5th IEEE Wkshp. on Mobile Computing Systems and Applications (WMCSA), Monterey, CA, Oct. 2003

Energy Efficiency through Burstiness

Athanasios E. Papathanasiou and Michael L. Scott
University of Rochester, Computer Science Department
Rochester, NY 14623, U.S.A.
{papathan, scott}@cs.rochester.edu

Abstract

OS resource management policies traditionally employ buffering to "smooth out" fluctuations in resource demand. By minimizing the length of idle periods and the level of contention during non-idle periods, such smoothing tends to maximize overall throughput and minimize the latency of individual requests. For certain important devices, however (disks, network interfaces, or even computational elements), smoothing eliminates opportunities to save energy using low-power modes. As devices with such modes proliferate, and as energy efficiency becomes an increasingly important design consideration, we argue that OS policies should be redesigned to increase burstiness for energy-sensitive devices.

We are currently experimenting with techniques to increase the disk access pattern burstiness of the Linux operating system. Our results indicate that the deliberate creation of bursty activity can save up to 78.5% of the energy consumed by a Hitachi DK23DA disk (in comparison with current policies), while simultaneously decreasing the negative impact of disk congestion and spin-up latency on application performance.

1. Introduction

Resource management is one of the key responsibilities of an operating system: the traditional management policy aims to share resources fairly among competing tasks in a way that maximizes the twin goals of throughput and responsiveness. For continuously renewable resources such as processor cycles and I/O bandwidth, the OS typically addresses these goals by making the global access pattern as *smooth* as possible. By spreading activity over time we avoid wasting resources during idle periods, and minimize the latency due to contention during active periods. Background computations, for example, are scheduled around more interactive applications, and I/O bursts are spooled through memory so they don't hit the disk all at once.

Smoothness may also serve, in important cases, to minimize energy consumption. Processors such as the Transmeta Crusoe and the Intel StrongARM support frequency and voltage scaling, allowing a clever scheduling algorithm to save dynamic energy by "squeezing out the idle time" in rate-based applications [28]. Such algorithms depend critically on the ability of the device (in this case the processor) to operate fully in its lower power modes. They also leverage the superlinear relationship between power and voltage: within typical operating ranges, an x% reduction in voltage yields more than an x% reduction in power.

Many important devices, however, including hard disks, network interfaces, and even DRAM chips, support *nonoperational* (standby) low-power modes. Requests can be handled only in full-power (active) mode; the various standby modes consume progressively less power, but take increasing amounts of time, and often energy, to return to active mode. Dropping into a standby mode makes sense only when we can stay in that mode long enough for the energy saved to exceed the energy required to return to active mode. Making efficient use of *nonoperational* low-power modes is extremely important in the mobile computing area, especially for resource-poor devices such as laptops and PDAs. However, by spreading accesses over time, a smooth access pattern may actually *waste* energy by eliminating opportunities to profit from low power modes. Specifically, in the case of hard disks, conservative prefetching algorithms, periodic update policies, and lack of coordination among I/O requests lead to access patterns that prohibit the use of a disk's low power states during the execution of several common applications.

To illustrate the effect of such kernel algorithms on energy consumption, consider the write behavior of a typical interval periodic update policy [20], as found, for example, in the Linux file system. Every 5 seconds, the Linux `kupdate` daemon scans for dirty pages that are more than 30 seconds old, and writes them to the disk. As a result of this policy, even during light write workloads the interval between successive disk operations is seldom longer than 5 seconds—too short to allow a modern laptop disk to save

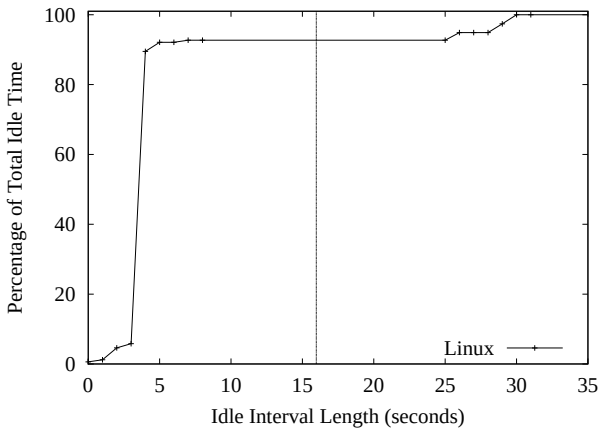

Figure 1. Distribution of idle time intervals for the Linux disk during a CD copy operation. The vertical bar indicates the break-even time of Hitachi DK23DA disk.

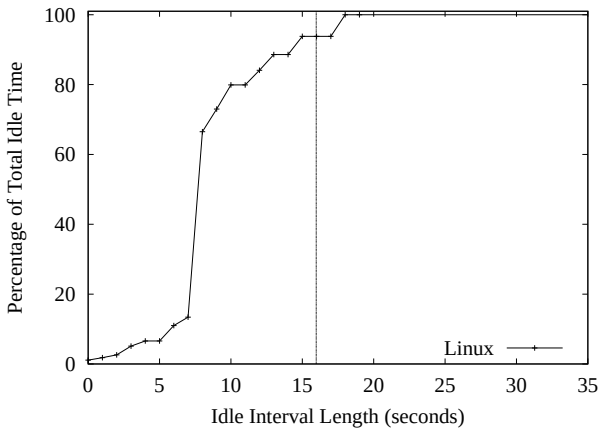

Figure 2. Distribution of idle time intervals for the Linux disk during MP3 playback.

energy by spinning down the disk. Figure 1 shows the disk idle times while copying a CD-ROM to disk. Almost no idle time interval is longer than five seconds, and the disk remains constantly active. This is unfortunate given that the sustainable bandwidth of the disk significantly exceeds that of the CD drive.

A similar problem arises in light read workloads. For sequential read accesses, Linux supports a conservative prefetching algorithm that reads up to 128 KB (32 4KB pages) in advance. For several common applications, such as MP3 playback and encoding, compression, and data copying, the 128 KB prefetching depth is not large enough to yield idle intervals above the disk break-even time. Figure 2 illustrates disk behavior for MP3 playback. The application consumes file data sequentially at a rate of approximately 16 KB/s (1MB/min), which the kernel translates into read requests for 128KB approximately every 8 seconds—still too short to justify a spin-down. In our experiment 66% of the total disk idle time (194 seconds out of 292) appears in intervals of less than 8 seconds, and only 6% appears in intervals that are larger than 16 seconds (the break-even time for a Hitachi DK23DA disk model[1]).

When multiple applications are executing concurrently, the fact that requests are not coordinated among applications means that we may not be able to use a low disk power mode even when each individual application has long periods of idle time. In addition, because of the conservative prefetching policy, increasing the system's memory does not improve the hard disk's energy consumption.

The goal of our work is to develop operating system mechanisms that increase the *burstiness* of disk access patterns. Toward this end we have modified the memory management and file system of the Linux 2.4.20 kernel. We extend the Linux operating system with novel algorithms and data structures that allow us to:

- Quickly identify the working set of the executing workload and dynamically control the amount of memory used for aggressive prefetching and buffering of dirty data.
- Coordinate the generation of disk requests among multiple concurrently running applications, so that they are serviced by the hard disk during the same small window of time.

The rest of this paper is structured as follows. Sections 2 and 3 describe the design of our prefetching and request deferring mechanisms. Section 4 presents experimental results. Section 5 discusses previous work. Section 6 summarizes our conclusions and directions for future work.

2. Design Guidelines

Modern hard disks for mobile systems support four different power states: Active, Idle, Standby, and Sleep. In the Idle state the disk is still spinning, but the electronics may be partially unpowered, and the heads may be parked or unloaded. In the Standby state the disk is spun down. The Sleep state powers off all remaining electronics; a hard reset is required to return to higher states. Individual devices may support additional states. The IBM TravelStar, for example, has three different Idle sub-states. Table 1 presents the power characteristics for four recent hard disks. Note

[1]Break-even time has been computed using the power consumption of the *low power* idle mode (Table 1).

that the IBM Microdrive and the Toshiba MK5002 are one-inch devices intended for use in cameras and PDAs.

Contemporary disks require on the order of one to three seconds to transition from Standby to Active state. During that spin-up time they consume 1.5–2X as much power as they do when Active. The typical laptop disk must therefore remain in Standby state for a certain amount of time to justify the energy cost of the subsequent spin-up. This *break-even time* is currently on the order of 5–20 seconds for laptop disks. The time and energy required to move from Idle to Active state are minimal for most devices, leading to a relatively short breakeven time (usually less than 3 seconds). In addition, the energy savings in the intermediate idle states are comparable to those of the standby state, making the idle low power mode an excellent alternative when disk idle periods are not large enough to justify a spin-down. However, none of the low power states may be used if the time between requests is very small.

The principal task of an energy-conscious buffer management policy is to extend the length of disk idle phases and force transitions to Standby state when this is likely to save significant energy. Inappropriate mode transitions can waste energy (if the disk remains idle for less than the break-even time), frustrate the human user (if the computer must frequently spin up the disk in order to service an interactive application), and possibly reduce the lifetime of the disk through wear and tear. On the other hand, even significant amounts of unnecessary disk traffic (e.g. prefetching of data that are never actually used) can be justified if they allow us to reduce the frequency of mode transitions or leave the disk in a low power mode for longer periods of time. Our goals are thus to (1) *maximize the length of idle phases* by prefetching aggressively and by postponing write requests, (2) *coordinate I/O across applications*, and (3) *maintain interactive responsiveness* through disk pre-activation.

2.1. Maximizing Idle Phases

Burstiness arises when we take accesses to an underloaded device and move them together in time. Some cases are easy: requests from non-interactive applications can be delayed for significant periods of time, though we need to be careful to maintain the fairness of long-term scheduling. Asynchronous requests can be similarly delayed, though for interactive applications we should generally aim to complete requests before the program waits for I/O completion. Upper levels of the I/O system will need to preserve sufficient information for lower levels to distinguish between urgent requests (those that might cause an interactive application to block) and delayable, non-urgent requests.

We can increase burstiness even for synchronous requests from interactive applications via aggressive prefetching and buffering (write-behind). Write-behind is relatively straightforward: it requires no prediction, and is limited only by the available buffer space and reliability considerations.

Prefetching for burstiness, on the other hand, is quite a bit more subtle. In comparison to traditional prefetching, which aims to reduce the latency of access to disks in active mode, a bursty prefetch algorithm must be significantly more aggressive in both quantity and coverage. A traditional prefetching algorithm can fetch data incrementally: its goal is simply to request each block far enough in advance that it will already be available when the application needs it. It will improve performance whenever its "true positives" (prefetched blocks that turn out to indeed be needed) is reasonably high, and its "false positives" (prefetched blocks that aren't needed after all) don't get in the way of fetching the "false negatives". By contrast, an energy-reducing prefetching algorithm must fetch enough blocks to satisfy all read requests during a lengthy idle interval. Minimizing "false negatives" has an increased importance compared to traditional prefetching, since the energy cost and performance penalty of power mode transitions is very high. These differences suggest the need to fetch much more data, and much more speculatively, than has traditionally been the case. Indeed, prefetching for burstiness more closely resembles prefetching for disconnected operation in remote file systems [14] than it does prefetching for low latency.

Currently, we do not have a complete solution that solves the problem of accurate file prediction. However, we believe that efficient file prediction may be achieved by monitoring past file accesses of applications and taking advantage of the *semantic locality* that appears in user behavior [16]. Upon initiation of the execution of an application its working set of files can be loaded into memory. An alternative method is to aggressively load into memory all small files found in the working directory of the application. Furthermore, keeping track of accesses to code pages with file accesses can improve prefetching in cases, such as scene and sound effect loading during game playing.

2.2. Coordinating across applications

Idle interval length can be limited because of a lack of coordination among requests generated by different applications. Write activity can be easily clustered because most write requests are issued by a single entity: the update daemon. Similarly, page-out requests are issued by the swap daemon. Read and prefetching requests, however, are generated within a process context independently of other applications. To coordinate prefetching requests across all running applications we introduce a centralized entity that is responsible for generating prefetching requests for all run-

Table 1. Energy consumption parameters for various disks. The power savings compared to the active state are shown in parenthesis for each low power mode. The characteristics of the IBM and Toshiba disks come from their respective data sheets, while those of the Hitachi disk have been computed through experimental measurements.

Disk	Hitachi DK23DA	IBM TravelStar	Toshiba MK5002 MPL	IBM Microdrive DSCM
Capacity	10-30GB	6-18GB	2GB	340MB-1GB
Active	2.1W	2.1W	1.3W	0.73W
Idle	1.6W (24%)	1.85W (12%)	0.7W (46%)	0.5W (31%)
Active Idle	NA	0.85W (59%)	NA	NA
Low Power Idle	0.6W (71%)	0.65W (69%)	0.5W (62%)	0.22W (70%)
Standby	0.15W (93%)	0.25W (88%)	0.23W (82%)	0.066W (91%)
Spin up	3.0W	3.33W	3.0W	0.66W
Spin up time	1.6s	1.8s	1.2s	0.5s

ning applications: the prefetch daemon. The prefetch daemon is equivalent to the update daemon and handles read activity. Through the prefetch thread the problem of coordinating I/O activity is reduced to that of coordinating three daemons.

Fundamental in achieving a bursty access pattern is a memory management system capable of coordinating requests generated by various system components (applications and kernel daemons) and reordering them with a goal to maximize the average length of idle intervals for devices supporting low power modes. The methods for reordering requests, aggressive prefetching and delayed-writes, increase significantly the memory pressure. Hence, prefetching aggressiveness and write delays should be controlled in a way that do not lead to increased paging or thrashing, evicting data that could be used earlier in time.

2.3. Maintaining responsiveness

For mobile systems, whose workloads are not usually I/O intensive, the principal downside to a bursty access pattern is a potential increase in the application-perceived latency of synchronous reads, due to spin-up operations. For interactive applications aggressive prefetching serves to reduce the number of visible delays. For rate-based and non-interactive applications, the same information that allows the operating system to identify opportunities for spin-down can also be used to *predict* appropriate times for spin-up, rendering the device available just in time to service requests. It can also be used to deactivate the disk after the end of a burst of activity.

For more I/O intensive workloads, burstiness can lead to increased latency due to congestion. This negative effect of burstiness was noted in the past for file systems using the 30 second periodic update policy, such as the Sprite file system [2]. In our current prototype, we keep track of pending I/O requests and initiate prefetching for each application in advance so that the time required for a request to wait in the disk queue is hidden.

In order to mitigate the impact of possible disk congestion on unpredicted cache misses, we introduce a notion of urgency and use it to order requests. The general idea is that requests for data that are going to be used earlier in time or requests that are important for system reliability should be serviced first. *Urgency-based* scheduling and request criticality have also been discussed in previous work [9] in the context of read latency reduction.

3. Prototype

The key idea behind our energy efficient prefetching and caching mechanism is the introduction of an Epoch-based algorithm in the basic memory management mechanisms of the Linux kernel. Each epoch consists of two phases: a *request generation* phase and an *idle* phase. During the request generation phase the operating system attempts to load into memory data that are going to be accessed in the near future. In order to achieve its goal the following tasks have to be completed:

1. Flush all dirty buffers.

2. Predict future data accesses. Prediction is based on access hints [23], which increase the overall prediction rate.

3. Compute the amount of memory that can be used for prefetching or storing new data. This step requires identifying quickly the currently useful in-memory data: the workload's working set and cached files.

4. Free the required amount of memory by unmapping pages and flushing dirty, mapped pages.

5. Prefetch or reserve buffers for writing new data proportionally to each executing application's data consumption or production rate. The goal of this step is to maximize the time to the next demand miss.

When the request generation phase completes, the idle phase of the epoch is initiated. During the idle phase, the data production and consumption rates of each application are monitored and, based on this information, a prediction about the time of the next request is made. If the time to the next request is predicted to be higher than the disk's break-even time, an early disk spin-down decision is made.

The start of a new epoch is triggered by one of the following events:

1. A new prefetching cycle has to be initiated.

2. A demand miss took place. In this case the prefetching algorithm has failed to load into memory all required data, and the application that issued the request may experience an increased delay if the disk has been placed into low power mode.

3. One or more dirty buffers have expired and it is time for them to be flushed.

4. The system is low on memory resources. The page freeing logic has to be executed.

3.1. The Prefetch Cache

We have augmented the kernel's page cache with a new data structure: the prefetch cache. Pages requested by the prefetch daemon are placed in the prefetch cache. Each page in the prefetch cache is associated with a timestamp that describes the time that it is expected to be accessed. Pages that get referenced or pages that do not get referenced before their expected access time are moved to the standard LRU list and thereafter are controlled by the kernel's page reclamation policy.

Intuitively, two parameters determine the number of pages that should be freed at the beginning of each epoch. First, the reserved amount of memory should be large enough to contain all predicted data accesses. Second, prefetching or future writes should not cause the eviction of pages that are going to be accessed sooner than the prefetched data. Since our goal is to maximize the length of the hard disk's idle periods, we use the type of the first miss during an epoch's idle phase to determine the size of the prefetch cache.

We categorize page cache misses as follows:

1. Compulsory miss: A miss on a page for which there is no prior information.

2. Prefetch miss: A miss on a page for which there was a prediction that it was going to be accessed. Such a miss suggests that a larger prefetch cache size could have been used during the current epoch.

3. Eviction miss: A miss on a page that used to reside in the page cache, but was evicted in favor of prefetching. Such a miss suggests that the prefetch cache used in the current epoch was too large.

In order to identify eviction misses, we have implemented a new data structure, called the *eviction cache*. The eviction cache maintains the metadata of recently evicted pages along with a unique serial number, called the eviction number. The serial number serves the purpose of keeping track of the number of pages that were evicted during the request generation phase of an epoch in favor of prefetching. During the idle phase, if an eviction miss takes place, the difference between the page's eviction number and the current epoch's starting eviction number provides an estimation of a suitable size for the prefetch cache in number of pages, and is used as the prefetch cache size for the upcoming epoch. The prefetch cache size does not change in the case of misses on pages that were evicted in past epochs, and it is increased by a certain constant if there were no misses, or there were only prefetch misses.

3.2. Update Policy

In our current implementation, we use a modified update daemon that flushes all dirty buffers once per minute. In addition, we have extended the `open` system call with an additional flag that indicates that write-behind of the dirty buffers belonging to a certain file can be postponed until the file is closed. Such a direction is useful for several common applications, such as compilations and MP3 encoding, that produce files that are not associated with strict reliability constraints. For such applications write-back can be delayed until the whole operation (eg. copying of a file or MP3 encoding of a CD track) has completed.

4. Experimental Evaluation

In this section, we compare our energy-conscious (*Bursty*) memory/disk management policy, which attempts to increase the burstiness of the disk's usage pattern through aggressive prefetching, to the standard *Linux* policies across systems with memory sizes ranging from 64MB to 492MB. The Linux kernel's update policy has been modified to flush all dirty buffers once per minute, since the original update daemon that flushes buffers every five seconds prohibits the

use of low power modes. The *Bursty* system takes advantage of the fact that data produced by the applications in our experiments are not associated with strict reliability constraints[2]. Hence, the time that write requests can be delayed is limited only by the amount of available memory. Dirty buffers are always flushed when a file is closed.

We assume a 10 second fixed threshold power management policy for the Linux kernel. However, the Linux memory management algorithms lead to very short periods of disk idle time, and hence, such a policy degenerates to the No-Spin-Down policy. Our kernel's power management policy is based on a predictive algorithm that monitors the I/O behavior of each running application and spins down the disk immediately when the predicted idle phase length is greater than the disk's break-even time. Our goal is to show how usage pattern re-shaping can lead to increased energy savings by providing longer intervals of disk idle time.

Our experiments were conducted on a Dell Inspiron 4100 laptop with 512 MB of total memory and a Hitachi DK23DA hard disk. Power measurements were collected from the two 5V supply lines of the Hitachi disk. To measure power, both the voltage and current need to be known. The voltage is assumed to be fixed at 5V. We used $100m\Omega$ resistors in order to dynamically measure the current through the supply lines. A voltmeter was used to measure the voltage drop across the resistors and compute the current. The sampling rate for each signal was 1000 samples/second.

The idle interval histogram graphs (figures 3 and 4) are based on traces collected from the ATA/IDE disk driver during the execution of our workloads scenarios. In order to avoid any disk activity caused by the tracing system, a pinned-down 20MB memory buffer that was periodically transmitted to a logging system through the network was used.

In the experimental evaluation we use two different workload scenarios with different degrees of I/O intensity. The first, MPEG playback of two 76 MB files, represents a relatively intensive read workload. The second is a read and write intensive workload, which involves concurrent MP3 encoding and MPEG playback. The MP3 encoder reads 10 WAV files with a total size of 626 MB and produces 42.9 MB of data. During the MP3 encoding process, the MPEG player accesses two files with a total size of 152 MB.

The metrics used in the comparisons are:

Length of idle periods Longer idle periods can be exploited by more power efficient device states. Increasing the length of idle periods can improve any underlying power management policy.

Energy savings We compare the energy savings achieved for Linux and our Bursty system for various memory sizes.

Slowdown A significant challenge for our Bursty memory management system is to minimize the performance penalties that may be caused by increased disk congestion and disk spin-up operations.

[2]The written data can be reproduced again automatically in case of a system crash for all the applications tested.

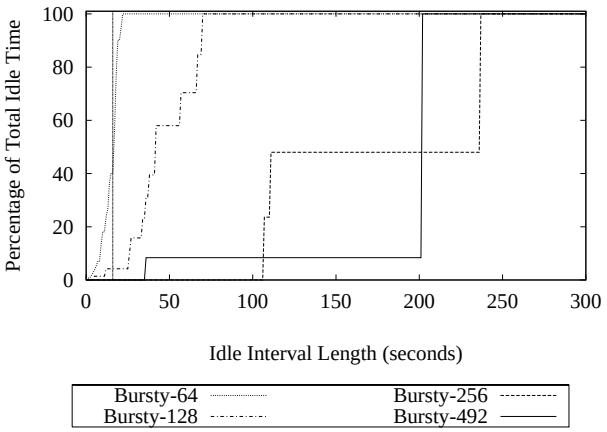

Figure 3. Cumulative distribution of disk idle time intervals during MPEG playback.

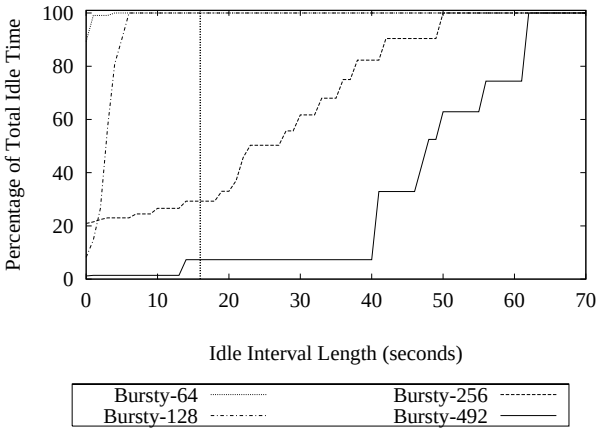

Figure 4. Distribution of disk idle time intervals during concurrent MPEG playback and MP3 encoding.

Figures 3-4 show the distribution of idle time intervals for our workload scenarios. We present results for our *Bursty* system using various memory sizes. In both graphs the straight vertical line represents the 16 second break-even point of the Hitachi hard disk. In the Linux case (not shown in the graphs), *100%* of the disk idle time appears in intervals of less than 1 second, independent of memory size,

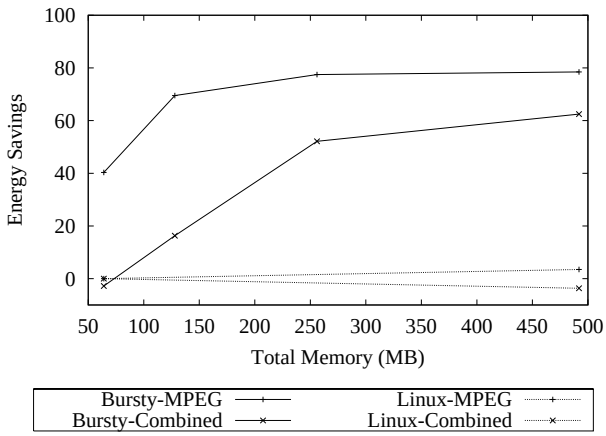

Figure 5. Disk energy savings as a function of total memory size. Results are shown for the two experimental workloads (*MPEG* and *Combined*) executing on the standard Linux kernel (*Linux*) and our bursty system (*Bursty*).

prohibiting the use of any low power mode. In contrast, larger memory sizes lead to longer idle interval lengths for the *Bursty* system, providing more opportunities for the disk to transition to a low power mode.

Figure 5 presents disk energy savings as a function of total system memory size. The base case used for the comparisons is the standard Linux kernel on a 64MB system. For Linux, increasing the system's memory size has a minor impact on the energy consumed by the disk, because of the lack of long idle intervals. Strangely, in the second more intensive workload (*Linux-Combined*) a memory size of 492 MB leads to a slight increase of 3.6% in disk energy consumption. In contrast, the savings achieved by the *Bursty* algorithm depend on the amount of available memory. For the less intensive workload, significant energy savings are achieved for all memory sizes. Even on the 64 MB system, the energy consumed by the disk is reduced by 40.3%. Despite the fact that most disk idle intervals are not long enough to justify a spin-down operation, they allow the disk to make an efficient use of the low-power idle state that consumes just 0.6 Watts. On the 492 MB system, the *Bursty* system loads the required data in just three very intensive I/O operations, allowing the disk to transition and remain in the spin-down state for significant periods of time, and leading to 78.5% disk energy savings. The results of the second workload are similar. However, because of the increased I/O intensity the energy savings are less pronounced. Energy consumption is reduced after the memory size exceeds 128 MB (15.9% energy savings). On a system with 492 MB energy savings reach 62.5%.

Figure 6 presents snapshots of the hard disk's power consumption during the execution of MPEG playback. For the Linux kernel we show results only for the 492 MB memory system (top), since in all cases the disk is constantly in the active or the first idle state (1.6 W). For Bursty, we present results for the 64 MB (middle) and 128 MB (bottom) cases. As memory size increases, the hard disk spends more time in the low-power idle or the spin-down state. The spikes in the middle and bottom graphs represent power-up and power-down operations. The power consumption behavior is similar for the second workload (not shown due to space limitations). During execution on the Linux kernel, the disk remains constantly in the active or the first idle state. The *Bursty* system manages to make more efficient use of the disk's low power modes as memory size increases. However, since the second workload is more intensive, efficient use of the low power modes starts at larger memory sizes (256 MB).

Figure 7 presents the total execution time for the two workload scenarios. For the first workload (left side of the graph) the *Bursty* system experiences a slowdown of 1% (for the 64 MB case) or less, when compared to Linux with 492 MB of memory (L-492). For the second workload (right side of the graph), the slowdown of the MP3 encoding process is 2.8% or less. The performance of the MPEG player stays within 1.6% of that on the Linux system in all cases except the 64 MB system (B-64), where it experiences a slowdown of 4.8%. Our performance results show that the prefetching algorithm manages to avoid successfully most of the delay that may be caused by disk spin-up operations or disk congestion.

5 Related Work

Power Management. The research community has been very active in the area of power-conscious systems during the last few years. Ellis *et al.* [6] emphasized the importance of energy efficiency as a primary metric in the design of operating systems. ECOSystem [30] provides a model for accounting and fairly allocating the available energy among competing applications according to user preferences. Odyssey [8, 21] provides operating system support for application-aware resource management. The key idea is to trade quality for resource availability.

Several policies for decreasing the power consumption of processors that support dynamic voltage and frequency scaling have been proposed. The key idea is to schedule so as to "squeeze out the idle time" in rate-based applications. Several researchers have proposed voltage schedulers for general purpose systems [7, 11, 28, 24]. Lebeck *et al.* [17] explore power-aware page allocation in order to make a more efficient use of memory chips supporting multiple power states, such as the Rambus DRAM chips.

Hard Disks. The energy efficiency of hard disks is not

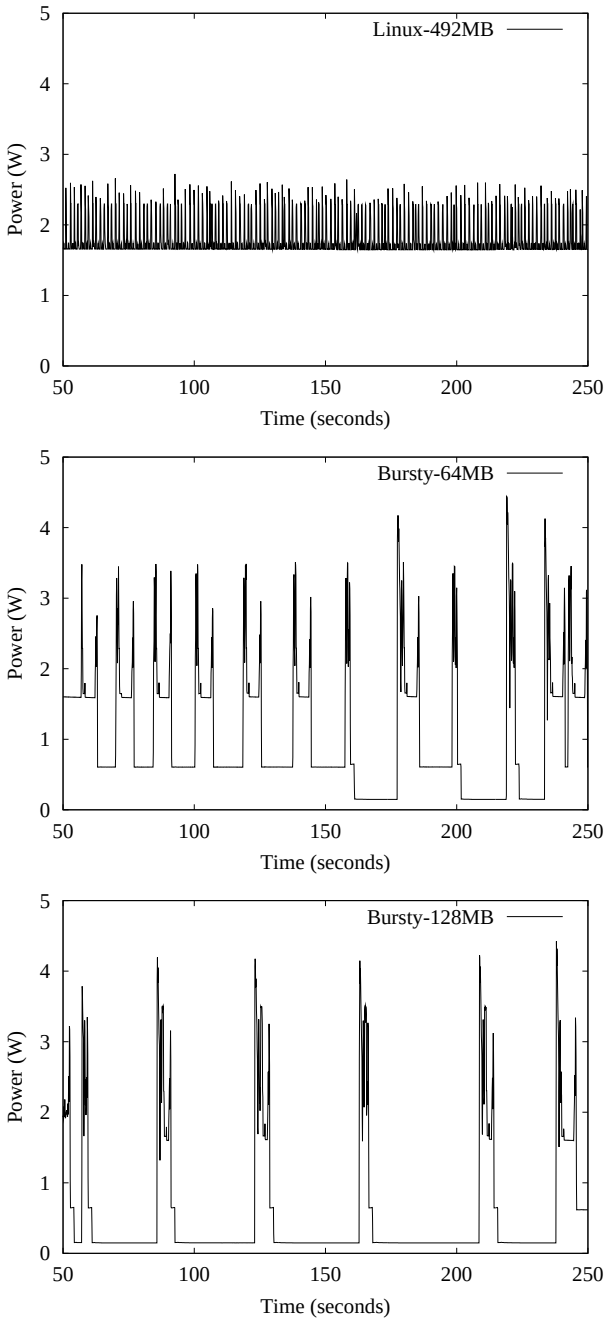

Figure 6. Snapshot of the hard disk's power profile during the execution of the MPEG player on *Linux* with 492MB (top), *Bursty* with 64MB (middle), and *Bursty* with 128MB (bottom).

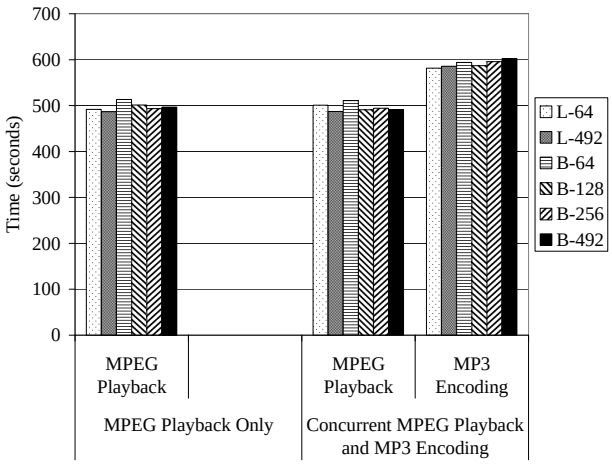

Figure 7. Execution time of the first workload scenario when ran on Linux (L) and Bursty (B) for various system memory sizes.

a new topic. The cost and risks of standby mode played a factor in the early investigation of hard-disk spin-down policies [4, 5, 13, 18]. Concurrently with our own work [22], several groups have begun to investigate the deliberate generation of *bursty* access patterns. Heath et al. [12] and Weissel et al. [29] propose user-level mechanisms to increase the burstiness of I/O requests from individual applications. Lu et al. [19] report that significant energy can be saved by respecting the relationship between processes and devices in the CPU scheduling algorithm. (We note, however, that given the many-second "break-even" times for hard disks, process scheduling can increase burstiness only for non-interactive applications, which can tolerate very long quanta.) Zeng et al. [31] propose "shaping" the disk access pattern as part of a larger effort to make energy a first-class resource in the eyes of the operating system.

Like Lu et al. and Zeng et al., we believe that the effective management of devices with standby modes requires global knowledge, and must be implemented, at least in part, by the operating system. Our work differs from that of Lu et al. by focusing on aggressive read-ahead and write-behind policies that can lead to bursty device-level access patterns even for interactive applications. Our work is more similar to that of Zeng et al., but without the notion of energy as a first-class resource. While we agree that energy awareness should be integrated into all aspects of the operating system, it is not clear to us that it makes sense to allocate joules to processes in the same way we allocate cycles or bytes. Rather than say "I'd like to devote 20% of my battery life to MP3 playback and 40% to emacs," we suspect that users in an energy-constrained environment will say "I'd like to extend my battery life as long as possible without suffering more than a 20% drop in sound quality or interactive responsiveness." It will then be the respon-

sibility of the operating system to manage energy *across applications* to meet these quality-of-service constraints.

Prefetching. Prefetching has been suggested by several researchers as a method to decrease application perceived delays caused by the storage subsystem. Previous work has suggested the use of hints as a method to increase prefetching aggressiveness for workloads consisting of both single [23] and multiple applications [27]. Cao *et al.* [1] propose a two-level page replacement scheme that allows applications to control their own cache replacement, while the kernel controls the allocation of cache space among processes. Curewitz *et al.* [3] explore data compression techniques in order to increase the amount of prefetched data. In the best of our knowledge, previously proposed prefetching algorithms do not address improved energy efficiency. In general, they assume a non-congested disk subsystem, and they allow prefetching to proceed in a conservative way resulting in a relatively smooth disk usage pattern.

6. Conclusion

In our study we investigated the potential benefits of increasing the burstiness of disk usage patterns in order to improve the energy efficiency of the disk power management policy. We suggested the use of aggressive prefetching and the postponement of non-urgent requests in order to increase the average length of idle phases. In addition, we presented a method to coordinate accesses of several concurrently executing tasks competing for limited memory resources so that requests are generated and arrive at the disk at roughly the same time. We have implemented the proposed ideas in the Linux kernel 2.4.20. Our experiments with streaming applications show that our techniques can increase the length of idle phases significantly compared to a standard Linux kernel leading to disk energy savings of up to 78.5%. The savings depend on the amount of available memory, and increase as the system's memory size increases. Our future work will focus on workloads that include applications with random access patterns. Such applications will require highly speculative prefetching algorithms similar to those used for prefetching for disconnected operation in remote file systems [14]. Although we may not see energy reduction at the order of 78.5% for such applications, we believe that even relatively short increases in the average idle interval length can lead to significant energy savings — mostly by making more efficient use of the intermediate low power states.

Despite of the fact that our current work has focused on hard disks, in principle increased burstiness can be used to improve the energy efficiency of other devices with non-operational low power modes. Network interfaces—for wireless networks in particular—are an obvious example, but they introduce new complications. First, in addition to standby modes, several wireless interfaces support multiple active modes, with varying levels of broadcast power suitable for communication over varying distances. The energy consumed by a wireless interface also depends on the quality of the channel, so communication bursts should be scheduled, when possible, during periods of high channel quality. Gitzenis *et al.* [10] consider the benefits of such scheduling under the assumption of a perfect predictor. We believe that our work will complement theirs nicely for wireless remote file access. For more general network traffic, we will need to develop techniques that increase the burstiness of browsers, streaming protocols, P2P searches, and other applications. We envision application-specific mechanisms that track past behavior and pass predictions to an OS daemon, which can in turn use global knowledge to shape the traffic pattern. Efficiently using the low power modes of wireless devices and the additional complication of accommodating externally initiated traffic will have a significant impact on the design of transport and physical layer protocols [15, 26]. An additional example arises in the case of Rambus DRAM memory chips. By consecutively scheduling processes, whose current working set resides on the same memory chip, one can increase the burstiness of accesses on a certain chip, and the idle intervals for the remaining chips. Such considerations affect the design of the processor scheduling and page placement algorithms [17].

Over time, we speculate that burstiness may become important in the processor domain as well. In a processor with multiple clock domains, for example [25], one can save dynamic power in a floating-point application by slowing down the (lightly-used) integer unit. Alternatively, by scheduling instructions for burstiness, one might save both dynamic *and* static power by gating off voltage to the integer unit during periods of inactivity.

Moreover, interdependencies among the components of a system should be taken into account when making scheduling and power management decisions. As an example, consider the case of an application transferring data over a slow wireless link to local storage (hard disk). On a traditional operating system, such an operation would probably result in a constantly active disk that spends most of its idle time waiting for data from the network link. However, buffering an increased amount of data through memory would provide longer disk idle periods that could be used for reducing energy consumption. Similar dependencies exist among all components of a computing system.

Finally, in a wide-open field of research, mobile wireless systems raise the possibility of enhancing performance, saving energy, or both, by off-loading computation to nearby servers. We plan to pursue this possibility in the context of work on *intuitive computing* (www.cs.rochester.edu/research/intuitive/), and to explore its implications for devices with standby modes.

References

[1] P. Cao, E. W. Felten, and K. Li. A Study of Integrated Prefetching and Caching Strategies. In *Proc. of the 1995 ACM Joint Int. Conf. on Measurement and Modeling of Computer Systems (SIGMETRCIS'95/PERFORMANCE'95)*, 1995.

[2] S. D. Carson and S. Setia. Analysis of the Periodic Update Write Policy For Disk Cache. *IEEE Transactions on Software Engineering*, 18(1):44–54, Jan. 1992.

[3] K. M. Curewitz, P. Krishnan, and J. S. Vitter. Practical prefetching via data compression. In *Proc. of the 1993 ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'93)*, May 1993.

[4] F. Douglis, P. Krishnan, and B. Bershad. Adaptive Disk Spindown Policies for Mobile Computers. In *Proc. of the 2nd USENIX Symposium on Mobile and Location-Independent Computing*, Apr. 1995.

[5] F. Douglis, P. Krishnan, and B. Marsh. Thwarting the Power-Hungry Disk. In *Proc. of the 1994 Winter USENIX Conference*, Jan. 1994.

[6] C. S. Ellis. The Case for Higher Level Power Management. In *Proc. of the 7th Workshop on Hot Topics in Operating Systems (HotOS VII)*, Mar. 1999.

[7] K. Flautner and T. Mudge. Vertigo: Automatic Performance-Setting for Linux. In *Proc. of the 5th USENIX Symp. on Operating Systems Design and Implementation (OSDI'02)*, Dec. 2002.

[8] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *Proc. of the 17th ACM Symp. on Operating Systems Principles*, Dec. 1999.

[9] G. R. Ganger and Y. N. Patt. Using System-Level Models to Evaluate I/O Subsystem Designs. *IEEE Transactions on Computers*, 47(6):667–678, June 1998.

[10] S. Gitzenis and N. Bambos. Power-Controlled Data Prefetching/Caching in Wireless Packet Networks. In *Proc. of the 21st Annual Joint Conf. of the IEEE Computer and Communications Societies (Infocom'02)*, June 2002.

[11] K. Govil, E. Chan, and H. Wasserman. Comparing Algorithms for Dynamic Speed-Setting of a Low-Power CPU. In *Proc. of the 1st Annual Int. Conf. on Mobile Computing and Networking (MobiCom'95)*, Nov. 1995.

[12] T. Heath, E. Pinheiro, J. Hom, U. Kremer, and R. Bianchini. Application Transformations for Energy and Performance-Aware Device Management. In *Proc. of the 11th Int. Conf. on Parallel Architectures and Compilation Techniques (PACT'02)*, Sept. 2002.

[13] D. P. Helmbold, D. D. E. Long, and B. Sherrod. A Dynamic Disk Spin-down Technique for Mobile Computing. In *Proc. of the 2nd Annual Int. Conf. on Mobile Computing and Networking (MobiCom'96)*, Nov. 1996.

[14] J. J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems*, 10(1), Feb. 1992. Earlier version presented at the 13TH *SOSP*, Oct. 1991.

[15] R. Krashinsky and H. Balakrishnan. Minimizing Energy for Wireless Web Access Using Bounded Slowdown. In *Proc. of the 8th Annual Int. Conf. on Mobile Computing and Networking (MobiCom'02)*, Sept. 2002.

[16] G. H. Kuenning and G. J. Popek. Automated hoarding for mobile computers. In *Proc. of the 16th ACM Symp. on Operating Systems Principles*. ACM Press, Oct. 1997.

[17] A. R. Lebeck, X. Fan, H. Zeng, and C. S. Ellis. Power Aware Page Allocation. In *Proc. of the 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS'00)*, Nov. 2000.

[18] K. Li, R. Kumpf, P. Horton, and T. Anderson. Quantitative Analysis of Disk Drive Power Management in Portable Computers. In *Proc. of the 1994 Winter USENIX Conference*, Jan. 1994.

[19] Y.-H. Lu, L. Benini, and G. D. Micheli. Power-Aware Operating Systems for Interactive Systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 10(1), Apr. 2002.

[20] J. C. Mogul. A Better Update Policy. In *Proc. of the USENIX Summer 1994 Technical Conference*, June 1994.

[21] B. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker. Agile Application-Aware Adaptation for Mobility. In *Proc. of the 16th ACM Symp. on Operating Systems Principles*, Oct. 1997.

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

[23] R. H. Patterson, G. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. Informed Prefetching and Caching. In *Proc. of the 15th ACM Symp. on Operating Systems Principles*, Dec. 1995.

[24] T. Pering, T. Burd, and R. Brodersen. Voltage Scheduling in the lpARM Microprocessor System. In *Proc. of the 2000 Int. Symp. on Low Power Electronics and Design (ISLPED'00)*, July 2000.

[25] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, and M. L. Scott. Energy Efficient Processor Design Using Multiple Clock Domains with Dynamic Voltage and Frequency Scaling. In *Proc. of the 8th Int. Symposium on High Performance Computer Architecture (HPCA-8)*, pages 29–40, Feb. 2002.

[26] E. Shih, S.-H. Cho, N. Ickes, R. Min, A. Sinha, A. Wang, and A. Chandrakasan. Physical Layer Driven Protocol and Algorithm Design for Energy-Efficient Wireless Sensor Networks. In *Proc. of the 7th Annual Int. Conf. on Mobile Computing and Networking (MobiCom'01)*, July 2001.

[27] A. Tomkins, R. H. Patterson, and G. Gibson. Informed multi-process prefetching and caching. In *Proc. of the 1997 ACM Joint Int. Conf. on Measurement and Modeling of Computer Systems (SIGMETRCIS'97)*. ACM Press, 1997.

[28] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for Reduced CPU Energy. In *Proc. of the 1st USENIX Symp. on Operating Systems Design and Implementation (OSDI'94)*, Nov. 1994.

[29] A. Weissel, B. Beutel, and F. Bellosa. Cooperative I/O: A Novel I/O Semantics for Energy-Aware Applications. In *Proc. of the 5th USENIX Symp. on Operating Systems Design and Implementation (OSDI'02)*, Dec. 2002.

[30] H. Zeng, X. Fan, C. S. Ellis, A. R. Lebeck, and A. Vahdat. ECOSystem: Managing Energy as a First Class Operating System Resource. In *Proc. of the 10th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS'02)*, Oct. 2002.

[31] H. Zeng, X. Fan, C. S. Ellis, A. R. Lebeck, and A. Vahdat. Currentcy: Unifying Policies for Resource Management. In *Proc. of the USENIX 2003 Annual Technical Conference*, June 2003.