feature of a hard disk. To the best of our knowledge, our work is the first attempt to design and incorporate a cost-aware cache management algorithm on the heterogeneous combination of a hard disk and a flash memory.

Our first goal is to investigate how our device-aware cache replacement algorithm can balance the I/O load between two heterogeneous devices on typical mobile workloads when the target system employs a hard disk and a flash memory as mass storage, compared with LRU. Second goal is to study how well our cache algorithm avoids cache pollution incurred by sequential block requests while balancing the I/O load.

We first tackle the design of a workload-aware cache replacement algorithm (in short, WAC) by introducing different cost per workload pattern similarly to the GreedyDual-Size algorithm. Then, we propose our re-partitioning policy on the total cache based on the cache miss counts at a fixed period and finally complete to embody our device-aware cache replacement algorithm (in short, DAC) combining these.

The rest of this paper is organized as follows. In Section 2, we review the features of a hard disk and a NAND flash memory to compose heterogeneous storage on mobile platforms and describe requirements for designing a device-aware cache replacement algorithm briefly. In Section 3, we describe our both workload-aware and device-aware algorithms in detail. Section 4 presents our simulation framework and simulation results. Related work is given in Section 5. Finally, we conclude in Section 6.

2 Motivation

2.1 Device Bandwidth and Sequentiality

Since our research targets heterogeneous storage systems with the configuration of a hard disk and a flash memory, it is necessary to examine the features of a hard disk and a flash memory. For this purpose, we simply take two typical devices as shown in Table 1, which are appropriate for mobile storage. Fujitsu MHT060BH has a 2.5″ form factor, a 60 GB capacity, and 5,400 RPM while Samsung K9K1208U is a NAND flash memory and has a 64 MB capacity, a block size of 16 KB, and a page size of 512 B. The throughputs of the flash memory were from [13] and those of the hard disk were obtained on our Sony VAIO laptop computer which embeds this disk using DiskSpd [14], which can measure disk I/O performance with various configurations including whether I/Os are sequential or random on Windows XP.

Table 1. Throughputs of a laptop disk and a NAND flash memory

Device			Hard disk	Flash memory
			MHT2060BH	K9K1208U
Throughput (MB/s)	Sequential	Read	30.47	14.3
		Write	30.47	1.78
	Random	Read	6.6	14.3
		Write	6.6	1.78

In Table 1, the disk shows a pretty good throughput for sequential I/Os and the value is about 5 times larger than that for random I/Os irrespective of the I/O type. In contrast to the disk, the flash memory doesn't concern sequentiality of I/Os and exhibits poor performance for write I/Os compared with reads. Therefore, when we design and use a heterogeneous storage system with such devices we surely need to meet performance imbalance which is likely to occur due to distinctly separable characteristics of these devices. This is because realistic workloads on mobile platforms often exhibit mixed patterns of sequential and random I/O operations like the case of concurrent execution of MP3 playing and program compiling. In addition, the conventional operating systems might not be designed well for I/O sequentiality coupled with this new and unfamiliar configuration of heterogeneous devices. For example, it seems that adequate management of sequentiality and I/O type for block requests across these heterogeneous devices in the viewpoint of performance may beyond the capability of the LRU algorithm as previously remarked.

2.2 Mobile Workloads and Sequentiality

Recent studies on mobile storage systems collected and utilized traces on applications typically used in mobile computing environments under feasible execution scenarios [5,10]. Among these, [10] gathered traces while executing real applications which can be used for a PDA immediately on an evaluation board similar to a PDA. The used execution scenario was repetition of *file transfer, email, file search, and sleep* (no operation). We examined the behavior of this mobile workload (hereafter, we will call *PDA trace*).

Since *file transfer* gives rise to disk reads (or writes) when files are sent to (or from) a network, the access pattern will be shown to be long sequential. The other applications except *sleep* are likely to exhibit random accesses (in this paper, a random access type means non-sequential one). Figure 1 shows the access pattern of the PDA trace, where x axis is *virtual time* (i.e., index of arriving requests) and y axis is *logical block address*. We notice that there are mixed accesses of a large number of sequential accesses, big and small loop-type accesses, and a small amount of temporal accesses. Similar access patterns can be found in the plots of traces gathered under programming and networking scenarios for mobile computing platforms in [5], though there is a different level of sequentiality compared with the PDA trace. Such observations drive us to need to deal with frequent sequential I/O operations together with random I/Os because if they weren't coped with adequately at the operating system software level there might occur critical performance degradation of the overall system.

3 Device-Aware Cache Replacement Algorithm

3.1 Workload-Aware Cache Algorithm

As was described in the previous section, it is requisite to deal with mixed access patterns which may frequently occur on generic mobile computing platforms

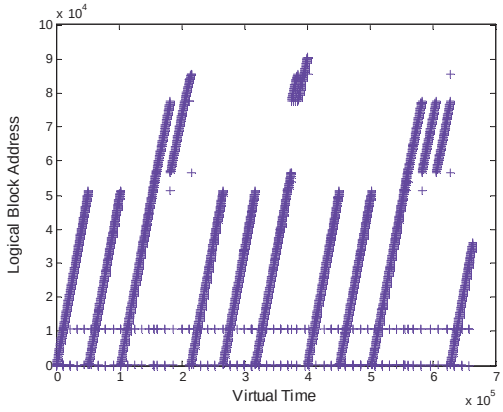


Fig. 1. Plot of the logical block address of each arriving IO request to its virtual time for the mobile trace used in [10]

as well as in our heterogeneous storage system with a hard disk and a flash memory. As a first step towards a solution, we first tackle the design of WAC, our workload-aware cache replacement algorithm using different cost per block according to workload patterns similarly to the GreedyDual-Size algorithm.

Figure 2 describes the overall algorithm of *WAC*. WAC combines different replacement cost and locality based on LRU. At the reference of a block x, when a cache miss occurs and it should be fetched to the cache WAC sets L to x's H. If there is no free block and block eviction is needed the cache block with the lowest H value is evicted and L is reset to this H value. If a cache hit occurs in the cache x's H is restored to L. How WAC updates the H values is shown in the subroutine *H_update*.

```
VARIABLE INITIALIZATION
L ← 0
C_SEQ ← α, C_RAND ← β (0<α < β)

MAIN ALGORITHM
At the reference of block x,
If x is hit in the cache
    H_update(x.H)
else
    If there is no free block in the cache
        L ← min{G| p∈T, G ← p.H}
        Evict p such that G = L
    Fetch x to the cache
    H_update(x.H)
```

```
SUBROUTINE H_update(x)
If x.pattern = SEQ
    x.H ← L + C_SEQ
else
    x.H ← L + C_RAND
```

Fig. 2. Proposed workload-aware cache replacement algorithm (WAC). In WAC, sequenitial I/O blocks have the most chances to stay in the cache, and random I/O blocks vice versa.

In designing the WAC algorithm, we tried to reflect the need that frequent and a large amount of sequential I/O requests which can be found in typical mobile workloads should be considered. Though there may be various ways in determining cost of each block in the cache while realizing this need, we simply divided the attribute, which each block can have in the cache, into 2: sequential and random. In the algorithm, the attribute is concreted by adding *C_SEQ* or *C_RAND* to the L value when the accessed block does take on sequentiality or not. Since when a block is sequential evicting it is more beneficial, we assign C_SEQ to a small positive value (in an actual implementation, we used 1). We expect more cache hits by keeping random blocks longer than those with sequentiality and thus C_RAND is assigned to a larger value than C_SEQ. Therefore, H values of cache blocks will be maintained relatively large if they are accessed recently or randomly and there will be more chances for such blocks to remain in the cache rather than blocks with little locality or sequentiality. This can be thought of a generalized version of LRU. In this paper, since we want to weight sequentiality for cache block replacement rather than I/O type, we do not consider more separated attributes like sequential and read accesses, random and write accesses, etc. Schemes using such more complex attributes will remain future work.

3.2 Evaluation: WAC and LRU

We evaluated performances of WAC and LRU in terms of cache hit rate using the PDA trace. For this, we built a trace-based cache simulator which implements WAC and LRU, and concatenated it and the simulator in [10] (Refer to subsection 4.1).

Figure 3 shows the hit rates of WAC and LRU, which were simulated for the PDA trace when the cache size varied from 5 to 60 MB with an incremental step of 5 MB except 55 MB (since the hit rate is already saturated around this size, we omitted it). We can notice that WAC outperformed LRU for all the cache sizes. This results apparently reflects the fact that WAC better maintains valuable (that is, causing more cache hits) blocks in the cache, which were not if they had been evicted early, and efficiently evicts less valuable blocks quickly, compared with LRU. Since we ascertain our workload-ware cache algorithm shows effectiveness for the mixed I/O request pattern of mobile workloads, our next task is to augment WAC such that it can be effective under mobile workloads in heterogeneous storage systems rather than single-device based storage systems (in this evaluation, a single disk was used).

3.3 Device-Aware Cache Replacement Algorithm

Our device-aware cache replacement algorithm (i.e., DAC) is mainly composed of 1) adjusting the sizes of partitions for a hard disk and a flash memory dynamically based on the performance index per device; 2) managing each partition according to the pattern of workloads by applying the WAC policy.

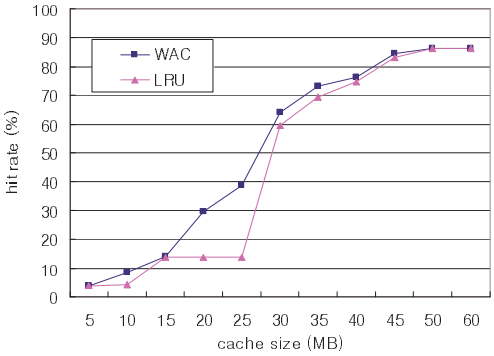


Fig. 3. Hit rates of WAC and LRU for the PDA trace when the cache size varies.

In designing the DAC algorithm, we took required cache management rules as follows: 1) the size of each partition should be adjusted so that the overall load is balanced considering cache miss counts as well as different miss penalties between a hard disk and a flash memory. 2) sequential I/O requests should be managed to be evicted earlier by applying WAC to cache blocks within each partition.

Detailed DAC's algorithm is shown in Figure 4. DAC has two LRU lists *T1* and *T2*, each of which represents a partition assigned to cache blocks for either of heterogeneous devices (in this paper, T1 is used for a hard disk and T2 for a flash memory). *T1_req* (*T2_req*) is the target size of the T1 (T2) partition and c is the total cache size. *PR* is the access time ratio between a hard disk and a flash memory. W represents a partition-adjusting period, which is compared with cumulated reference counts (*cumulated_ref_count*) for re-partitioning.

DAC largely consists of 1) a cache replacement algorithm executed at every block reference; 2) a cache partitioning algorithm executed at W block references. At each block access, DAC determines the access pattern and whether the block is missed or hit. If a hit occurs in T1 (or T2), the block is moved to the MRU position of T1 (or T2). If there occurs a miss, the subroutine *List_update* is invoked with *miss_flag* which indicates in which device the missed block resides. Briefly, the mechanism of List_update is to control T1 and T2 so that their sizes follow T1_req and T2_req well, respectively. This is quite important process because good control of each partition size is based on the harmony of adjustment issuing and its follow-up. During such operations, we employ the subroutine H_update, which was seen in WAC, for the purpose of setting values to the cache blocks within each partition. In a period of W block references, the algorithm re-partitions the cache depending on miss counts of random-type blocks multiplied by PR. Multiplying PR is needed because two devices have different miss penalties (i.e., access times).

The strong points of the DAC algorithm come from its being device-aware as well as workload-aware: it adjusts the partition sizes based on performance

```
VARIABLE INITIALIZATION
x : referenced block
c: cache size
W: partition-adjusting period
PR: performance ratio
|T1| ← 0, |T2| ← 0, T1_req ←c/2, T2_req ← c/2
T1.rand_misses ← 0, T2.rand_misses ← 0
T1.L ← 0, T2.L ← 0
miss_flag ← 0
cumulated_ref_count ← 0

MAIN ALGORITHM
At every reference of blocks,
Find the access pattern:
    x.pattern ← SEQ or RAND
Increase cumulated_ref_count
If a cache hit occurs in T1
    Move x to the MRU in T1
else if a cache hit occurs in T2
    Move x to the MRU in T2
else
    If x resides in the disk
        miss_flag ← 1 (a cache miss in T1 is set)
    else
        miss_flag ← 2 (a cache miss in T2 is set)
    List_update (miss_flag, T1, T2)

At a period of W references,
    If cumulated_ref_count = W
        If T1.rand_misses* PR > T2.rand_misses
            delta ← T1.rand_misses – T2.rand_misses
            If T1_req + delta > c
                T1_req ← c
            else
                T1_req ← T1_req + delta
                T2_req ← c – T1_req
        else
            delta ← T2.rand_misses – T1.rand_misses
            If T2_req + delta > c
                T2_req ← c
            else
                T2_req ← T2_req + delta
                T1_req ← c – T2_req
```

```
SUBROUTINE List_update ( miss_flag, T1, T2 )
If miss_flag = 1
    If |T1| > |T1_req| and |T1| is not zero
        T1.L ← min{y.H| y ∈ T1}
        Evict y which satisfies y.H = in T1.L
        Decrease |T1|
    Fetch x to the cache and move it to the MRU
    position in T1
    H_update(x, T1)
    Increase |T1|
    If |T1|+|T2|>= c and |T2| >= T2_req
    and |T2| is not 0
        T2.L ← min{y.H| y ∈ T2}
        Evict y which satisfies y.H = in T2.L
        Decrease |T2|
    Increase T1.rand_miss_times
else
    If |T2| > T2_req and |T2| is not zero
        T1.L ← min{y.H| y ∈ T1}
        Evict y which satisfies y.H = in T2.L
        Decrease |T2|
    Fetch x to the cache and move it to the MRU
    position in T2
    H_update(x, T2)
    Increase |T2|
    If |T1|+|T2|>= c and |T1| >= T1_req
    and |T1| is not 0
        T1.L ← min{y.H| y ∈ T1}
        Evict y which satisfies y.H = in T1.L
        Decrease |T1|
    Increase T2.rand_miss_times
miss_flag ← 0

SUBROUTINE H_update ( x, Ti ), i= 1 or 2
If x.pattern = SEQ
    x.H ← Ti.L + C_SEQ
else
    x.H ← Ti.L + C_RAND
```

Fig. 4. Proposed device-aware cache replacement algorithm (DAC).

skewness and manages cache blocks according to worth in the aspect of performance by taking into account the access pattern. Thus, we expect that it may improve the system performance better by dealing with performance imbalance efficiently in heterogeneous storage systems, compared with LRU. We also expect that DAC may be more helpful in enhancing the performance by evicting sequential blocks early.

There are several challenges in the DAC algorithm. First, we found that the value of PR can vary according to the degree of temporal locality through experiments in the viewpoint of the overall system performance. Therefore, in the experiments we simply (not optimally) changed the value of PR statically in order to obtain a better performance. Second, when we calculate *delta* we also found that it was sometimes more beneficial to weight the larger value of two random miss counts of T1 and T2 depending on the degree of temporal local-

ity. We simulated while varying this value statically. Finally, the period of W affected the overall performance and needs to vary depending on workload patterns. However, building a fully automatically-tunable DAC to maintain optimal parameters is a problem rather beyond the scope of this paper and will remain future work.

4 Simulation and Results

4.1 Simulation Environment

We developed a trace-based cache simulator which incorporates LRU, WAC, and finally DAC. In order to link cache simulation with the operation of a heterogeneous storage system, we augmented the multi-device power and performance simulator in [10]. The hard disk model we used is the MK4004GAH with a $1.8''$ form factor and 4,200 RPM [10] and the flash model is the K9K1208U shown in Table 1.

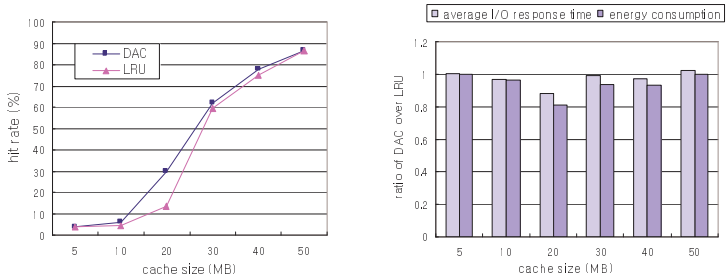
We also built a synthetic trace generator, which can generate three types of traces by controlling sequentiality and temporal locality: SEQUENTIAL, TEMPORAL, and COMPOUND. Our synthetic trace generator can also control various parameters such as control request rate, read/write ratio, file size, and request size. We ran our trace generator, varying default parameters. Default parameter setting is as follows: average interval time between I/O requests = 70 (ms), trace time = 80 (min), maximum file size = 5 (MB), total file size = 350 (MB), and write ratio = 0.5. Default I/O access pattern is set to COMPOUND (i.e., mixed of sequentiality and temporal locality).

For simulation, we used the PDA trace and two synthetic traces (we call *trace1* and *trace2*): trace1 uses the default parameters and trace2 also does except that the average interval time and the maximum file size are set to 20 ms and 1 MB, respectively. The PDC trace, trace1, and trace2 have working set sizes of 44, 23, and 57 MB and trace file sizes of 30, 2.8, and 9.6 MB, respectively. We evaluated the cache hit rate and average system I/O response time for DAC and LRU as metrics. We assumed that the overheads of re-partitioning and handling blocks per partition in DAC are acceptable in comparison with LRU and set W to 200 and PR to 35.

4.2 Simulation Results

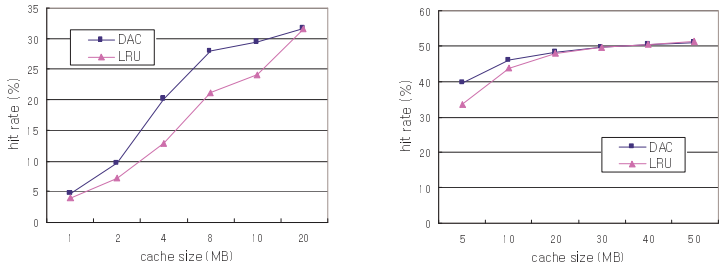
In Figure 5, plots (a) and (b) show the hit rates of DAC and LRU and average system I/O response times and energy consumptions of DAC normalized over LRU, respectively, for the PDA trace with the cache size varied. In the plot (a), DAC has higher hit rates than LRU in all cases. The higher hit rates of DAC affected the average I/O response times and these values of DAC appeared smaller than those of LRU overall, as shown in the plot (b).

Two exceptions are when the cache sizes are 5 and 50 MB. We found that though there occurred the same device accesses for DAC and LRU the degree of



(a) (b)

Fig. 5. Simulation results of DAC and LRU for the PDA trace when the cache size varies: (a) Hit rates of DAC and LRU (b) Average system I/O response times and energy consumptions of DAC, which are normalized over LRU.



(a) (b)

Fig. 6. Simulation results of DAC and LRU for synthetic traces when the cache size varies: (a) Hit rates of DAC and LRU for the trace1 (b) Hit rates of DAC and LRU for the trace2.

clustering in the device (exactly, disk) queue was lower for DAC, that is, a little more non-sequential accesses occurred and they caused more seek time. This phenomenon rather seems to be related with adjustment of parameters described in subsection 3.3 for the case of two extremes of the cache sizes. We also notice that there were more chances for power-down in the devices. Consequently, the energy consumption of DAC was observed to be smaller than that of LRU almost always.

Figure 6 shows the hit rates of DAC and LRU for two synthetic traces with the cache size varied: (a) for the trace1 and (b) for the trace2. We notice that DAC showed almost equal or better results in hit rates for both traces. Comparing the hit rates in the plots (a) and (b) depending on working set sizes and varying cache sizes, we can notice that the trace2 has more temporal I/O accesses. This means that DAC might be effective regardless of the amount of sequentiality. To examine this, we evaluated two more synthetic traces with temporal access patterns, which have the same parameters of trace1 and trace2 except that the I/O access pattern is set to TEMPORAL (we call these traces *trace1_temp* and

trace2_temp). For the trace1_temp, we found that the hit rates of DAC and LRU with a 4 MB cache were 53.0% and 54.1% (actually, with different setting of W and PR, we could obtain the almost same hit rate). For the trace2_temp, the hit rates were 97.1% for both DAC and LRU with a 10 MB cache. We omitted the average system I/O response time and energy consumption due to the space limit, but we found that DAC has better performance in these two metrics than LRU similarly to the results of the PDA trace.

5 Related Work

[3,4,5,9] have all proposed using flash memory as a non-volatile cache, maintaining blocks which are likely to be accessed in the near future, and thus allowing a hard disk to spin down for longer time. [4] focused on the use of a flash memory as a write buffer cache, while [5] has recently studied a technique of partitioning a flash memory into a cache, a prefetch buffer, and a write buffer to save energy. [9] mainly considered reducing the power consumption of a main memory by using a flash memory as a second-level buffer cache. Hybrid HDD solution co-developed by Samsung and MS uses a flash memory as an on-board non-volatile cache in addition to a hard disk, which aims at performance boosting, low power, and high reliability on mobile computers [6,7].

Our work is distinct from the above research in that it studies performance improvement in a heterogeneous storage system which uses a flash memory together with a hard disk as secondary storage. Our approach suggests an effective buffer cache management algorithm aiming at performance improvement, depending on both device-awareness and workload-awareness.

6 Conclusions

We have proposed a novel buffer cache replacement algorithm which targets a mobile computing system with a heterogeneous storage pair of a hard disk and a flash memory. The proposed algorithm partitions the cache per each device and adjusts the size of each partition based on the performance indices of the devices, and manages each partition according to workload patterns.

Trace-based simulations showed that the proposed technique can lead to up to a two times higher hit rate than LRU according to the cache size with a pair of a 1.8″ hard disk and a NAND flash memory on realistic mobile traces. In addition, our algorithm reduced the average system I/O response time and energy consumption by up to 12% and 19%, respectively, compared with LRU.

As future work, we plan to study software techniques including cache algorithms in order to mitigate the write/erase cycles of a flash memory while maintaining the performance. We also plan to research the performance and energy consumption using DAC under various data layouts.

Acknowledgments. This research was supported by the MIC (Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Assessment). And the ICT at Seoul National University provided research facilities for this study.

References

1. L. D. Paulson. Will hard drives finally stop shrinking? *IEEE Computer*, Vol. 38., No. 5. pp.14–16, 2005.
2. G. Lawton. Improved flash memory grows in popularity. *IEEE Computer*, Vol. 39., No. 1. pp.16–18, 2006.
3. B. Marsh, F. Douglis, and P. Krishnan. Flash memory file caching for mobile computers. in *Proc. of the 27th Hawaii International Conference on System Sciences*, Hawaii, USA, pp.451–460, Jan. 1994.
4. T. Bisson and S. Brandt. Reducing energy consumption with a non-volatile storage cache. in *Proc. of International Workshop on Software Support for Portable Storage (IWSSPS)*, held in conjunction with *the IEEE Real-Time and Embedded Systems and Applications Symposium (RTAS 2005)*, San Francisco, California, March, 2005.
5. F. Chen, S. Jiang, and X. Zhang. SmartSaver: turning flash drive into a disk energy saver for mobile computers. in *Proc. of the 11th ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED'06)*, Tegernsee, Germany, October 4-6, 2006.
6. Microsoft, ReadyDrive and Hybrid Disk. `http://www.microsoft.com/whdc/device/storage/hybrid.mspx`.
7. `http://www.samsung.com/Products/HardDiskDrive/news/HardDiskDrive_20050425_0000117556.htm`.
8. R. Panabaker. Hybrid Hard Disk & ReadyDrive™ Technology: Improving Performance and Power for Windows Vista Mobile PCs. in *Proc. of Microsoft WinHEC 2006*, June 2003. `http://www.microsoft.com/whdc/winhec/pres06.mspx`.
9. T. Kgil and T. Mudge. FlashCache: A NAND flash memory file cache for low power web servers. in *Proc. of 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES '06)*, Seoul, Korea, October 22-25 2006.
10. Y.-J. Kim, K.-T. Kwon, and J. Kim. Energy-efficient file placement techniques for heterogeneous mobile storage systems. in *Proc. of the 6th ACM & IEEE Conference on Embedded Software (EMSOFT)*, Seoul, Korea, October 22-25 2006.
11. P. Cao and S. Irani. Cost-aware WWW proxy caching algorithms. in *Proc. of USENIX Symposium on Internet Technology and Systems*, December, 1997.
12. B. Forney, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Storage-aware caching: revisiting caching for heterogeneous storage systems. in *Proc. of the 1st. USENIX Conference on File and Storage Technologies (FAST)*, Jan. 2002.
13. H. G. Lee and N. Chang. Low-energy heterogeneous non-volatile memory systems for mobile systems. *Journal of Low Power Electronics*, Vol. 1, Number 1, pp. 52–62, April, 2005.
14. `http://research.microsoft.com/BARC/Sequential_IO/`.