# Adaptive disk spin-down for mobile computers *

David P. Helmbold, Darrell D.E. Long, Tracey L. Sconyers and Bruce Sherrod

*Department of Computer Science, Jack Baskin School of Engineering, University of California, Santa Cruz, CA 95064, USA*

We address the problem of deciding when to spin down the disk of a mobile computer in order to extend battery life. One of the most critical resources in mobile computing environments is battery life, and good energy conservation methods increase the utility of mobile systems. We use a simple and efficient algorithm based on machine learning techniques that has excellent performance. Using trace data, the algorithm outperforms several methods that are theoretically optimal under various worst-case assumptions, as well as the best fixed time-out strategy. In particular, the algorithm reduces the power consumption of the disk to about half of the energy consumed by a one minute fixed time-out policy. Furthermore, the algorithm uses as little as 88% of the energy consumed by the best fixed time-out computed in retrospect.

## 1. Introduction

As one of the main limitations on mobile computing is battery life; minimizing energy consumption is essential for portable computing systems. Adaptive energy conservation algorithms can extend battery life by "powering down" devices when they are not needed. Several researchers have even considered dynamically changing the speed of the CPU in order to save power [7,22]. We show that a simple algorithm for deciding when to power down the disk drive is even more effective in reducing the energy consumed by the disk than the best fixed time-out value computed in retrospect.

Douglis et al. [4] showed that the disk sub-system on portable computers consumes a major portion of the available energy; Greenawalt [8] states 30% or more. It is well-known that spinning the disk down when it is not in use can save energy [4,5,14,23]. As spinning the disk back up consumes a significant amount of energy, spinning the disk down immediately after each access is likely to use more energy than is saved. An intelligent strategy for deciding when to spin down the disk is needed to maximize the energy savings.

Current mobile computer systems use a fixed time-out policy. A timer is set when the disk becomes idle and if the disk remains idle until the timer expires then the disk is spun down. This time-out can be set by the user, and typical values range from 30 s up to 15 min.

We use a simple algorithm called the *share algorithm*, a machine learning technique developed by Herbster and Warmuth [10], to determine when to spin the disk down. Our implementation of this algorithm dynamically chooses a time-out value as a function of recent disk activity. As the algorithm adapts to disk access patterns, it is able to exploit the bursty nature of disk activity.

We show that the share algorithm reduces the power consumption of the disk to about *one half* of the energy consumed by a one minute fixed time-out. In other words, the simulations indicate that battery life is extended by more than 17% when the share algorithm is used instead of a one minute fixed time-out.[1] A more dramatic comparison can be made by examining the energy wasted (i.e., the energy consumed minus the minimum energy required to service the disk accesses) by different spin-down policies. The share algorithm wastes only about 26% of the energy wasted by a one minute fixed time-out.

As noted earlier, large fixed time-outs are poor spin-down strategies. One can compute (in retrospect) the best fixed time-out value for a sequence of accesses and then calculate how much energy is consumed when this best fixed time-out is used. On the trace data we analyzed, the share algorithm performs better than even this best fixed time-out; on average it wastes only 77.5% of the energy wasted by the best fixed time-out.

The share algorithm is efficient and simple to implement. It takes constant space and time per access. For the results presented in this article, the total space required by the algorithm is never more than about 2400 bytes. Our implementation is about 100 lines of C code.

The rest of this article proceeds as follows. Section 2 contains a brief survey of related work. We formalize the problem and define our performance metrics in section 3. Section 4 describes the share algorithm. We present our empirical results in section 5, and present a justification for our approach to best fixed time-out calculations in section 6. Finally, we present future work in section 7, and conclusions in section 8.

* This research was supported by the Office of Naval Research under Grant N00014-92-J-1807, the National Science Foundation under Grant CCR-9704348, and by research funds from the University of California, Santa Cruz.

[1] We assume that the disk with a one minute time-out uses 30% of the energy consumed by the entire system. Thus, if $t$ is the time the system can operate on a single battery charge with 1 min time-outs, and $t'$ is the time the system can operate using the share algorithm, we have $t = 0.7t' + 0.15t'$. So $t' > 1.176t$ and the battery life is extended by more than 17%.

## 2. Related research

One simple disk spin-down policy picks a fixed time-out value and spins-down after the disk has remained idle for that period. Most current mobile computers use this method, with a fixed time-out as long as several minutes [4]. It has been shown that energy consumption can be improved dramatically by picking a shorter fixed time-out, such as a few seconds [4,5]. For any particular sequence of idle times, a *best fixed time-out* is the fixed time-out that causes the least amount of energy to be consumed over the sequence of idle times. The best fixed time-out depends on the particular sequence of idle times; this information about the future is unavailable to the spin-down algorithm *a priori*.

We can compare the energy used by algorithms to the minimum amount of energy required to service a sequence of disk accesses. This minimum amount of energy is used by the *optimal algorithm*, which "peeks" into the future before deciding what to do after each disk access. If the disk will remain idle for only a short time, then the optimal algorithm keeps the disk spinning. If the disk will be idle for a long time, so that the energy used to spin it back up is less than the energy needed to keep the disk spinning, then the optimal algorithm immediately spins it down. The optimal algorithm uses a long time-out when the idle time will be short and a time-out of zero when the idle time will be long.

Although both the optimal algorithm and the best fixed time-out use information about the future, they use it in different ways. The optimal algorithm adapts its strategy to each individual idle time, and uses the minimum possible energy to service the sequence of disk requests. The best fixed time-out is non-adaptive, using the same strategy for every idle time. In particular, the best fixed time-out waits some amount of time before spinning down the disk, and so uses more energy than the optimal algorithm. Although both the best fixed time-out and the optimal algorithm are impossible to implement in any real system, they provide useful information for evaluating the performance of other algorithms.

A critical value of the idle period is when the energy cost of keeping the disk spinning equals the energy needed to spin the disk down and then spin it back up. If the idle time is exactly this critical value then the optimal algorithm can either immediately spin-down the disk or keep it spinning; both actions incur the same energy cost.

We measure the energy cost to spin-down and then spin-up the disk in terms of the number of seconds that this amount of energy would keep the disk spinning. We call this number of seconds the *spin-down cost* for the disk drive.

One natural algorithm uses a fixed time-out equal to the spin-down cost of the disk drive. If the actual idle time is shorter than the spin-down cost, then this algorithm keeps the disk spinning and uses the same amount of energy as the optimal algorithm on that idle period. If the length of the idle time is longer than the time-out, then this algorithm waits until the time-out expires and then spin down the disk. This uses exactly twice the spin-down cost in total energy (to keep it spinning before the spin-down, and then to spin it back up again). This is also twice the energy used by the optimal algorithm on that idle period, since the optimal algorithm would have immediately spun down the disk. Therefore this algorithm never consumes more than twice the energy used by the optimal algorithm.

An algorithm is called $c$*-competitive* or has a *competitive ratio* of $c$ if it never uses more then $c$ times the energy used by the optimal algorithm [11,19]. The preceding algorithm is 2-competitive, and we refer to it as the *2-competitive algorithm*. It is easy to see that the 2-competitive algorithm has the best competitive ratio of all constant time-out algorithms.[2]

Because the 2-competitive algorithm uses a fixed time-out, its performance never surpasses the best fixed time-out. The 2-competitive algorithm uses *one* predetermined time-out, guaranteeing that it is reasonably good for *all* sequences of idle times, while the best fixed time-out is computed for a particular sequence of idle times.

Randomized algorithms can be viewed as selecting time-out values from some distribution, and can have smaller (expected) competitive ratios. Although the competitive ratio is still based on a worst-case idle time between accesses, the energy used is averaged over the algorithm's random choice of time-out.

Karlin et al. [12] give an expected $(e/(e-1))$-competitive randomized algorithm. If the spin-down cost is $s$, their algorithm chooses a time-out at random from $[0, s]$ according to the density function

$$\Pr(\text{time-out} = x) = \frac{e^{x/s}}{e-1}$$

and is optimal in the following sense: every other distribution of time-outs has an idle time for which the distribution's expected competitive ratio is larger than $e/(e-1)$.

Both the 2-competitive algorithm and the $(e/(e-1))$-competitive randomized algorithm perform competitively even if the idle times are selected adversarially. Without making some assumptions about the nature of the data, and thus abandoning this worst case setting, it is difficult to improve on these results.

One can get better bounds by assuming that the idle times are drawn independently from some fixed (but unknown) probability distribution instead of chosen adversarially. With this assumption, the good time-out values for the past idle times should also perform well in the future. Krishnan et al. [13] introduced an algorithm that operates in two phases. The first phase predicts arbitrarily while building a set of candidate time-outs from the idle

[2] Any algorithm that uses a larger time-out, say $(1+\Delta)s$, for a spin-down cost $s$, is only $(2+\Delta)$-competitive when the idle times are large; an algorithm that uses time-out smaller than $s$ is less than 2-competitive when the idle time is between its time-out and $s$.

Table 1
Energy and loss statistics during each trial.

$$\text{Energy used by time-out} = \begin{cases} \text{idle time} & \text{if idle time} \leqslant \text{time-out} \\ \text{time-out} + \text{spin-down cost} & \text{if idle time} > \text{time-out} \end{cases}$$

$$\text{Energy used by optimal} = \begin{cases} \text{idle time} & \text{if idle time} \leqslant \text{spin-down cost} \\ \text{spin-down cost} & \text{if idle time} > \text{spin-down cost} \end{cases}$$

$$\text{Excess energy} = \text{Energy used by time-out} - \text{Energy used by optimal}$$

$$\text{Loss} = \frac{\text{Excess energy}}{\text{Spin-down cost}}$$

times. After obtaining enough candidate time-outs, the algorithm then tracks the energy used by the candidates and chooses the best candidate as its time-out. Their full algorithm repeatedly restarts this basic algorithm with each restart using more candidates and running for exponentially increasing periods. When processing the $t$th idle period the full algorithm tracks $O(\sqrt{t})$ candidates, taking $O(\sqrt{t})$ time. They prove that the energy used by this full algorithm *per disk access* approaches that of the best fixed time-out under probabilistic assumptions.

The idle times in disk traces do not appear to be drawn according to a simple fixed distribution. So, in contrast to the above algorithms, we assume that the data is time dependent, having both busy and idle periods. We use a simple adaptive algorithm that exploits these different periods by shifting its time-out after each trial. In trace-driven simulations, our algorithm performs better than all of the algorithms described above, and even conserves more energy than the best fixed time-out.

Douglis et al. [3] studied some incrementally adaptive disk spin-down policies. The policies they consider maintain a changing time-out value. When the disk access pattern indicates that the current time-out value may be too long or too short, the time-out is modified by an additive or multiplicative factor. While we concentrate on the energy used by the spin-down algorithm, they pay particular attention to those spin-downs likely to inconvenience the user and analyze the trade-off between energy consumed and these undesirable spin-downs. Golding et al. [5] evaluate similar incrementally adaptive policies.

## 3. Problem description

We interpret disk spin-down algorithms as computing, after each disk access, a delay or *time-out* indicating how long an idle disk is kept powered up before spinning it down. We treat the problem as a sequence of trials, where each trial represents the idle time between two consecutive accesses to the disk. The disk is spun down if and only if it remains idle for longer than the computed time-out.

We measure the performance of the algorithms in terms of "seconds of energy" used, a measure introduced by Douglis et al. [4]. One "second of energy" is the difference in energy consumed between a spinning disk and a spun down disk over one second. One second of energy corresponds to some number of joules, depending on the model of disk drive used. Using seconds of energy allows us to discuss disk drives in general while avoiding a joules/second conversion factor.

We use the term "spin-down cost" to refer to the total cost of choosing to spin down the disk. This cost equals the energy required to spin the disk down (if any), plus the energy needed to spin the disk back up. We measure the spin-down cost in seconds of energy so that a spin-down cost of $s$ means that spinning the disk down and starting it up again consumes as much energy as keeping the disk spinning for $s$ seconds. If we assume that a mobile computer user's disk usage is independent of the type of disk, then the *spin-down cost* $s$ is the only statistic about the physical disk that we need for our simulations. Douglis et al. [4] compute this value for two disks, giving spin-down costs of 5 and 14.9 s. Golding et al. [5] give disk statistics that correspond to a spin-down cost of 9 or 10 s.

We define the metrics in table 1 for measuring and comparing the performance of algorithms. The energy use of an algorithm on a given trial depends on whether or not the algorithm spins down the disk. The *excess energy* used by the algorithm is the amount of additional energy used by the algorithm over the optimal algorithm. We find it convenient to scale the excess energy, and denote this scaled quantity for a time-out $x$ as *Loss*($x$).

## 4. Algorithm description

The *share* algorithm is a member of the multiplicative-weight algorithmic family. This family has excellent performance for a wide variety of on-line problems [2,15,16,20]. Algorithms in this family receive as input a set of "experts", other algorithms which make predictions. On each trial, each expert makes a prediction. The goal of the algorithm is to combine the predictions of the experts in a way that minimizes the total error, or loss, over the sequence. Algorithms typically keep one weight per expert, representing the quality of that expert's predictions, and predict with a weighted average of the experts' predictions.

After each trial the weights of the experts are updated: the weights of misleading experts are reduced (multiplied

by some small factor), while the weights of good experts are usually not changed. The more misleading the expert the more drastically the expert's weight is reduced. This method causes the predictions of the algorithm to quickly converge to the those of the best expert.

Herbster and Warmuth developed a "sharing update" [10] that takes some of the weight of each misleading expert and "shares" it among the other experts. Thus, an expert whose weight was severely slashed, but is now predicting well, quickly regains its influence on the algorithm's predictions. This adaptability allows the algorithm to exploit the bursty nature of disk accesses and perform better than the best fixed time-out.

For the disk spin-down problem, we usually interpret each expert as a different fixed time-out, although we use algorithms as experts in section 5.8. In our experiments we used $n = 25$ experts whose predictions are exponentially spaced fixed time-outs between zero and the disk's spin-down cost. Although it is easy to construct traces where the best fixed time-out is larger than the spin-down cost, this does not seem to happen in practice. Reducing the space between experts tends to improve the algorithm's performance. On the other hand, the running time and memory requirements of the algorithm increase as additional experts are added.

We denote the predictions of the experts as $x_1$ to $x_n$ (which are fixed). The weights of the experts, denoted by $w_1$ to $w_n$, are initially set to $1/n$. We use *Loss*$(x_i)$ to denote the loss of expert $i$ on a given trial.

The share algorithm uses two additional parameters. The learning rate, $\eta > 1$, controls how rapidly the weights of misleading experts are reduced. The share parameter, $0 < \alpha < 1$, governs how rapidly a poorly predicting expert's weight recovers when it begins predicting well. These parameters must be chosen carefully to prove good worst-case bounds on the learning algorithm. However, the real-world performance of multiplicative weight algorithms appears less sensitive to the choice of parameters; see Blum [1] for another example. In our experiments, akin to a "train and test" regimen, we used a small portion of the data (the first day of one trace) to find good values for $\eta$ and $\alpha$, and then use those settings on the rest of the data (the remaining 62 days). We chose $\eta = 4.0$ and $\alpha = 0.08$. Small perturbations in these parameters have little effect on our results. The performance of the algorithm on our baseline disk changes by less than a factor of 0.0023 as $\alpha$ varies in the range 0.05 to 0.1. Similarly, different values of $\eta$ between 3.5 to 4.5 cause the algorithm's performance to change by at most a factor of 0.0007.

We can now precisely state the variant of Herbster and Warmuth's [10] variable-share algorithm that we use. We denote the time-outs used by the $n$ experts as $x_1, \ldots, x_n$, the time-out computed for each trial as *time-out*, the spin-down cost as *spin-down*, and the idle time and optimal energy for each trial as *idle* and *optimal*, respectively.

On each trial the algorithm:

1. Uses a time-out equal to the weighted average of the experts' time-outs
$$\textit{time-out} = \frac{\sum_{i=0}^{n} w_i x_i}{\sum_{i=0}^{n} w_i}.$$
2. Computes the loss of each expert $x_i$
$$\textit{Energy}(x_i) = \begin{cases} \textit{idle} & \text{if } \textit{idle} < x_i, \\ \textit{time-out} + \textit{spin-down} & \text{otherwise,} \end{cases}$$
$$\textit{Loss}(x_i) = \frac{\textit{Energy}(x_i) - \textit{optimal}}{\textit{spin-down}}.$$
3. Reduces the weights of poorly performing experts
$$w_i' = w_i \mathrm{e}^{-\eta \textit{Loss}(x_i)}.$$
4. Shares some of the remaining weights
$$\textit{pool} = \sum_{i=1}^{n} w_i' \left(1 - (1-\alpha)^{\textit{Loss}(x_i)}\right),$$
$$w_i'' = (1-\alpha)^{\textit{Loss}(x_i)} w_i' + \frac{1}{n}\textit{pool}.$$

The new $w_i''$ weights are used in the next trial.

The algorithm runs in constant time and space where the constants depend linearly on $n$, the number of experts. However, the algorithm as stated above will continually shrink the weights towards zero. Our implementation avoids underflow problems by bounding the ratio between weights and periodically rescaling them.

## 5. Experimental results

We present trace-driven simulation results showing that the share algorithm outperforms the best strategies currently available. We extend our previous work [9] by examining the spacing between and the number of experts, varying parameters that govern how quickly the algorithm learns, and adding other types of experts, such as adaptive algorithms. We present improvements on the share algorithm that allow it to run with fewer experts and use less energy than previously reported.

### *5.1. Methodology*

We used traces of HP C2474s disks collected from April 18, 1992 through June 19, 1992 (63 days) as described by Ruemmler and Wilkes [18]. These traces came from three different Hewlett-Packard computer systems, all running release 8 of the HP-UX operating system, which uses a version of the BSD fast file system [17]. The trace data were obtained using a kernel-level trace facility built into HP-UX that is extremely light-weight and adds no noticeable processor load to the system. The data were logged to dedicated disks to avoid perturbing the system being measured. Each trace record contained the following data about a single physical I/O:

- timings, to microsecond resolution, of enqueue time (when the disk driver first sees the request); start time (when the request is sent to the disk) and completion time (when the request returns from the disk);
- disk number, partition and device driver type;
- start address (in 1 kilobyte fragments);
- transfer size (in bytes);
- the drive's request queue length upon arrival at the disk driver, including the current request;
- flags for read/write, asynchronous/synchronous, block/character mode;
- the type of block accessed ($i$-node, directory, indirect block, data, super block, cylinder group bit map).

For the initial train-and-test regimen, one disk was selected and its trace data for the first day was used to find reasonable settings for the parameters. The values determined from this partial trace are 25 exponentially distributed experts, $\eta = 4$, and $\alpha = 0.08$. These were used to test the share algorithm against the other algorithms, including the 2-competitive algorithm, the randomized $(e/(e-1))$-competitive algorithm, (an approximation to) the best fixed time-out, and the optimal algorithm, all described in section 2. As the best fixed time-out is very expensive to compute, we compute only an approximation. The possible error in this approximation is analyzed in section 6. The optimal algorithm's performance provides an indication of how far we have come and how much room is left for improvement.

After confirming that the share algorithm outperformed the others using these parameter settings, we then varied several properties of the share algorithm. These variations included the distribution of the experts (uniform, harmonic, and exponential), the number of experts $5 < n < 100$, the types of experts (fixed value and four incrementally adaptive algorithms), the learning rate $1 < \eta < 30$ (range of 1–30), and the share rate $0.03 < \alpha < 0.9$. Each parameter was examined independently and the most promising variations were tried in combination.

Although very few experts performed relatively poorly, our tests indicate that 10 experts predict nearly as well as 100 experts. The distribution of the experts also has a large impact on the share algorithm's performance. For all parameter settings, exponentially spaced experts outperformed uniformly spaced experts. We also found that when exponentially spaced experts were used, the share and learning rate could be varied within reasonable ranges without significantly increasing the energy used by the algorithm. Exponentially spaced experts allow fewer resources to be used, since fewer experts are needed, and tolerate a wide range of values for the learning and share rate, thus reducing the need to fine tune the algorithm to match the data.

### 5.2. Share versus fixed time-out

Figure 1 summarizes the main experimental results of this article. For each value of the spin-down cost, we show the daily energy use (averaged over the 62 day test period) for all the algorithms described in this section: the 2-competitive algorithm, the $(e/(e-1))$-competitive randomized algorithm, the approximate best fixed time-out, and the share algorithm using 25 exponentially spaced experts, with $\eta = 4$ and $\alpha = 0.08$. We also include the energy use of the optimal algorithm, one minute and 30-second fixed time-outs for comparison and to give some idea of scale and the limits on possible improvements. The figure shows that the share algorithm is better than the other practical algorithms, and even outperforms the best fixed time-out. Our implementation uses between 88% (at spin-down cost 1) and 96% (at spin-down cost 20) of the energy used by the best fixed time-out. When averaged over the 20 time-outs, our implementation uses only 94% of the energy consumed by the best fixed time-out.

Figure 2 plots the *excess energy* used by each algorithm beyond that used by the optimal algorithm. This indicates how much more energy was used by the algorithm than the theoretical minimum energy required. Compared to the per-

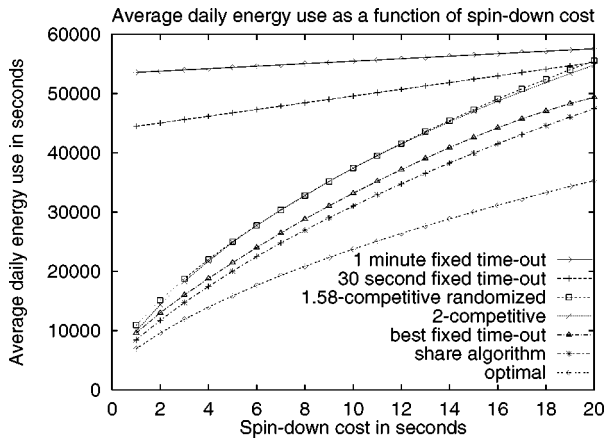


Figure 1. Average energy use per day as a function of the spin-down cost.

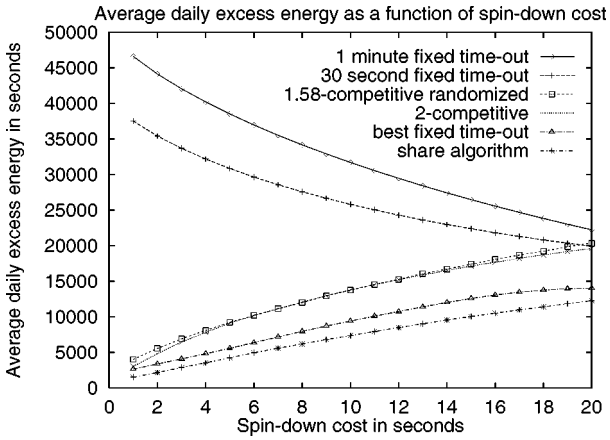


Figure 2. Average excess energy per day as a function of the spin-down cost.

formance of a one minute fixed time-out, we see a dramatic improvement. Over the twenty spin-down costs, the share algorithm averages an excess energy of merely 26.1% the excess energy of the one minute time-out. In terms of total energy, the share algorithm uses 54.7% of the total energy used by the one minute fixed time-out (averaged over the different spin-down costs). Stated another way, if the battery is expected to last 4 h with a one min time-out (where $1/3$ of the energy is used by the disk) then almost 45 min of extra life can be expected when the share algorithm is used.

Notice that the $(e/(e-1))$-competitive randomized algorithm performs slightly worse than the 2-competitive algorithm. This is easily explained when we consider the distribution of idle times in our traces. Most idle times are short – much shorter than the spin-down cost – and the 2-competitive algorithm performs optimally on the short idle times. Only when the disk stays idle for longer than the spin-down cost does the 2-competitive do the wrong thing. The $(e/(e-1))$-competitive randomized algorithm performs $(e/(e-1))$-competitively for all idle times, including those shorter than the spin-down cost.

Another class of algorithms adapts to the input patterns, deducing from the past which values are likely to occur in the future. This approach was taken by Krishnan et al. [13] (see section 2). Their algorithm keeps information that allows it to approximate the best fixed time-out for the entire sequence, and thus its performance is about that of the best fixed time-out. We use an algorithm that takes this approach a step further. Rather than looking for a time-out that has done well on the entire past, the share algorithm attempts to find a time-out that has done well on the recent past. As figure 2 shows, the share algorithm consistently outperforms the best fixed time-out, consuming an average of 77% of the excess energy consumed by the best fixed time-out. The share algorithm outperforms the best fixed time-out by exploiting time dependencies in the input values.

### 5.3. Share versus other adaptive algorithms

Douglis et al. [3] considered a family of incrementally adaptive spin-down schemes. These schemes change the time-out after each idle time by either an additive or multiplicative factor. When an idle time is "long" the time-out is decreased to spin down the disk more aggressively. When an idle time is "short" the time-out is increased to reduce the chance of an inappropriate spin-down.

The traces are composed mostly of short idle times, so there is the danger that an adaptive algorithm's time-out value will quickly become too large to be effective (see table 3). To prevent this problem we upper bound the time-out at 10 s, the same value as the largest expert used by the share algorithm. The incrementally adaptive algorithms perform poorly without this bound.

We consider three ways of increasing the time-out: doubling ($\times 2.0$), adding 1 s ($+1.0$), and adding 0.1 s ($+0.1$). Similarly, we decrease the time-out three ways: halving ($\div 2.0$), decreasing by 1 s ($-1.0$), and decreasing by 0.1 s ($-0.1$). This gives us nine variations in the incrementally adaptive family. Similar variations were also used in Douglis et al. [3] and Golding et al. [6]. The time-out should never become negative, so we constrained the time-out to be at least one decrement amount above zero.

We compared each of these nine algorithms with the share algorithm and the daily best time-outs on three different traces. The results are presented in table 2.

Table 2
Spin-downs and energy costs in seconds of the adaptive algorithms, averaged over the two-month traces. The spin-down cost is 10 s.

| Algorithm | | Trace 1 | | Trace 2 | | Trace 3 | |
|---|---|---|---|---|---|---|---|
| | | SDs | cost | SDs | cost | SDs | cost |
| Daily best fixed | | 2036 | 32566 | 741 | 12163 | 409 | 4887 |
| Share algorithm | | 1894 | 30436 | 707 | 11676 | 371 | 4784 |
| 10 s time-out | | 1358 | 36890 | 494 | 13833 | 294 | 6613 |
| Adaptive | | | | | | | |
| increase | decrease | | | | | | |
| $+0.1$ | $-0.1$ | 1378 | 37412 | 496 | 13792 | 295 | 6376 |
| $+0.1$ | $-1.0$ | 1472 | 34925 | 649 | 12964 | 312 | 5630 |
| $+0.1$ | $\div 2.0$ | 1775 | 31424 | 788 | 12551 | 357 | 4923 |
| $+1.0$ | $-0.1$ | 1378 | 37418 | 494 | 13813 | 294 | 6577 |
| $+1.0$ | $-1.0$ | 1379 | 37261 | 505 | 13478 | 294 | 6445 |
| $+1.0$ | $\div 2.0$ | 1441 | 35590 | 564 | 12666 | 303 | 5742 |
| $\times 2.0$ | $-0.1$ | 1361 | 36859 | 499 | 13783 | 295 | 6594 |
| $\times 2.0$ | $-1.0$ | 1392 | 36548 | 549 | 13431 | 299 | 6502 |
| $\times 2.0$ | $\div 2.0$ | 1542 | 35529 | 686 | 13239 | 320 | 6158 |

Table 3
Frequencies of idle time ranges in a typical day of the trace. There are 142,694 idle times in this day.

| Idle time (s) | Frequency count |
|---|---|
| 0 | 37,252 |
| 0–1 | 102,146 |
| 1–10 | 1,747 |
| 10–30 | 917 |
| 31–100 | 498 |
| 100–600 | 131 |
| $>600$ | 3 |

The better incrementally adaptive algorithms decrease the time-out rapidly but increase it only slowly. Decreasing the time-out rapidly allows greater savings if the next idle time is long. The disadvantage of a rapid decrease is that an inappropriate spin-down may occur if the next idle time had an intermediate duration. However, the preponderance of small idle times (see table 3) makes this unlikely. A slow increase in the threshold allows the algorithm to perform well when two nearby long idle times are separated by one or two short idle times.

Table 2 shows that some of the incrementally adaptive algorithms are primarily exploiting the 10 s bound on their time-outs. Since any spin-down done by the 10 s time-out is also done by all of the other algorithms, we can infer that some of the add/subtract algorithms do exactly the same spin-downs as the 10 s time-out, although sometimes they

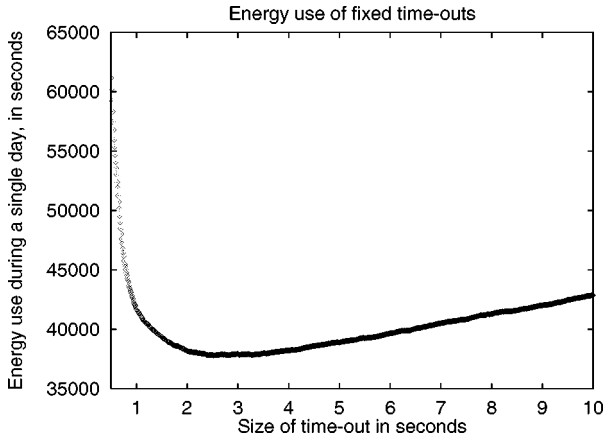


Figure 3. Energy cost of fixed time-outs, Monday, April 20 using a spin-down cost of 10.

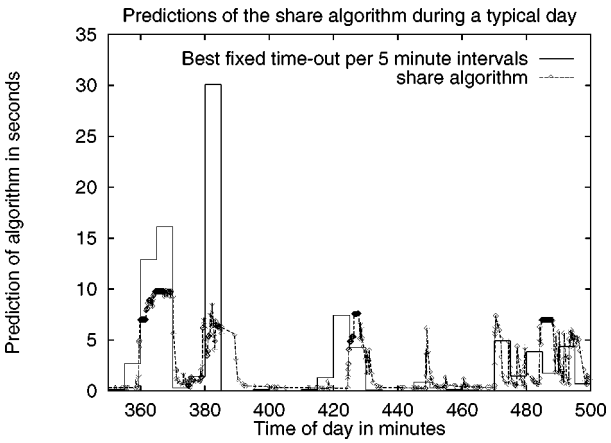


Figure 4. Time-outs suggested by the algorithm, on each successive trial during a 24 h trace, with a spin-down cost of 10.

may do these spin-downs with a slightly smaller time-out.

An interesting property of table 2 is that the daily best fixed time-out uses slightly less energy than the share algorithm on trace 2. This is due in part to the fact that the best fixed time-out is recalculated for each day's data. On trace 2, it varies from about 1 s to about 6 s depending on the day. If the same time-out was used every day then energy consumed would certainly be larger than that used by the share algorithm.

### 5.4. Predictions of the share algorithm

Figure 4 shows the predictions of the algorithm during a portion of a typical day (Monday, April 20), using a spin-down cost of 10 and 100 uniformly spaced experts.[3] The figure also shows the best fixed time-out for each 5 min interval. Note that it is unreasonable to expect any practical algorithm to perform as well as the best fixed time-outs on small intervals. This figure illustrates some interesting aspects of the share algorithm. We can see that the time-outs

[3] Since the improvements using exponentially spaced experts are in the 3–5% range, the comparison here should also apply to exponentially spaced experts.

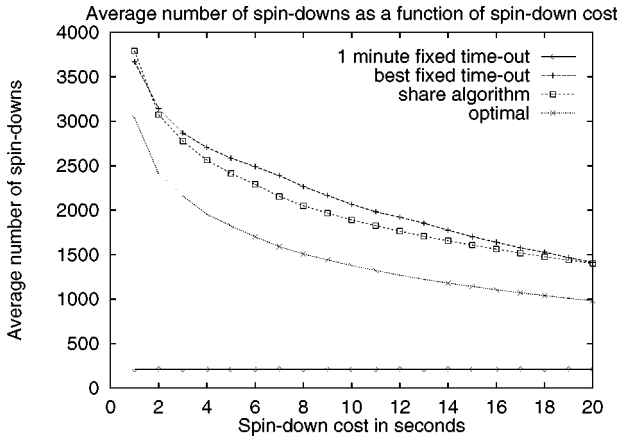


Figure 5. Average spin-downs for each algorithm as a function of spin-down cost.

used by the share algorithm vary dramatically, by at least an order of magnitude. While the disk is idle, the predictions become smaller (smaller than the best fixed time-out), and the algorithm spins the disk down more aggressively. When the disk becomes busy, the algorithm quickly jumps to a much longer time-out. These kinds of shifts occur often throughout the trace, sometimes every few minutes. The time-outs issued by the share algorithm tend to follow the best fixed time-outs over five minute intervals, which in turn reflects the bursty nature of disk accesses. Due to our choice of experts, the share algorithm can only make predictions less than or equal to the spin-down cost (10 in this example). However, as figure 4 shows, sometimes the best fixed time-out is not in this range. Yet, our performance remains good overall. The explanation for this is that when the best fixed time-out is large there is a large region which is nearly flat, with all time-outs in the region using similar amounts of energy.

### 5.5. Spin-downs

One way to measure intrusiveness is by the number of spin-downs. Figure 5 shows the average number of spin-downs used per day as a function of the spin-down cost for the share algorithm, the best fixed time-out, the optimal algorithm, and the one minute fixed time-out. We can see from this figure that the share algorithm recommends spinning the disk down less often than the (daily) best fixed time-outs. Our simulations show that the share algorithm tends to predict when the disk will be idle more accurately than the best fixed time-out, allowing it to spin the disk down more quickly when both algorithms spin down the disk. In addition, the share algorithm more accurately predicts when the disk will be needed, enabling it to avoid disk spin-downs ordered by the best fixed time-out. Because the share algorithm spins the disk down less often, it is also likely to minimize inconvenience to the user.

### 5.6. Spacing of experts

The performance of the share algorithm changes when different spacings of the experts are used. We looked at three distributions of constant value experts: uniform, harmonic, and exponential. Both the harmonic and exponential spacings have more experts with small time-out values. As shown in figure 6, this bias decreases the average average energy used. Exponentially spaced experts with a base of 2.0 perform as well as or better than harmonically spaced experts, which performed better than uniformly spaced experts for all spin-down costs.

The relative energy consumed between the types of expert spacings dependends on the values chosen for the learning and share rate. When $\eta = 4$ and $\alpha = 0.08$, all three spacings performed approximately the same. In general, higher share rates were better when using exponential spacing while uniform spacing preferred lower share rates. However, the average energy used by exponential spacing was almost always lower than that used by uniform spacing.

The base used for the exponential spacing interacts with the other parameters in complicated ways. Figure 7 shows the energy used as a function of the base for two different settings of the other parameters. When a small base is used, the resulting distribution is closer to uniform, and the algorithm is more sensitive to the choices of the other parameters. When large bases are used, the algorithm is more stable and tends to perform better with larger values for the other parameters.

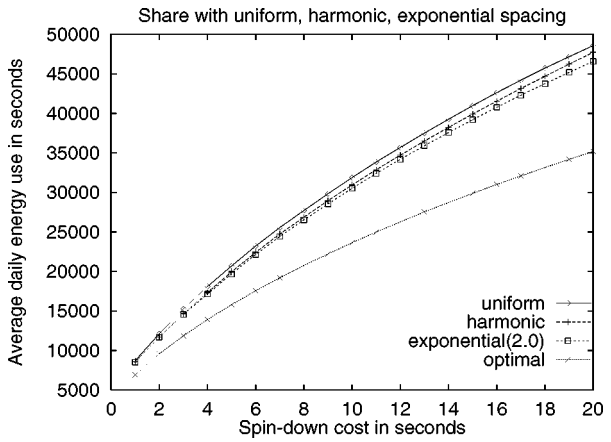


Figure 6. Share algorithm with varying expert spacings (50 experts, $\eta = 1$, $\alpha = 0.8$).

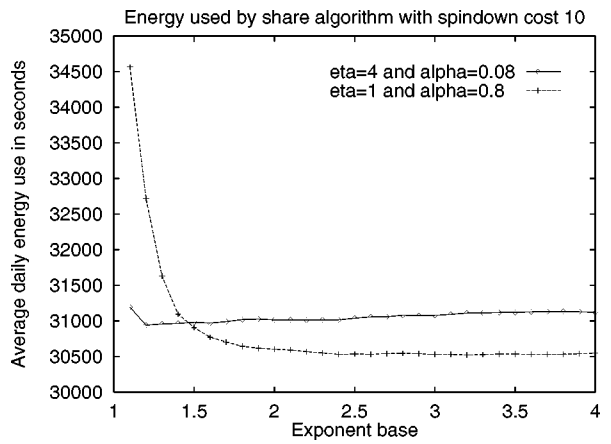


Figure 7. Share algorithm with exponential spacing and a spin-down cost of 10.

We ran several experiments where the learning rate was fixed at $\eta = 4$ (a good value for all spacings) while the share rate was varied. The exponentially spaced experts performed best with higher share rate, and that both spacings were somewhat insensitive the the actual share rate chosen. Interestingly, base 1.5 and 2.0 both performed steadily better as the share rate was increased. For uniform spacing, the optimal value was somewhere around 0.08. Thus, uniform spacing and exponential spacing performed best with share rate values at the opposite ends of the spectrum.

Finally, we fixed the share rate at $\alpha = 0.08$ and varied the learning rate. For all spin-down costs and spacings, a learning rate $\eta$ between about 3.5 and 4.5 produced the best results. In almost all cases, the exponentially spaced experts used slightly less energy than the uniformly spaced experts for moderate $\eta < 5$. Although the performance of both spacings degraded with larger $\eta$, when $\eta > 9$, the uniformly spaced experts were more robust against extremely large $\eta$ values.

In summary, exponentially spaced experts using either base 1.5 or 2.0 perform nearly the same when $1 < \eta < 9$, and though uniformly spaced experts perform a little worse in this wide range, they do perform reasonably well. For the share rate, exponentially spaced experts performed the best with the largest share rate, while uniformly spaced experts performed best with a much lower share rate of about 0.08. The algorithms performed most efficiently with an exponential spacing using either base 1.5 or base 2.0, and with $\eta$ between 3 and 5, and $\alpha$ at 0.9. Picking exponential spacing over uniform spacing seemed to have a greater impact on the energy used than selecting the particular learning and share rate from a reasonable range. Although the $\eta$ and $\alpha$ parameters must be carefully chosen for the worst-case bound proofs, for practical purposes the choice of these parameters tended to have only a small impact on the algorithm's performance. This observation is in agreement with other empirical work on multiplicative weight algorithms [1].

### 5.7. Number of experts

We examine how the share algorithm performs when the number of experts varies, as the running time of the algorithm is proportional to the number of experts used. Figure 8 shows that for uniform spacing there is a noticeable jump in energy usage between 5 and 10 experts with an average energy increase of 1.5%. However, the difference between 10 and 100 experts is small, only about 0.4%.

We observe similar behavior when using exponentially spaced experts with a base of 1.5. However, if the base is

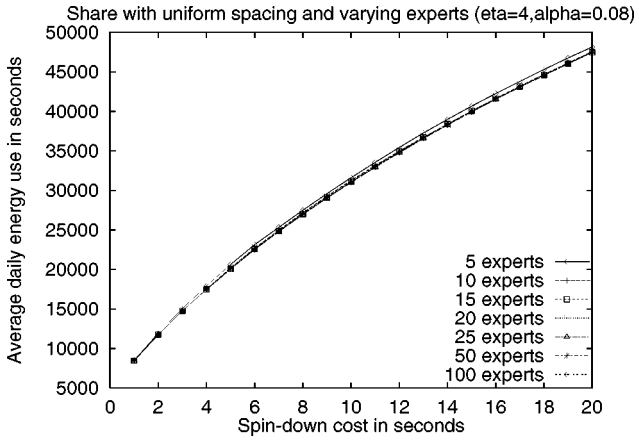


Figure 8. Share algorithm with uniform spacing and varying number of experts.

larger (2.0) then the difference between 5 and 10 experts drops to only 0.1%. These results lead us to believe the share algorithm can be reliably configured to use as few as 10 experts, allowing the algorithm to run faster and use fewer resources.

### *5.8. Adding other adaptive algorithms*

We also attempt to improve performance by using adaptive experts in addition to the fixed constant experts. We consider a set of incrementally adaptive algorithms first used by Douglis et al. [3] (see section 2). We use four variations of this algorithm: for "long" idle times, we either decrease the time-out by subtracting 1 s or by halving it, and for "short" idle times, we either increase the time-out by adding 1 s or doubling it. We add each of the four adaptive algorithms singly and in combination to the pool of constant value experts in the share algorithm with both uniform and exponential spacing. When added singly, none of the four algorithms makes a noticeable difference in the average daily energy usage. The "decrease by half, increase by adding one" algorithm has the most noticeable effect, but even it decreased the average daily energy used over all spin-down costs by only 0.2%. Similarly, when we look at the effect of including all four adaptive algorithms for uniformly spaced constant experts, the average improvement in energy is only 0.5%, with most of this improvement in the lower spin-down costs.[4]

A similar pattern appears when using exponential base 1.5 spacing for the pool of constant value experts, though the improvement in energy usage is slightly higher, averaging 1.6% over all spin-down costs.[5]

[4] For spin-down costs between 1 and 7, the average improvement is 1.6%, but for spin-down costs between 8 and 14, the most likely costs for real disk usage, this improvement is only 0.2%. For spin-down costs from 15 to 20, the energy usage *increases* by 0.3% when the adaptive algorithms are added.

[5] For spin-down costs between 1 and 7, the average improvement is a noticable 4.3%, but for spin-down costs between 8 and 14, the improvement shrinks to 0.5%. For spin-down costs from 15 to 20, the energy usage *increases* by 0.2% when the adaptive algorithms are added.

From this we conclude that adding adaptive algorithms makes little difference in the performance of the algorithms for most spindown costs, and in certain situations, may even increase the amount of energy consumed. Since the adaptive algorithms add complexity, there is no significant benefit in including them in the pool of experts, unless the spindown cost is very small.

## 6. Theoretical justification

The best fixed time-out could be any of the idle times in the trace, so straightforward techniques for computing the best fixed time-out require order $n^2$ calculations, where $n$ is the number of idle times in the trace. A day's trace data often contains millions of idle times, ruling out this approach. Instead, we use an approximation to the cost of the best fixed time-out in the previous section. Our approximation method calculates the energy use of 10,000 time-outs evenly spaced between 0 and 100 s on each day of the trace. Because the energy cost is a discontinuous and nonmonotonic function of the time-out, it is not immediately clear that this method closely approximates the energy used by the best fixed time-out. In this section we bound the error in our approximation.

There are two possible reasons why the approximation may severely overestimate the energy used by the best fixed cutoff. First, the best fixed cutoff may be larger than 100 seconds. We upper bound the value of the best fixed time-out in section 6.1 to ensure that our approximation method uses large enough time-outs. Second, the best fixed time-out is likely to fall between two of the time-outs analyzed. We bound the error due to this effect in section 6.2.

### *6.1. Maximum possible best fixed time-out*

In our experiments, the share algorithm uses experts whose time-out values are no larger than the spin-down cost. This natural choice[6] is appropriate for the trace data available, but may not be adequate in general. Here we determine analytically a tight upper bound on the best fixed time-out in terms of the length (in multiples of the spin-down cost) spanned by the sequence of disk requests. In particular, if the sequence of idle times is $t$ spin-down costs long then there is a best fixed time-out at most $H_t$, the $t$th harmonic number. We show that this upper bound can be achieved, and that every sequence of disk accesses has a best fixed time-out no greater than this upper bound. We present the analysis for a spin-down cost of one second. This analysis can be generalized by re-scaling the time unit.

We use particular sequences of idle times in our analysis. We call these idle times the *harmonic idle times* to emphasize their connection with the harmonic numbers, $H_n$ ($H_n = \sum_{i=1}^{n} 1/i \approx \ln n$, and $H_0 = 0$). In particular, the

[6] It is easy to see that when a single idle time is considered, the best time-out value is either 0 (if the idle time is greater than the spin-down cost) or the spin-down cost (if the idle time is smaller).

harmonic idle times of order $t$ are the set of $t$ idle times $\{h_{0,t}, h_{1,t}, \ldots, h_{t-1,t}\}$, where each $h_{i,t} = H_t - H_i$. Note that $h_{0,t}$ is the largest idle time in the set and $h_{t-1,t}$ is the smallest.

We show in the appendix (theorem 1) that the time-out $H_t$ is a best fixed time-out for the harmonic idle times of order $t$. By perturbing the harmonic idle times we get a sequence of idle times of length $t-\varepsilon$ (for any $\varepsilon > 0$) whose smallest best fixed time-out is $H_t - \varepsilon$. Furthermore, we use these harmonic idle times to show that every sequence of idle times of length $t$ has a best fixed time-out less than $H_t$ (theorem 2). This implies that the best fixed time-out for a 24 h trace is less than 100 s when the spin down cost is 10 or less.

### 6.2. Bounding the overestimation of the best fixed time-out's cost

As noted earlier, computing the exact cost of the best fixed time-out is prohibitively expensive, so we overestimate of this cost. In this section we show that the estimated cost is not more than 0.05% above the actual cost of the best fixed time-out for spin-down costs above 6, and always within 0.25% of the best fixed time-out's cost.

Assume that we have computed the cost of two time-outs, $t_0$ and $t_g = t_0 + g$ (we use $g = 1/100$ of a second). Let $t_\varepsilon = t_0 + \varepsilon$ for some $0 \leqslant \varepsilon < g$ be the best time-out between $t_0$ and $t_g$. We use information from the trace to bound the cost of $t_\varepsilon$ in terms of the cost of $t_0$. We prove our bound only for case where $t_0 = 0$. The generalization to $t_0 > 0$ is straightforward once one realizes that time-outs $t_\varepsilon$, $t_0$, and $t_g$ all keep the disk spinning until the idle time exceeds $t_0$.

We define some additional notation before proceeding:

- $n_1$ is the number of idle times falling between $t_0$ and $t_\varepsilon$.
- $n_2$ is the number of idle times falling between $t_\varepsilon$ and $t_g$.
- $n$ is the number of idle times falling between $t_0$ and $t_g$, $n = n_1 + n_2$.
- $l_1$ is the sum of the lengths of the $n_1$ idle times falling between $t_0$ and $t_\varepsilon$.
- $l_2$ is the combined length of the $n_2$ idle times falling between $t_\varepsilon$ and $t_g$.
- $l$ is the combined length of the $n$ idle times falling between $t_0$ and $t_g$, $l = l_1 + l_2$.
- $m$ is the total number of idle times that are larger than $t_g$.
- $s$ is the spin down cost, we assume that $s > g$.

The situation we consider is shown in figure 9.

We want a bound on the cost of $t_\varepsilon$ that depends only on information easily computed from the trace. In particular, our bound depends only on $n$, $m$, $l$, and the cost of $t_0$. We assume nothing about the distribution of lengths of the $n$ idle times, other than that their total length is $l$.

The cost of time-outs $t_0$ and $t_\varepsilon$ can be related as follows. On the $n_1$ idle periods between $t_0$ and $t_\varepsilon$, time-out $t_\varepsilon$ keeps the disk spinning. This saves $n_1$ spin-downs at the cost of $l_1$ extra spin time compared with time-out $t_0$. On each of the $n_2 + m$ longer idle times, time-out $t_\varepsilon$ waits an extra $\varepsilon$ seconds before spinning down the disk. This costs $\varepsilon(n_2 + m)$ more seconds of energy compared with $t_0$. Thus,

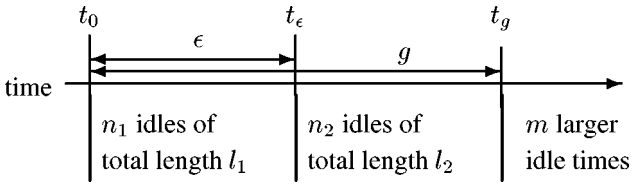


Figure 9. Notation for section 6.2.

$$\begin{aligned}\text{cost}(t_\varepsilon) &= \text{cost}(t_0) - n_1 s + l_1 + \varepsilon(n_2 + m)\\ &= \text{cost}(t_0) - (n - n_2)s + l - l_2 + \varepsilon(n_2 + m).\end{aligned}$$

Recall that $l_2$ is the total length of the $n_2$ idle periods with lengths between $\varepsilon$ and $g$, so $l_2 \leqslant g n_2$. Continuing with this substitution we have

$$\begin{aligned}\text{cost}(t_\varepsilon) &> \text{cost}(t_0) - (n - n_2)s + l - g n_2 + \varepsilon(n_2 + m)\\ &> \text{cost}(t_0) + l + \varepsilon m - ns + n_2(s - g + \varepsilon). \quad (1)\end{aligned}$$

We assumed that $s > g$, so the factor multiplying $n_2$ is positive. We consider two cases based on the value of $\varepsilon$.

First, if $\varepsilon \geqslant l/n$ then we can underestimate $n_2$ by 0 to obtain

$$\begin{aligned}\text{cost}(t_\varepsilon) &> \text{cost}(t_0) + l + \varepsilon m - ns\\ &> \text{cost}(t_0) + l + \frac{lm}{n} - ns. \quad (2)\end{aligned}$$

For the second case, $0 \leqslant \varepsilon \leqslant l/n$. As $l_2 \leqslant g n_2$,

$$n_2 \geqslant \frac{l_2}{g} \geqslant \frac{l - l_1}{g} \geqslant \frac{l - \varepsilon n}{g}.$$

In the last line we used the fact that $l_1$ is the combined length of the $n_1$ idle times, each of which is less than $\varepsilon$ long, so $l_1 \leqslant \varepsilon n_1 \leqslant \varepsilon n$. Substituting this in inequality (1) yields

$$\begin{aligned}\text{cost}(t_\varepsilon) &> \text{cost}(t_0) + l + \varepsilon m - ns + \frac{l - \varepsilon n}{g}(s - g + \varepsilon)\\ &> \text{cost}(t_0) - ns + \frac{ls}{g} + \varepsilon\left(m + n - \frac{ns - l}{g}\right)\\ &\quad - \varepsilon^2 \frac{n}{g}.\end{aligned}$$

This last bound is quadratic in $\varepsilon$ with a negative second derivative, so it is minimized at one of the endpoints, $\varepsilon = 0$ or $\varepsilon = l/n$. Thus, after cancelation,

$$\begin{aligned}&\text{cost}(t_\varepsilon)\\ &> \min\left[\text{cost}(t_0) - ns + \frac{ls}{g},\ \text{cost}(t_0) - ns + \frac{lm}{n} + l\right]. \quad (3)\end{aligned}$$

The bound of inequality (2) (the first case) is exactly equal to the second term of the minimum, so the cost of any $t_\varepsilon$ from either case is underestimated by inequality (3).

For the disk traces used in most of our results, inequality (3) verifies that we never overestimate the cost of the best fixed time-out by more than 0.25%. Inequality (3) tends to be weakest for smaller spin-down costs. When the spin-down cost is at least 6, then our estimates of the best fixed time-out's costs are no more than 0.05% (one-twentieth of one percent) greater than its actual value on any particular day.

## 7. Future work

Several variations on the share algorithm have the potential to dramatically improve its performance. In particular, we are interested in better methods for selecting the algorithms learning rate and share rate. It may be possible to derive simple heuristics that provide reasonable values for these parameters. A more ambitious goal is to have the algorithm self-tune these parameters based on its past performance. Although cross validation and structural risk minimization techniques [21] can be used for some parameter optimization problems, the on-line and time-critical nature of this problem makes it difficult to apply these techniques.

One issue that deserves further study is how the additional latency imposed by disk spin-down impacts the user and their applications, but this is difficult to quantify. A second important question is how algorithms like the share algorithm perform on other power management and systems utilization problems, such as transceiver power management.

## 8. Conclusions

We have shown that a simple machine learning algorithm is an effective solution to the disk spin-down problem. The algorithm adapts to the pattern of recent disk activity, exploiting the bursty nature of user activity. This algorithm performs better than all other known algorithms, using less than half the energy consumed by a standard one minute time-out. The algorithm even outperforms the best fixed time-out that requires knowledge of the future.

Other algorithms for the disk spin-down problem either make worst-case assumptions or attempt to approximate the best fixed time-out over an entire sequence. Although these algorithms have good worst case bounds, they do not necessarily perform well in practice. Our simulations show that our learning algorithm outperforms the worst-case algorithms.

We believe that the disk spin-down problem is just one example of a wide class of rent-to-buy problems for which our learning algorithm is well suited. Other problems in this class are of importance to mobile computing, such as: power management of a wireless interface, admission control on shared channels, and a variety of other power management problems. Other problems where the algorithm can be applied include: deciding when a thread that is trying to acquire a lock should busy-wait or context switch and computing virtual circuit holding times in IP-over-ATM networks [13].

Our implementation of the share algorithm is efficient, taking taking constant space and constant time per trial. This constant is adjustable, and adjusts the accuracy of the algorithm. For the results presented in this article, the total space required by the algorithm is never more than about 2400 bytes, and our implementation in C requires about 100 lines of code. This algorithm could be implemented on a disk controller, in the BIOS, or in the operating system.

## Acknowledgements

We are deeply indebted to J. Wilkes, R. Golding, and the Hewlett-Packard Company for making their file system traces available to us. We are also grateful to M. Herbster and M. Warmuth for several valuable conversations, and the anonymous reviewers for their many helpful comments.

## Appendix. Theoretical justification for harmonic idle times

We first verify that the harmonic idle times of order $t$ have total length equal to $t$.

$$\sum_{i=0}^{t-1} h_{i,t} = \sum_{i=0}^{t-1} (H_t - H_i) \tag{A.1}$$

$$= tH_t - \sum_{i=0}^{t-1} H_i \tag{A.2}$$

$$= tH_t - tH_t + t \tag{A.3}$$

$$= t. \tag{A.4}$$

To get line (A.4) requires the following identity,

$$\sum_{i=0}^{k-1} H_i = kH_k - k, \tag{A.5}$$

which is easily proven by induction.

We use two simple facts about best fixed time-out times in our analysis. First, adding an idle interval of length 0 to a set of idle times never changes the set of the best fixed time-outs. This is because the additional idle time adds the same cost, namely 0, to all fixed time-outs. The second fact is given in the following lemma.

**Lemma 1.** Every best fixed time-out is either 0, one of the idle times, or greater than any of the idle times.

*Proof.* By contradiction. Suppose that a sequence of idles times has a best fixed time-out $c$ that is positive, less than

the largest idle time in the sequence, and not equal to any of the idle times in the sequence. From the first fact we can assume that there is a zero-length idle time in the sequence, so the sequence contains at least one idle time less than $c$. We now consider the time-out $c'$ equaling the largest idle time in the sequence that is smaller than $c$. The costs of time-outs $c$ and $c'$ are the same on those idle times less than $c$ (recall that the disk drive is spun down only if the idle time exceeds the time-out). Furthermore, the cost of $c'$ is strictly less than the cost of $c$ on the larger idle times. Therefore, $c'$ is a strictly better fixed time-out than $c$, and $c$ is not a best fixed time-out. □

We are now ready to show that the best fixed time-outs for the harmonic idle periods of order $t$ have cost equal to $t$.

**Theorem 1.** When the sequence of idle times is the harmonic idle periods of order $t$, every fixed time-out has cost at least $t$.

*Proof.* From lemma 1 above, we need only show that time-outs equal to idle times in the sequence have cost at least $t$. Consider a time-out $c$ equal some $h_{i,t}$ for an arbitrary $0 \leqslant i < t$. The cost of this time-out $c$ is

$$\begin{aligned}
&\sum_{j=0}^{i-1}(1+h_{i,t}) + \sum_{j=i}^{t-1} h_{j,t} \\
&\quad = i(1+h_{i,t}) + \sum_{j=0}^{t-1} h_{j,t} - \sum_{j=0}^{i-1} h_{j,t} \\
&\quad = i(1+H_t - H_i) + t - \sum_{j=0}^{i-1}(H_t - H_j) \\
&\quad = i + iH_t - iH_i + t - iH_t + \sum_{j=0}^{i-1} H_j \\
&\quad = i - iH_i + t + iH_i - i = t.
\end{aligned}$$

We used the fact that the sum of the idle times is $t$ in the third line, and equation (A.5) in the fourth line. □

By slightly perturbing the harmonic idle periods of order $t$ (i.e., decreasing the length of the longest one by some $\varepsilon$) we obtain a sequence of idle times spanning $t - \varepsilon$ seconds where the best fixed time-outs are all at least $H_t - \varepsilon \approx \ln t$. The following theorem shows that every sequence of idle times spanning no more than $t$ time units has a best fixed time-out that is at most $H_t$.

**Theorem 2.** If $S = \{i_0, i_1, \ldots, i_{n-1}\}$ is a non-empty sequence of idle times where $\sum_{j=0}^{n-1} i_j \leqslant t$, then $S$ has a best fixed time-out less than $H_t$.

*Proof.* We assume to the contrary that some non-empty sequence of idle times $S$ with total length at most $t$ has no best fixed time-out less than or equal to $H_t$. Without loss of generality, we index the $n$ idle times in $S$ such that $i_0 \geqslant i_1 \geqslant \cdots \geqslant i_{n-1}$. Because additional length zero idle times do not affect the best fixed time-outs, we assume that $S$ contains at least $t$ idle times (so $n \geqslant t$).

Furthermore we assume that the smallest best fixed time-out for $S$ is equal to $i_0$. This assumption is valid because if some $i_j$ is the smallest best fixed time-out for $S$ then $i_j$ remains the smallest best fixed time-out for the new sequence of idle times created when all longer idle times ($i_0$, $i_1, \ldots, i_{j-1}$) are reduced to $i_j$ as the costs of time-out $i_j$ and all smaller time-outs are unchanged. The new sequence preserves the original properties of $S$ as well as satisfying this last assumption.

Note that $i_0$, the longest idle time in $S$, is greater than $H_t = h_{0,t}$. Let $k$ be the largest index such that $i_k > h_{k,t}$. Because $\sum_{j=0}^{t-1} h_{j,t} = t \geqslant \sum_{j=0}^{t-1} i_j$, there will be some $j < t$, where $i_j \leqslant h_{j,t}$.

We now consider the cost of the fixed time-out $h_{k,t}$ on sequence $S$. Note that for each idle time $i_j$, if $j < k$ then the disk will be spun down after waiting $h_{k,t}$ time units, and if $j \geqslant k$ then the disk will remain spinning for the duration of the idle time. Therefore,

$$\begin{aligned}
\text{cost}_{h_{k,t}}(S) &= k(h_{k,t}+1) + \sum_{j=k}^{n-1} i_j \\
&= kH_t - kH_k + k + \sum_{j=k}^{n-1} i_j \\
&= kH_t - \sum_{i=0}^{k-1} H_i + \sum_{j=k}^{n-1} i_j \\
&= \sum_{j=0}^{k-1}(H_t - H_j) + \sum_{j=k}^{n-1} i_j \\
&= \sum_{j=0}^{k-1} h_{j,t} + \sum_{j=k}^{n-1} i_j \leqslant \sum_{j=0}^{n-1} i_j \qquad \text{(A.6)}
\end{aligned}$$

and the cost of time-out $h_{k,t}$ is less than the total length of the sequence $S$. But no time-out can have a cost less than a best fixed time-out, and the cost of the best fixed time-out $i_0$ on $S$ is simply the total length of $S$, because the disk is always kept spinning. This contradicts inequality (A.6). □

## References


[1] A. Blum, Empirical support for Winnow and weighted-majority-based algorithms: Results on a calendar scheduling domain, in: *Proc. of the 12th Internat. Conf. on Machine Learning* (Morgan Kaufmann, San Mateo, CA, 1995) pp. 64–72.

[2] N. Cesa-Bianchi, Y. Freund, D. Haussler, D.P. Helmbold, R.E. Schapire and M.K. Warmuth, How to use expert advice, Technical Report UCSC-CRL-94-33, University of California, Santa Cruz (1994).

[3] F. Douglis, P. Krishnan and B. Bershad, Adaptive disk spin-down policies for mobile computers, in: *Proc. of the 2nd Usenix Sympo-*

*sium on Mobile and Location-Independent Computing*, Ann Arbor, MI (Usenix Association, April 1995).
[4] F. Douglis, P. Krishnan and B. Marsh, Thwarting the power-hungry disk, in: *Proc. of the Usenix Technical Conf.*, San Francisco, CA (Usenix Association, 1994) pp. 292–306.
[5] R. Golding, P. Bosch, C. Staelin, T. Sullivan and J. Wilkes, Idleness is not sloth, in: *Proc. of the Usenix Technical Conf.*, New Orleans (Usenix Association, January 1995) pp. 201–212.
[6] R. Golding, P. Bosch and J. Wilkes, Idleness is not sloth, Technical Report HPL-96-140, Hewlett Packard Computer Systems Laboratory (1996).
[7] K. Govil, E. Chan and H. Wasserman, Comparing algorithms for dynamic speed-setting of a low-power cpu, in: *The 1st Annual Internat. Conf. on Mobile Computing and Networking (MobiCom)*, Berkeley, CA (ACM, 1995) pp. 13–25.
[8] P. Greenawalt, Modeling power management for hard disks, in: *Proc. of the Conf. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (IEEE, January 1994) pp. 62–66.
[9] D. Helmbold, D. Long and B. Sherrod, A dynamic disk spin-down technique for mobile computing, in: *Proc. of the 2nd Annual ACM Internat. Conf. on Mobile Computing and Networking* (ACM/IEEE, November 1996).
[10] M. Herbster and M.K. Warmuth, Tracking the best expert, in: *Proc. of the 12th Internat. Conf. on Machine Learning*, Tahoe City, CA (Morgan Kaufmann, San Mateo, CA, 1995) pp. 286–294.
[11] A. Karlin, M. Manasse, L. Rudolph and D. Sleator, Competitive snoopy caching, in: *Proc. of the 27th Annual IEEE Symposium on the Foundations of Computer Science*, Toronto (ACM, October 1986) pp. 224–254.
[12] A. Karlin, M.S. Manasse, L.A. McGeoch and S. Owicki, Competitive randomized algorithms for non-uniform problems, in: *Proc. of the ACM-SIAM Symposium on Discrete Algorithms* (1990) pp. 301–309.
[13] P. Krishnan, P. Long and J.S. Vitter, Adaptive disk spin-down via optimal rent-to-buy in probabilistic environments, in: *Proc. of the 12th Internat. Conf. on Machine Learning (ML95)*, Tahoe City, CA (Morgan Kaufman, San Mateo, CA, July 1995) pp. 322–330.
[14] K. Li, R. Kumpf, P. Horton and T. Anderson, A quantitative analysis of disk drive power management in portable computers, in: *Proc. of the Usenix Technical Conf.*, San Francisco (Usenix Association, 1994) pp. 279–291.
[15] N. Littlestone, Learning when irrelevant attributes abound: A new linear-threshold algorithm, Machine Learning 2 (1988) 285–318.
[16] N. Littlestone and M.K. Warmuth, The weighted majority algorithm, Information and Computation 108(2) (1994) 212–261.
[17] M.K. McKusick, W.N. Joy, S.J. Leffler and R.S. Fabry, A fast file system for UNIX, ACM Transactions on Computer Systems 2 (August 1984) 181–197.
[18] C. Ruemmler and J. Wilkes, UNIX disk access patterns, in: *Proc. of the Usenix Technical Conf.*, San Diego, CA (Usenix Association, 1993) pp. 405–420.
[19] D.D. Sleator and R.E. Tarjan, Amortized efficiency of list update and paging rules, Communications of the ACM 28 (February 1985) 202–228.
[20] V. Vovk, Aggregating strategies, in: *Proc. of the 3rd Annual Workshop on Computational Learning Theory*, Rochester, NY (Morgan Kaufmann, San Mateo, CA, 1990) pp. 371–383.
[21] V. Vapnik, *Estimation of Dependencies Based on Empirical Data* (Springer, Berlin, 1982).
[22] M. Weiser, B. Welch, A. Demers and S. Shenker, Scheduling for reduced CPU energy, in: *Proc. of the 1st Symposium on Operating Systems Design and Implementation (OSDI)*, Monterey, CA (Usenix Association, November 1994) pp. 13–23.
[23] J. Wilkes, Predictive power conservation, Technical Report HPL-CSP-92-5, Hewlett-Packard Laboratories (February 1992).

**David Helmbold** is a Professor in the Computer Science Department of the University of California, Santa Cruz. He received the B.A. degree from the University of California, Berkeley, in 1981 and the Ph.D. in computer science from Stanford University in 1987. His main research interests are in the area of algorithmic learning theory, and he is currently working on boosting methods for improving learning algorithms. Professor Helmbold was recently a member of the Computational Learning Theory (COLT) steering committee, and currently serves on the editorial board for Machine Learning.

**Darrell D.E. Long** is a Professor of Computer Science at the University of California, Santa Cruz. He also serves as Associate Dean in the Jack Baskin School of Engineering. He received his B.S. degree in computer science from San Diego State University in 1984, and his M.S. and Ph.D. degrees in computer science and engineering from the University of California, San Diego, in 1986 and 1988, respectively. His research interests include distributed systems, particularly high speed storage systems, fault tolerance, performance evaluation and mobile computing. He is the Director of the Concurrent Systems Laboratory which is part of the Jack Baskin School of Engineering. His research is supported by the Office of Naval Research, the Naval Research Laboratory, the National Science Foundation, the Usenix Association and by International Business Machines Corporation (IBM).

**Tracey L. Sconyers** received the Masters in computer science from the University of California, Santa Cruz, in 1998, and is now working at Alta Vista.

**Bruce Sherrod** received the Masters in computer science from the University of California, Santa Cruz, in 1997, and is now working in industry.