Joint Power Management of Memory and Disk

Le Cai, Yung-Hsiang Lu

Joint Power Management of Memory and Disk *

Le Cai and Yung-Hsiang Lu
School of Electrical and Computer Engineering, Purdue University
{lc, yunglu}@purdue.edu

Abstract

This paper presents a scheme to combine memory and power management for achieving better energy reduction. Our method periodically adjusts the size of physical memory and the timeout value to shut down a hard disk for reducing the average power consumption. We use Pareto distributions to model the distributions of idle time. The parameters of the distributions are adjusted at run-time for calculating the corresponding timeout value of the disk power management. The memory size is changed based on the inclusion property to predict the number of disk accesses at different memory sizes. Experimental results show more than 50% *energy savings compared to a 2-competitive fixed-timeout method.*

1. Introduction

Power reduction has become a major goal in system designs. Among all components, the storage hierarchy (cache, memory, and disks) consumes a significant percentage of power. In a data center, storage devices can consume over 25% power [21]. Many studies have been conducted to reduce memory's power consumption [6, 8] or disks' power consumption [4, 21]. However, the close relationship between memory management and power management has not been fully investigated. We use an example to illustrate the importance of combining memory management and power management.

Adding memory often improves performance by reducing the numbers of disk accesses. However, when power management is adopted, more memory may actually degrade performance and increase power consumption. For example, when a system's physical memory increases, more data reside in memory so the disk becomes idle and enters a low-power ("sleep") state more often. If the memory is not large enough, the system accesses the disk soon. Since it has to wake up on-demand, the disk saves insufficient energy to compensate the spin-up energy and the system actually consumes more energy. The minimum spin-down time to save energy is called *break-even time* [9]. Moreover, the performance degrades due to the disk's wakeup delay—usually several seconds or billions of cycles. This example shows that adding more memory may increase the overall power consumption and degrade performance even though disk accesses occur less frequently. If we further increase the memory size, the disk can remain spinned-down longer to save more energy but the savings may not compensate the power consumed by the additional memory. It is essential to combine memory management and power management to reduce power consumption and to improve performance. This joint management method is different from previous adaptive disk power management methods [2, 10] because they respond to the changes of disk accesses without actively adjusting memory sizes.

This paper presents a technique using operating systems to perform joint power and memory management for physical memory and hard disks. We assume that part of memory can be turned on and off, as suggested in [6, 8], to reduce power and to improve performance. We change the memory size by allocating or releasing reserved memory pages in operating systems since reserved pages are unavailable to application programs. We adopt the timeout policy to manage the disks for three reasons: (a) Timeout is widely used and outperforms many policies [9]. (b) It assists some other policies, such as time-index stochastic optimization [17]. (c) It is used as

* This project is supported in part by Purdue Research Foundation and National Science Foundation Career CNS 0347466.

a "backup" scheme when a prediction policy has insufficient history to predict [4]. The objective of this study is to determine two parameters at run-time: the timeout value for shutting down the disk and the size of the physical memory. Our method considers the interactions between the memory and the disk to minimize the energy consumption.

Two major challenges exist in the joint management: (a) determining disk IO (the number and the interarrival time) at different sizes of memory without running the same programs multiple times and (b) adjusting the timeout values without offline analysis. We use the concept of the ghost buffer [11, 13] to predict the number of disk accesses at different memory sizes. The ghost buffer records replaced memory pages as if they are stored in additional physical memory. With the recorded information, the ghost buffer can determine how many disk accesses can be eliminated by adding certain amount of memory. We model the distributions of interarrival time of disk accesses as Pareto distributions. This is because previous studies [16, 18] and our analysis of HP disk traces [15] show Pareto distributions can closely model disk IO. We evaluate the joint management using a web server and use SPECweb99 as the benchmark. Our experiment results show that the joint method can save more than 50% energy compared with a 2-competitive fixed-timeout method.

2. Background

2.1. Power-Aware Storage Hierarchy

Hu et al. [6] turn off individual cache lines whose data are unlikely to be accessed again. Their method is 2-competitive by turning off a cache line 10000 cycles after the last access. Lebech et al. [8] use multiple power modes of RDRAM and dynamically turn off memory chips with power-aware page allocation in operating systems. These studies suggest that future memory should support power management to turn on/off part of the memory. The energy to turn on and off memory chips is negligible compared with the memory's static energy. For example, a 64MB RDRAM chip consumes less than 1μJ energy to change from a power-down state to an active state [8]. Zhu et al. [21] show that lower miss rates do not necessarily save disk energy. A power model of hard disk is presented by Zedlewski et al. [19]. Papathanasiou et al. [12] use prefetching to prolong the idle time of hard disks so that they may stay off to save power. Two recent studies calculate the sizes of prefetching buffers to optimize power savings [1, 14]; both studies assume streaming data that are not reused. Gurumurthi et al. [5] propose adjusting the rotation speed of disk platters to reduce power consumption. Their approach is similar to frequency scaling in processors. Timeout is often used to predict long idle periods. Simunic et al. [16] present a time-indexed Markov chain stochastic model to optimize power management policies based on Pareto distribution. Douglis et al. [2] dynamically adapt the disk timeout to save energy while meeting the performance requirements. Lu et al. [10] divide disk accesses into different sessions; the method dynamically changes the timeout based on the predicted duration of each session. These two methods respond to the changes of disk IO patterns. In contrast, our method proactively changes disk IO by adjusting the size of physical memory.

2.2. Memory Size and Disk IO

Changing memory size can affect disk IO and power consumption because more memory may reduce disk accesses and allow the disk to sleep longer. One approach to obtain these relationships is running the same program multiple times with different memory sizes. This is time-consuming and impossible if the programs need run-time inputs. Franklin et al. [3] use Markov chains to model the probabilities of page faults with different memory sizes. Their method needs all memory references in advance so it only applies to offline analysis. Mattson et al. [11] use the *inclusion property* of many memory replacement algorithms: the content of smaller-size memory is a subset of the content of larger memory for the same memory access sequence. We use the least recently used (LRU) replacement algorithm as an example. An LRU list records the most recently accessed memory pages. If there are q pages, the LRU list stores q most recently accessed pages. If there are s pages and $s < q$, the list of s pages is a subset of the list of q pages. Any reference to the s most recently accessed pages hits in memory when the memory size is either s or q pages. When the memory size is s pages, the references to the $(s+1)^{th}$ to q^{th} most recently accessed pages are misses. These accesses hit in memory when the size is q page. If we count the references to the $(s+1)^{th}$ to q^{th} most recently accessed pages, we can obtain the relationship of memory sizes and the number of disk accesses. Specifically, we know the additional number

of disk accesses when the memory size changes from q pages to s pages. This method can apply to determining the effect of shrinking the physical memory. The same method is used to online predict page miss ratio [20].

To determine the effect of enlarging the physical memory, a "ghost buffer" [13] is used to record replaced pages. All pages in the ghost buffer form a doubly-linked list and they are ordered based on the time when they are replaced. Different from the LRU list, the ghost buffer stores only the tags, such as their inode IDs and offsets in the file; no actual data are stored. Using the tags, we can uniquely identify a page. When disk accesses happen, the tags of the required pages are compared with the tags stored in the ghost buffer. A match means that an earlier replaced page is accessed again. This memory access would not cause disk IO if the memory size is larger and the page was not replaced. If the matched page is the i^{th} page from the head of the ghost buffer, i pages memory is needed to keep this page in memory. Hence, by recording the position of the matched pages and the number of matches, this algorithm can determine the number of reduced disk accesses when adding memory pages.

3. Joint Management

The joint manager periodically determines the appropriate size of physical memory and the timeout value to spin-down the disk. We use T to represent the period length. Every period we record the number of disk accesses. Using this number and the information from the LRU list and the ghost buffer, our method predicts the disk accesses and the length distribution of the idle time during next period for the chosen memory size. The proper timeout and memory size are chosen based on the their relationships with the average power consumption. Table 1 lists the symbols used in our method.

3.1. Idle Time of Disks

Since disk accesses often occur in bursts, many idle periods are very short and provide no opportunity for power management. We remove these short idle periods by using a sliding window. If one disk access is followed by another and the idle time is smaller than the window length, this idle period is ignored. We use 0.1s as the window length since the break-even time of most disks is much larger than this value. Let ℓ be a random variable representing the lengths of the disk's idle time. We model the probability by Pareto distributions:

symbol	description
N	observed number of disk accesses
n	predicted number of disk accesses
T	period (s)
t_o	disk timeout (s)
t_{be}	break-even time of disk (s)
t_s	off time each T (s)
n_s	times to turn off disk each T
m	memory size (page)
e_d	disk dynamic energy (J/page)
p_d	disk static power (W)
p_m	static power of memory (W/page)

Table 1. Symbols and their meanings.

$$f(\ell) = \frac{\alpha\beta^\alpha}{\ell^{\alpha+1}}, \ell > \beta, \alpha > 1 \quad (1)$$

Here, β is the length of the sliding window so all idle periods are longer than β. The value of α determines the distribution of ℓ. When α is small, it is more likely to have long idle periods. At run time, we can calculate the average idle time to determine α's value. Let $E(\ell)$ represent the mean of ℓ: $E(\ell) = \frac{\alpha\beta}{\alpha-1}$. The value of α is $\frac{E(\ell)}{E(\ell)-\beta}$. We estimate $E(\ell)$ by the average length of idle time. For duration T, the average idle time is the ratio of T and the number of accesses: $E(\ell) = \frac{T}{n}$. Here, n is the predicted number of disk accesses each T. We use the predicted value when the memory size changes (to be explained later). Hence, we obtain

$$\alpha = \frac{T}{T - \beta n} \quad (2)$$

This paper considers only two power modes: on and off, but the method can be extended to more modes. To obtain the energy consumption of the disk, we compute the disk's off time from (1). All idle periods longer than the timeout t_o will trigger the disk to be turned off. The off time is the difference between the idle time and t_o, i.e. $\ell - t_o$. For the idle periods shorter than t_o, there is no off time. The probability of an idle period longer than t_o is $\int_{t_o}^{\infty} f(\ell)d\ell$. The average off time of each idle period is $\int_{t_o}^{\infty}(\ell - t_o)f(\ell)d\ell$ and there are n idle periods:

$$t_s = n\int_{t_o}^{\infty}(\ell - t_o)f(\ell)d\ell = n(\frac{\beta}{t_o})^{(\alpha-1)}\frac{\beta}{\alpha-1} \quad (3)$$

If β is smaller than t_o ($\frac{\beta}{t_o} < 1$), the off time is longer with a smaller α. From (1), we also compute the average number of the times to turn off disk each T. This number is used to compute the energy consumed to turn on and off disk.

$$n_s = n \int_{t_o}^{\infty} f(\ell) d\ell = n(\frac{\beta}{t_o})^{\alpha} \tag{4}$$

A smaller α means a larger n_s because long idle periods are more likely. In each T the memory and the disk consume: (a) static energy for the disk and m pages of memory: $p_m mT + p_d(T - t_s)$, (b) dynamic energy: $e_d n$, and (c) turn-on energy: $p_d t_{be} n_s$. Each time turning on and off disk consumes $p_d t_{be}$ based on the definition of break-even time. We ignore the energy consumed to turn on and off memory because this energy consumption is much smaller than the static energy consumption of memory. We add (a) (b) and (c) to obtain the total energy in T:

$$p_m mT + p_d(T - t_s) + e_d n + p_d t_{be} n_s \tag{5}$$

We use (3) and (4) to replace t_s and n_s in (5) to obtain the energy consumption represented by t_o:

$$p_m mT + p_d[T - \frac{n\beta^{\alpha}}{(\alpha - 1)t_o^{(\alpha-1)}}] + e_d n + p_d t_{be} n(\frac{\beta}{t_o})^{\alpha} \tag{6}$$

We calculate the derivative of (6) with respect to t_o and make it equal to zero.

$$\begin{array}{l} p_d n(\frac{\beta}{t_o})^{\alpha} - p_d n(\frac{\beta}{t_o})^{\alpha} \frac{\alpha t_{be}}{t_o} = 0 \\ \Rightarrow \quad t_o = \alpha t_{be} \end{array} \tag{7}$$

The second derivative is positive when $t_o = \alpha t_{be}$. Thus, we obtain the minimum power consumption when $t_o = \alpha t_{be}$. Since a larger α indicates more short idle periods, increasing timeout can reduce the probability to turn off the disk during short idle periods. The timeout value also increases when the break-even time becomes larger. Larger break-even time means more energy is consumed to turn on and off disk; the power manager should increase timeout to avoid turning off the disk.

3.2. Number of Disk Accesses

The joint manager decides the memory size and the corresponding number of disk accesses using the LRU list and the ghost buffer. The manager counts the number of disk accesses in each T and predicts the access number for the next T. Let n_i be the predicted number of accesses during $[iT, (i+1)T]$. We use a recursive formula to predict n_i.

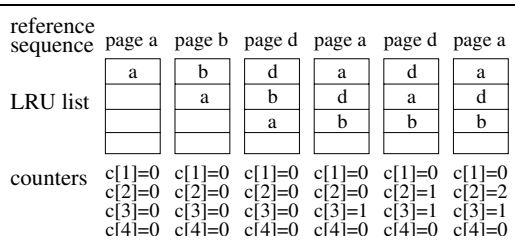

Figure 1. The reference count of LRU list.

$$n_i = kN_{i-1} + (1 - k)n_{i-1} \tag{8}$$

The values of N_{i-1} and n_{i-1} are the observed and predicted numbers of accesses in the previous period. We choose $k = 0.5$ to balance the recent number and the previous number since n_{i-1} is predicted from previous numbers. This is similar to the prediction used in [7]: exponentially reducing the effect of previous observations and predictions.

Formula (8) represents the number of disk accesses when the memory size remains constant. We use the information from the LRU list and the ghost buffer to obtain the values of n_i with different memory sizes. Figure 1 illustrates how to count the references to the LRU list for a system with four pages of physical memory. A counter array $c[.]$ is used to record the reference number to each page in the LRU list. When the referenced page is the i^{th} page from the head of the LRU list, the i^{th} counter $c[i]$ increases one. For example, the memory is accessed by a sequence of six references $\{a, b, d, a, d, a\}$. The first three references need disk accesses since pages $\{a, b, d\}$ are not in memory. After these three references, the values of all counters are zero and the LRU list is $\{d, b, a\}$. The fourth reference is for page a, the third page in the list. The counter $c[3]$ increments by one. The fifth access is d and it is the second page in the list; thus $c[2]$ increments by one. Finally, the sixth access is page a again and it is the second in the list; $c[2]$ increments by one. The final values of counters are $\{0, 2, 1, 0\}$. These values indicate that there is no access to the first and fourth page, two accesses to the second page, and one access to the third page. Since $c[4] = 0$, no additional disk access occurs if we reduce the memory to three pages. Because $c[3]$ is nonzero, reducing the memory to two pages will add one ($c[3] = 1$) more disk ac-

symbol	value	symbol	value
β	0.1s	p_m	5×10^{-5}W/page
T	60s	e_d	0.42J/page
t_{be}	7.3s	p_d	3.99W

Table 2. Parameters in experiments.

cess and require totally four accesses. The value of each counter indicates the additional disk accesses when the corresponding page is removed. The ghost buffer has the similar function except that the ghost buffer records the numbers of the accesses to the replaced pages. Thus, the ghost buffer can predict how many disk accesses can be removed by enlarging the memory size. We enumerate possible memory sizes with 1MB as the unit and compare the energy consumption. This enumeration needs little time because the memory size is generally within several thousand MBs and it is fast to compute the energy consumption from (6).

4. Experiment

4.1. Experiment Configuration

We implement our method in Linux 2.4.20. Each memory page has one entry in both the LRU list and the ghost buffer. Each entry uses 8 bytes for two integer variables recording the inode ID and the offset in file and 16 bytes for four pointers storing previous and next entries in two double-linked lists. Two integer counter arrays are used to count the references to the entries in the LRU list and the ghost buffer. We use the LRU list in Linux and only add the ghost buffer and two counter arrays. When the physical memory is 1GB, the ghost buffer and each counter array have 0.25M entries for 4KB pages. The overall memory usage is $0.25*(8+16+4+4) = 8$MB. An Apache web server and SPECweb99 benchmarks are used in the experiments. The numbers of simultaneous connections generate different amounts of workload. We conduct experiments under different connection numbers and each experiment lasts for one hour. One minute is chosen for T since it performs well in our experiments. Because of the space limitation, we will not discuss the detail of determining T. Table 2 lists the parameters in the experiments. The main memory is Micron MT48LC1M16A1S DRAM and we obtain its static power from the specification. We divide this power by the number of the memory pages to estimate the value of p_m. We use p_m to estimate the power consumption of different sizes of physical memory since no existing commercial memory chips allow run-time resizing. Therefore, the system actually has a constant size of physical memory. Our modified operating system restricts the available size to application programs in order to create the effects of varying memory sizes. In the experiments, the hard disk switches between the idle and the standby modes. The disk's off status refers to standby and p_d is the power difference between the idle and the standby states. We use a Seagate ST340810A hard disk and it consumes 5.13W and 1.14W power in the idle and standby modes, respectively; the value of p_d is $5.13 - 1.14 = 3.99$W. To obtain the dynamic energy of the hard disk (e_d) for accessing one page, we measure the disk's energy consumption under different workloads. After subtracting the static energy, we divide the difference by the number of accessed pages. The quotient is used as the disk's dynamic energy per page. The value of e_d varies with the rate of disk accesses. Our experiments use the average value. We measure the energy consumed by the hard disk when it is switched from idle to standby and back to idle. This energy is divided by p_d to calculate t_{be}. The formulas derived in Section 3 are used to compute the energy consumption of the memory and the disk. Our method is compared with the 2-competitive timeout methods using t_{be} as the timeout value. These timeout methods use the physical memory of the size from 64MB to 1024MB.

4.2. Energy Consumption

Figure 2 shows the power consumption of the joint management and the fixed-timeout methods with different memory sizes. When the memory size is 1 GB, the system consumes the most power and we use its power as the base (100%) for comparison. The workload varies from 10 to 70 client connections because the server starts denying requests when the number of connections exceeds 70. The figure shows that our method achieves the best power savings, from 48% to 62%. When the numbers of connections are below 30, the timeout method with 128 MB memory consumes comparable power with our management scheme. However, the power increases dramatically as the number of connections grows if the memory size remains 128 MB. This is because that the small memory size causes many disk accesses. In contrast, the power increases much slower with our scheme because it chooses appropri-

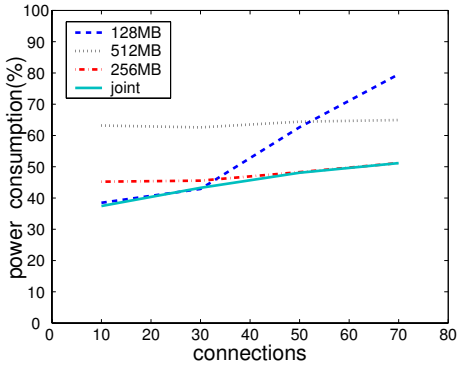

Figure 2. Power consumption of the joint method and fixed-timeout methods. The power consumption percentage is based on the timeout method with 1GB memory.

C	10	20	30	40	50	60	70
m	113	123	126	244	246	247	248
t_o^m	7.4	7.4	7.4	7.4	7.4	7.5	7.5
t_o^M	9.0	11.2	14.5	14.6	15.5	16.4	19.7

Table 3. Memory size and timeout used in the joint method. C:connections; m:average memory size(MB); t_o^m and t_o^M: minimum and maximum timeout(s).

ate memory sizes and timeout values as the workload varies. Table 3 lists the average memory sizes and the minimum and maximum timeout values used in the joint method. When the workload is light, the joint method uses a smaller size of memory to save power. When the workload becomes heavier, the joint method uses more memory to reduce disk IO and to reduce the disk's power consumption. For memory sizes above 256 MB, the power consumption remains almost constant. This is because large memory cannot completely eliminate all disk accesses. The workload varies even in an experiment with a constant number of client connections. The joint method adapts the timeout to the workload variation.

5. Conclusion

In this paper, we present a joint method for memory management and power management. This method uses the close relationship between memory management and disk IO to save power. Our method provides an analytic model to compute the proper disk timeout to minimize the energy consumption. We predict the number and the inter-arrival time of disk IO for different memory sizes without offline analysis. Our experimental results show that the joint method can save more power than timeout methods with fixed memory sizes.

References

[1] L. Cai and Y.-H. Lu. Dynamic Power Management Using Data Buffers. In *DATE*, pages 526–531, 2004.

[2] F. Douglis, P. Krishnan, and B. Bershad. Adaptive Disk Spin-down Policies for Mobile Computers. In *USENIX Symposium on Mobile and Location-Independent Computing*, pages 121–137, 1995.

[3] M. A. Franklin and R. K. Gupta. Computation of Page Fault Probability From Program Transition Diagram. *Communications of The ACM*, 17(4):186–191, April 1974.

[4] C. Gniady, Y. C. Hu, and Y.-H. Lu. Program Counter Based Techniques for Dynamic Power Management. In *HPCA*, pages 24–35, 2004.

[5] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. DRPM: Dynamic Speed Control for Power Management in Server Class Disks. In *Proc. Int. Symp. on Computer Architecture*, pages 169–181, 2003.

[6] Z. Hu, S. Kaxiras, and M. Martonosi. Let Caches Decay: Reducing Leakage Energy Via Exploitation of Cache Generational Behavior. *ACM Transactions on Computer Systems*, 20(2):161–190, May 2002.

[7] C.-H. Hwang and A. C.-H. Wu. A Predictive System Shutdown Method for Energy Saving of Event-driven Computation. *ACM Transactions on Design Automation of Electronic Systems*, 5(2):226–241, April 2000.

[8] A. R. Lebeck, X. Fan, H. Zeng, and C. Ellis. Power Aware Page Allocation. In *ASPLOS*, pages 105–116, 2000.

[9] Y.-H. Lu, E.-Y. Chung, T. Simunic, L. Benini, and G. D. Micheli. Quantitative Comparison of Power Management Algorithms. In *DATE*, pages 20–26, 2000.

[10] Y.-H. Lu and G. D. Micheli. Adaptive Hard Disk Power Management on Personal Computers. In *Great Lakes Symposium on VLSI*, pages 50–53, 1999.

[11] R. Mattson, J. Gecsei, D. Slutz, and I. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems Journal*, 12(2):78–117, 1970.

[12] A. E. Papathanasiou and M. L. Scott. Energy efficient prefetching and caching. In *USENIX Annual Technical Conference*, 2004.

[13] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. Informed Prefetching and Caching. In *ACM SOSP*, pages 79–95, 1995.

[14] N. Pettis, L. Cai, and Y.-H. Lu. Power Management by Prefetching Streaming Data. In *ISLPED*, pages 62–65, 2004.

[15] C. Ruemmler and J. Wilkes. UNIX Disk Access Patterns. In *USENIX Winter Conference*, pages 405–420, 1993.

[16] T. Simunic, L. Benini, P. Glynn, and G. D. Micheli. Dynamic Power Management for Portable Systems. In *MobiCom*, pages 11–19, 2000.

[17] T. Simunic, L. Benini, P. Glynn, and G. D. Micheli. Event-Driven Power Management. *IEEE Transactions on CAD*, 20(7):840–857, July 2001.

[18] W. Vogels. File System Usage in Windows NT 4.0. In *ACM SOSP*, pages 93–109, 1999.

[19] J. Zedlewski, S. Sobti, N. Garg, F. Zheng, A. Krishnamurthy, and R. Wang. Modeling Hard-Disk Power Consumption. In *Conference on File and Storage Technologies*, pages 217–230, 2003.

[20] P. Zhou, V. Pandey, J. Sundaresan, A. Raghuraman, Y. Zhou, and S. Kumar. Dynamic tracking of page miss ratio curve for memory management. In *ASPLOS*, pages 177–188, October 2004.

[21] Q. Zhu, F. M. David, C. Devaraj, Z. Li, Y. Zhou, and P. Cao. Reducing Energy Consumption of Disk Storage Using Power-Aware Cache Management. In *HPCA*, pages 118–129, 2004.