

A Dynamic Disk Drive Power Management Algorithm

Yang Yu Walter Burkhard
Gemini Storage Systems Laboratory
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0114, USA
{yyu,burkhard}@cs.ucsd.edu

Abstract

Disk drives consume a significant amount of energy in mobile computers. We address the problem of reducing disk energy usage in a mobile computer by adaptively spinning down the disk. We devise a computationally effective dynamic histogram algorithm that analyzes disk access patterns, evaluates the expected energy saving of various spin-down policies and selects a policy that maximizes the energy saving. Our simulations on real-world traces show our algorithm saves more disk energy than the best existing algorithm. It saves between 38% and 88% energy for various disk models and requires significantly less computational overhead than the best existing algorithm.

1 Introduction

Energy consumption is a major concern in mobile computer design due to portability and battery cost. High energy consumption also raises temperature and deteriorates computer reliability. A significant portion of the energy is used by hard disks [5]. In mobile computers, the workloads are often not I/O intensive. The bursty disk access pattern allows disks to be spun down during idle time to save energy.

Modern disk drives in mobile computers typically have several power modes: *active*, *idle*, *standby* and *sleep*, each respectively utilizing less power [1]. In the active mode, a disk is actively seeking, writing or reading with the platters are spinning at full speed. In the idle mode, the platters remain spinning but the read/write electronics are off and the disk head may be parked. In the standby mode, the platters are not rotating. The sleep mode shuts off all remaining electronics. Some disk varieties support additional modes [10]. Switching between levels requires some energy as well as time depending on the transition. The switch between active and idle is almost instantaneous with little energy expense; accordingly, the disk can immediately switch between the idle and active modes. We do not consider these two modes distinct following [6, 7, 12, 14]. In addition, most power management policies assume only two power states: one working state (the active/idle state) and one low power state (the standby state). We make the same assumption as well.

The standby mode consumes less power over time; however, to serve a request, a disk must return to the active mode which requires both time and energy. The transition to standby mode is referred to as *spin down* and the return to active mode *spin up*. A disk must remain in standby mode for a period of time, the *break-even* time, to compensate for the additional energy expense. The break-even time for a disk drives depends on its operational parameters.

Since the spin up process requires additional energy, transit immediately to the standby mode when the disk is idle may utilize more energy than saved if the next request arrives too soon. A good disk power management scheme must accurately predicate when to spin down the disk. The time interval between completion of the most recent request and the spin down is referred to as the

threshold. If the disk access intervals are drawn independently from an unknown, fixed distribution, an optimal threshold exists [11, 13, 14].

However, the assumption that the access intervals are independently drawn from a fixed distribution is unrealistic. Upon examination of real traces, we find the disk accesses are bursty. In other words, the distribution of the access intervals is time-dependent rather than fixed. To address this issue, we present a simple but very efficient algorithm called the *dynamic histogram algorithm* determining the threshold online. The algorithm dynamically selects a threshold based on historical disk access patterns, with greater emphasis on recent activities. The disk spins down if it is idle for a period longer than the threshold and spins up when a new request arrives. Our simulation results on disk access traces collected from HP workstations [18] show our algorithm saves more energy than all existing fixed threshold or adaptive threshold algorithms. It saves between 38.4% and 87.7% energy for various disk models and requires significantly less computational overhead than the best existing algorithm. Furthermore, we show the algorithm is straightforward to implement and efficient.

The remainder of the paper is organized as follows: Section 2 models the problem and describes the dynamic histogram algorithm. In section 3 we discuss the experimental methodology, including performance metrics. We also compare the dynamic histogram algorithm with existing power management algorithms. The dynamic histogram algorithm is further investigated in section 4. Section 5 presents related work. Finally, we conclude and present future work in section 6.

2 Dynamic Disk Power Management Algorithm

2.1 Problem Statement

We define the disk spin-down problem as follows: Given the past disk access pattern and the power characteristics of the disk, the algorithms determines when to spin down the disk to minimize energy usage. As mentioned before, the disk requires extra energy and time to spin down and then spin up. For a particular disk, there is a period during which it costs the same amount of energy to keep the disk in the idle mode or to spin the disk down, stay in the standby mode and then spin it up. We call this time the *break-even time*. Disk activities can be viewed as a sequence of disk accesses, each of which is represented by its access time. The time period between two successive accesses is called an *idle interval* in this paper.

Assume the disk consumes p_1 watts in the idle mode and p_2 watts in the standby mode. Also assume the spin up and spin down consumes $\mathcal{E}$ joules (watt-seconds) and takes s_1 seconds. The break-even time s , as defined above, satisfies the following constraints:

$$sp_1 = (s - s_1)p_2 + \mathcal{E} \quad (1)$$

and

$$s > s_1 \quad (2)$$

From equation 1 we have

$$s = (\mathcal{E} - s_1 p_2) / (p_1 - p_2) \quad (3)$$

which always satisfies restraint 2 given $\mathcal{E}/s_1 > p_1$, that is, the power during transition is naturally higher than that of the idle mode. Clearly, s is determined only by the power characteristics of the disk drive variety. To determine when to transition from the working state to the low-power state, the table algorithm employs the time-out method widely used in practice and previous studies [7, 12]. A timer is triggered when the disk becomes idle and after the threshold elapses, the disk transitions to standby mode.

2.2 The Oracle Algorithm

The goal of disk power management is to conserve energy by switching to standby mode whenever possible. If the access sequence can be known in advance, the power management policy can make optimal decisions. This is referred to as the optimal algorithm [7] or the Oracle algorithm [16]. We use the name Oracle algorithm because it is based on the unrealistic assumption that the access sequence is known in advance. The Oracle algorithm gives an lower-bound on the energy consumed for a given access sequence, assuming there is no prefetching or caching and the access sequence is not altered. The Oracle algorithm begins by calculating the break-even time of the disk. The algorithm spins down the disk immediately after serving the current request if the next idle interval is longer than the break-even time. If the next idle interval is shorter than the break-even time, the disk stays in the idle state. The amount of energy $O(t)$ used by the Oracle algorithm in an idle interval of length t is shown in the following equations:

$$\begin{aligned} O(t) &= \begin{cases} t \times p_1 & \text{if } t < s, \text{ no spin-down} \\ t - s_1 \times p_2 + \mathcal{E} & \text{if } t \geq s, \text{ immediate spin-down.} \end{cases} \\ &= \begin{cases} t \times p_1 & \text{if } t < s, \text{ no spin-down} \\ t \times p_2 + s \times (p_1 - p_2) & \text{if } t \geq s, \text{ immediate spin-down.} \end{cases} \end{aligned}$$

Other algorithms use more energy than the Oracle algorithm. The amount of additional energy used by an algorithm is called the *excess energy* as defined by Helmbold *et al.* [7].

2.3 The Table Algorithms

Given an access sequence, the Oracle algorithm consumes the least amount of energy. The Oracle algorithm does not exist in reality because it is impossible to predicate future arrivals perfectly. A more realistic assumption is the idle intervals are independently drawn from a fixed, known distribution [11]. We can determine a threshold that minimizes the expected energy consumed by the disk.

Suppose the probability that the idle interval is exactly t seconds is $I(t)$, then the complementary cumulative probability distribution function, that is, the probability that the idle interval is at least t seconds can be derived as $F(t) = \int_t^\infty I(t)dt$. It is easy to verify that $F(0) = 1$, $F(\infty) = 0$ and $dF(t)/dt = -I(t)$.

If the threshold is y seconds, and the idle interval is t seconds, then we use the following $E(y, t)$ joules energy where:

$$\begin{aligned} E(y, t) &= \begin{cases} t \times p_1 & \text{if } t < y, \text{ no spin-down} \\ (t - s_1 - y) \times p_2 + y \times p_1 + \mathcal{E} & \text{if } t \geq y, \text{ spin-down after } y \text{ seconds.} \end{cases} \\ &= \begin{cases} t \times p_1 & \text{if } t < y, \text{ no spin-down} \\ t \times p_2 + (y + s) \times (p_1 - p_2) & \text{if } t \geq y, \text{ spin-down after } y \text{ seconds.} \end{cases} \end{aligned}$$

Compared to the strategy that never spins down the disk, which always consumes $p_1 t$ energy, the energy saved $S(y, t)$ is:

$$S(y, t) = \begin{cases} 0 & \text{if } t < y, \text{ no spin-down} \\ (t - y - s) \times (p_1 - p_2) & \text{if } t \geq y \text{ spin-down after } y \text{ seconds.} \end{cases}$$

$S(y, t)$ depends on both the idle interval t and the threshold y . Considering the distribution of the idle interval t , the expected saved energy with a threshold y is a function of y only:

$$\begin{aligned}
E[S(y)] &= \int_y^\infty (p_1 - p_2)(t - y - s)I(t)dt \\
&= (p_1 - p_2)\left(\int_y^\infty tI(t)dt - (y + s) \int_y^\infty I(t)dt\right) \\
&= (p_1 - p_2)\left(\int_y^\infty F(t)dt - \int_y^\infty d(tF(t)) + (y + s) \int_y^\infty d(F(t))\right) \\
&= (p_1 - p_2)\left(\int_y^\infty F(t)dt + yF(y) - \lim_{t \rightarrow \infty} tF(t) - (y + s)F(y)\right) \\
&= (p_1 - p_2)\left(\int_y^\infty F(t)dt - sF(y) - \lim_{t \rightarrow \infty} tF(t)\right)
\end{aligned}$$

Since $\lim_{t \rightarrow \infty} tF(t)$ is independent of y , we must find y that maximizes $P(y) = \int_y^\infty F(t)dt - sF(y)$.

Based on the assumption that the idle intervals are independently drawn from a fixed distribution, we can estimate the distribution using a simple but effective histogram table technique [11, 20]. The histogram table contains m bins, each of which is associated with a time segment $[t_i, t_{i+1})$ with the $[0, t_1)$ being the first and $[t_{m-1}, \infty)$ being the last. There are various methods to decide the length of each bin. We choose a simple method in which all bins except the last one have a same length, which we call the *granularity* of the histogram, denoted g . Let n be the total number of idle intervals observed and n_i the number of intervals of length between $[t_i, t_{i+1})$. The i^{th} bin has the value $f(i) = \sum_{j \geq i} n_j / n$, which represents $F(t_i)$, the probability an idle interval being at least t_i . All the bins together approximate the complementary cumulative distribution function F . Similarly, $P(t_i)$ can be approximated by $p(i) = \sum_{j \geq i} f(j) - sf(i)$. We construct the histogram in this way to facilitate the estimation of the optimal threshold.

We propose the following intermediate algorithm based solely on the estimated interval distribution:

1. Create the table of size m . For each bin i , set $t_i = i \times g$, where g is the granularity.
2. Initialize the table, set $n = 0$, $f(0) = 1$, $f(i) = 0$ for all $i > 0$.
3. Each time the disk is idle, calculate the last idle interval t before the last request.
4. $n = n + 1$.
5. For each bin i , $f(i) = f(i) \times (n - 1) / n$;
6. For each bin i satisfying $t \geq t_i$, $f(i) = f(i) + 1 / n$.
7. For each bin i , calculate $p(i) = \sum_{j=i}^{m-1} f(j) - sf(i)$.
8. Find the i that maximize $p(i)$, set the threshold to be $i \times g$.

Our performance results for the algorithm, hereafter called the *static histogram algorithm* are presented in section 3. The static histogram scheme can be further improved in the following way. On observing the disk access patterns, we find the assumption that the idle intervals are independently drawn from a fixed distribution is rather weak. Figure 1 shows number of requests every 5 minutes in two 24 hours disk access traces collected from a workstation[18]; it suggests that the idle interval distribution is time-dependent. The static histogram algorithm records every interval in

the past but fails to recognize the shift of arrival rates over the time. Therefore, the threshold given by the algorithm approximates the best fixed threshold for the entire past, but may not approximate very well the best threshold of a particular instance.

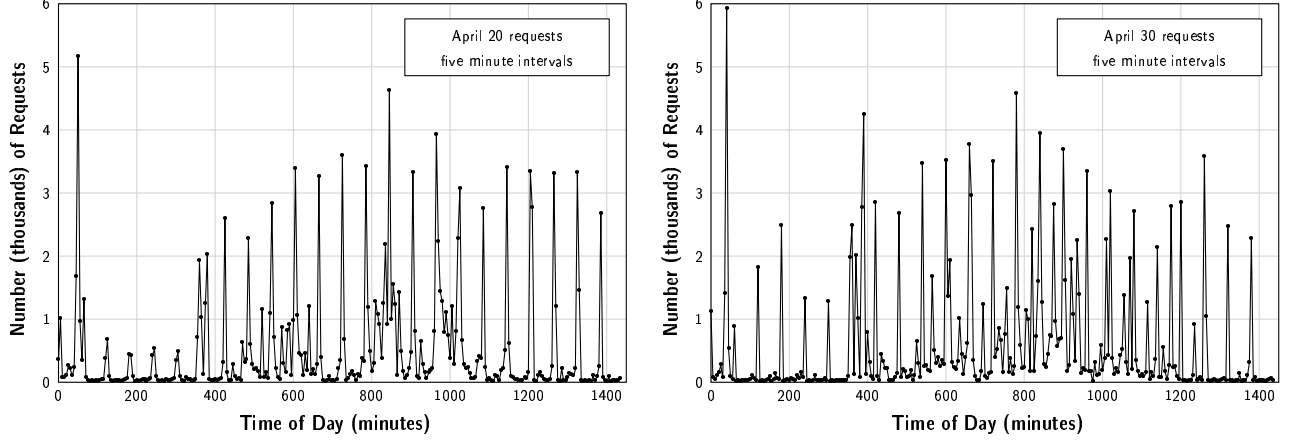


Figure 1: Number of requests per 5 minutes during two days on a disk drive

To address the time dependent nature of idle intervals, we propose an adaptive algorithm that gives more weight to recent intervals. The new algorithm, called the *dynamic histogram algorithm*, is described; there is one additional parameter α which will be discussed subsequently.

1. Create the table of size m . For each bin i , set $t_i = i \times g$, where g is the granularity of each bin.
2. Initialize the table, set $f(0) = 1$ and $f(i) = 0.5f(i - 1)$.
3. Each time the disk is idle, calculate the last idle interval t before the last request.
4. For each bin i , $f(i) = f(i) \times (1 - \alpha)$, where α is a value between 0 and 1.
5. For each bin i satisfying $t \geq t_i$, $f(i) = f(i) + \alpha$.
6. For each bin i , calculate $p(i) = \sum_{j=i}^{m-1} f(j) - sf(i)$.
7. Find the i that maximize $p(i)$, set the threshold to be $i \times g$.

The dynamic histogram algorithm chooses the threshold from the range $r = [0, g(m - 1)]$, which we call the *effective range*. Among the three parameters of the table: the number of entries m , the effective range r and the granularity g , only two are independent. In the remainder of the paper, we consider m a factor decided by r and g because the latter two directly decides how the histogram table approximates the interval distribution.

The table could be initialized as it is in the static histogram algorithm, which effectively assumes all previous intervals are all shorter than the granularity g . Alternatively, we can set $f(0) = 1$ and $f(i) = \beta f(i - 1)$ for all other i , assuming the previous intervals are drawn from an exponential distribution. We find the the algorithm is insensitive to the way the table is initialized because it is able to quickly approximate the table to the current distribution. In the experiments, we use the second initialization method with $\beta = 0.5$;

The effective range and granularity of the histogram table are subject to discussion. Helmbold *et al.* argue a threshold cannot be greater than s , the break-even time, in most situations. However,

our experiments show that larger threshold can slightly improve performance. A table with a finer granularity can better approximate the distribution, but the computation cost is increased as well. The impact of the effective range and granularity on the performance is discussed in section 4.

Comparing the static and dynamic histogram algorithms, one will notice the dynamic histogram algorithm does not utilize the number of intervals observed. In the static histogram algorithm the number needs to be counted because all idle intervals, regardless of their orders, have the same weight. By comparison, the weights of intervals deteriorate over the time in the dynamic histogram algorithm. The parameter α decides how rapidly the weights deteriorate. A new interval always has a weight that equals α , and all previous intervals have a total weight of $1 - \alpha$. The larger α , the faster the weights of previous intervals deteriorate. The impact of α is discussed in section 4.

3 Experimental Results

In this section, we present the trace driven simulation results showing our dynamic histogram algorithm saves more energy than all existing disk power management algorithms consistently on a variety of disk energy models.

3.1 Methodology

We run our simulations on disk access traces collected from April 18, 1992 to June 19, 1992 on the cello workstation in the HP lab[18]. The daily workloads change significantly during the period as illustrated in figure 2, allowing the traces to represent various types of disk workloads. Figure 1 shows numbers of requests per 5 minutes in two day's traces. The disk access traces have been used in evaluating the performance of power management algorithms in previous studies [7, 3].

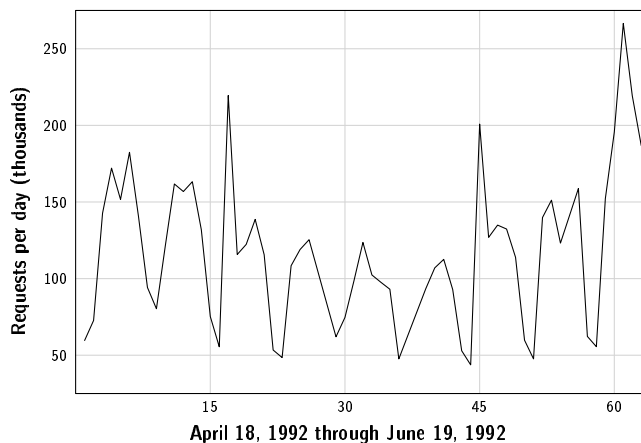


Figure 2: Number of requests per day between April 18, 1992 and June 19, 1992

Several existing algorithms are compared with both the static and the dynamic histogram algorithms. The 2-competitive algorithm [12] is non-adaptive, always using the break-even time as the threshold. The randomized algorithm [12], randomly chooses a threshold no larger than the break-even time, has proven best worst-case performance. The share algorithm is a machine learning algorithm [7] which adaptively selects a threshold based on disk access patterns and has the best average-case performance so far. The exponential average algorithm [8] predicates the length of the next interval and sets the threshold to be either zero or the break-even time accordingly. We also

compare our algorithms to the Oracle algorithm. Although the Oracle algorithm relies on knowledge of future disk accesses and cannot be applied in practice, it indicates how well our algorithms perform as well as indicating how much improvement is possible.

In this paper, disk energy is measured in terms of 'seconds', a measurement introduced by Douglass *et al.* [5] and used by others [6, 7]. A 'second of energy' is the difference between energy consumed by a standby disk and an idle disk in one second. The actual amount of one 'second of energy' in joules depends on the power characteristics of the disks. In using the term 'seconds of energy', we can discuss disk drive energy consumption while avoiding the joules/seconds conversion factor varying among disk drives varieties.

In the simulation, the disk spends t seconds of energy if it is never spun down during a period of t seconds. The disk consumes s seconds of energy for each spin-down, where s is the break-even time. Thus, if an algorithm decides to spin down the disk and it stays in the idle mode for t seconds, it saves $t - s$ seconds energy compared to the strategy that never spins down.

We measure the energy saving performance using two metrics: the savings ratio and the competitiveness. The savings ratio is the energy saved by an algorithm as a percentage of the overall energy consumed if the disk is never spun down. The competitiveness is defined as the ratio between the energy used by an algorithm and the energy used by the Oracle algorithm. The performance are measured on a variety of disk models, each of which is represented by a distinctive break-even time ranging from 1 second to 25 seconds. Table 1 presents the power characteristics and break-even time for some typical disk drives [9, 4]. The Ultrastar 36Z15 disk is a SCSI model designed for servers and have a relative long break-even time. The Travelstar 2000 and ST94011A are for mobile computers and have shorter break-even times.

Manufacturer Model	IBM/Hitachi Ultrastar 36Z15	IBM/Hitachi Travelstar 2000	Seagate ST94011A
Capacity(GB)	18.4	18	40
Idle Power(W)	10.2	1.85	1.2
Standby Power(W)	2.5	0.25	0.36
Spin-up Time(sec)	10.9	1.8	2.6
Spin-up Energy(J)	135	6.0	7.8
Spin-down Time(sec)	1.5	NA	NA
Spin-down Energy(J)	13	NA	NA
Break-even Time(sec) ⁰	15.2	3.5	8.2

Table 1: Power characteristics of several hard drives

Although disk power management policies can save substantial energy, they inevitably disrupt disk usage and delay response to disk requests arriving during the standby mode. Frequent spin-downs also wear the disk and reduce its lifespan. These delays might adversely impact the user perceived performance. A metric of user perceived intrusiveness used in previous studies is the number of spin-downs; accordingly, we measure this as an indication of intrusiveness for the user as well as the disk.

Since our algorithms use several parameters, we considered a range of parameters and select a set of reasonable combination. For both the static histogram and dynamic histogram algorithms, we set the effective range $r = 2s$ where s is the break-even time. The histogram table size m is fixed at 21, making the granularity $g = r/(m - 1) = 0.1s$. For the dynamic histogram algorithm the weight of new interval α is 0.01. Our experimental results indicate the algorithms are insensitive to the change of the parameters over a large range. For example, when the break-even time is 10 seconds, the competitive ratio of the dynamic histogram algorithm varies between 1.304 and 1.297

⁰The break-even time in table 1 are calculated using the equations in section2.1

when the effective range changes from 10 seconds to 29. The details of the impact of parameters can be found in section 4.

We have used the parameters published within previous papers for a critical comparison. For the share algorithm [7], the number of experts is 100, the learning rate η is 4, and the share rate α is 0.08. For the exponential average algorithm [8], the weight of a new interval a is 0.5, and the saturation limit c is 2.

3.2 Energy Saving Performance

Figure 3 shows the saving ratio of various power management algorithms; Figure 3 also shows their competitiveness. The results indicate the dynamic histogram algorithm saves more energy than all previous practical algorithms. The algorithm saves between 38.4% (when the break-even time is 25 seconds) and 87.7% (when the break-even time is 1 second) overall disk energy. It consistently outperforms the share algorithm, the best existing threshold based algorithm. The advantage is most significant with disk models of large break-even time: with a break-even time of 25 seconds, the dynamic histogram algorithm saves 4.1% overall energy more than the share algorithm, equivalent to about 1 hour battery life per day. Compared to the Oracle algorithm, the dynamic histogram algorithm has a competitive ratio between 1.19 and 1.31 (1.27 on average), meaning it only spends averagely 27% more energy than the Oracle algorithm. By contrast, the share algorithm has an average 1.33 competitive ratio. Compared to other algorithms, the advantage of the dynamic histogram algorithm is even more significant.

The static histogram algorithm performs worse than the share algorithm except when the break-even time is large, confirming our hypothesis that the distribution is time-dependent. The randomized algorithm has a relatively constant competitiveness ratio of about 1.58, which is compatible with the analytical results [12]. Although it has the best worst-case performance, the results show algorithms with knowledge of the history can have better results in practice. The exponential average algorithm performs worst among the algorithms due to the difficulty of predicting the length of the next idle interval. It has a sharp reduction of competitiveness as well as number of spindowns (see figure 4) when the break-even time is larger than 15 seconds. This can be explained as follows. An exceptionally high number of intervals in the traces are 30 seconds long, preceded and succeeded by very short intervals. The exponential average algorithm, observing the 30 seconds interval and the previous ones, predicts the next interval to be 15 seconds. This prediction causes a false spin-down when the break-even time is less than or equal to 15 seconds, but does no harm otherwise. This in turn improves the algorithm's performance when the break-even time is larger than 15 seconds.

3.3 Number of Spin-downs

The average daily number of spin-downs are shown in figure 4 a measurement of intrusiveness. The 2-competitive algorithm has the same number of spin-downs as the Oracle algorithm because both algorithms spin down if and only if the next interval is larger than the break-even time. Compared to the share algorithm, the dynamic histogram algorithm spins down more frequently when the break-even time is small, but less frequently when the break-even time is large. In general, the intrusiveness of the dynamic histogram algorithm is better than or comparable to existing algorithms.

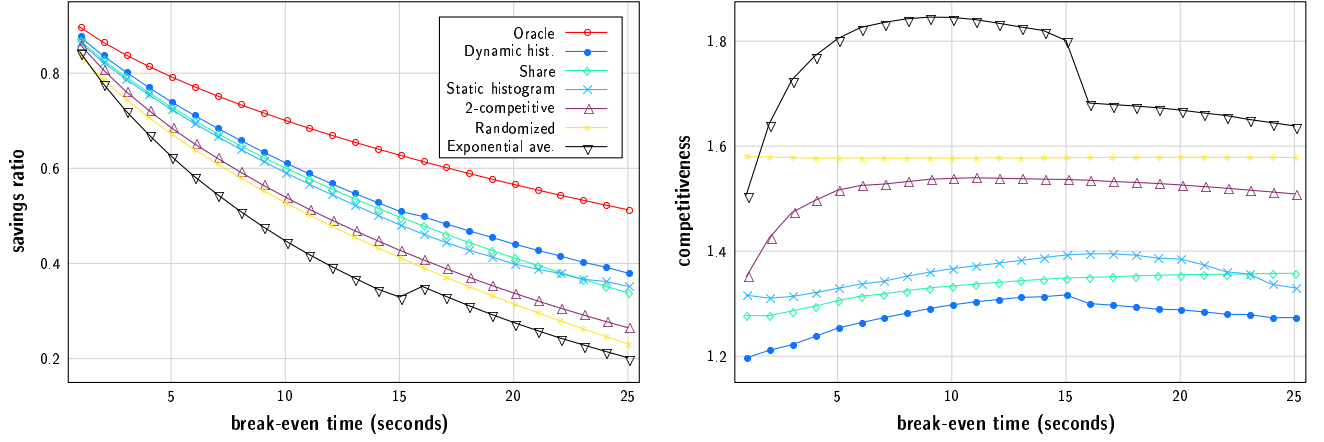


Figure 3: Savings ratio and competitiveness of various disk power management algorithms as a function of break-even time

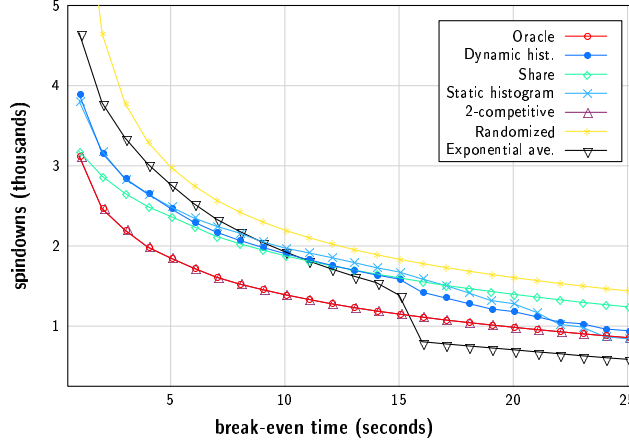


Figure 4: Average number of spin-downs per day vs break-even time

4 Discussion

4.1 Weight of New Intervals

The weight of the new interval α determines how rapidly a histogram table adjusts to the current idle interval distribution. If α is too large, the histogram may only reflect the statistics of a few previous intervals. If α is too small, the histogram may include obsolete information.

In the following experiment, we examine the impact of α on the performance of the dynamic histogram algorithm. We set the break-even time s to be 5 seconds and 10 seconds. The effective range of the table is $2s = 20$ seconds and the granularity of each entry g is one tenth of the break-even time. The results are shown in figure 5.

The result indicates the dynamic histogram's performance is relatively insensitive to α in a wide range between 0.01 and 0.2. A value outside this range can negatively affect the performance. However, even with a value as large as 0.5 or as small as 0.002, the dynamic histogram algorithm's performance still is compared favorably to the share algorithm, which has a competitiveness of 1.306 and 1.334 when the break-even time is 5 seconds and 10 seconds respectively.

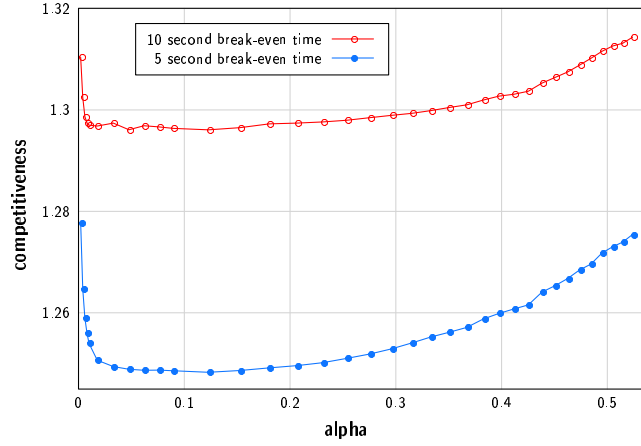


Figure 5: The competitiveness of the dynamic histogram algorithms as function of α

4.2 Effective Range and Granularity of the Histogram Table

As stated in section 2, both the effective range and granularity of the histogram table can affect the performance of the algorithm. We conduct a set of experiments with the break-even time set to 5 seconds and 10 seconds with $\alpha = 0.01$ to choose the parameters.

The histogram table has m entries and the granularity of g seconds. Each entry in the histogram table represents a range of g seconds with the exception of the last one. The dynamic histogram algorithm chooses the threshold from the effective range $[0, g(m - 1)]$. We first determine how granularity affects the performance of the dynamic histogram algorithm by fixing the effective range at $r = 2s$ seconds. The experimental results are shown in figure 6.

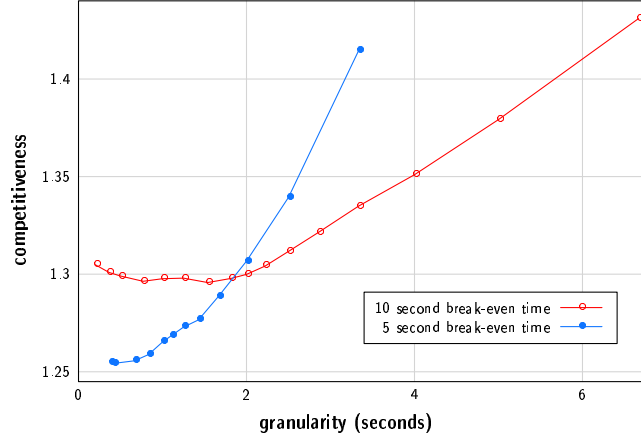


Figure 6: The competitiveness of the dynamic histogram algorithm as function of entry granularity

The results indicate a coarse granularity can substantially hurt the performance because the histogram cannot represent the interval distribution accurately enough. However, changing the granularity has little impact on performance if it is lower than one fifth of the break-even time. The results in section 3 are obtained with a granularity of one tenth of the break-even time, which gives satisfactory results.

Second, we fix the granularity at one tenth of the break-even time and determine the ideal effective range. Since most intervals are very short, the lower bound of the effective range has to be 0. The experiment only determines the upper bound of the effective range. Again the break-even time

is 5 seconds and 10 seconds and the α value is 0.01. We have experiments with a wide range of the upper bound. The results are shown in figure 7. The x-axis of the results indicates the ratio between the upper bound and the break-even time.

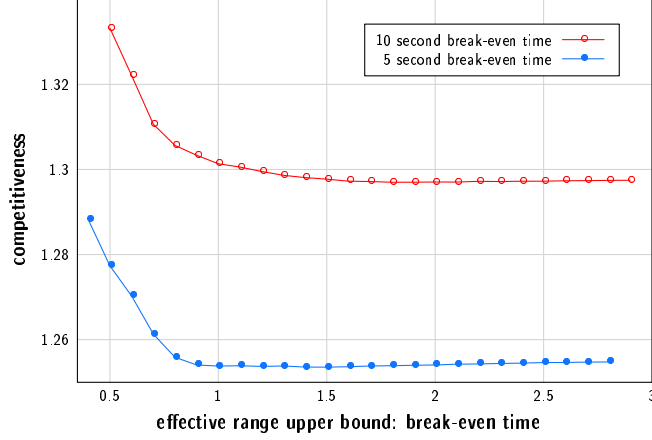


Figure 7: The competitiveness of the dynamic histogram algorithm as a function of the effective range

Helmbold *et al.* [7] suggested an effective range of $[0, s]$ where s is the break-even time. This is generally confirmed by the experimental results in figure 7. However, when the break-even time is 10 seconds, we find a larger effective range of $[0, 2s]$ can slightly improve the performance.

4.3 Implementation and Complexity Analysis

The dynamic histogram algorithm has been presented in section 2. Here we discuss implementation issues. The only dynamic memory required by the algorithm is for the histogram table, which can be implemented as a fixed size array with m entries and requires $O(m)$ space. We have shown the algorithm performs well for relative small $m = 21$. Therefore, the space complexity can be considered constant.

The algorithm can be further simplified. Since we are only interested in finding the i that maximize $p(i)$, we don't need to compute the exact value of $p(i)$. Note

$$p(i+1) - p(i) = \sum_{j=i+1}^{m-1} f(j) - sf(i+1) - \sum_{j=i}^{m-1} f(j) + sf(i) = (s-1)f(i) - sf(i+1),$$

the dynamic histogram algorithm can be simplified as follows:

1. Create the table of size m . For each bin i , set $t_i = i \times g$, where g is the granularity of each bin.
2. Initialize the table.
3. Each time the disk is idle, calculating the last idle interval t before the last request.
4. Let $p = 0$; $besti = 0$; $largestp = 0$.
5. For each bin i
 - $f(i) = f(i) \times (1 - \alpha)$;
 - If $t \geq t_i$, $f(i) = f(i) + \alpha$.

6. For each i

$$p = p + (s - 1)f(i) - sf(i + 1);$$
 If $p > largestp$, let $largestp = p$ and $besti = i + 1$.
7. Set the threshold to be $besti \times g$.

Each time the algorithm is invoked, m entries of the table are updated and p is computed m times. Since m is a small fixed value, the threshold decision process can be considered having constant time complexity.

Unlike the share algorithm, which requires exponential calculation in the algorithm, the dynamic histogram algorithm only requires simple arithmetic calculations. Therefore, it requires less computation cost and can be implemented effectively inside the disk controller. We record the running time of simulations of various power management algorithms on the HP traces. The simulations are done on a PC with a 1G Hz Athlon processor and 256 MB memory. An implementation of the share algorithm takes an average of 6.54 minutes to run the whole trace. The dynamic histogram algorithm only takes 2.08 minutes, a little longer than the 1.43 minutes taken by the very simple 2-competitive algorithm. Because of computational overhead such as I/O operations shared by the implementations, the real efficiency between the share algorithm and the dynamic histogram algorithm is even larger. The algorithm is also easy to implement: our implementation has less than 50 lines of code. Because of its simplicity and constant time/space complexity, the algorithm can be easily implemented on the disk controller or in a driver program.

5 Related Work

Disk power management policies can be roughly categorized into two categories: transition policies and load change policies. A transition policy determines when to spin down the disk to the standby state and when to spin up. A load change policy modifies the workload to increase burstiness. The static and dynamic histogram algorithms discussed in this paper are transition policies.

Load change policies reduce disk energy usage by change its workload. Such changes are often achieved through the change of the size or usage of the cache [15, 20]. Prefetching and delaying service of non-urgent operations are also effective strategies to cluster disk accesses [17]. The increased burstiness allows more aggressive transition policies.

The simplest and most widely used transition policy is the fixed threshold policy. The disk enters the standby mode when the disk is inactive after a fixed threshold. Manufacturers often suggest a fixed threshold on the order of minutes. However, researchers discovered that power consumption could be minimized if the thresholds are set as low as 1-10 seconds [5, 15].

Among the fixed threshold policy, the 2-competitive algorithm [12] is of particular interest. An algorithm is called c -competitive if it never uses more than c times the energy used by the Oracle algorithm that has the perfect knowledge of the future. The 2-competitive algorithm simply choose the threshold that equals the break-even time. It has been proved that the 2-competitive [12] has the best worst-case competitive ratio among all fixed threshold algorithms.

Other algorithms dynamically change the threshold. A randomized algorithm [12] selects the threshold randomly from $[0, s)$ where s is the break-even time according to the following density function $p(x) = e^{x/s}/(e - 1)$. This algorithm has a competitive ratio of $e/(e - 1)$. This competitive ratio is optimal in the sense that for all other distribution of thresholds, there exist access intervals for which the distribution's competitive ratio is larger than $e/(e - 1)$.

The randomized algorithm has the best worst-case performance. It has the best result if the access intervals are chosen by an adversary. However, it does not necessarily have the best performance on

real cases. If an algorithm obtains knowledge of the disk access pattern, it is likely to have a better performance than the randomized algorithm on real cases.

One common assumption is the idle intervals are drawn from an unknown but fixed distribution [13, 11]. With this assumption, the distribution can be learned by collecting historical interval data. In one of the article [11], the distribution of past intervals is estimated with a histogram of variable bin sizes. The algorithm then chooses a threshold to minimize energy usage. The algorithm is very close to the static histogram algorithm. In another article [13], the algorithm first builds up a set of candidate thresholds, each associated with a value indicating its effectiveness. At any point, the algorithm choose the candidate that performs best during the past. The algorithm requires only constant space but requires increasing time to update its data structure because of the increased amount of historical data.

Another strategy is also based on a set of candidates [7]. Initially, all candidates are given equal weights. After each access, the candidates' weights are increased or decreased according to how well they performed compared to the Oracle algorithm. The threshold is then set as the weighted average of all candidates. The algorithm performs better than fixed threshold algorithms and adaptive threshold algorithms stated in [12, 5, 13] on real world traces.

The above algorithms all use an inactivity threshold explicitly as our algorithm does. The Oracle algorithm, however, only has two choices: spin down immediately or never spin down. Inspired by this observation, some researchers try to predict the exact length of the next interval and make the decision based on the prediction. One method used the average lengths of previous intervals weighted by an exponential sequence as the predicted interval [8]. Another strategy uses an adaptive learning tree to predict if the length of the next interval is larger than a certain value [3]. However, these strategies generally save less energy than adaptive threshold strategies possibly because it is hard to have an exact prediction.

Other strategies model the arrival of requests and change of power states as stochastic processes. Requests arrive with a probability and the state changes with another probability. Minimizing energy usage becomes a stochastic optimization problem [2, 19, 16]. These strategies often rely on off-line analysis of the disk access trace and assume the access intervals are drawn from a known distribution (exponential [2] or Pareto [19]). The stochastic optimization process also requires significant computation cost.

6 Conclusion and Future Work

In this paper we present the dynamic histogram algorithm for disk power management. The algorithm adaptively learns the time-dependent disk arrival interval distribution and subsequently determines a time-out threshold. Our simulation results show the algorithm saves more energy than existing power management algorithms and requires little computational overhead. We show the algorithm can be implemented efficiently and requires only constant time and space.

In the future, we plan to expand the research in several directions. It is more accurate to model the disks with several energy states. We will investigate how to apply the dynamic histogram algorithm in a multiple states disk model situation. Disk power management algorithms can be easily applied to a multiple states model if we assume the disk can only spin down to the immediate lower energy mode [11]. In a more realistic model, the disk can spin down to any lower mode. Applying dynamic histogram algorithm in this model would be more challenging.

We are also interested in reducing the intrusiveness of the disk power management algorithm. First, the dynamic histogram algorithm is designed to minimize energy usage and needs modification to meet the new requirement. Second, the intrusiveness in this paper is measured as the number of

spin downs. This measurement has been widely used in research but is not accurate. For example, a user may tolerate two spin-downs if they are separated by 10 minutes, but might be agitated if they appear in one minute. Therefore, a new measurement combining the number of spin-downs and the intervals between spin-downs is needed as well.

We concentrated on disk energy management in this paper. However, the principles behind the dynamic histogram algorithm can be applied in other domains as well. In particular, the algorithm can find application in energy management of other electronic devices in a mobile computer, such as the display and the wireless interface. In principle, the algorithm can be applied to conserve energy on all devices with bursty user patterns.

References

- [1] ACPI. Advanced Configuration and Power Interface. Also available at <http://www.apci.info>.
- [2] Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. Dynamic power management of electronic systems. In *Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*, pages 696–702. ACM Press, 1998.
- [3] Eui-Young Chung, Luca Benini, and Giovanni De Micheli. Dynamic power management using adaptive learning tree. In *Proceedings of the 1999 IEEE/ACM international conference on Computer-aided design*, pages 274–279. IEEE Press, 1999.
- [4] Seagate Corp. Momentus drive specification. Also available at <http://www.seagate.com/support/disc/manuals/ata/momentus/pm.pdf>.
- [5] Fred Douglass, P. Krishnan, and Brian Marsh. Thwarting the power-hungry disk. In *USENIX Winter*, pages 292–306, 1994.
- [6] Fred Douglass, Padmanabhan Krishnan, and Brian Bershad. Adaptive disk spin-down policies for mobile computers. In *Proc. 2nd USENIX Symp. on Mobile and Location-Independent Computing*, 1995.
- [7] David P. Helmbold, Darrell D. E. Long, Tracey L. Sconyers, and Bruce Sherrod. A dynamic disk spin-down technique for mobile computing. In *Proceedings of the Second Annual ACM International Conference on Mobile Computing and Networking*, November 1996.
- [8] Chi-Hong Hwang and Allen C.-H. Wu. A predictive system shutdown method for energy saving of event-driven computation. *IEEE Trans. VLSI Syst.*, 8(3):28–32, 1997.
- [9] IBM. IBM hard disk drives. <http://www.storage.ibm.com/hdd/index.htm>.
- [10] IBM. Adaptive power management for mobile hard drives, 1999. Also available at <http://www.almaden.ibm.com/almaden/mobile-hard-drives.html>.
- [11] S. Irani, R. Gupta, and S. Shukla. Competitive analysis of dynamic power management strategies for systems with multiple power savings states. In *Proceedings of the conference on Design, automation and test in Europe*, page 117. IEEE Computer Society, 2002.
- [12] A. Karlin, M. Manasse, L. McGeoch, and S. Owicki. Competitive randomized algorithms for nonuniform problems. *Algorithmica*, pages 542–571, 1994.
- [13] P. Krishnan, P. M. Long, and J. S. Vitter. Adaptive disk spindown via optimal rent-to-buy in probabilistic environments. Technical report, Duke University, 1995.

- [14] P. Krishnan, P. M. Long, and J. S. Vitter. Adaptive disk spindown via optimal rent-to-buy in probabilistic environments. *Algorithmica*, 23(1):31–56, January 1999.
- [15] Kester Li, Roger Kumpf, Paul Horton, and Thomas E. Anderson. A quantitative analysis of disk drive power management in portable computers. In *USENIX Winter*, pages 279–291, 1994.
- [16] Yung-Hsiang Lu and Giovanni De Micheli. Comparing system-level power management policies. *IEEE Design and Test of Computers*, 18(2):10–19, March 2001.
- [17] A. Papathanasiou and M. Scott. Increasing disk burstiness for energy efficiency. Technical report, University of Rochester, 2002.
- [18] C. Ruemmler and J. Wilkes. UNIX Disk Access Patterns. In *Winter 1993 Usenix Conference*, 1993.
- [19] Tajana Simunic, Luca Benini, Peter Glynn, and Giovanni De Micheli. Dynamic power management for portable systems. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 11–19. ACM Press, 2000.
- [20] Q. Zhu, F. David, C. Devaraj, Z. Li, Y. Zhou, and P. Cao. Reducing energy consumption of disk storage using power-aware cache management. In *10th International Symposium on High Performance Computer Architecture*, 2004.