Adapting Predictions and Workloads for Power Management

Jeffrey P. Rybczynski   Darrell D. E. Long[†]
Storage Systems Research Center
University of California, Santa Cruz

Ahmed Amer[‡]
Department of Computer Science
University of Pittsburgh

Abstract

Power conservation in systems is critical for mobile, sensor network, and other power-constrained environments. While disk spin-down policies can contribute greatly to reducing the power consumption of the storage subsystem, the reshaping of the access workload can actively increase such energy savings. Traditionally reshaping of the access workload is a result of caches passively modifying the workload with the aim of increasing hit ratios and reducing access latency. In contrast, we present the a shifting predictive policy that actively reshapes the workload with the primary goal of conserving disk energy consumption. By reshaping the disk workload to explicitly lengthen idle periods, the disk can remain spun-down longer, saving more energy. We show that our approach can save up to 75% of disk energy compared to the common fixed-timeout spin-down policies. Our shifting algorithm dynamically shifts to the most energy efficient cache prefetching policy based on the current workload. This best shifting *prefetching policy is shown to use 15% to 35% less energy than traditional disk spin-down strategies and 5% to 10% less energy than the use of a fixed (non-shifting) prefetching policy.*

Keywords: Power-Aware Computing, Energy Conservation, Access Prediction, Disk Management.

1 Introduction

The use of mobile computers has become increasingly pervasive. With the popularity of wireless networks, the number of locations with network access has greatly increased. Allowing mobile users to stay connected via such networks increases communication and productivity. Such mobile applications, and also sensor networks, depend heavily on the ability to operate on battery power. While laptop batteries have greatly improved, they still only allow for a short time period of disconnected operation before requiring a power source (*i.e.* a wall outlet). Depending on system usage patterns, system components, and peripherals, a laptop user can get anywhere from less than an hour up to 5 hours of system operation per battery charge. Even with five hours of usable power, it is still advantageous to use energy conservation strategies. Along with lengthening the duration of time one can use a laptop disconnected from a continuous source of power, with less power being used batteries could be lighter, and the total energy cost to operate the computer would be lower.

Many innovations have been proposed to conserve system power for laptops, such as slowing down processor and bus speeds under light system use, dimming the display and adding low power states, along with more integrated solutions that are designed for mobile and low-power environments. While processors are still the main consumer of system power, it has been shown that the hard disk can use up to 30% of the total system energy [13], making the disk subsystem a prime candidate for energy conservation.

[†]Supported in part by the National Science Foundation under award CCR-0204358

[‡]Supported in part by the National Science Foundation under award ANI-0325353.

2 Disk Energy Conservation

Disk systems use a significant amount of energy. Unlike most electronics in a computer, the disk has mechanical components. The spinning disk platters and the actuator arm require a considerable amount of energy to operate. To use the disk, the disk's platters need to be spun at the proper rate (4200–5400 RPMs for typical laptop drives [10]) by a spindle motor. The constant spinning of the disk to keep it ready for incoming I/O requests is a significant waste of energy if there are no current requests. Conserving energy at the disk level can be done by effectively turning the disk (or parts of it, such as the spindle motore or electronics) off. Most disk drives typically have one or mores spin-down, or standby states. In such a state the disk platters are spun down and most other non-essential electronics are turned off. Such a spin-down state can only be used if the disk is not needed and no requests are pending. We will consider two distinct states of disk operation, the *on* or *spinning*

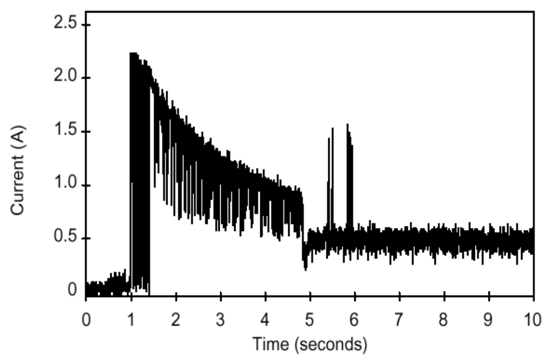


Figure 1. This graph shows the amount of power used by a used by a 12 volt IDE disk during a spin-up (starting at time 1) and during normal operation (starting after time 5.). Source: Seagate datasheet.

state for normal disk I/O operations, and the *spin-down* or *standby state* for when the disk is turned off and idle.

If a disk I/O request is sent to the disk while in this spun down state, the disk must "wake up" by bringing the platters up to the proper speed, and servicing the incoming request. The transition to an active state introduces new overheads. Spinning up the disk requires about 1.5 to 3 times more energy than the energy used to keep the disk spinning for the same amount of time, as shown in Figure 1 [30]. Getting the platters up to speed also takes time, which translates to access latency for the user (3 to 9 full seconds, depending on the drive [10, 30]). Thirdly, disks are rated for a specific number of cycles, where one spin up and one spin down equal one cycle. While most laptop disks have a failure rating on the order of half a million cycles, IDE desktop drives are only rated for around 50,000 cycles [30]. Excessive spin-down of the disks can cause premature disk failure. Given these observations, the selection of an effective spin-down strategy is crucial to the efficient, reliable, and energy-conserving operation of the disk subsystem.

In order to save energy we must spin down the disk, but knowing when to do so is the real issue. Since we do not know when disk I/O requests will arrive, we do not know when the disk will be used in the future. If we aggressively spin down for short idle periods and have disk requests arriving shortly after, we will not only incur the added energy cost for spinning the disk back up again, but will add significant access latencies and physical wear to the drive. Ideally, we need to spin down the disk only when the idle periods are long enough to make it worthwhile, and we would go further and express the need to actively create longer idle periods whenever possible.

2.1 Disk Energy Consumption

While different disks offer different active, idle, or standby states, for generality we will assume two basic states: active and standby. Equation 1 shows how the energy consumption of the disk will be determined. As different disk drives use differing amounts of energy for each state. We will assume that the disk uses one unit of active (spinning) disk energy for every second the disk is active. For the standby state we assume that the disk uses zero units of energy per second while spun down. Due to differences in actual hard disk specifications of the extra energy cost caused by the spin-down processes, we define this as a parameter of our model. Douglis *et al.* [7] and Golding *et al.* [11] have also used disk spin-down costs computed in seconds. Since most hard disks take 3 to 7 seconds to spin up to the proper speed and use 1.5 to 3 times the amount of normal active energy in this spin up state, this value is typically between 5 and 15 seconds (units) of energy depending on the hard disk model. This means that for a spin-down to be worth the energy cost it takes to spin the disk back up, the disk needs to remain spun down for at least 5 to 20 seconds depending on the individual hard disk's specifications and energy usage. For our tests, we use a spin down cost of 10 units of energy, corresponding to a spin up that takes 5 seconds and uses twice the normal active energy, which we find to be typical of current disk drives [10, 30].

$$E_{Total} = (C_{up} \times T_{up}) + (C_d \times T_d) + (C_s \times S) \qquad (1)$$

$$\begin{aligned}
E_{Total} &= \text{Total Energy} \\
C_{up} &= \text{Cost of Spinning (in seconds)} \\
C_d &= \text{Cost of Not Spinning (in seconds)} \\
C_s &= \text{Cost of a Spin Up (in seconds)} \\
T_{up} &= \text{Time disk was spinning} \\
T_d &= \text{Time disk was not spinning} \\
S &= \text{Number of Spin up cycles}
\end{aligned}$$

2.2 Active Workload Reshaping

Disk request patterns are bursty by nature, with periods of activity and idle periods of little to no activity. To further increase the energy savings of a disk spin-down policy, we need to lengthen idle periods, allowing the disk to remain spun down for longer. This requires actively reshaping the disk workload. This is done by moving the individual disk requests to either earlier or future burst periods. By moving these disk accesses to points in time where the disk is already active, we are effectively batching these requests together, thus creating busier burst periods and longer idle periods. To clarify how this movement is achieved, and when it is feasible, we must first understand the different types of disk accesses. Disk write and disk reads must be treated differently.

2.2.1 Read & Write Requests

For a write disk request, it is desirable to write the data to disk as soon as possible. To ensure data reliability, we do not want to cache the writes in the volatile write buffer for excessively long periods of time. It is typical to wait for a fixed period of time before writing back modified data, to avoid unnecessary writing of data that might be updated soon. If we wait for a variable amount of time, based on the current state of the disk, then we can postpone these requests to a future point in time, thus extending the current idle period [33].

For example, if our disk is in the active state and a write request arrives, then we should service it immediatly, instead of holding it in a write buffer, where it would only need to be written to the disk later (at which point the disk could already be spun down). Conversely, if the disk is in the standby state, spun down, then we do not want to spin the disk up immediately, but would rather postpone the write as much as our flexible write-back period will allow, or until a spin-up is necessitated by a demand for data (a read operation). By allowing flexible write buffer time-outs based on the current disk state, we can batch write requests together with not only other write requests, but also with read requests.

For a read disk request, the user or system needs the data immediately. This means that such requests cannot be postponed to the future, and must be serviced immediately. Read requests can normally only be moved backwards, to previous points in time, which is achieved by predicting and prefetching future file accesses.

2.2.2 Predictive Prefetching

The only way to move disk accesses earlier in time is to predict future accesses, prefetch, and then cache the data for future use. There are many different schemes for predicting future file accesses. Most of them use past file access behavior to predict future file accesses.

These prefetching policies are used to predict and prefetch possible future disk accesses. The prefetched files are then placed in the system cache, alongside normal demand fetched files. The cache then uses LRU to decide which files should be evicted. Thus prefetching incorrect files can adversely effect the performance of the cache and reduce the length of idle periods. That is why it is imperative to use prefetching policies that are accurate and effective. We have implemented six different prefetching policies. Each uses an underlying LRU cache eviction algorithm while predicting and prefetching files in accordance with their individual models.

The **Unmodified** caching policy predicts and prefetches nothing. It simply uses the Least Recently Used (LRU) cache eviction policy to store files as they are accessed. For some cases it can be shown that future disk accesses cannot be predicted accurately by any means.

Last Successor assumes that two files accessed in succession will be accessed in the same order in the future. For each file access the policy dynamically records the following file access, and when the first file is accessed again, its past successor is prefetched if not already in the cache. The predicted last successor is a dynamic successor due to the fact that a files last successor can change dynamically every time that particular file is accessed. It has been shown that last successor-based policies can be enhanced to increase their accuracy and reduce their likelihood of misprediction [35, 1, 2].

For each file in the **First Successor** model, we record the first file accessed after a given file access. Unlike the last successor model, this policy does not attempt to change its prediction of a particular file's successor once it has selected the initial one. While prefetching the first successor to a file may seem to be intuitively less accurate the last successor, Amer and Long [1] showed that prefetching first successors may be a better choice for certain workloads. In such workloads continuously changing the successor would lead to frequent deviation from a single good prediction.

Stability (Noah) is another successor-based predictor created by Amer *et al.* [1, 2]. It adds stability to the last successor model by avoiding the continuous switching of the predicted successor. Stability employs a general stability parameter, while Noah is the original form of this predictor which employs both static and dynamic predictions and a selector to choose which one to pick. Depending on which predictor the selector is pointing to, either the static predictor (first successor) is prefetched, or the dynamic (last successor) is prefetched. The selector uses past history to determine which prediction to choose.

The **Finite Multi-Order Context (FMOC)** model is a context modeling predictor, as described by Kroeger and Long [19]. It employs the use of a trie data structure [17] to keep track of file accesses and the context in which they were observed. Each node of the trie has a specific likelihood of being next in the sequence and the children of each node are the possible successors to that given the previous files. Each branch of the trie provides a context for current and future file accesses. FMOC prefetches files which have a likelihood of near-future access above a given threshold.

The **Extended Partition Context Model**, or **EPCM**, is a variant of the FMOC model and an extension of the earlier Partioned Context Model (PCM) [18, 20]. These models differ from FMOC in that they partition the trie into sections so that prediction metadata is more manageable, and so that files are not excessively prefetched, as they can be with FMOC.

2.3 Limits of Energy Savings

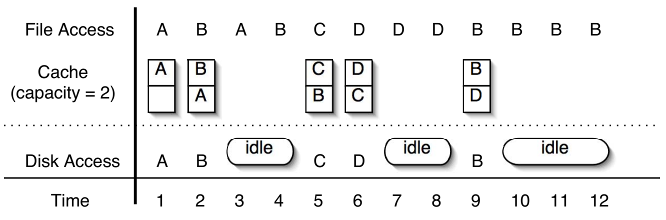
To allow our spin-down policy the longest idle periods, we need to optimally prefetch and batch together disk requests. To give us a minimum amount of energy required by a given workload, and to measure the effectiveness of our prefetching policies, we used a prefetching and spin-down oracle. The oracle is aware of all future accesses, and returns the best possible request batching and spin-down scheme to derive the best energy usage for a given trace. The optimal prefetching strategy can be broken into a smaller optimal problem plus the last prefetch, shown in Equation 2. The oracle uses dynamic programming to obtain the best possible ordering of disk requests and the ideal times at which to prefetch them. Simply prefetching the next *N* files (even if you are perfectly accurate) is not always the best strategy.

$$O_T = \min_i (P_i + O_{T-1}) \qquad (2)$$

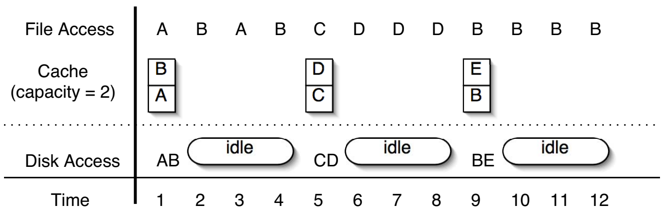
$$O_T = \text{Optimal Sequence at time T}$$
$$P_i = \text{Cost created by prefetching sequence } i$$

Assume that we have the file access pattern which contains files A, B, C and D and we are trying to create long disk idle periods so we can spin the disk down. Now assume the following file access sequence: ABABCDDDBBBB (shown in Figure 2) and that we have a cache with a capacity of 2 files. If we assume that all files are predictable, then by predicting the next files that fit in the cache will get us an access pattern as shown in Figure 2(b), with three idle periods of length three. As we can see from the sequence, if we fetch D and prefetch B towards the end of the trace, then we can reduce this to only two idle periods, one of length three and the other of length six (Figure 2(c)). The dynamic programming aspects allow the oracle to try every option for prediction and prefetching and find the optimal solution to batch together disk requests and maximize the idle period length between burst periods rather than relying on simply prefetching the next *N* files which as shown in Figure 2(b) may not always be optimal.

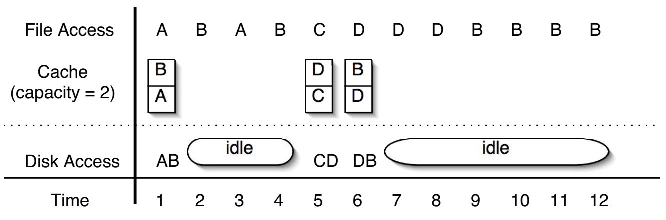
The oracle works on the principal that all file accesses are in one of three categories, either they are predictable, unpredictable, or may be delayed. Predictable files are files that have been seen before and if a prefetching algorithm was smart enough, could predict and choose to prefetch. An unpredictable file access, whether it is a write that needs to happen immediately or just a file that has not been seen before, are files that can never be predicted and will always result in a disk access. File accesses that can be delayed are a special case. These are file accesses whose resulting disk access can be postponed for at most a given time period. After that time period expires, if the file has yet to be written to disk or accessed, it becomes an unpredictable file access that must happen immediately. An example of a file accesses that can be postponed is a write request stored in a write buffer and waiting for a flexible time-out before being written to disk. This allows the file system to batch writes with other requests based on the current state of the disk so that they can all go to the disk at the same time, creating a busier burst period and a possibly longer idle period. Weissel *et al.* [33] have also shown that flexible write time-outs can be used to batch disk requests and save disk power.



(a) No prefetching



(b) Perfect greedy prefetching



(c) Optimal Prefetching

Figure 2. The initial file access pattern with even spaces between each access and the corresponding disk accesses for: no prefetching, greedy prefetching, and optimal prefetching.

Once the oracle reads in all the file accesses in the trace into a table and marks each of them as predictable, unpredictable, or postponable. It can then process the trace again and compute the new idle periods created. These idle periods are the maximum idle periods that could be found by the oracle. Once all the new idle periods are created, the oracle performs a second pass through the traces and finds the optimal path, or sequence, of disk accesses that results in the minimum energy usage. The oracle then uses backtracking to compute the series of disk accesses and prefetches that comprise the optimal sequence for that workload. This is our optimal disk access schedule, assuming 100% accurate prefetching for the trace. The oracle also optimizes for the spin-down policy. If the idle period is long enough to make a spin-down energy efficient, then the oracle assumes a spin-down occurs at the start of the idle period. This allows the oracle to represent not only an optimal prefetching and request batching strategy, but also an optimal spin down strategy as well. The energy value estimated by the oracle for the given workload is a strict lower bound on how much energy a given trace would require.

3 The *Best Shifting* Policy

Each of the implemented prefetching policies use past file events to predict future file accesses. Different prefetching policies work better for different workload patterns. The Figures 5, 6, 7, and 8 show how different prefetching policies work better for different days, using day long traces from instructional computers at the University of California, Berkeley [28], and traces taken from research systems (a Windows laptop and a desktop PC) at the University of California, Santa Cruz during early 2005 [29]. Both sets of traces demonstrate that a single prefetching policy does not always perform best. This is typical of all traces we have considered. Computers run many different programs and can have many different users who perform different file accesses depending on the task, causing workloads to change rapidly. Limiting yourself to one fixed prefetching policy is suboptimal because current workload determines the best policy.

Since it can be shown that different prefetching policies can perform best for different workloads, and that file access patterns change, it would be best to use an algorithm that would automatically switch to the best prefetching policy in response to the current workload. is the basis of our *Best Shifting* prefetching policy, which shifts which prefetching policy it uses based on which policy can save the most energy for the current file access workloads.

The *Best Shifting* policy uses machine learning techniques to choose which policy out of the six implemented (unmodified, last successor, first successor, Stability, FMOC, and EPCM) works best for the file access workload at that point in time. The Best Shifting policy dynamically chooses the best policy, not based upon hit ratio, but based on an energy savings metric.

To keep track of the performance of the different predictors, each policy has its own virtual cache, containing the files it would have in the cache if it was the system's prefetching policy. The virtual cache then allows us to derive which file accesses would cause disk accesses for the different policies. Each policy then stores these disk accesses in its own Disk Access Window, which is a snapshot of all the disk accesses a given policy would have created in the past *N* seconds had it been the system's prefetching policy. From this window, we can directly calculate our energy saving metrics.

3.1 Tracking Policy Performance

To keep track of the performance of each policy, the Best Shifting algorithm keeps virtual caches for each workload reshaping policy. Each prefetching policy has a virtual cache which keeps track of all the files that would be in the cache if the policy was currently being used by the system cache. By using virtual caches, the file accesses that need to go to disk can be derived for each prefetching policy. We then know the past disk access pattern and idle periods created by each of the different policies. This allows us to come up with the idle period length and approximate energy usage for the recent history of each policy. This information can then be used to decide which policy will give us the maximum energy conservation.

Virtual caches were used previously by Ari *et al.* [3] and Gramercy *et al.* [12]. Our strategy, however, is different. In previous work, the virtual caches were used to keep track of cache hit and miss ratios. These metrics were then used to determine which policy is the best and should be chosen for the current workload. Our algorithm uses these virtual caches to determine the disk accesses each policy would create had it been the system cache's policy. Then, keeping track of these disk accesses for each policy, we can derive the idle periods and an estimate of the energy used by the policy. The policy with the lowest energy saving metric is chosen as the winner and is adopted as the policy for the main cache.

To keep track for the disk accesses each policy creates, we use a Disk Access Window. For each policy, the disk accesses in the past *N* seconds, where *N* is a user defined variable, are kept track of in a window. Typical windows sizes range from 120 seconds (2 minutes) to 600 seconds (10 minutes). From this window of disk accesses, idle periods can be derived and estimations can be made on the energy usage. These Disk Access Windows allow us to compute energy saving metrics for each policy that we then use in our machine learning process. The metrics are: number of idle periods, potential energy savings, average idle period length and maximum idle period length. These metrics can be calculated directly from the idle periods in the Disk Access Window for each policy. These metrics are then used to compute the loss of each of the policies, which then is used to recompute the weight, using a multiplicative weight algorithm with a *share* update, of each policy and choose a winner.

3.2 Policy Evaluation & Selection

The Disk Access Window for each policy allows us to derive the energy saving metrics for each policy, which are used to compute the loss of the given policy. Equation 3 shows how the loss for a policy is computed using the metrics drawn from the Disk Access Window. If the policy has the best metric out of all the policies, then its loss will be 0. Otherwise, its loss will be anywhere from 0 to 1, depending on the performance of the prediction policy at moving future disk accesses and creating longer idle periods. This loss is then used to recompute the weights for each policy. We use a *share* update function [16], which is a member of the multiplicative weight family of algorithms. The update uses a Vovk exponential update [32] to first compute the next weight, reducing the weights of poorly performing experts, then shares some of the remaining weights equally with all the policies. It has parameters of α and η which are defined as the share parameter and the learning rate, respectively. Equation 4 shows how the new weights are computed using the loss for each policy and the share update functions.

$$Loss_i = \frac{\max(Metric) - Metric_i}{\max(Metric)} \tag{3}$$

$$W_i = W_i \times e^{-\eta Loss_i}$$

$$pool = \sum_{all i} (1 - (1-\alpha)^{Loss_i} \times W_i))$$

$$W_i = ((1-\alpha)^{Loss_i} \times W_i) + (\frac{1}{\text{\# of Policies}}) \times pool \tag{4}$$

These two parameters control the action of the update. The learning rate, η, controls how rapidly the weights of incorrect experts are decreased while the share parameter, α, controls how fast a previously poorly predicting expert recovers when its performance increases. In our application and previous works [15], the precise values of these parameters have little effect on the overall performance of the algorithm. To find the best values for η and α, tests where run on the traces to evaluate a range of values. The best values of these parameters were found to be approximately 0.08 for α and approximately 10.0 for η.

When the main cache's policy changes, there are two strategies that we can perform to realize this change. First we can simply change the policy without affecting the contents of the cache. This changes the way future predicted files will be prefetched, though it does nothing to the files that are already in the cache. The second strategy is to "roll over" the cache. This is the process of synchronizing the virtual cache of the winning policy with the main cache by prefetching files that are not in the system cache yet are in the winning policy's virtual cache. This operation, however, can take many disk accesses to perform, and so is only to be attempted if the disk is in the active state. If the cache policy changes while the disk is down, roll-over does not take place. If the disk is already active, then we can fetch the files we need to synchronize the cache in the background without causing an unnecessary disk spin-up. Roll-over can quickly help cache performance after the policy is switched due to a workload change that favors the new policy.

4 Experimental Results

To test our dynamic prefetching policy, Best Shifting, we used file system traces from a varied selection of sources. A cache emulator is employed to model the system cache whilst keeping track of the demand-fetched and prefetched files, along with cache statistics. If a trace entry asks for a file that is not in the cache, a disk request is created. Our cache emulator records the timing of file accesses that result in cache misses and require physical disk activity. This output is then used as the input to a spin-down algorithm, and workload's energy usage can then be calculated. For our tests, we used a cache emulator with a typical 30 second write buffer time-out. The output for this emulator was then run through a dynamic spin-down time-out algorithm, implemented as described by Helmbold *et al.* [15]. The decisions of the spin-down algorithm were then used to calculate the energy usage.

We applied our dynamic prefetching algorithm to traces from varied workloads. These included file traces of instructional systems at the University of California, Berkeley [28], and file system traces recorded from Windows systems at the University of California, Santa Cruz [29].

The bulk of single user systems today run Microsoft Windows as their operating system. To obtain typical workloads from a typical single user system, we decided to trace both desktop and laptop PCs. running Microsoft Windows XP. We collected these traces to serve as typical workload for our dynamic prefetching algorithm. These traces, were collected in January and February of 2005. Both desktop and laptop machines are single-user systems. The raw traces taken from these systems contain file activity in the form of file opens, and disk activity, in the form of disk read and write operations. Through post-processing of these traces, we derive an accurate picture of the file operations performed and the time at which they occurred, allowing us

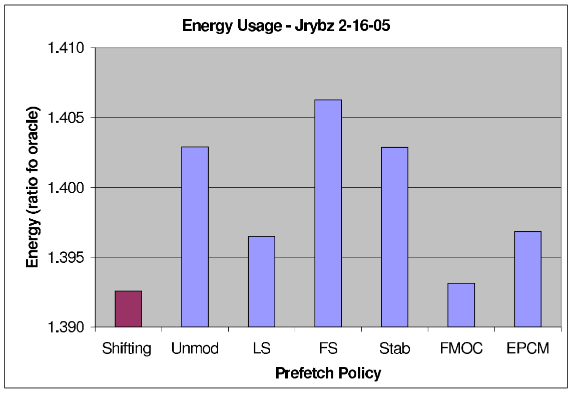


Figure 3. The energy usage for a fixed cache size, write buffer size and time-out,and spindown policy for mobile host *Jrybz* at the University of California, Santa Cruz on February 16, 2005.

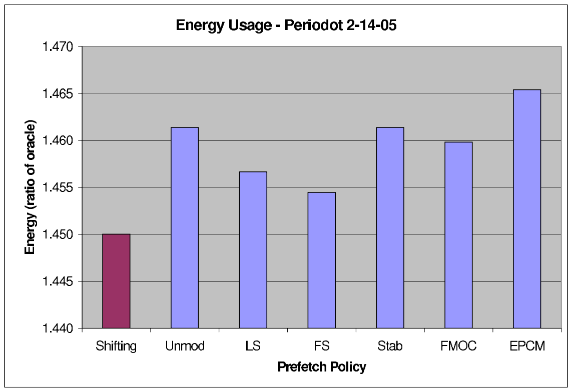


Figure 4. The energy usage for a fixed cache size, write buffer size and time-out,and spindown policy for host *Periodot* at the University of California, Santa Cruz on February 14, 2005.

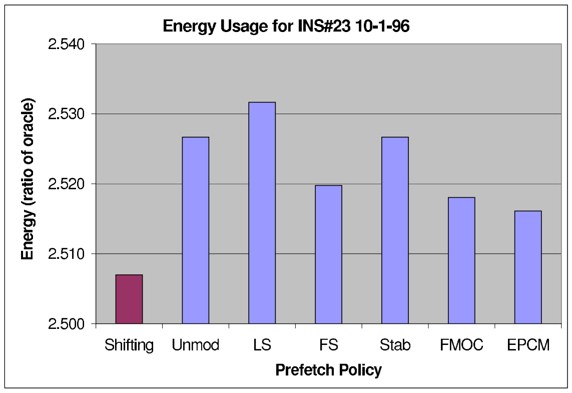


Figure 5. The energy usage for a fixed cache size, write buffer size and time-out,and spindown policy for host *INS#23* at the University of California, Berekely on October 1, 1996.

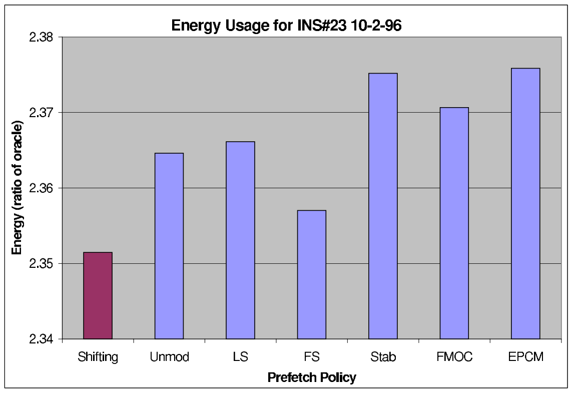


Figure 6. The energy usage for a fixed cache size, write buffer size and time-out,and spindown policy for host *INS#23* at the University of California, Berekely on October 2, 1996.

to have a workload containing file read and write operations to test our dynamic prefetching policy.

The Berkeley traces contain file system requests from three different workloads, graduate research computers, undergraduate instructional machines, and web servers. They were taken from machines in computer labs at the University of California, Berkeley campus from late 1996 to early 1997. Roselli and Anderson [28] used these traces in their argument for optimizing disk layout to improve read performance. The traces contain OPEN, EXEC, READ, WRITE, CREATE, and CLOSE file operations along with file size and user ID information for all file activity.

Host *INS#23* is one of many typical workloads we saw in the Berkeley traces. A Windows laptop PC, *Jrybz* and *Periodot*, a Windows Desktop PC, were used as file access workloads from the the University of California, Santa Cruz traces. Energy usage results for these workloads are shown in Figures 5 through 4.

Figures 5 through 8 and Figures 3 and 4, show energy usage our dynamic prefetching algorithm, which dynamically chooses the "best" prefetching policy for the given workload, based on energy usage. These results demonstrate that our algorithm produces disk behavior that is at worst the same in terms of energy usage as an *a priori* best fixed prefetching policy, and in most cases outperforms all the fixed prefetching policies. By dynamically choosing the "best" prefetching policy for the workload at a given point, we are able to modify the workload and increase the length of idle periods, allowing our algorithm to spin the disk down for longer periods of time, conserving more energy than with an unmodified trace.

While our algorithm combines workload reshaping with a dynamic spin-down algorithm, it is reasonable to con-

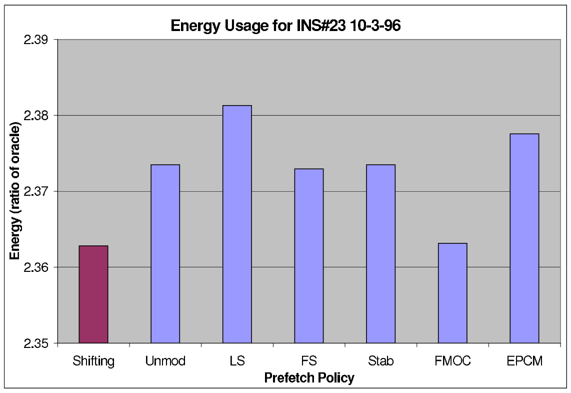


Figure 7. The energy usage for a fixed cache size, write buffer size and time-out,and spindown policy for host *INS#23* at the University of California, Berekely on October 3, 1996.

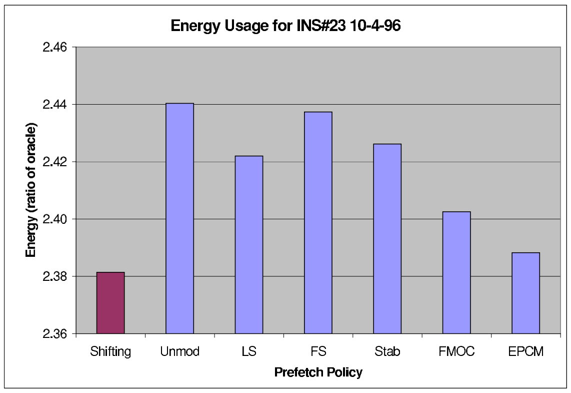


Figure 8. The energy usage for a fixed cache size, write buffer size and time-out,and spindown policy for host *INS#23* at the University of California, Berekely on October 4, 1996.

sider other combinations. Figure 9 shows the relative performance of our combination of prefetching and disk spin-down policies compared to other policy combinations. The value being represented is a ratio of the energy used by under the given policy, to the minimum possible energy usage as computed by the oracle. Our oracle gives us the absolute minimum energy usage for the given workload by calculating optimal busy burst periods and long idle periods. The oracle considers both optimatl prefetching and request batching along with an optimal spin-down policy for the given workload. These results demonstrate the possibility of saving as much as 65% to 75% energy over traditional non-prefetching and fixed-time-out spin-down algorithms. Achieving this ideal would require perfect and complete knowledge of the future, and yet our dynamic prefetching policy and our dynamic spin-down time-out algorithm allowed us to save a substantial 15% to 35% energy.

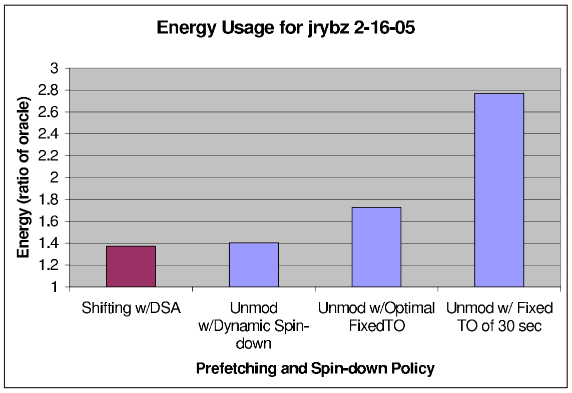


Figure 9. This figure shows a day long trace's disk energy usage given different prefetching and disk spin-down policies. This behavior is representative of the workloads tested.

5 Related Research

By creating bursty disk patterns using prediction and prefetching, our disk spin-down policy could save more energy by aggressively spinning down the disk. We have employed the batching of disk requests, the prediction and prefetching of data, and the judicious application of disk spin-down policies. Significant research has been done in each of these areas individually.

Effectively spinning down the disk drive can save valuable energy in a mobile environment [23, 11]. We have used a dynamic spin-down algorithm which adjust the time-out value based on past disk request history. Helmbold *et al.* [15] described another dynamic spin-down timeout algorithm that employed a machine learning algorithm to adjust the timeout. Bisson and Brandt demonstrated the practicality of implementing such an algorithm [4].

The nature of the access workload, and its interaction with the underlying cache and disk, is crucial for for effective disk power management. Zhu *et al.* [36] showed that simply minimizing cache misses does not necessarily result in the minimum energy usage for a given cache replacement policy. They proposed four different power-aware caching policies that can save up to 16% disk energy over a traditional LRU cache policy. Creating busier burst periods and longer idle periods allows the disk to be spun down for longer periods of time. The exploitation and promotion of such bursty behavior has been explicitly attempted by Weisel *et al.* [33], and Papathanasiou and Scott [27, 26, 25]. They found that traditional OS resource management policies tend to "smooth out" these burst and idle periods. *The Milly Watt Project* [8] shows that because application needs are the driving force behind power management strategies, it is useful to propagate energy efficiency information to the application. Nobel's implementation, *Odyssey* [24], showed a factor of five increase in performance over three different benchmarks. More specifically Flinn *et al.* [9] showed that collaboration between the operation system and applications can be used to achieve longer battery life and less energy consumption. Their implementation in the Linux kernel monitored energy supply and demand to select a trade-off between energy conservation and performance. Others have also used collaboration between the operation system and applications the increase energy efficiency and performance [33, 31].

A great deal of work has been done in the area of accurate file prediction and prefetching. The *Seer* project applied prediction to automate mobile file hoard construction [22, 21]. Griffioen *et al.* [14] applied a graph-based algorithm to predict file accesses, while Curewitz *et al.* [6] first proposed the use of data compressions techniques. Kroeger and Long [20] used similar techniques to create a practical, space-efficient and more adaptive prediction model, that was the implemented in the Linux kernel. A multi-expert prediction technique was proposed by Whittle *et al.* [34]. Brandt *et al.* [5] demonstrated another form of multi-expert prediction algorithm that further considered non-prediction as a mechanism to reduce mispredictions.

6 Conclusion and Future Research

By creating burstier disk access patterns and longer disk idle periods we can significantly increase the disk energy savings. This in turn allows us to use aggressive disk spin-down policies to effectively conserve energy. We have implemented a dynamic prefetching policy, *Best Shifting*, which dynamically chooses among six basic policies. The individual policy that creates the most potential energy savings is deemed best and selected. We then combined this policy with a dynamic spin-down mechanism, described by Helmbold *et al.* [15], with the resulting approach using 5% to 10% less energy than the dynamic spin-down policy alone. This combined strategy results in 15% to 35% less energy usage than traditional predictive prefetching and spin-down policies.

While we have shown that by dymanically selecting the prefetching strategy, we can lengthen disk idle periods and save energy, our oracle results show that there is more energy yet to be saved. Future investigation will be aimed at further reducing energy consumption. By implementing different prefetching strategies, with the goal not only to make accurate predictions but to create busier burst periods, we aim to come closer to optimal disk energy savings.

References

[1] A. Amer and D. D. E. Long. Noah: Low-cost file access prediction through pairs. In *Proceedings of the 20th IEEE International Performance, Computing and Communications Conference (IPCCC '01)*, pages 27–33. IEEE, Apr. 2001.

[2] A. Amer, D. D. E. Long, J.-F. Pâris, and R. C. Burns. File access prediction with adjustable accuracy. In *Proceedings of the International Performance Conference on Computers and Communication (IPCCC '02)*, Phoenix, Apr. 2002. IEEE.

[3] I. Ari, A. Amer, R. Gramacy, E. L. Miller, S. A. Brandt, and D. D. E. Long. ACME: adaptive caching using multiple experts. In *Proceedings in Informatics*, volume 14, pages 143–158. Carleton Scientific, 2002.

[4] T. Bisson and S. A. Brandt. Adaptive disk spin-down algorithms in practice. In *Proceedings of the 2004 Conference on File and Storage Technologies (FAST)*, 2004.

[5] K. Brandt, D. D. E. Long, and A. Amer. Predicting when not to predict. In *Proceedings of the 12th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '04)*, pages 419–426, Volendam, The Netherlands, Oct. 2004.

[6] K. M. Curewitz, P. Krishnan, and J. S. Vitter. Practical prefetching via data compression. In *Proceedings of ACM - SIGMOD Conference on Management of Data*, pages 257–266, 1993.

[7] F. Douglis, P. Krishnan, and B. Marsh. Thwarting the power-hungry disk. In *Proceedings of the Winter 1994 USENIX Technical Conference*, pages 292–306, San Francisco, CA, Jan. 1994. USENIX.

[8] C. S. Ellis. The case for higher-level power management. In *HOTOS '99: Proceedings of the The Seventh Workshop on Hot Topics in Operating Systems*, page 162, Washington, DC, USA, 1999. IEEE Computer Society.

[9] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *Symposium on Operating Systems Principles*, pages 48–63, 1999.

[10] Fujitsu. *MHT2080 Disk Drives Product Manual*, 2003.

[11] R. Golding, P. Bosch, C. Staelin, T. Sullivan, and J. Wilkes. Idleness is not sloth. In *Proceedings of the Winter 1995 USENIX Technical Conference*, pages 201–212, New Orleans, LA, Jan. 1995. USENIX.

[12] R. B. Gramacy, M. K. Warmuth, S. A. Brandt, and I. Ari. Adaptive caching by refetching. In *Advances in Neural Information Processing Systems 15*, pages 1465–1472. MIT Press, 2003.

[13] P. M. Greenawalt. Modeling power management for hard disks. In *Proceedings of the 2nd International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '94)*, pages 62–66, Durham, NC, Jan. 1994. IEEE.

[14] J. Griffioen and R. Appleton. Reducing file system latency using a predictive approach. In *Proceedings of the Summer 1994 USENIX Technical Conference*, pages 197–207, 1994.

[15] D. P. Helmbold, D. D. E. Long, T. L. Sconyers, and B. Sherrod. Adaptive disk spin-down for mobile computers. *ACM/Baltzer Mobile Networks and Applications (MONET)*, 5(4):285–297, 2000.

[16] M. Herbster and M. K. Warmuth. Tracking the best expert. In *Proceedings of the 12th International Conference on Machine Learning*, pages 286–294, Tahoe City, CA, 1995. Morgan Kaufmann.

[17] D. E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, 1973.

[18] T. M. Kroeger and D. D. E. Long. Predicting file-system actions from prior events. In *Proceedings of the Winter 1996 USENIX Technical Conference*, pages 319–328, San Diego, Jan. 1996.

[19] T. M. Kroeger and D. D. E. Long. The case for efficient file access pattern modeling. In *Proceedings of the 7th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VII)*, pages 14–19, Rio Rico, Arizona, Mar. 1999.

[20] T. M. Kroeger and D. D. E. Long. Design and implementation of a predictive file prefetching algorithm. In *Proceedings of the 2001 USENIX Annual Technical Conference*, pages 105–118, Boston, Jan. 2001.

[21] G. H. Kuenning. Seer: Predictive file hoarding for disconnected mobile operation. Technical Report 970015, University of California, Los Angeles, 20, 1997.

[22] G. H. Kuenning, G. J. Popek, and P. L. Reiher. An analysis of trace data for predictive file caching in mobile computing. In *Proceedings of the Summer 1994 USENIX Technical Conference*, pages 291–303, 1994.

[23] K. Li, R. Kumpf, P. Horton, and T. Anderson. A quantitative analysis of disk drive power management in portable computers. In *Proceedings of the Winter 1994 USENIX Technical Conference*, pages 279–291, San Francisco, CA, Jan. 1994.

[24] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker. Agile application-aware adaptation for mobility. In *Sixteen ACM Symposium on Operating Systems Principles*, pages 276–287, Saint Malo, France, 1997.

[25] A. E. Papathanasiou and M. L. Scott. Increasing disk burstiness for energy efficiency. Technical Report 792, Department of Computer Science, University of Rochester, Nov. 2002.

[26] A. E. Papathanasiou and M. L. Scott. Energy efficiency through burstiness. In *Proceedings IEEE Workshop on Mobile Computing Systems and Applications, 2003*, page 44, Monterey, CA, Oct. 2003.

[27] A. E. Papathanasiou and M. L. Scott. Energy efficient prefetching and caching. In *Proceedings of the 2004 USENIX Annual Technical Conference*, pages 255–268, Boston, MA, June 2004.

[28] D. Roselli and T. E. Anderson. Characteristics of file system workloads. Research report, University of California, Berkeley, June 1996.

[29] J. P. Rybczynski. Tracing Windows with tracelog and tracedmp, Feb. 2005.

[30] Seagate Technology LLC. *Barracuda Hard Drive Product Manual*, 2004.

[31] A. Vahdat, A. Lebeck, and C. Ellis. Every joule is precious: The case for revisiting operating system design for energy efficiency, Sept. 2000.

[32] V. Vovk. Aggregating strategies. In *Proceedings of the 3rd Annual Workshop on Computational Learning Theory*, pages 371–383, Rochester, NY, 1990. Morgan Kaufmann.

[33] A. Weissel, B. Beutel, and F. Bellosa. Cooperative I/O: A novel I/O semantics for energy-aware applications. *SIGOPS Oper. Syst. Rev.*, 36(SI):117–129, 2002.

[34] G. A. S. Whittle, J.-F. Pâris, A. Amer, D. D. E. Long, and R. Burns. Using multiple predictors to improve the accuracy of file access predictions. In *Proceedings of the 20th IEEE / 11th NASA Goddard Conference on Mass Storage Systems and Technologies*, pages 230–240, Apr. 2003.

[35] T. Yeh, D. D. E. Long, and S. A. Brandt. Caching files with a program-based last n successors model. In *Proceedings of the Workshop on Caching, Coherence and Consistency (WC3 '01)*, Sorrento, Italy, June 2001. ACM.

[36] Q. Zhu, F. M. David, C. F. Devaraj, Z. Li, and Y. Zhou. Reducing energy consumption of disk storage using power-aware cache management. In *10th International Symposium on High Performance Computer Architecture (HPCA'04)*, pages 118–129, Madrid, Spain, Feb. 2004. Cisco Systems Inc.