SmartSaver: Turning Flash Drive into a Disk Energy Saver for Mobile Computers

Feng Chen
The Ohio State University
Columbus, OH 43210, USA
fchen@cse.ohio-state.edu

Song Jiang
Wayne State University
Detroit, MI 48202, USA
sjiang@ece.eng.wayne.edu

Xiaodong Zhang
The Ohio State University
Columbus, OH 43210, USA
zhang@cse.ohio-state.edu

ABSTRACT

In a mobile computer the hard disk consumes a considerable amount of energy. Existing dynamic power management policies usually take conservative approaches to save disk energy, and disk energy consumption remains a serious issue. Meanwhile, the flash drive is becoming a must-have portable storage device for almost every laptop user on travel. In this paper, we propose to make another highly desired use of the flash drive — saving disk energy. This is achieved by using the flash drive as a standby buffer for caching and prefetching disk data. Our design significantly extends disk idle times with careful and deliberate consideration of the particular characteristics of the flash drive. Trace-driven simulations show that up to 41% of disk energy can be saved with a relatively small amount of data written to the flash drive.

Categories and Subject Descriptors

D.4.2 [**Storage Management**]: Secondary Storage

General Terms

Design, Experimentation, Performance

Keywords

Hard disk, flash drive, energy saving, mobile computer

1. INTRODUCTION

As one of the major energy consumers in a mobile computer [9], the hard disk accounts for a considerable amount of energy consumption. Constrained by its mechanical nature, hardware support for disk energy conservation has not changed much over the years. Existing *Dynamic Power Management* policies are still using a simple timeout strategy to save disk energy: Once disk is idle for a specific period (timeout threshold), it is spun down to save energy. Upon arrival of a request, the disk is spun up to service the request. This strategy is also the basis of most existing disk energy-saving schemes for mobile computers [3, 4, 6, 13, 15].

Recently, with a rapid technology improvement, the flash drive has quickly taken the place of the floppy disk as a convenient portable storage device. Attracted by its compact size and low price, almost every mobile computer user now carries a flash drive on travel. Different from the hard disk, the flash drive is made of solid-state chips without any mechanical components, such as disk platters, which consume a considerable amount of energy. In addition, as a non-volatile storage device, the flash drive does not need power to maintain its data as main memory does. Compared with the hard disk, the energy consumption of the flash drive is almost negligible (its standby energy consumption is only around 1% of disk standby power consumption [14]). Unfortunately, in current mobile systems the flash drive is only used for file transfer or temporary storage, while its low energy consumption advantage has not yet been well exploited. In this paper we present a novel and practical scheme to utilize the low-price and ubiquitous flash drive to achieve a highly desired goal — saving disk energy.

This idea is challenged by two particular characteristics of the flash drive. First, the bandwidth of the flash drive is usually much lower than the peak bandwidth of the hard disk. For example, the Transcend TS1GJF2A Flash drive has a read bandwidth of 12MB/sec and write bandwidth of 8MB/sec [14], while the 4200RPM Hitachi-DK23DA hard disk can achieve a bandwidth of 35MB/sec. Second, the flash drive has a limited number of erasure (rewrite) cycles. Typically, a flash memory cell could wear out with over 100,000 overwrites. These two characteristics of the flash drive must be carefully considered in the scheme design to effectively use it for saving disk energy.

In this paper we present a comprehensive disk energy-saving scheme, called *SmartSaver*. This scheme uses the flash drive as a standby buffer for caching and prefetching disk data to service requests without disturbing disk. Although main memory can also be used as a buffer, it is undesired to use main memory for disk energy saving, as main memory itself is a big energy consumer [11].

Compared with existing disk energy-saving schemes, *SmartSaver* has the following merits: (1) It effectively exploits the low power consumption feature of the flash drive to save disk energy. (2) It carefully takes the flash drive's particular characteristics into consideration. (3) It is designed to be widely applicable in various operating systems with minimal changes to existing OS kernels. (4) Our experiments show that it can achieve significant disk energy saving by effectively extending disk idle times.

2. RELATED WORK

Existing disk energy-saving schemes for mobile computers can be classified into three groups. The first group of work focuses on the selection of timeout threshold, which could be a fixed time period, such as 2-5 seconds [4], or be adaptively adjusted at runtime [3, 6]. These schemes passively monitor disk I/O operations without extending disk idle time, which greatly limits their energy saving potential. The second group of work customizes system or application software for saving

ISLPED'06, October 4–6, 2006, Tegernsee, Germany.

disk energy [13,15]. The proposed scheme in [13] uses aggressive prefetching to create bursty disk accesses for long disk idle periods. While the scheme can extend disk idle time, significant changes are required to modify existing buffer cache management policies in OS kernels. Furthermore, as the buffer cache management policy is a performance-critical component in a kernel, practitioners have to be very cautious about potential performance degradation when they orient the component towards energy saving. In contrast, our scheme can effectively extend disk idle time with minimal changes to existing kernel policies. The third group of work proposes hardware designs to dynamically change disk rotational speed on the fly so that disk energy can be saved by reducing disk speed [5]. Our solution is complementary to this hardware mechanism, while such speed-adjustable disks are not yet available in the mainstream commercial market.

Using the flash memory for disk energy saving was discussed in an earlier work [10]. However, the work only provides a preliminary algorithm and many important design issues are missing, including file prefetching, balanced flash space allocation among cached and prefetched blocks, and consideration of the characteristics of the flash memory. Some of their design choices are inappropriate from today's point of view. For example, they write every missed block into flash memory when it is read from disk. Today's applications, such as movie player, tend to access a large volume of disk data. If all the data have to go through the flash memory, its low write bandwidth and limited erasure cycles would pose a serious problem. Another work [1] proposes to redirect write-back traffic to the flash drive when the disk is spun down so that the number of costly disk spin-up/downs could be reduced. However, this approach simply uses the flash drive as a write-only cache and it has no caching or prefetching mechanisms. Some other work also mentions the low power consumption characteristics of flash memory [12, 16]. However, no further detailed design considering the particular characteristics of the flash drive for disk energy saving was proposed. In this paper we provide a comprehensive flash memory management scheme that addresses these issues.

3. ISSUES AND CHALLENGES

A straightforward approach [10] to use the flash drive for disk energy saving is to simply use the flash drive as a new layer between the main memory and the hard disk in the storage hierarchy. This so-called 'cache all' policy simply caches every byte that is read from disk or evicted from memory into the flash drive, and uses the LRU replacement algorithm to free space once the flash drive is full.

After carefully examining this approach, here we summarize its several critical weaknesses, which also serve as technical bases for us to design an efficient flash-drive-based disk energy-saving scheme.

1. **Flash drive does not fit well in the storage hierarchy.** Since the write bandwidth of the flash drive is usually far less than that of the hard disk, simply caching all data that are transferred between the disk and memory in the flash drive can easily make the flash drive a bottleneck in the storage hierarchy.

2. **One-time access data should not be cached at all**. The 'cache all' approach inevitably stores one-time access data in the flash drive, which is a waste of its space and its precious erasure cycles.

3. **What to cache is critical**. Energy saving can be maximized by caching two types of data in the flash drive to effectively extend disk idle times: (1) very frequently reused data, and (2) data that move slowly from/to the disk, such as multimedia data with think time between disk reads and widely scattered data in the disk with long seek times to access. The 'cache all' approach pays no special attentions to these two types.

4. **What to replace is also critical**. Following the same rule mentioned above, we'd better not replace these two types of data when the flash drive is full. However, a standard LRU replacement algorithm could easily replace the second type of data that are infrequently used.

To address these issues, we must hold two principles in the design of our scheme. (1) A flash drive is not simply a 'smaller disk' or a 'larger memory'. Its low write bandwidth and limited erasure cycles need to be carefully considered. (2) Our scheme needs to balance the benefit (how much energy is saved) and the cost (how much flash memory space is demanded). In other words, it should identify and store the most valuable data in the flash drive for energy saving.

4. THE DESIGN OF SMARTSAVER

Typically, the hard disk has four power-consumption states — *active*, *idle*, *standby*, and *sleep*. To save energy, a disk is spun down to the standby state if it is idle for a specific period, and it can be spun up later to the active state for servicing a request. We call the time period between the disk spin-up and its consecutive spin-down a *busy period*, and the time period between the disk spin-down and its consecutive spin-up a *quiet period*. Since spinning up/down disk consumes substantial energy, the disk has to stay in the standby state for a sufficiently long period to compensate the energy overhead. The minimum interval to pay off the overhead is referred to as *break-even* time. Obviously, the longer disk is idle, the more energy can be saved.

To effectively extend disk idle periods, *SmartSaver* uses the flash drive as a buffer to store disk data, which can be used to service requests without disturbing the disk. The available flash drive space managed by *SmartSaver* plays three roles: (1) A caching area for holding data that is likely to be reused; (2) A prefetching area for storing data preloaded from the disk; and (3) A writeback area for temporarily storing the dirty blocks flushed from memory. Accordingly, *SmartSaver* partitions available flash drive space into three areas for caching, prefetching, and writeback, respectively.

4.1 Energy Saving Rate

The goal of caching is to avoid a future busy period by holding disk data that are likely to be reused in the flash drive. The energy that could be saved by avoiding a busy period may vary greatly. For example, *gzip* can compress a 20MB file in a few seconds, while *glimpse* may need a few minutes to build an index for the same file. Obviously, avoiding the busy period of *glimpse* can keep disk idle longer and save more energy than avoiding the busy period of *gzip*, though they have the same flash space cost.

To quantitatively measure the energy-saving potential of avoiding a busy period, we introduce a metric, *Energy Saving Rate* (*ESR*). ESR is the energy that could be saved if a busy period was avoided over the amount of data accessed during the busy period. The amount of saved energy equals to the energy spent in the busy period (the sum of disk active energy, disk idle energy, and the energy overhead of spinning up/down disk) subtracted by the disk standby energy spent in a quiet period of the same length. ESR describes the amount of saved energy each cached block could contribute if its busy period was avoided.

4.2 Caching

It is important to manage all the blocks accessed during one busy period as a whole. If we cache only a part of the accessed data blocks, the disk still has to be spun up for accessing the remaining blocks uncached in the flash drive, thus the effort of avoiding a busy period is foiled. To this end,

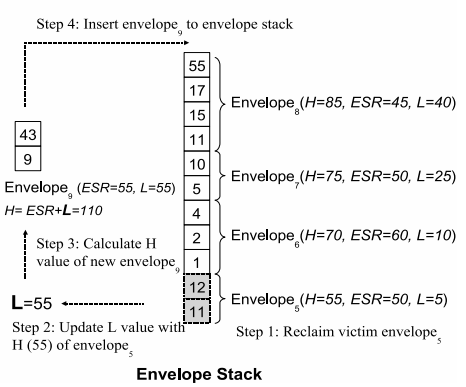


Figure 1: An example of envelope stack with four envelopes, among which $envelope_5$ with the smallest H value (55) is at the stack bottom and $envelope_8$ with the largest H value (85) is at the stack top. When a new envelope $envelope_9$ is to be inserted into the full envelope stack, the algorithm performs four steps as shown in the figure.

SmartSaver uses a data structure, called *envelope*, to record the metadata of all accessed data blocks during a busy period. Each envelope is associated with a busy period. When a block is requested from the disk during a busy period, its metadata is recorded in its associated envelope. The blocks in an envelope are organized in the LRU order. When the busy period is completed, its ESR value can be calculated. If we decide to cache the envelope, all the data blocks in it are written into the flash drive to maintain the integrity of the busy period. Cached envelopes are placed in a queue, called *envelope stack*, as shown in Figure 1.

When the caching area is full, we need to identify and reclaim the most 'valueless' blocks in terms of energy saving. A block's value for energy saving is determined not only by how much energy could be saved, which can be presented by ESR, but also by how likely it is to be accessed. The challenge is to evaluate and compare the blocks' value for energy saving simultaneously using these two orthogonal metrics.

To address the challenge, we adopt an algorithm similar to *GreedyDual-Size* [2], which is originally designed for web-caching. In the algorithm when an envelope e is inserted into the envelope stack or accessed in the stack, it is given an H value, where $H(e) = ESR(e) + L$. In the formula $ESR(e)$ is its corresponding busy period's ESR value, L is a global inflation value and is set to the H value of the most recently reclaimed envelope. L is initialized to 0. Envelopes in the stack are sorted in ascending order of their H values from the bottom to the top. When the caching area is full and a replacement is needed, the envelope at the stack bottom is selected, and the blocks in the envelope are reclaimed one by one in the LRU order. Once all blocks in an envelope are reclaimed, the envelope is destroyed, and L is updated using the H value of the destroyed envelope. Thus, the L value keeps being inflated. The most recently accessed block always uses the up-to-date L value, which represents its locality, and its ESR value, which represents its energy-saving potential, to calculate its H value. The H value determines the block's position in the stack and its timing to be reclaimed. Therefore, the block with the smallest H value is the most 'valueless' block and is located at the stack bottom for reclamation.

4.3 Prefetching

Prefetching is used for disk energy saving through *condensing* the sequential disk accesses at the beginning of an access sequence, which is called *stream*. After an initial prefetching of data into the flash drive, future requests to the data can be satisfied from the flash drive without accessing data on the disk. In other words, prefetching makes the *evenly* distributed disk accesses more *bursty*. In this section, we first explain how to identify the sequential disk accesses and then describe the prefetching mechanism itself.

Effective prefetching in *SmartSaver* requires an accurate identification of sequential accesses of files. In order to avoid intrusive changes to existing buffer cache management in OS kernels, *SmartSaver* does not conduct its own sequential access pattern detection. Instead, it relies on the kernel to provide hints of file access patterns to conduct its prefetching. In Linux, the file-level prefetching mechanism uses two readahead windows to detect access patterns, and the sizes of readahead windows are dynamically adjusted according to current accesses. *SmartSaver* monitors the readahead window sizes and uses them as the hints for its prefetching decisions. When the windows are enlarged to their full sizes, which indicates the kernel concludes that the file is being accessed sequentially, *SmartSaver* initiates a prefetching stream for it. When the windows are shrunk, which indicates a change of file access patterns, *SmartSaver* terminates the associated prefetching stream. If there exist multiple prefetching streams, we consider them as one aggregate stream to create a common disk idle period. When the disk is spun up or a stream uses up its prefetched blocks, all the streams are refilled.

Considering the low write bandwidth and limited erasure cycles of the flash drive, we need to reduce the amount of data written to the flash drive by avoiding inefficient prefetching. In a mobile computing environment, many applications are rate-based applications [13], which hold a steady rate of data consumption, such as *mplayer* and *xmms*. For such applications, *Prefetching Efficiency* for energy saving (the percentage of energy that can be saved through prefetching) is affected by two factors: data consumption rate and prefetching time (the maximum interval during that requests can be serviced with prefetched data in the flash drive). To guarantee efficient prefetching, we set some rules as follows: First, if the data consumption rate of a stream is not lower than the flash write bandwidth or the data consumption rate is so large that we can not use available space in the prefetching area to keep the disk idle longer than the disk break-even time, *SmartSaver* does not prefetch data for the stream. Second, *SmartSaver* sets the lower bound of prefetching time to the disk break-even time, and the upper bound to 300 seconds, as the prefetching time that goes too large brings diminishing benefits.

In a multi-task system, other concurrent disk events may break a long disk idle period created through prefetching. For example, we prefetch 4 minutes of movie clips in an attempt to keep the disk idle for that long period of time. However, if a virus-scanning application touches the disk every 10 seconds during that period, the expected 4 minutes of idle period cannot be realized. To coordinate prefetching with other disk events, we classify disk events that spin up a standby disk into two types: (1) *Bumps*, which are disk spin-up events that take place when streams use up their prefetched data and need to be refilled. A bump can be avoided by increasing prefetching time. (2) *Holes*, which are the other disk spin-up events waking up a standby disk, such as cold misses. If one bump takes place, which indicates that the disk could be kept idle longer if the prefetching time was increased, we then double the current prefetching time so that the prefetching becomes more aggressive. When a hole appears, which indicates that the current prefetching has been overshot, we reduce the prefetching time to the length of the most recent interval between two holes to avoid overshooting the expected disk idle period time.

4.4 Writeback

In most OS kernels, dirty blocks in main memory are periodically flushed to the disk to avoid losing data in volatile memory. For example, Linux writes dirty blocks older than 30 seconds back to the disk every 5 seconds. Such periodical disk accesses conflict with disk energy-saving efforts. Linux lap-

top mode recommends to accommodate a large ratio of dirty pages in memory and reduce the frequency of writing back of dirty pages. However, this solution increases the risk of losing data in volatile memory. *SmartSaver* solves this problem by redirecting the writeback traffic to the flash drive, a non-volatile buffer, to avoid breaking a long disk idle period.

In *SmartSaver*, the writeback area serves as a destaging buffer for temporarily holding dirty blocks. When a disk is in the standby state, dirty blocks are written to the writeback area to avoid spinning up the disk. Otherwise, dirty blocks are directly written to the hard disk. When the disk is back to the active state and over 90% of the writeback area is used, the dirty blocks in the writeback area are flushed to the disk. When the writeback area is overflowed, the disk is spun up to write back all the dirty blocks. In this way, *SmartSaver* achieves both energy saving and data safety purposes.

4.5 Balancing the three areas

As the flash drive is partitioned into the caching, prefetching, and writeback areas, *SmartSaver* employs a balancing mechanism to dynamically adjust the sizes of these three areas to optimize overall energy-saving efficiency. The principle for the mechanism is that the area that can save more energy with the same amount of additional buffer space should be given more buffers. *SmartSaver* monitors accesses to each area and periodically evaluates the amount of energy saved due to the addition of another N blocks. Each time N blocks are reclaimed from the least 'productive' area, where adding N more blocks can achieve less increased energy saving than adding them to other areas, and allocated to the most 'productive' one.

5. PERFORMANCE EVALUATION

5.1 Simulation

We wrote a trace-driven simulator to evaluate our disk energy-saving scheme. It simulates the management of three storage devices: main memory, hard disk, and flash drive. The simulator emulates the policies used for Linux buffer cache management, including the 2Q-like memory page replacement algorithm, the two-window readahead policy that prefetches up to 32 pages, and the I/O request clustering mechanism for grouping consecutive blocks in multiple requests into a large request.

The disk simulated in our experiment is the Hitachi-DK23DA hard disk [7]. It has a 30GB capacity, 4200 RPM and 35MB/sec peak bandwidth. Its average seek time is 13.0 ms, and its average rotation time is 7.1ms. Its energy consumption parameters are listed in Table 1. The timeout threshold for disk spin-down is set as 20 seconds, the default value for Linux laptop mode.

P_{active}	Active Power	2.00 W
P_{idle}	Idle Power	1.60 W
$P_{standby}$	Standby Power	0.15 W
E_{spinup}	Spin up Energy	5.00 J
$E_{spindown}$	Spin down Energy	2.94 J
T_{spinup}	Spin up Time	1.60 sec
$T_{spindown}$	Spin down Time	2.30 sec

Table 1: The energy consumption parameters for the Hitachi-DK23DA hard disk.

The simulated flash drive is the Transcend TS1GJF2A flash drive [14] with a 1GB capacity. Its read and write bandwidths are 12MB/sec and 8MB/sec, respectively. Its maximum active power consumption, 0.37W, is conservatively adopted as both read and write power consumptions in our simulations. Its sleep power consumption is 0.60mW.

To measure the energy consumed in the disk, we need to evaluate the period length of each disk state. The period length of disk active state can be broken down to three components: transfer time, rotation time, and seek time. Among them, the transfer time is the amount of requested data divided by disk bandwidth. As the rotation time is variable, we use a random value between 0 and 14.2ms (the maximum rotation delay). The seek time is determined by the seek distance between two consecutive requests. To estimate the seek time, we trace the latency of real disk accesses with different seek distances and plot a seek profile as done in [8]. We can then estimate seek times for various seek distances using the seek profile curve.

In the experiments, besides *SmartSaver* we also simulate the disk energy-saving scheme presented in [10], which is denoted as *Baseline*, and the original Linux laptop mode without a flash drive, which is denoted as *Linux*.

5.2 Traces

We modified the *strace* utility in Linux to collect traces to drive our simulator. The modified *strace* can intercept system calls related to file operations, such as open(), close(), read(), write(), and lseek(). For each system call, we collected the following information: PID, file descriptor, inode number, offset, size, type, timestamp, and duration. Blocks of the traced files are sequentially mapped to a simulated disk with a small random distance between files to simulate an actual layout of files on disk.

Eight traces of seven applications that are typically used in a mobile computing environment were collected, as listed in Table 2. Besides the five single-application traces, *thunderbird*, *scp-r*, *make*, *grep*, and *xmms*, we also collected traces *scp-mplayer* and *ftp-mplayer*, where two applications ran concurrently as representatives of multi-stream cases.

Name	Description	# of files	Size
thunderbird	an email client tool	283	188.0
scp-r	a remote copy tool	12669	191.0
make	a Linux kernel make tool	2579	72.5
grep	a text search tool	1332	50.4
xmms	a mp3 player	116	47.9
ftp-mplayer	a ftp client tool & a movie player	91	364.7
scp-mplayer	a remote copy tool & a movie player	91	364.7

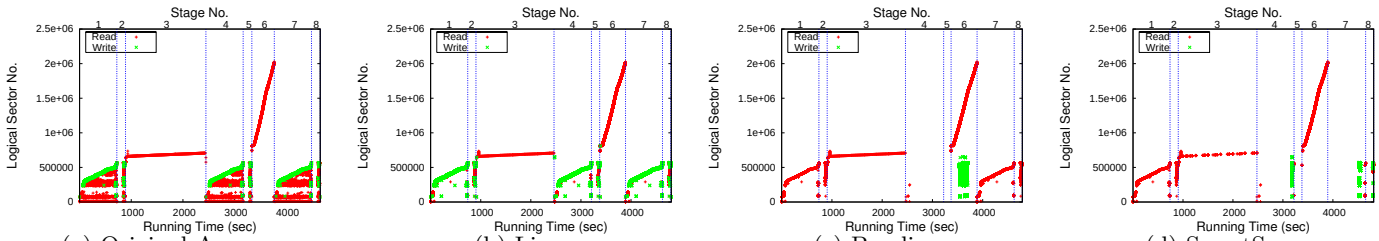
Table 2: Trace descriptions. Sizes are in units of MBs. Both *scp-mplayer* and *ftp-mplayer* concurrently run two programs to access two separate sets of identical files. *Scp* transfers data in 8MB/sec, and *ftp* transfers data in 20KB/sec.

We synthesized two typical mobile computing scenarios by concatenating individual application traces one by one. The first experiment simulates a programming scenario, where a user is programming, searching codes, listening to music, and performing remote file transfer. The second experiment simulates a networking scenario, where a user is checking and searching emails, performing remote file transfer, using FTP service, and watching a movie. They are referred to as *programming* and *networking*, respectively.

5.3 Case 1: Programming

The *programming* scenario is composed of eight stages. Each stage is a replay of one trace, *make*, *grep*, *xmms*, *make*, *grep*, *scp-r*, *make*, and *grep*, in that order. This scenario consists of both non-sequential accesses on a large number of small files (*make*, *grep*, and *scp-r*), and relatively long sequential accesses on large files (*xmms*).

Since disk idle time is critical for energy saving, we plot the Cumulative Distribution Function (CDF) curves of disk idle times for three schemes, as shown in Figure 2. The vertical line in the figure is the disk break-even time. Because disk energy could be saved only when the disk is idle longer than



(a) Original Accesses (b) Linux (c) Baseline (d) SmartSaver

Figure 3: Case 1 *programming*: Disk I/O accesses in original application, *Linux*, *Baseline*, and *SmartSaver* schemes. All experiments are configured with a 64MB memory and a 128MB flash drive.

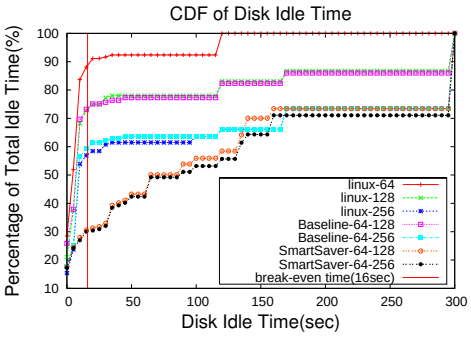


Figure 2: The CDF curves of disk idle times for *programming*. Linux-64, Linux-128, and Linux-256 are the original *Linux* scheme with a 64MB, 128MB, and 256MB main memory, respectively. Baseline-64-128 and Baseline-64-256 are the *Baseline* scheme with a 64MB memory and a 128MB or 256MB flash drive, respectively. SmartSaver-64-128 and SmartSaver-64-256 are the *SmartSaver* scheme with a 64MB memory and a 128MB or 256MB flash drive, respectively. The vertical line is the disk break-even time, 16 seconds.

Scheme	Mem	Flash	Energy(J)	Flash R/W
Linux	64	N/A	7325.0	N/A
Linux	128	N/A	6317.3	N/A
Linux	256	N/A	5231.6	N/A
Baseline	64	128	6365.7	64/493
Baseline	64	256	5312.9	104/442
SmartSaver	64	128	4318.0	96/263
SmartSaver	64	256	4046.2	110/316

Table 3: The energy consumption and the Read/Write volume in the flash drive for *programming*. The memory sizes, flash drive sizes, and flash R/W volumes are in units of MBs.

the break-even time, we call the percentage of the sum of idle periods that are shorter than the break-even time over the total disk idle time *Unusable Idleness Percentage* (UIP). The smaller an UIP is, the more energy could be saved.

Figure 2 shows that *Linux* with a 64MB memory has an UIP as large as 89.1%. This percentage is reduced to 73.3% and 57.2% with a 128MB and 256MB memory, respectively. This is because the working sets of *make* and *grep* are more likely to be held in a larger memory to avoid disk accesses. In a 64MB memory system, *Baseline* with a 128MB flash drive has an UIP of 73.9%, which is almost identical to that for *Linux* with a 128MB memory. This is because *Baseline* manages the flash drive as a larger memory to hold all data blocks transferred between memory and disk. In contrast, *SmartSaver* with a 128MB flash drive has an UIP of only 30.8%. Correspondingly, *SmartSaver* consumes only 4318.0J disk energy, compared with 6365.7J for *Baseline* and 7325.0J for *Linux*, as shown in Table 3.

SmartSaver attempts to save the limited erasure cycles of the flash drive. With a 128MB flash drive *SmartSaver* writes only 263MB data to the flash drive, which is 46.6% less than 493MB for *Baseline*. Meanwhile, with a 128MB flash drive *SmartSaver* reads 96MB data from flash drive, while *Baseline* reads only 64MB data. The read/write ratios clearly show that *SmartSaver* uses the flash drive more efficiently. This is a much desired advantage considering limited erasure cycles of the flash drive. Using existing wear-levelling techniques, such as JFFS2, *SmartSaver* can distribute a small amount of overwrites evenly on the flash drive. For *programming*, if we use a 1GB flash drive and configure *SmartSaver* with 128MB flash space for energy saving, one cell could be overwritten once approximately every 4.5 hours.

For a detailed analysis of disk accesses at each stage, we plot the disk I/O activities with a setting of a 64MB memory and a 128MB flash drive in Figure 3. As shown in this figure, there are disk accesses at all stages for *Linux*, because *Linux* cannot accommodate the working set in memory. *Baseline* performs better with a flash drive. For example, at stage 4 and stage 5 the disk keeps idle for 822 seconds, as the working set of *make* and *grep* is held in the flash drive. However, since *Baseline* caches every byte transferred between memory and disk, at stage 6 *scp-r* flushes the data of *make* out from the flash drive, which causes the disk to be spun up to serve *make* at the next stage. In contrast, *SmartSaver* protects the data of *make* from being replaced by the data of *scp-r* and the disk remains idle at stage 7. This figure also shows that most writeback requests are absorbed by the flash drive and reshaped as a few bursts, which helps maintain a long disk idle period.

5.4 Case 2: Networking

The *networking* scenario includes three stages, *thunderbird*, *scp-mplayer*, and *ftp-mplayer*. Except *thunderbird*, which accesses one or multiple email files, each sequentially and non-continuously, *scp*, *mplayer*, and *ftp* sequentially access files.

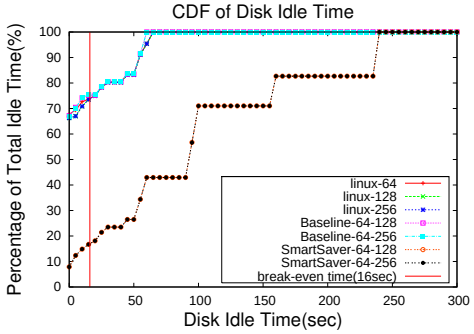
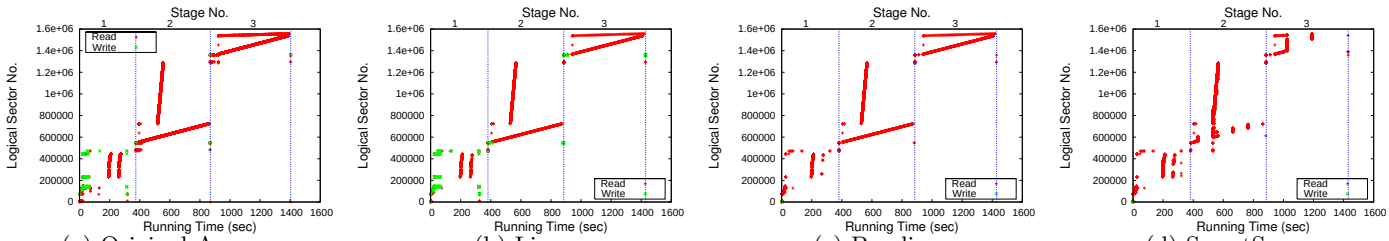


Figure 4: The CDF curves of disk idle times for *networking*.

Figure 4 shows the CDF curves of disk idle times. Both *Linux* and *Baseline* have an UIP of around 73%, and no improvements can be achieved with additional memory or flash



(a) Original Accesses (b) Linux (c) Baseline (d) SmartSaver

Figure 5: Case 2 *networking*: Disk I/O accesses in original application, *Linux*, *Baseline*, and *SmartSaver* schemes. All experiments are configured with a 64MB memory and a 128MB flash drive.

Scheme	Mem	Flash	Energy(J)	Flash R/W
Linux	64	N/A	2076.4	N/A
Linux	128	N/A	2052.5	N/A
Linux	256	N/A	2052.1	N/A
Baseline	64	128	2103.2	26/574
Baseline	64	256	2102.2	26/552
SmartSaver	64	128	1207.5	210/443
SmartSaver	64	256	1208.0	210/447

Table 4: The energy consumption and Read/Write volume in the flash drive for *networking*. The memory sizes, flash drive sizes, and flash R/W volumes are in units of MBs.

drive space, because the disk activities in this scenario are dominated by one-time accesses. In contrast, *SmartSaver* significantly extends disk idle time through prefetching. With a 128MB flash drive, it reduced the UIP to 16.7%. As shown in Table 4, *Linux* with a 64MB memory consumes 2076.4J, while *Baseline* with a 64MB memory and a 128MB flash drive consumes 2103.2J, which is even more than that for *Linux*. This is because *Baseline* wastes energy on caching the data that are never to be reused in the flash drive. With the same setting, *SmartSaver* consumes only 1207.5J, which is 41.8% less than that for *Linux*. We also observe that *SmartSaver* is not sensitive to the size of the flash drive. This is because the upper-bound of prefetching time limits the consumption of flash drive space by avoiding too aggressive prefetching. As in *programming*, *SmartSaver* with a 128MB flash drive writes 22.8% less data to the flash drive (443MB) than *Baseline* (574MB), but reads 707.6% more data (210MB) than *Baseline* (26MB).

Figure 5 plots the disk I/O activities for *networking*. Both *Linux* and *Baseline* cannot avoid disk busy periods in each stage. In contrast, *SmartSaver* reshapes most evenly distributed disk accesses into several bursts of disk accesses through prefetching. Figure 5(d) shows that during the overlapped 36 seconds of *scp* and *mplayer* at stage 2, there exist disk accesses. This is because *SmartSaver* does not prefetch for *scp*, whose data consumption rate is 8MB/sec, the flash write bandwidth. The prefetching for *mplayer* is not carried out either, because *scp* keeps the disk busy and prefetching for *mplayer* is meaningless at that time. After *scp* completes, the prefetching time for *mplayer* quickly increases to the upper bound. At stage 3, the prefetching is conducted for both *ftp*, whose consumption rate is only 20KB/sec, and *mplayer*.

6. CONCLUSION

In this paper, we identify some critical issues involved in the flash memory management to save disk energy. We provide a comprehensive set of solutions to maintain an effective use of the flash drive for three goals: (1) accommodating the unique properties of the flash memory; (2) minimizing OS kernel changes; and (3) significantly improving disk energy saving. Trace driven simulations for typical mobile computing scenarios demonstrate that our scheme can save up to 41% disk energy compared with existing policies used in Linux.

7. ACKNOWLEDGMENT

We thank anonymous reviewers for their comments and suggestions. We appreciate William L. Bynum for reading the paper and his comments. Some technical discussions with Xiaoning Ding are also helpful. This work is partially supported by the National Science Foundation under grants CNS-0098055 and CNS-0405909.

8. REFERENCES

[1] 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*, San Francisco, CA, March 2005.

[2] P. Cao and S. Irani. Cost-aware www proxy caching algorithms. In *Proc. of USENIX'97*, Dec. 1997.

[3] F. Douglis, P. Krishnan, and B. Bershad. Adaptive disk spin-down policies for mobile computers. In *Computing Systems*, volume 8(4), pages 381–413, 1995.

[4] F. Douglis, P. Krishnan, and B. Marsh. Thwarting the power-hungry disk. In *Proc. of USENIX'94*, Jan. 1994.

[5] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. Drpm: Dynamic speed control for power management in server class disks. In *Proc. of ISCA'03*, 2003.

[6] D. P. Helmbold, D. D. E. Long, and B. Sherrod. A dynamic disk spin-down technique for mobile computing. In *Proc. of the 2nd Annual International Conference on Mobile Computing and Networking*, 1996.

[7] HITACHI. http://www.hitachigst.com/tech/techlib.nsf/products/DK23DA_Series.

[8] H. Huang, W. Hung, and K. G. Shin. Fs2: dynamic data replication in free disk space for improving disk performance and energy consumption. In *Proc. of SOSP'05*, Brighton, U.K., Oct. 2005.

[9] Intel. Intel mobile platform vision guide for 2003.

[10] B. Marsh, F. Douglis, and P. Krishnan. Flash memory file caching for mobile computers. In *Proc. of the 27th Hawaii Conference on Systems Science*, Hawaii, 1994.

[11] MICRON. http://download.micron.com/pdf/technotes/TN4603.pdf.

[12] E. B. Nightingale and J. Flinn. Energy-efficiency and storage flexibility in the blue file system. In *Proc. of OSDI'04*, SF, CA, Dec. 2004.

[13] A. E. Papathanasiou and M. L. Scott. Energy efficient prefetching and caching. In *Proc. of USENIX'04*, 2004.

[14] TRANSCEND. http://www.transcend.com.tw/Support/DLCenter/Datasheet/TS1GJF2A.pdf.

[15] A. Weissel, B. Beutel, and F. Bellosa. Cooperative io - a novel io semantics for energy-aware applications. In *Proc. of OSDI'02*, Dec. 2002.

[16] F. Zheng, N. Garg, S. Sobti, C. Zhang, R. E. Joseph, A. Krishnamurty, and R. Y. Wang. Considering the energy consumption of mobile storage alternatives. In *Proc. of MASOCTS'03*, Oct. 2003.