Energy Efficient Management Scheme for Heterogeneous Secondary Storage System in Mobile Computers

Shu Liu Xu Cheng Xuetao Guan Dong Tong
Microprocessor Research and Development Center of Peking University
Room 1818, Science Building 1, School of Electronics Engineering and Computer Science
Beijing, P.R. China, 100871
{liushu, chengxu, guanxuetao, tongdong}@mprc.pku.edu.cn

ABSTRACT

Flash memory is widely used because of its shock-resistance and power-efficient features. However, it cannot replace hard disks as secondary storage devices due to their greater cost per unit storage and low capability. In this paper, we propose an energy efficient heterogeneous secondary storage system management scheme for mobile systems. We employ flash memory device as a file cache of hard disk and extend existing data cache management algorithms to distribute files between two devices with consideration of file level cache restrictions. As a result, most file accesses are conducted in flash memory device and disk is spun-down to save energy. We develop a trace-driven simulator to evaluate our scheme in comparison with other alternatives. Results demonstrate that with the help of our scheme, energy consumption of secondary storage system can be saved by up to 90% and I/O access time is improved. Furthermore, the file cache management algorithms can result in high hit ratios.

Categories and Subject Descriptors

D.4.2 [**Operating Systems**]: Storage Management – *secondary storage.*

General Terms

Design, Management, Measurement, Experimentation

Keywords

Energy saving, flash memory, hard disk, heterogeneous secondary storage.

1. INTRODUCTION

Power conservation is a critical issue in mobile storage system design. The hard disk has been the dominant form of secondary storage device for decades. However, its mechanical components induce high read/write latency, reliability problem and make it one of the major energy consumers of battery power. Though a large main memory is helpful to reduce disk accesses, DRAM also contributes to a large portion of overall system power.

Fortunately, non-volatile, shock resistant and power-economical flash memory has been introduced. With recent technology improvements in both capacity and reliability, flash memory storage systems are much more usable than ever. Flash memory devices such as NAND flash memory and flash SSD are widely used as storage devices[1]. Mobile computers equipped with flash SSD have hit the market, such as ASUS EeePC. Table 1 shows the latency, throughput and power parameters of three devices from Samsung in the same form-factor[2]. We notice that flash SSD exhibits the lowest power consumption and latency. However, flash memory device is not an ideal replacement for hard disks yet due to its low capacity and high expense. In September 2008, Intel launched a range of 1.8" and 2.5" SATA flash SSDs with 80GB capacity priced at around $595. The larger its capacity is, the more expensive it is. The rising popularity of audio and video applications makes the storage demands of mobile computers are eclipsing the capabilities that flash memory can provide at low cost. So, it is not suitable for these systems to use large flash SSD alone as secondary storage device. Though technology improvements will make this solution increasingly feasible, combining flash memory and hard disk is more practical nowadays.

The trade-offs associated with hard disks and flash memory devices, either in the form of price, capacity, or performance, motivate lots of storage system designs. Many researchers have proposed the use of flash memory as a non-volatile cache to prolong the disk spin down time[3-8]. However, these schemes treat flash memory as complement of DRAM buffer cache, and only a subset of data blocks are cached in flash memory, then the disk is used quite frequently due to cache misses or flushing. As flash memory's capacity increases, a heterogeneous secondary storage solution is expected to be more effective[9] and techniques such as PB-PDC are proposed. Different from data block level cache, flash memory stores files and can be accessed independently in heterogeneous secondary storage system. Previous research has shown that about 5% of the total files in disk are accessed over a 24 hour period[10] and it is possible to accurately predict which files are to be accessed during a work session[11, 12]. If files are accessed from flash memory, spinning down the disk as long as possible can reduce the system energy consumption greatly. Furthermore, the capacity and cost problems are solved. This inspires our research. Though file caching has been studied in Coda[11]and hoarding systems[12] to ensure data availability during network disconnect, we examine the energy and performance effect of file cache management scheme on heterogeneous secondary system consisting of flash memory and hard disk in this paper.

In our proposed system, the flash memory device is a disk file cache which caches files which will be accessed in near future. We adapt traditional data cache management algorithms to the file level by taking file cache restrictions into account and evaluate their effects. With the help of our management scheme, files are accessed from the cache device and the hard disk is spun-down. The hard disk is accessed in the event of file cache miss. The system monitors file accesses and decides which file should be cached in the flash memory device dynamically. When the cache is full, victim files are written back to hard disk. From the user's perspective, the capacity of secondary storage is equal to the capacity of hard disk. By adding a cheap and relatively small flash memory device, the secondary storage system's energy consumption is as much as that of flash memory device. We develop a trace-driven simulator to compare our scheme with employing larger DRAM and PB-PDC. Evaluations show that our policy can save secondary storage system's energy most efficiently. In addition, extended data cache management algorithms work well for our file-level cache.

Table 1 Characteristics of hard disk and flash SSD

Device(2.5")		**PC disk**	**notebook disk**	**Flash SSD**
latency (ms)	seek	12.0	14.0	-
	read	5.6	8.3	0.2
	write	5.6	8.3	0.2
Throughput (MB/s)	Read	107	80	90
	write	107	80	25~70
power (W)	seek	2.6	1.3	-
	active	2.4	1.8	0.5/0.6*
	idle	0.7	0.4	0.2
	standby	0.25	0.12	0.2

*read/write power

The rest of this paper is organized as follows. Section 2 provides a brief overview of related work. Section 3 describes heterogeneous secondary storage system organization and our management scheme in details. Section 4 presents the experimental environment and results. Finally, conclusions and future work are in section 5.

2. RELATE WORK

This section describes researches concerning the combination of flash memory and hard disk, and power saving mechanisms in disk storage system.

The different features of flash memory and hard disk have motivated lots of researches. A typical method is using flash memory as non-volatile secondary buffer cache which maintains blocks to be accessed in the near future[3-8]. March et al.[3] place the flash memory between DRAM and disk. All read and write requests to disk are cached in the flash memory. Bission et al.[4,5] focus on the redirection of write requests to a flash memory device when the hard disk is spun-down. Chen et al.[6] partition the flash memory into a read cache, a prefetch buffer, and a write buffer to save energy. The Hybrid Hard Disk Drive(H-HDD) [7] and Robson technology[8] adopts flash memory as a cache to improve performance, power and reliability. All these solutions cache data at block granularity and treat the flash memory device as a second level buffer cache. In this paper, flash memory device is a sub-component of secondary storage, which is operated on file level and can be used as an independent storage device.

Employing flash memory as a secondary storage device is not a novel idea. Flash memory has been widely used in embedded devices and heterogeneous secondary storage mechanism consisting of flash memory and hard disk has been proposed. Kim et al.[9] propose energy-efficient file placement technique PB-PDC. Frequently-read data are migrated to the flash memory device and frequently-written data are migrated to hard disk. As file access patterns changed, file migration between two devices could induce many disk accesses. The idea of PB-PDC is similar to data migration method in multi-disk system[13]. In contrast, we employ cache device organization like MAID where a subset of disks are treated in the storage system as cache disks to absorb I/O traffic[14]. Though our storage system organization is similar to disk/tape system, we aim not only to increase storage capacity, but also to provide an energy efficient storage system at low cost, since the hard disk system has nearly 26 times higher energy cost than tape system.

There are many researches on saving energy for a single disk in mobile system[15][16]. Some focus on the selection of timeout threshold used to switch the disk from active state to lower power modes and others customize systems or applications to reorganize idle periods in I/O request sequences by delaying or prefetching requests for saving disk energy. In this paper, we try to save energy at the architecture level by taking advantage of the low power feature of flash memory. All previous algorithms can be used to manage the hard disk in our storage system. So, our work is complementary to above algorithms.

3. HETEROGENEOUS SECONDARY STORAGE MANAGEMENT

In this section, we describe the organization of heterogeneous secondary storage system, and algorithms for managing flash memory device cache.

3.1 Basic Concept

As shown in Figure 1, the heterogeneous secondary storage system consists of a flash memory device and a disk. In this storage architecture, DRAM is block level buffer cache whereas the flash memory device and the hard disk are file level storage devices. From an organizational standpoint, the first decision is how we should distribute files between two devices. Similar to multi-disk storage system management, two methods can be used: migration and cache. For simplicity, some of the symbols used in our analysis are shown in Table 2. We define these two methods as follows.

Definition 1 Migration.heterogeneous storage(MHS). If a file x is contained in set F, it is not be contained in set D, and vice versa.

Properties:

(1) $F \cap D = \emptyset$; (2) The capacity of storage system is equal to $C_f + C_d$.

Definition 2 Cache heterogeneous storage(CHS). If a file x is contained in set F, it is contained in set D.

Properties:

(1) F $\subseteq$ D; (2) The capacity of storage system is equal to C_d.

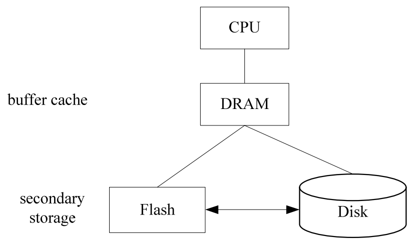

Figure 1 The heterogeneous secondary storage system

If MHS is used, the intent would be moving some files to the flash memory device; those files would not be duplicated on the hard disk. The management scheme PB-PDC is an example of MHS, where files are moved between two devices according to their read/write frequency. In contrast, files in flash memory are only a copy of that on disk in CHS. Though migration provides more usable storage, since no duplications exist, the disk will be accessed frequently because of regular file accesses which are placed in the disk or file migration from flash memory to disk.

Table 2 Symbol definition

Symbol	Parameter
C_f / C_d	Capacity of flash memory device/hard disk
E_f / E_d	Energy consumption of flash memory device/hard disk
F	The set of files that are stored in flash memory device
D	The set of files that are stored in hard disk
f	A set of functions or algorithms used to decided file distribution between flash memory and hard disk

We argue that CHS is more efficient than MHS. Let's consider two storage systems organized in these two methods respectively, where files {a, b, c, d, e, f, g} are stored. If function *f* decides that files a, e, and f should be stored in flash memory, these files are moved to flash. Operation processes of two methods are almost the same, except that files a, e, and f should be deleted from hard disk in MHS. According to above definition, F={a, e, f} and D={b, c, d, g} in MHS, while F={a, e, f} and D={a, b, c, d, e, f, g} in CHS. Then a series of read/write operations are conducted in each storage system. When files in flash memory have to be replaced, non-dirty files must be written back to hard disk in MHS whereas they are only discarded in CHS. CHS can avoid some unnecessary disk accesses to save energy. Both PB-PDC and our scheme are evaluated in the experimental section, which proves our deduction.

In the above description, we assume that files are already stored in hard disk. What happens if new files are created? In MHS, they can be placed in any device and properties of MHS are also maintained. According to the definition of CHS, new files should be placed in the hard disk. If they are in the flash cache, a copy should be stored in the hard disk. To reduce accesses of hard disk further, CHS can be extended to CHS-U. The management of CHS-U involves additional mechanism to synchronize files between two devices. For example, when a file is created, the operating system has to access the hard disk to make sure that this file doesn't exist. The efficiency of CHS-U is hard to evaluate. We focus on the basic CHS method, which can demonstrate the feasibility of flash file cache scheme and be extended to CHS-U.

Definition 3 Cache heterogeneous storage with union (CHS-U). F is the union of a subset of D and newly created file set.

Then we investigate management scheme details based on CHS. We explore the locality of file accesses and place copies of files on flash memory device. File system monitors file access history and caches hot files in the flash memory device. If request data are not in DRAM buffer cache, secondary storage devices are accessed. In most cases, files are accessed from flash cache and hard disk is spun-down. Then the hard disk becomes more lightly loaded, allowing for energy saving. In the event of a flash memory file cache miss, the disk is spun-up and files are accessed from it. If it is a hot file, we fetch it to flash memory. When flash memory device cache is full, some files are replaced and written back to disk.

The energy consumption of secondary storage system is the sum of E_f and E_d. And E_f/E_d is the product of the power need to access files from that device, which is related to the hit ratio in the cache device. The more files accessed from flash memory are, the more energy efficient the secondary storage system is. The device access behavior is affected by several important algorithms and parameters. The first one is the capacity of the flash memory device C_f. Large capacity allows more files to be placed in the cache to increase the hit probability. Since flash memory capacity is sacrificed to get low power for the secondary system, we must make trade-offs in price, performance, and power. The second is cache and replacement algorithms. These algorithms are critical to explore file access locality. In a word, the more files hit in the flash memory device cache, the longer time the disk can be powered off, thereby reducing power consumption.

3.2 Cache Management Algorithms

Over the years, cache management algorithms have been most widely studied in the context of CPU caches and main memory buffer cache. However, there are some special features in our design. First, existing cache management algorithms are at the unit of data block. In contrast, we operate at the file level. Second, traditional caches fetch data on a miss or when prefetching. Files on disk should not be brought into the flash memory cache once it is missed. Instead, only hot ones should be cached to reduce file copy overhead. Third, there are fixed cache entries in data caches, and replacement happens when there's no empty entry. In our scheme, file sizes are not determined. So, the number of files that can be cached is not fixed. The free space of flash memory is the only criteria to judge whether replacement is needed.

3.2.1 Cached file selection

The cached file selection algorithm decides files to be cached in flash. Usually, both static and dynamic types of selections can be used. The static approach is more suitable for files which users almost always need access, for example, the operating system file, compiler and some C libraries. The method for specifying such "permanently cached files" can be implemented by preloading a list of names kept in a special file. The cached file selection algorithm we describe following is a dynamic method.

It is impossible to accurately predict the possibility of file accesses. In data caches, the historical frequency of access plays a role that rivals that of temporal locality. File accesses exhibit similar rules. Consider the behaviors when we use computers. Certain files may be repeatedly re-accessed at short-interval. For example, your work may concentrate on the same directory for several days. We employ a simple algorithm to identify hot files on the hard disk, which are assumed to be accessed in the near future. All the accessed files are maintained in a LRU list, and associated with a reference count. Each element is denoted by {*inode*, *count*}. When a file is accessed, its count is increased by one. If its reference count exceeds a predefined value T, we copy it to flash memory device. Then it is deleted from the LRU list. Because the files in the disk are often too many to trace, consuming tremendous memory space, we set the maximum number of files to be traced. Once the maximum number M is reached, the trace information of file in the LRU position is removed.

Furthermore, file usage predictions from prior researches on providing the illusion of seamless network connectivity during network disconnect can be adapted to cache file working set[12,13].

3.2.2 File cache replacement algorithm

When the remaining capacity of flash memory cache device reaches a threshold value, such as 10% of its total size or it is not enough for the next cached file, replacement is needed. The main guideline for replacement algorithm is that files accessed less frequently and will not be accessed in near future should be removed from flash memory cache. If the victim file is dirty, it should be written back to disk. The oldest and yet still widely used algorithm in cache management is LRU. It is designed on the assumption that recently accessed data are likely to be accessed again in the near future. However, LRU only consider access recency, without regard to frequency.

We adopt the FBR (Frequency-Based Replacement)[23] algorithm, which weighs both of the frequency of a file's access and its last access in file replacement. FBR management algorithm is illustrated in Figure 2. Each block in the figure denotes an accessed file. The basis of the algorithm is LRU, in which the most recently referenced file is inserted at the head of the list. A reference count is maintained for each file. The LRU list is divided into three sections: new section, middle section, and old section. When a file is referenced, one and only one of following cases will happen. Then, the file is moved to the MRU position.

(1) The file is new to the list. It is brought into the new section, and initialized with a count of one;

(2) If the file is in the new section, its reference count is not increased. But it is increased by one if the file is in the middle or old section.

The middle section provides sufficiently long interval for file aging out of the new section to build up their reference counts even if they were relatively frequently referenced. Victim files are only choose in the old section. The one with the smallest reference count is evicted.

There are three parameters in cache replacement algorithm with FBR model: B1, the fraction of files in the new section, B2, the fraction files in the old section and R, a percentage of flash memory size to decide when to trigger file replacement. Since LRU list size is variable, we set B1 and B2 to the percentage of total list size. Therefore, the numbers of files in three sections are dynamically adjusted.

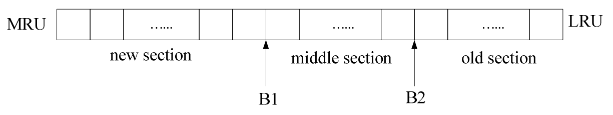

Figure 2 Flash memory device cache management

4. EXPERIMENTAL EVALUATION

In this section, we describe our simulation environment, and then evaluate our management scheme in comparison with alternative ones. Furthermore, results with different parameter configurations are examined.

4.1 Simulation Environment

4.1.1 Simulator

It is hard to measure the power consumption and performance of storage system accurately. Although the performance can be easily timed in real system, there is no flexible mechanism to count system energy. Moreover, system behaviors in real system are hard to be re-produced when comparing different storage schemes in the same operation scenario. Therefore, we implement a trace driven-simulator to evaluate our scheme. The file access from a program to storage in Linux kernel is shown in Figure 3(a). Corresponding to it, our simulator architecture is shown in Figure 3(b). The simulator uses file system level trace as input. It models our file cache management algorithms and emulates policies used for Linux buffer cache management. The buffer page size is set to 4KB. We interfaced our simulator with DiskSim 4.0[18]. The number of files accessed from each device, energy and performance can be reported by the simulator.

Flash SSD is used as an example of flash memory device in our experiment. If raw NAND flash memory is employed, the out-of-place update, garbage collection and wear-leveling processes should be considered carefully. Then the results will highly depends on concrete flash memory management algorithms. Flash SSD implements these policies in hardware-level and present a traditional hard disk interface to users. As a result, we can evaluate our scheme without regarding these problems.

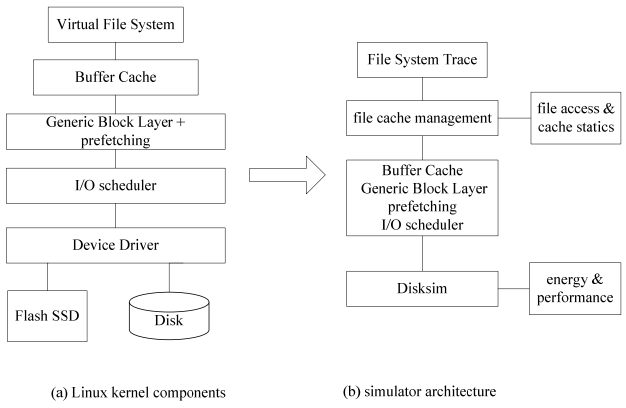

Figure 3 Kernel and Simulator architecture

DiskSim provides a large number of timing and configuration parameters for specific disks and the controllers/buses for I/O interface. Our simulator inherits the performance simulation module from DiskSim and we incorporated TPM power model[19] for studying the energy consumption. Detailed power and performance parameters of hard disk used in our simulation are given in Table 3. We assume that the spin down operation does not consume any power and that transition from the *Active* to *Idle* state takes zero time and power as in [19]. Also, we assume that the spin up time is 5 sec, which is the drive ready time. The total energy consumption is calculated as the sum of the energy used in each state and the energy consumed during transition periods. The energy in each state is estimated as power multiply the time spent in that state. For SSD, we assume a similar energy module, except that the cost of transitions to and from low-power mode is assumed to be negligible.

4.1.2 Traces

Traces are collected on a mobile computer environment with 512MB DDRII memory. Its system disk is 40GB. We modify Linux kernel 2.6.23 to trace open, read, write, and close file operations in the system disk. Information such as inode number, file size, access offset, and count are recorded. A relay mechanism is used to transfer data logged in kernel to user space. A user application collects the trace data and appends the data to a file on another disk to avoid affecting the trace itself. We collect five-day normal use trace including applications such as visiting websites, playing MP3, editing documentations and programming. There are about 13 million records on 11379 distinct files in the trace. The ratio of read operation to write operation is 2.8.

Table 3 Simulation Parameters

Parameter	Disk	SSD
Manufacturer	SAMSUNG	SAMSUNG
Interface	SATA 3.0Gb/s	SATA 3.0Gb/s
Buffer DRAM size	8MB	-
Byte per sector	512	-
Rotational speed	5400RPM	-
Meida to/from Buffer (max)	860Mbits/s	-
Buffer to/from Host (max)	300MB/s	-
Drive ready time	5 sec	0
Seek power	2.6W	0
Spinup power	5W	0
Read/Write power	2.4W	0.5W
Idle power	0.7W	0.2W
Standby power	0.25W	0.2W

4.2 Experimental Results

We evaluate our scheme in comparison with two solutions: disk-alone secondary storage system and PB-PDC heterogeneous secondary storage. The former one is treated as baseline.

4.2.1 Results

First, we examine the behavior of a single disk secondary storage system with various DRAM sizes. Figure 4 displays the page cache hit ratio and Table 4 lists the number of I/O operations to hard disk and its energy consumption, respectively. We can see that increasing DRAM size is helpful to improve page hit ratio. Large DRAM size absorbs more I/O requests. Then actual accesses to the secondary storage system are reduced and the hard disk's energy consumption is saved. However, there is marginal improvement after a certain point. Assuming a mobile system equipped with a 512MB DRAM, its page cache hit ratio is 95% and its energy consumption is 4930 joules. If a 1GB DRAM is employed, the page hit ratio is increased by only 1%. Though the energy consumption of hard disk is saved by 33%, the main memory's energy consumption is increased.

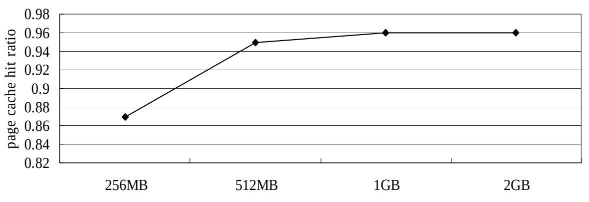

Figure 4 DRAM buffer cache hit ratio

Table 4 The impact of DRAM size

DRAM size	# of ios	Energy consumption(J)
256MB	1009843	21698.952455
512MB	300579	4930.048072
1GB	228972	3294.851941
2GB	224225	3424.951805

Then we evaluate our management scheme and PB-PDC. To compare with the single disk solution with the consideration of DRAM size, we set DRAM page cache size to 512MB. In our proposed policy, parameters such as C_f, access threshold T in cached file selection algorithm, and B1, B2, R in cache replacement algorithm have effects on hit ratio. The default values used in our evaluation are: flash memory size (512MB), T(3), R(0.7), B1(30), B2(80). Both LRU and FBR file cache replacement algorithms are examined.

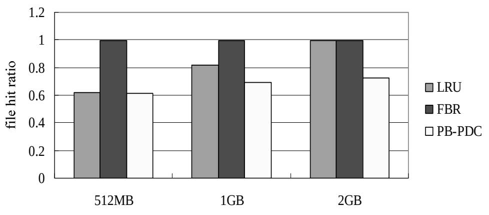

Figure 5 File hit ratios in flash memory device

Figure 5 lists the results hit ratios with various flash memory sizes. For LRU, FBR and PB-PDC, as flash size becomes larger, more files can be retained in the flash memory device. As a result, more file access requests can be fulfilled in flash. Our file cache management scheme results in higher file hit ratio than PB-PDC. This is because PB-PDC only stores read frequently files in flash memory. If a file has high read/write frequency rank, it will be moved to another device only if its operation rank is changed to some extent. So, it takes a long time to detect file access pattern changes. In contrast, our file cache management scheme mainly takes access frequency into consideration, caching files likely to access in near future. Furthermore, the reference count policy in

FBR avoids a file residing in flash memory too long, adapting to new patterns quickly. This is the reason that hit ratios of FBR are higher than those of LRU.

Figure 6 shows the energy consumptions results. Suppose the energy consumed by the disk without flash memory device is x, and the energy consumed by our scheme is y. The saved energy is x-y, and accordingly, the energy saving ratio is (x-y)/x. The energy saving ratio of each policy is increased with flash memory device size. Our file cache schemes perform better than PB-PDC and FBR outperforms LRU. Noticeable, with our management scheme, adding a 512MB flash memory device to single disk system saves energy by up to 90%, which is more energy efficient than employing a 1GB DRAM.

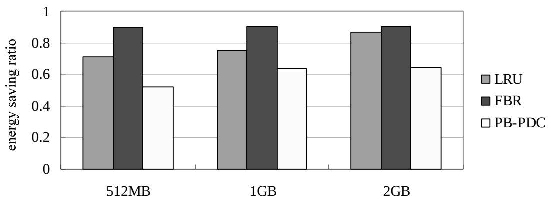

Figure 6 Energy saving ratio

I/O access time is defined as the sum of flash memory access time and hard disk access time. Normalized I/O time is depicted in Figure 7, where the time of disk-alone system is treated as baseline. As listed in Figure 5, only a small portion of I/O requests are directed to hard disk in LRU and FBR. Thus latencies induced by seeking and rotating of hard disk are eliminated. However, because most operations in the trace are read and read frequently files are stored in hard disk in PB-PDC, the reduction of its I/O access time is not as much as our scheme.

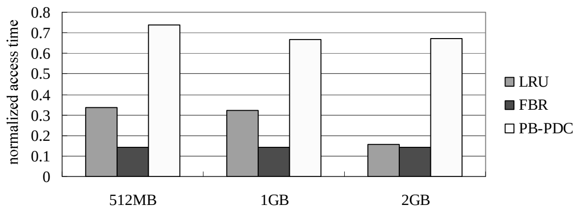

Figure 7 Normalized I/O access time

From above discussion, we can conclude that heterogeneous secondary storage is more energy and cost efficient than employing larger DRAM with negligible influence on page cache hit ratio. And our scheme performs better than PB-PDC.

4.2.2 Parameters

We use one day trace as an example to illustrate the impact of some parameters. The first one is hot threshold in cached file selection algorithm. Hit ratios and energy consumption results of LRU and FBR with various hot threshold value are listed in Table 5. Noticeably, FBR outperforms LRU. If the criteria to judge whether a file is hot is not so stringent, more files are stored in flash. However, false decisions may cause inefficient file copies. So, the hit ratio is increased with decreases in the number of hot threshold value. Energy consumption shows the same trend.

The second is replacement threshold value R Table 6 shows the results. A low one can not take full advantage of flash space and. In contrast, a high one gives less flexibility to future cache operation. In the experiment, 0.7 is a peak point.

Table 5 The impact of hot threshold

hot threshold	**hit ratio**		**energy consumption (J)**	
	LRU	**FBR**	**LRU**	**FBR**
1	0.9311	0.9841	2850.18	220.23
2	0.9311	0.9841	2850.18	220.23
3	0.9298	0.9803	2888.62	375.18
4	0.9276	0.9776	2877.23	355.17
5	0.8932	0.9750	3112.29	371.22
6	0.9242	0.9723	2774.04	432.34

Table 6 The impact of replacement threshold

replacement threshold	**hit ratio**		**energy consumption (J)**	
	LRU	**FBR**	**LRU**	**FBR**
0.5	0.8674	0.9718	4185.23	544.58
0.6	0.9081	0.9765	3182.93	467.26
0.7	0.9298	0.9803	2888.61	375.18
0.8	0.8937	0.9708	3157.37	1238.08
0.9	0.9680	0.9729	2376.56	1156.60

Figure 8 illustrates hit ratios for some combination configurations of B1 and B2. B2 is related to the replacement window and the difference between B1 and B2 decides the middle window. For a certain B1 value, the smaller B2 is, the higher its hit ratio is. This is because less frequently accessed files are chosen from a large window. In contrast, for a fixed B2 value, we find out that it is not suitable to use a too large or too small middle window. Though energy consumption doesn’t have direct relationship with hit ratio, there is a high probability that B1/B2 combination with high hit ration is energy efficient, as energy consumption results shown in Figure 9.

5. CONCLUSION AND FUTURE WORK

In this paper, we propose a heterogeneous secondary storage management scheme, where the flash memory device is treated as file level cache of hard disk. File cache management algorithms borrowed from traditional data cache management are described and evaluated. Evaluation shows that our scheme can reduce the energy consumption significantly. Furthermore, the simple file cache management algorithms are efficient and the I/O access time also benefits from our scheme. The proposed heterogeneous secondary storage system scheme weighs the effect of DRAM cache and secondary storage device, and takes advantage of flash memory, providing an energy efficient solution to mobile computers.

Our work can be extended in several directions. We are trying to collected more traces to explore the access pattern of file accesses and evaluating the proposed techniques in more detail, such as read/write/erase operations on flash memory. Since in this paper, we suppose that the flash memory device-aware management is implemented in file system level or flash SSD hardware level. These aspects are not quantified. Furthermore, we are also investigating more cache management algorithms and prefetching policies based on sufficient traces. From the results in this paper, we believe that the heterogeneous secondary storage management scheme can provide us a low cost, high performance, low energy consumption and large capacity storage solution for mobile computers. Moreover, it can be used in PCs and servers to alleviate the energy problem.

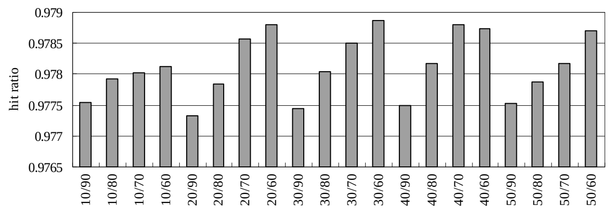

Figure 8 Hit ratio for B1/B2

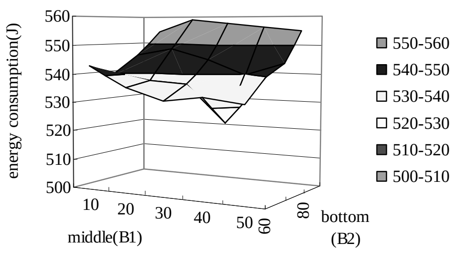

Figure 9 Energy consumption for B1/B2

6. REFERENCES

[1] A. Leventhal. Flash storage memory. *Communications of the ACM*, 51, 7 (2008), 47-51.

[2] SAMSUNG. Flash-SSD products. 2009. http://www.samsung.com/global/business/semiconductor/products/flash/Products_FlashSSD.html.

[3] B. Marsh, F. Douglis, and P. Krishnan. Flash memory file caching for mobile computers. *In Proceedings of the Twenty-Seventh Hawaii Internation Conference on System Sciences,*Wailea, HI, USA, 1994, 451-460.

[4] T. Bisson, S.A. Brandt, and D.D.E. Long. NVCache: Increasing the effectiveness of disk spin-down algorithms with caching. *In MASCOTS 2006,* Monterey, USA, 2006, 422- 432.

[5] T. Bission and S. Brandt. Reducing energy consumption with a non-volatile storage cache. *In Proceeding of International Workshop on Software Support for Portable Storage*, San Francisco, California, 2005.

[6] F. Chen, S. Jiang, and X. Zhang. SmartSaver: Turning Flash Drive into a Disk Energy Saver for Mobile Computers. *In Proceedings of the 2006 International Symposium on Low Power Electronics and Design,* Tegernsee, Germany, 2006, 412-417.

[7] R. Panabaker. Hybrid Hard Disk And ReadyDrive™ Technology: Improving Performance And Power For Windows Vista Mobile PCs. *In Proceeding of Microsoft WinHEC*, 2006.

[8] M. Trainor. Overcoming disk drive access bottlenecks with intel robson technology. 2006.

[9] Y.-J. Kim, K.-T. Kwon, and J. Kim. Energy-efficient file placement techniques for heterogeneous mobile storage systems. *In Proceedings of the 6th ACM & IEEE International conference on Embedded software*,Seoul, Korea, 2006, 171 – 177.

[10] C. Ruemmler and J. Wikes. A trace-driven analysis of disk working set sizes. Technical Report HPL-OSR-93-23, 1993.

[11] M. Satyanarayanan, The evolution of Coda. *ACM Transactions on Computer System*s, 20, 2, 2002, 85-124.

[12] G.H. Kuenning and G. J. Popek. Automated hoarding for mobile computers. *In Proceedings of the sixteenth ACM symposium on Operating System principles*, 1997, 264-275.

[13] E. Pinheiro and R. Bianchini. Energy conservation techniques for disk array-based servers. *In Proceedings of the 18th annual international conference on Supercomputing,* Malo, France, 2004, 68-78.

[14] D. Colarelli and D. Grunwald. Massive arrays of idle disks for storage archives. *In Proceedings of the 2002 ACM/IEEE conference on Supercomputing,* Baltirnore, Maryland, 2002, 1-11.

[15] D.P. Helmbold, D.D.E. Long, and B. Sherrod. A dynamic disk spin-down technique for mobile computing. *In Proceedings of the 2nd annual international conference on Mobile computing and networking*, Rye, New York, United States, 1996, 130-142.

[16] A.E. Papathanasiou and M.L. Scott. Aggressive prefetching: an idea whose time has come. *In Proceedings of the 10th conference on Hot Topics in Operating Systems,* 2005.

[17] J.T. Robinson and M.V. Devarakonda. Data cache management using frequency-based replacement. *In Proceedings of the 1990 ACM SIGMETRICS conference on Measurement and modeling of computer systems,* Boulder, Colorado, United States, 1990, 134-142.

[18] J.S. Bucy, J. Schindler, S. W.Schlosser, and G.R. Ganger, *The DiskSim Simulation Environment Version 4.0 Reference Manual*. Parallel Data Laboratory, Carnegie Mellon University: Pittsburgh, PA, 2008.

[19] Y.-H. Lu, E.-Y. Chung, et al. Quantitative comparison of power management algorithms. *In Proceedings of the conference on Design, Automation and Test in Europe*, Paris, France, 2000, 20-26.