PS-BC: Power-saving Considerations in Design of Buffer Caches Serving Heterogeneous Storage Devices

Feng Chen and Xiaodong Zhang
Department of Computer Science & Engineering
The Ohio State University
Columbus, OH 43210, USA
{fchen, zhang}@cse.ohio-state.edu

ABSTRACT

Under a replacement policy, existing operating systems identify and maintain most frequently used storage data in buffer caches located in main memory, aiming at low-latency I/O data accesses. However, replacement policies can also strongly affect energy consumptions of various connected storage devices, which has not been a consideration in the design and implementation of buffer cache management. In this paper, we present a system framework for an energy-aware buffer cache replacement, called PS-BC (power-saving buffer cache). By considering several critical factors affecting system energy consumption, PS-BC can effectively improve system energy efficiency, while it is able to flexibly incorporate conventional performance-oriented buffer cache replacement policies for different performance objectives. Our experimental studies based on a trace-driven simulation show that the PS-BC framework embedded with the CLOCK replacement policy can achieve an energy saving rate of up to 32.5% with a minimal overhead for various workloads.

Categories and Subject Descriptors

D.4.2 [**Storage Management**]: Main Memory

General Terms

Design, Experimentation, Performance

Keywords

Hard disk, buffer caches, energy saving, power management

1. INTRODUCTION

Buffer cache is a performance-critical component in operating systems. As an intermediate layer bridging the performance gap between processors and storage devices, based on its prediction, buffer cache maintains a data set with high locality in main memory to avoid long I/O latency.

The decision maker in buffer cache management is its replacement policy, which distinguishes locality strengths of blocks and selectively caches the blocks that are most likely to be reused again. Following this basic performance oriented principle, the advancement of buffer cache management has gone three different stages during the last 40 years. LRU and CLOCK [11] represent an early development in the field. They are effective for strong locality data accesses and have been widely used. The major limitation of LRU-like policies is the inability to handle workloads with weak locality (e.g. one-time accesses and loop-like accesses) [15]. To address this serious issue, researchers have made many years' efforts. Representative algorithms and implementations are LIRS [15] and CLOCK-Pro [13], which have been gradually adopted in production systems. The most recent development to improve buffer cache management is to make spatial locality into consideration in replacement policies, such as sequential accesses versus random accesses in disks [14].

Computer system development has been highly performance driven, and power saving issues have been paid attention only recently. Buffer cache management can significantly influence the energy consumption of computer systems due to its *filtering effect* – accessing a block in the buffer cache can be satisfied in memory without accessing the storage device. As a result, the replacement policy, which decides the selection of blocks for caching, is capable of shaping the data access sequence to storage devices, in terms of both the amount of accessed data and the pattern in which data are accessed. In practice, different data access sequences can lead to completely divergent energy consumptions, due to the following three reasons:

1. **Accessing different storage devices incurs disparate energy consumptions**: Modern computer systems often interact with a broad set of storage devices, which have various energy consumption requirements, such as flash disks and hard drives. Even the same type of devices can have different power consumption requirements due to variable performance specifications. For example, a 15,000 RPM IBM Ultrastar 36Z15 hard disk has an active power of 13.5 Watts, while a 4200RPM Hitachi DK23DA laptop disk has a power of only 2 Watts in the same power mode. Thus caching blocks from high-power devices can save more energy than caching blocks from low-power devices.

2. **Different data access patterns lead to different energy consumptions**: The energy consumption of accessing storage devices is determined not only by the amount of accessed data but also by the access patterns. For example, a bursty workload on a hard disk is more likely to consume less energy than a non-bursty workload, because its long idle periods can create more opportunities for the built-in Dynamic Power Management (DPM) unit to transition the disk to a low power mode. Thus caching blocks being accessed in a non-bursty pattern is more energy efficient.

3. **Energy consumption of the base system is also I/O performance dependent**: Besides storage devices, the base system, including the power-hungry CPU and memory, also consumes a significant amount of energy. For example, the SUN Fire X4100 Server base system has a power of 218 Watts, while a high performance disk has a power of only around 10 Watts. Though storage devices in such a system only account for a small portion of the overall energy consumption, the high storage access latency can cause the whole system to waste energy by waiting for slow I/O operations. In other words, improving I/O performance can effectively reduce system-wide energy consumption, especially for data-intensive applications [16]. Therefore

caching blocks accessing which would incur high I/O latency is also beneficial for energy-saving purpose.

Having carefully considered the three factors, we propose a system framework for an energy-aware buffer cache replacement, called PS-BC (power-saving buffer cache). PS-BC can effectively optimize energy efficiency based on the key factors affecting the system energy consumption, meanwhile, it is able to flexibly incorporate conventional performance-oriented buffer cache replacement policies for different performance objectives. Our trace-driven simulation based studies show that PS-BC embedded with the CLOCK replacement policy can achieve an energy saving rate of up to 32.5% with a minimal overhead under various scenarios.

2. ENERGY SAVING RATE

Modern storage devices often provide multi-level power modes. For example, a typical hard disk usually has four power modes: *active*, *idle*, *standby*, and *sleep*. Similarly, wireless network interface cards (WNIC) provide continuous-aware mode (CAM) and power saving mode (PSM). For the purpose of energy saving, the built-in DPM mechanism can transition a device from a high power mode to a low power mode, once the device has been idle for a 'time-out' period.

Without loss of generality, we call the periods of time when a storage device stays at the high power mode *busy periods*, and the rest *quiet periods*. If all blocks accessed during a busy period were already held in memory by an energy-aware caching policy, the busy period could be avoided and become a quiet period, since the device would be kept at the low power mode. In other words, each block accessed during a busy period consumes a certain amount of energy, and caching it in memory could save the same amount of energy. Our goal is to identify and cache the blocks caching which could save the most energy.

$$ESR = \frac{E_{high}(d) - E_{low}(d) + E_{trans}(d) + P_{base} \times L(b)}{S(b)} \tag{2.1}$$

Figure 1: Energy Saving Rate (ESR). $E_{high}(d)$ **and** $E_{low}(d)$ **are energy consumptions of a device** d **at high and low power modes, respectively.** $E_{trans}(d)$ **is the energy overhead of power mode transition.** P_{base} **is the base system power.** $L(b)$ **is the I/O latency of accessing data during a busy period,** b**.** $S(b)$ **is the number of blocks accessed during the busy period.**

To quantitatively describe the amount of energy that caching a block could save, we introduce a metric, *Energy Saving Rate* (*ESR*), as shown in Equation (2.1). In this equation, the involved parameters either can be found in the device specifications (e.g. $P_{base}(d)$), or can be measured or calculated out during runtime. For example, $E_{high}(d)$ is a product of the device high power (in Watts) and the duration of the measured busy period time (in Seconds).

ESR describes the three key factors affecting energy saving in one single metric. In particular, $E_{high}(d) - E_{low}(d) + E_{trans}(d)$ reflects the power consumption characteristics of the storage device, $P_{base} \times L(b)$ describes the impact of I/O latency to the energy consumption of the base system, and $S(b)$ represents the burstiness of data accesses. In essence, ESR is a model describing the average energy consumed for accessing a block, and it also can be flexibly extended to incorporate other device power models (e.g. the devices with multi-level power modes), if needed.

In the end of a busy period, i.e. when a device is transitioned to the low power mode, we can know the measured I/O latency, the amount of accessed data, and the duration of the busy period. The ESR value for this busy period can be calculated using Equation (2.1). Each block accessed during the busy period is associated with this ESR value. In PS-BC, we use the ESR value associated with a block as an *indicator* to describe its value for energy saving. In particular, the larger an ESR value is, the more valuable the associated block is regarded for energy saving.

3. THE DESIGN OF PS-BC FRAMEWORK

As a flexible system framework, PS-BC can incorporate any advanced replacement policies designed for optimizing performance, while it introduces the capability of being energy aware to buffer cache management. In this section, we will explain how to achieve this goal in our design.

3.1 The PS-BC Framework

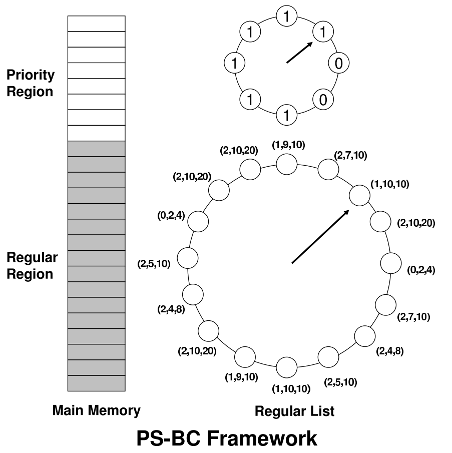


Figure 2: The PS-BC framework for energy-aware buffer cache replacement. Each circle in the list represents a memory block. The numbers beside blocks in the regular list are its REF counter, AGE counter, normalized ESR value, respectively.

In the PS-BC framework, the buffer cache is segmented into two regions, *regular region* and *priority region* (see Figure 2). The blocks in the two regions are managed separately by using different replacement policies. As the names indicate, the priority region manages its blocks as 'first-class citizens' by using a performance-oriented replacement policy, e.g. CLOCK, to optimize system performance. The blocks in the regular region are regarded as 'second-class citizens' and managed by an energy-oriented replacement policy (shown in Section 3.3) for energy saving.

Initially, a block fetched into the buffer cache is inserted into the regular region. If it is recognized as a big 'energy-saving contributor' (e.g. it is accessed frequently or carries a large ESR value), the block is *promoted* to the priority region. Accordingly, a block, which is regarded as least valuable for I/O performance, is *demoted* from the priority region to the regular region. Upon a memory miss, a victim block is selected from the regular region.

Partitioning the buffer cache with different region sizes can affect the buffer cache management. For example, a large regular region means the overall replacement is more energy-saving oriented. As a configurable option, we set the regular region size twice as large as the priority region size to make the replacement more energy effective. In our experimental studies we found this setting largely balances performance and energy-saving and works pleasantly well in practice. We also can apply techniques similar to that in [9] to automatically tune this parameter during runtime.

3.2 Managing the Priority Region

The priority region adopts a performance-oriented replacement policy, which manages the blocks to optimize I/O performance. The PS-BC framework flexibly allows any performance-oriented replacement policy (e.g. CLOCK-Pro [13] or LIRS [15]) to be integrated into the management of the priority region, which has several merits. First, adopting a performance-oriented replacement policy to manage the priority region can effectively hold the most performance-critical blocks in memory and avoid introducing high I/O latency. Second, the blocks are promoted to the priority region due to their high contribution for energy saving, so system energy efficiency would not be affected much without further considering their energy-saving properties. Third, this design can also flexibly incorporate different replacement policies for various design objectives. In this paper, we show a case of embedding the CLOCK algorithm [11] in the PS-BC framework to manage the priority region.

3.3 Managing the Regular Region

A block's value for energy saving is determined not only by how much energy it could save, its associated *ESR* value, but also how likely it would be accessed, its *locality.* For example, caching a frequently accessed block with a small ESR value can save a substantial amount of energy comparable to caching an infrequently accessed block with a large ESR value. Therefore, the regular region manages the blocks by considering both the blocks' energy-saving properties and temporal locality.

Akin to the CLOCK algorithm [11], we link the blocks in the regular region in a circular list, called *regular list*, and a clock hand points to the list end. Each block is associated with two counters, *REF*, which records how many times the block has been accessed, and *AGE*, which records how many times the block has been swept over by the clock hand. Both counters are initialized to zero when the block is inserted into the list. A block's ESR value, which represents its value for energy saving, is normalized to an integer, *nESR*, as a threshold to determine how long this block is allowed to stay in the buffer cache. In the next section, we will introduce the normalization of ESR values.

```
/* initialize the promotion threshold */
Initialize promo_thld = 0;

get_victim_on_regular_list()
{
 diff = INFINITE;
 victim = NULL;
 for each block in the regular list{
   block->AGE ++;
   /* return the first met block that is over-aged */
   if(block->AGE > block->nESR){
      return block;
   }
   if(block->nESR - block->AGE < diff){
     diff = block->nESR - block->AGE;
     victim = block;
   }
 }
 return victim;
}
hit_block_on_regular_list(block)
{
 block->REF ++;
 block->AGE = 0;
 if(block->REF*block->ESR > promo_thld){
   victim = get_victim_block_from_priority_region();
   promo_thld = p*promo_thld + \\
               (1-p)*(victim->REF*victim->ESR);
   demote_block_to_regular_region(victim);
   promote_block_to_priority_region(block);
 }
}
```

Figure 3: Operations on blocks in the regular region

Figure 3 shows the pseudocode of two key operations in the regular list. Upon a memory miss, we need to identify and evict the most 'valueless' block in the regular region as a victim block. Each time, we scan the blocks in the regular list starting from the block that the clock hand points to and move the clock hand in the clockwise direction. Each swept block has its AGE counter incremented by one to depreciate its priority of staying in the buffer cache. If the block's AGE counter exceeds its normalized ESR value, *nESR*, it is identified as the victim block. Otherwise, we move the clock hand to the next block and repeat this process until all blocks are scanned. While scanning the blocks in the list, we record the block whose AGE is closest to its nESR value. This block is returned as the victim block, if all blocks in the list are scanned and no qualified victim block is found. This brings two benefits: First, it avoids multiple scans for reclaiming one block, which reduces overhead, especially when there are a large number of blocks in the list. Second, it ensures that a block is penalized by incrementing its AGE counter at most once for a memory miss.

Upon a hit on a block in the regular list, the block's REF counter is incremented by one to track the number of accesses to it, and its AGE counter is cleared to restart the aging process. If a block has saved a substantial amount of energy (e.g. it is frequently hit in memory or each hit to it saves a lot of energy), we grant it a high priority and promote it into the priority region, where it is given more privilege to stay in memory. To this end, we check if the block's accumulated energy saving exceeds a threshold *promo_thld*, as shown in the pseudocode. If true, this block is promoted to the priority region, and a block is demoted from the priority region accordingly. The threshold is automatically determined and dynamically updated by the energy saving contributed by the blocks that are recently demoted from the priority region. The reason behind this is that, if a block saves even more energy than another block that is resident in the priority region with a higher priority, it should be promoted to the priority region.

3.4 Normalizing ESR

As mentioned in the previous section, a block's ESR value needs to be normalized to an integer, *nESR*, which determines how long the block can stay in the regular list. We use Equation (3.2) to normalize the ESR values.

$$nESR(esr) = \left\lfloor \frac{(esr - ESR_{floor}) \times Resolution}{ESR_{ceiling} - ESR_{floor}} + 1 \right\rfloor \tag{3.2}$$

It is non-trivial to select a proper parameter set $\{ESR_{floor}$, $ESR_{ceiling}$, $Resolution\}$ due to several reasons. First, the ESR values can vary greatly, especially considering the diversity of storage devices and workloads. Second, the blocks in the regular region can move in and out frequently. A proper parameter set for the current set of blocks may not be a sound choice at a later time. Third, *resolution* determines how much difference between ESR values we can distinguish. The smaller *resolution* is, the less difference we can tell. As an extreme case, setting *resolution* 1 would disregard the ESR values for replacement, and the caching policy would behave similar to the CLOCK algorithm. Thus, the normalized ESR value, nESR, should represent the *relative* difference between the blocks' ESR values. Ideally, we should 'zoom in' when the difference between ESR values is small, and 'zoom out' if ESR values are distributed sparsely.

3.4.1 ESR_{floor} and $ESR_{ceiling}$

In order to determine ESR_{floor} and $ESR_{ceiling}$, we track the number of blocks moved into the regular region. Once m blocks in the regular list change, we re-evaluate the two parameters. Simply using the minimum and the maximum of ESR values can be suboptimal, because a small portion of blocks may carry ESR values that deviate greatly from the other blocks. Alternatively, we calculate a cumulative distribution function (CDF) of ESR values. ESR_{floor} is the maximum of the s smallest ESR values, and $ESR_{ceiling}$ is set as the minimum of the s largest ESR values. In experiments, we set m and s as 10% and 5% of the number of blocks in the regular list, which works well in practice. Since ESR values can be collected when the regular list is scanned for searching a victim block, no extra scanning is needed.

3.4.2 *Resolution*

The parameter *resolution* determines how likely a caching policy is to be energy-aware. A high *resolution* makes caching more 'energy-saving oriented', as a slight difference between ESR values can be distinguished. Thus, if we can yield more benefits by setting the caching policy more energy-saving oriented, *resolution* should be tuned up, otherwise, *resolution* should be tuned down.

To estimate the effectiveness of tuning *resolution*, we set an additional buffer, called *victim buffer*, to record the meta-data of blocks evicted from the regular region and its size is set the same as the regular region. An energy saving counter ES_{victim} is associated to the victim buffer and initialized to zero. We also associate another counter $ES_{regular}$ to the regular region. Once a memory miss occurs, we check whether the on-demand block is resident in the victim buffer. If true, ES_{victim} is incremented by the block's ESR value. If a block is hit in the regular region, $ES_{regular}$ is updated in the same way. Periodically, we compare the two counters. If $ES_{regular}$ is larger than ES_{victim}, which means the last tuning is beneficial, we tune *resolution* linearly to the same direction as we did last time. If $ES_{regular}$ is smaller than ES_{victim}, which means *resolution* is over-tuned last time, we tune it exponentially back to the opposite direction.

3.5 Adapt to Run-time Dynamics

In practice, many run-time dynamics, such as cold misses, may prevent us from powering down a device. If that happens, a busy period cannot be converted to a quiet period as expected. In order to adapt to such situations, we associate each storage device with a *scale*, a floating point value between 0 and 1, to describe how much likely we can achieve the expected optimal case. In particular, a small *scale* value means that we are unlikely to achieve the optimal case. The *scale* value is determined as follows.

In the optimal case, a busy period can be avoided, so the lowest energy consumed by a storage device is its energy consumption in the low power mode, denoted as $Energy_{opt}$. In practice, the achievable lowest energy consumption is the energy consumed when all available memory space is allocated to hold blocks from the device exclusively, denoted as $Energy_{pseudo-opt}$. We also denote the energy actually consumed by the storage device as $Energy_{real}$. With the three values, *scale* can be calculated by using Equation (3.3).

$$scale = \frac{Energy_{real} - Energy_{pseudo-opt}}{Energy_{real} - Energy_{opt}} \tag{3.3}$$

In this equation, $Energy_{opt}$ and $Energy_{real}$ can be calculated based on the storage device specification data and the observed data access sequence. $Energy_{pseudo-opt}$ can be estimated by assuming all the memory space is allocated to hold blocks from the storage device exclusively. Periodically, *scale* for each device is updated to reflect the most recent change, and we use $scale \times ESR$ rather than the original ESR value for replacement. As each block's ESR value does not need to be scaled until it is swept over by the clock hand, scaling ESR values would incur little run-time overhead.

4. EVALUATION

4.1 Simulation

We conducted a trace-driven simulation to evaluate the effectiveness of the PS-BC framework. The simulator, called *iosim*, emulates the policies used in the Linux system, including the 2Q-like memory page replacement algorithm, the two-window based readahead policy, the I/O request clustering mechanism, and the periodic write-back scheme, etc. We also simulated the policies of the Linux laptop mode [2] as a baseline case to show the energy saving achieved by the existing power management mechanism.

In our simulation, the write-back interval is set 10 minutes, the sync delay is set 5 seconds, the highest ratio of dirty blocks is set 40%, and the lowest ratio of dirty blocks is set 5%. The disk spin-down timeout is set 20 seconds.

4.2 Storage Devices

We simulated five different types of storage devices, including a flash drive, a wireless interface card, a laptop disk, a high performance SCSI disk, and a flash disk.

The simulated flash drive is a Transcend TS1GJF2A flash drive with a 1GB capacity [8]. Its read and write bandwidths are 12MB/sec and 8MB/sec, respectively. Its maximum active power is 0.37W, and its sleep power is 0.6mW.

The simulated wireless card is a Cisco Aironet 350 with a bandwidth of 11Mbps [4]. It has two power modes, PSM mode and CAM mode. If being idle for more than 800 msec, the wireless card switches to the PSM mode. If more than one packet is received, it switches back to the CAM mode. Its power consumption parameters are shown in Table 1.

Mode	Power Consumption
PSM(idle/recv/send)	0.39 W /1.42 W /2.48 W
CAM(idle/recv/send)	1.41 W /2.61 W /3.69 W
CAM to PSM(Delay/Energy)	0.41sec/0.53J
PSM to CAM(Delay/Energy)	0.40sec/0.51J

Table 1: The Cisco Aironet 350 wireless card [4].

Two types of hard disks are simulated, a laptop disk and a high performance disk. The simulated laptop disk is a Hitachi DK23DA laptop disk which has a 30GB capacity, 4200 RPM and 35MB/sec bandwidth [9]. The simulated high performance disk is an IBM Ultrastar 36Z15 SCSI disk which has a 18.4GB capacity, 15000 RPM and 53MB/sec bandwidth [9]. Their power consumption parameters are listed in Table 2. The spin-down timeout for both disks is set 20 seconds, the default value for Linux laptop mode.

Power mode	Hitachi DK23DA	IBM Ultrastar
Active Power	2.00 W	13.5 W
Idle Power	1.60 W	10.2 W
Standby Power	0.15 W	2.5 W
Spin up	1.6 sec/5.00 J	10.9 sec/135 J
Spin down	2.30 sec/2.94 J	1.5 sec/13 J

Table 2: The hard disk power parameters [9].

The simulated flash disk is an Adtron A35FB flash disk with a capacity of 128GB [1]. Its read and write bandwidths are 65MB/sec and 55MB/sec, respectively. Its active power is 2.0W and idle power is 1.75W.

4.3 Traces

In our simulation, we designed two typical computing scenarios to show how PS-BC behaves under different environments. The first experiment simulates a mobile computing scenario, in which a laptop user is programming, searching codes, and listening to music. The base system is an IBM T20 laptop, which has a power of 15.8 Watts [4]. Three applications, *make*, *grep*, and *xmms*, are traced by using our modified *strace* utility [8], which can intercept I/O related system calls, such as read() and write(). The details of three applications are shown in Table 3.

Name	Description	File #	Size(MB)
make	a Linux kernel compiler	2579	72.5
grep	a text search tool	1332	50.4
xmms	a mp3 player	116	47.9

Table 3: Trace description of applications

This workload is synthesized by concatenating the collected traces of the aforesaid applications. As shown in Figure 4(a), it includes seven stages, each of which is a replay of one or multiple traces on different storage devices, individually or simultaneously. Three storage devices are used in this case, including a laptop disk, a flash drive, and a wireless network card. In this workload, a laptop user first compiled a Linux kernel binary (*make*) on the disk and on the flash drive. Later, the user re-compiled the Linux kernel binary on the disk again, and connected to a remote storage server via the wireless card and searched in a set of Linux source code files (*grep*) stored on the remote server, then the user made another compilation on the disk again. Finally the user opened the *xmms* media player to enjoy the music while compiling the codes on the remote storage.

The second experiment is designed to evaluate PS-BC on an enterprise server running real-world workloads. It simulates a Sun Fire X4100 server, which has a power of 218 Watts (the aggregate power consumed by all components, including CPU and memory, except storage devices) [3]. It is equipped with two storage devices, a 15,000RPM IBM Ultrastar 36Z15 disk and an Adtron flash disk. For servers equipped with more hard disk drives, we expect to see more significant gains on energy saving, since energy consumption would increase with the number of attached hard disks. This server provides web searching service and NFS file service. The search engine I/O trace [5] is collected from a popular search engine and it features intensive random data accesses due to high access concurrency. The NFS trace [10] is collected in a typical office/research environment, which shows a strong locality. As done in practice, we dedicate the flash disk to host the search engine workload to speedup random data accesses. The NFS workload is held on the IBM disk.

4.4 Case 1: Mobile Computing

This workload is designed as a micro-benchmark to specifically show how PS-BC behaves to achieve the design goals. In addition to PS-BC, we also simulate the 2Q-based memory replacement algorithm adopted in current Linux kernel, and GreedyDual [7], which is a cost-aware caching algorithm. Respectively, they are referred to as *Linux* and *GreedyDual*. For a fair comparison, we use the ESR value, which is used in PS-BC, to label the cost of each block in GreedyDual. We set the memory size 64MB to trigger the buffer cache replacement by creating a high memory pressure. In practice, available memory size is often larger, but the demand for memory space is much higher too. Also note that the memory used by the OS is not included here.

	Linux	GreedyDual	PS-BC
Total Energy(J)	20148.2	17598.0	15366.7
System Energy(J)	9508.7	8999.8	7036.4
Storage Energy(J)	10639.5	8598.1	8330.3
Disk Energy(J)	6786.6	4873.5	5124.5
Flash Energy(J)	6.6	12.5	6.6
WNIC Energy(J)	3846.2	3712.1	3199.0

Table 4: Case 1 *mobile computing*: Energy consumptions of *Linux*, *GreedyDual*, and *PS-BC*. The system energy consumption is measured during I/O time.

As Table 4 shows, *PS-BC* consumes 15366.7J energy in total, which is 23.7% less than that of *Linux* and 12.6% than that of *GreedyDual*. *PS-BC* optimizes the access stream to the laptop disk and wireless network card, which are both energy-inefficient devices. More specifically, compared to *Linux*, *PS-BC* consumes 24.4% less hard disk energy and 16.8% less wireless card energy. Although *PS-BC* consumes a little more (5.1%) disk energy than *GreedyDual*, it is paid back by consuming 13.8% less energy for the wireless card. Since the number of memory misses is reduced, not only the system performance is improved due to lower I/O latency, but the energy consumed by the base system during I/O time is also reduced by 26%, from 9508.7J to 7036.4J.

In order to clearly show how energy saving is achieved by *PS-BC*, we plot the storage activities of all three algorithms in Figure 4. At the first stage, all three algorithms have to activate the hard disk to load data. At stage 2, building the Linux kernel binary on the flash drive leads to cold misses for all three policies. At this stage, *Linux* behaves differently from the other two energy-aware algorithms. *Linux* replaces the blocks loaded from the disk, since the blocks loaded from the flash drive are more recently used and have better temporal locality. Consequently, at stage 3, when the disk blocks are accessed again, *Linux* has to spin up the high power disk to service requests, while *GreedyDual* and *PS-BC* can keep the disk at the standby mode. The green spikes shown on Figure 4(c) and 4(d) are delayed write-back of dirty blocks. At stage 4, *grep*, which scans the Linux source code, accesses the remote storage via the wireless card. Though power consumption of the wireless card is comparable to the laptop disk, the access pattern of *grep* is completely different from that of *make*. *Grep* references a large amount of data in a few seconds, which makes it much more bursty than *make*. Both *PS-BC* and *GreedyDual* recognize this difference and protect the disk blocks, while *Linux* ignores this fact and evicts the disk blocks out of memory. As a result, at stage 5, substantial energy is saved by *PS-BC* and *GreedyDual*. At stage 6, we activate the ESR scaling to show how *PS-BC* adapts to the runtime dynamics. At this stage, the user begins to use *xmms* to listen to music, which accesses the hard disk constantly and makes the disk unable to be spun down. Simultaneously, the user is building a Linux kernel binary on the remote storage. As *xmms* competes with *make* for memory space, *Linux* cannot keep the whole working-set of *make* in memory. *GreedyDual* relies solely on the static ESR values and cannot dynamically adapt to the change of device status. Thus *GreedyDual* still offers disk blocks a high priority to stay in memory, although *xmms* keeps the disk active and no further energy saving can be achieved. In contrast, *PS-BC* quickly recognizes that the disk cannot be spun down, and thus it responds by biasing the disk blocks and evicting them from memory. Thus, at the last stage, PS-BC avoids most accesses to the remote storage, while GreedyDual and Linux have to activate the wireless card, which raises substantial energy consumption.

4.5 Case 2: High Performance Server

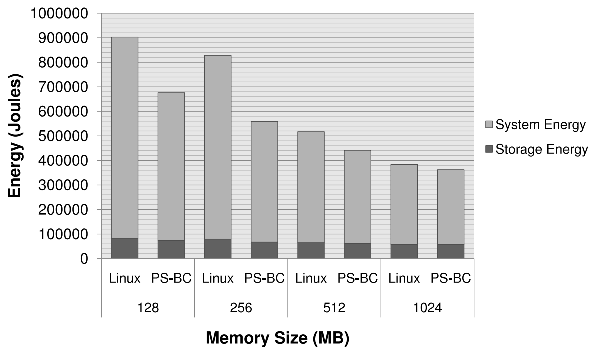


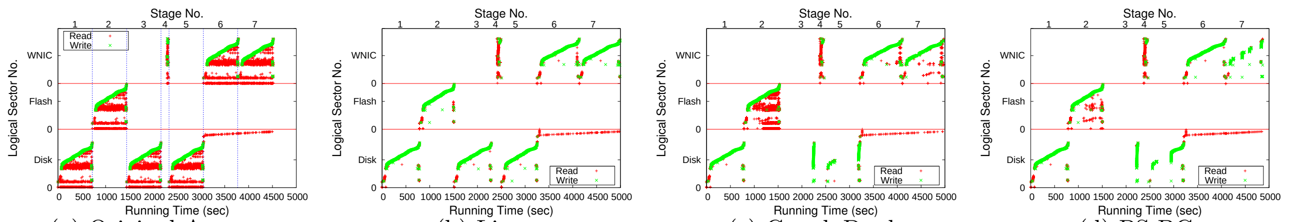
Figure 5: Case 2 *server*: The storage and system energy consumptions of *Linux* and *PS-BC*. The system energy consumption is measured during I/O time.

The second workload is designed to show how *PS-BC* works on enterprise servers running real-world workloads. In the previous section we have shown how *PS-BC* benefits from the scalable ESR values, compared to *GreedyDual*, which uses static ESR values. In this section, we will focus on comparing *Linux* and *PS-BC* with more details.

	Linux	PS-BC
Flash Hit Ratio	0.17%	0.02%
Flash I/O latency(sec)	137.8	138.0
Flash Energy(Joules)	10066.1	8867.8
Disk Hit Ratio	75.7%	83.2%
Disk I/O latency(sec)	3296.4	2112.9
Disk Energy(Joules)	69462.4	58670.1

Table 5: Case 2 *server*: Experimental results of *Linux* and *PS-BC* (256MB Memory).

Figure 5 shows the energy consumptions of the storage devices and the base system for *Linux* and *PS-BC* with various amount of memory. The base system energy consumption is measured during the I/O time. On a system with 128MB memory, Linux consumes 902938.9J energy, while PS-BC



(a) Original Accesses (b) Linux (c) GreedyDual (d) PS-BC

Figure 4: Case 1 *Mobile computing*: Data accesses in original applications, *Linux*, *GreedyDual*, and *PS-BC* schemes. All experiments are configured with memory size of 64MB.

consumes only 676127.6J energy, which is an energy saving of 25.1%. The most significant energy saving is observed on a system with 256MB memory. *PS-BC* consumes 558271.7J energy, which is 32.5% less than that of *Linux* (828222.4J). More specifically, *PS-BC* reduces the energy consumption of storage devices from 79528.6J to 67537.9J, which is a 15% energy saving. More significant energy saving is achieved for the base system — the base system of *Linux* consumes 748693.7J, which is nearly 52.5% more than that of PS-BC (490733.7J). In this workload, the base system, SUN Fire X4100 server, has a power of as high as 218 Watts, which means reducing I/O latency can save a substantial amount of energy. Since *PS-BC* takes the base system energy consumption as an important factor in the calculation of ESR values, it attempts to not only optimize the energy consumption of storage devices but also reduce the I/O latency, which in turn reduces the base system energy consumption. In Table 5, we can see that *PS-BC* apparently biases the blocks loaded from the flash disk. The accesses to the flash disk data has a hit ratio of only 0.02% for *PS-BC*, while *Linux* has a much higher hit ratio (0.17%). Unfortunately, the higher hit ratio only leads to negligible reduction of I/O latency, 0.2 seconds. In contrast, *PS-BC* manages memory blocks in a more efficient way. It improves the hit ratio of disk blocks from 75.7% to 83.2%, which not only leads to a significant reduction of I/O latency but also a substantial energy saving for both the power-hungry hard disk and the base system. With additional memory, energy consumption of both *Linux* and *PS-BC* is reduced, but PS-BC still achieves 14.6% and 5.5% energy saving with 512MB memory and 1024MB memory, respectively.

5. RELATED WORK

There are some recent research efforts on energy-aware caching policies for power saving in storage systems, including PA-LRU [17], PB-LRU [18], CBSM [6], and C-Burst [9]. PS-BC differs from these schemes in three aspects and effectively addresses their limits. First, PS-BC evaluates energy-saving properties at a fine-grained level of blocks, thus energy consumption is optimized not only for different storage devices but also for each individual storage. Second, PS-BC carefully takes the base system energy consumption as an important factor in the design consideration for achieving system-wide energy conservation, which has not been considered in most previous studies. Third, PS-BC is designed as a general caching policy without any requirement on special hardware, such as the speed-adjustable disks [12].

6. CONCLUSION

In this paper, we present an effective system framework for energy-aware buffer cache management, called PS-BC. PS-BC carefully considers several critical factors affecting energy consumption efficiency for accessing storage devices. It achieves the energy saving purpose by holding the most valuable blocks for energy saving in memory and adapts to the dynamic storage status. Our experimental studies show that PS-BC can achieve an energy saving rate of up to 32.5% for both mobile computing workloads and real-world server workloads with a minimal overhead.

7. ACKNOWLEDGMENTS

We thank anonymous reviewers for their constructive comments. This work has been partially supported by the National Science Foundation under grant CCF-072380.

8. REFERENCES

[1] http://www.adtron.com/pdf/A35FB-spec061206.pdf.
[2] Linux laptop mode document. http://lxr.linux.no/source/Documentation/laptop-mode.txt.
[3] Sun. http://www.sun.com/servers/entry/x4100/calc/.
[4] M. Anand, E. Nightingale, and J. Flinn. Self-tuning wireless network power management. In *Proc. of Mobicom'03*, 2003.
[5] K. Bates, B. McNutt, and SPC. http://traces.cs.umass.edu/index.php/Storage/Storage.
[6] L. Cai and Y. Lu. Power reduction of multiple disks using dynamic cache resizing and speed control. In *Proc. of ISLPED'06*, 2006.
[7] P. Cao and S. Irani. Cost-aware WWW proxy caching algorithms. In *Proc. of USENIX'97*, Dec. 1997.
[8] F. Chen, S. Jiang, and X. Zhang. SmartSaver: Turning flash drive into a disk energy saver for mobile computers. In *Proc. of ISLPED'06*, 2006.
[9] F. Chen and X. Zhang. Caching for bursts (C-Burst): Let hard disks sleep well and work energetically. In *Proc. of ISLPED'08*, 2008.
[10] D. Ellard, J.Ledlie, P. Malkani, and M. Seltzer. Passive NFS tracing of email and research workloads. In *Proc. of FAST03*, 2003.
[11] F.J.Corbato. A paging experiment with the multics system. In *MIT Project MAC Report MAC-M-384*, May 1968.
[12] 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.
[13] S. Jiang, F. Chen, and X. Zhang. CLOCK-Pro: An effective improvement of the CLOCK replacement. In *Proc. of USENIX'05*, April 2005.
[14] S. Jiang, X. Ding, F. Chen, E. Tan, and X. Zhang. DULO: An effective buffer cache management scheme to exploit both temporal and spatial localities. In *Proc. of FAST'05*, 2005.
[15] S. Jiang and X. Zhang. LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance. In *Proc. of SIGMETRICS '02*, June 2002.
[16] P. Sehgal, V. Tarasov, and E. Zadok. Evaluating performance and energy in file system server workloads. In *Proc. of FAST'10*, 2010.
[17] Q. Zhu, F. M. David, C. F. Devaraj, Z. Li, Y. Zhou, and P. Cao. Reducing energy consumption of disk storage using power-aware cache management. In *Proc. of HPCA'04*, 2004.
[18] Q. Zhu, A. Shankar, and Y. Zhou. PB-LRU: A self-tuning power aware storage cache replacement algorithm for conserving disk energy. In *Proc. of ICS'04*, 2004.