Conserving Energy in Real-Time Storage Systems with I/O Burstiness

ADAM MANZANARES, XIAOJUN RUAN, SHU YIN, and XIAO QIN
Auburn University
ADAM ROTH
Big Tribe Corporation
and
MAIS NAJIM
University of Southern Mississippi

Energy conservation has become a critical problem for real-time embedded storage systems. Although a variety of approaches for reducing energy consumption have been extensively studied, energy conservation for real-time embedded storage systems is still an open problem. In this article, we propose an energy management strategy, I/O Burstiness for Energy Conservation (IBEC), exploiting the burstiness of real-time embedded storage systems applications. Our approach aims at combining the IBEC energy-management strategy with a Linux-based disk block-scheduling mechanism to conserve the energy of storage systems. Extensive experiments are conducted involving a number of synthetic disk traces as well as real-world data-intensive traces. To evaluate the energy efficiency of IBEC, we compare the performance of IBEC against three existing strategies, namely, PA-EDF, DP-EDF, and EDF. Compared with the alternative strategies, IBEC reduces the power consumption of real-time embedded disks system by up to 60%.

Categories and Subject Descriptors: D.4.2 [**Operating Systems**]: Storage Management—*Secondary storage*; D.4.4 [**Operating Systems**]: Communications Management—*Buffering*; D.4.8 [**Operating Systems**]: Performance—*Simulation*

General Terms: Design, Performance

Additional Key Words and Phrases: Disk scheduler, energy efficiency, linux

The work reported in this article was supported by the US National Science Foundation under Grants No. CCF-0742187, No. CNS-0757778, No. CNS-0831502, No. OCI-0753305, No. DUE-0621307, and No. DUE-0830831, and Auburn University under a start-up grant and the Intel Corporation under Grant No. 2005-04-070.

Author's addresses: A. Manzanares, X. Ruan, S. Yin, and X. Qin, Department of Computer Science & Software Engineering, Auburn University, Auburn, AL 36849-5347; email: {acm0008, xzr0001, szy0004, xqin}@auburn.edu; A. Roth, Big Tribe Corporation, San Francisco, CA 94107; email: aroth@bigtribe.com; M. Nijim, School of Computing , The University of Southern Mississippi, Hattiesburg, MS 39406; email: mais@orca.st.usm.edu.

DOI 10.1145/1698772.1698778 http://doi.acm.org/10.1145/1698772.1698778

ACM Reference Format:
Manzanares, A., Ruan, X., Yin, S., Qin, X., Roth, A., and Najim, M. 2010. Conserving energy in real-time storage systems with I/O burstiness. ACM Trans. Embedd. Comput. Syst. 9, 3, Article 20 (February 2010), 21 pages.
DOI = 10.1145/1698772.1698778 http://doi.acm.org/10.1145/1698772.1698778

1. INTRODUCTION

Because of the rapid advances in computational power, disk performance, and high-speed networks, storage systems have attracted increased interest. This trend can be attributed to the rapidly growing demands of data-intensive applications, which include but are not limited to video surveillance [Avitzour 2004], remote-sensing database systems [Chang et al. 1997], out-of-core applications [Qin et al. 2005], digital libraries [Sumner and Marlino 2004], and long-running simulations [Tanaka 1993]. The main components of real-time embedded systems include VLSI chips and hard-disk drives [Benini et al. 2000]. Recent developments in magnetic disk manufacturing techniques have made it possible to provide real-time embedded systems with small form-factor disk drives with high capacities [Zedlewski et al. 2003]. Often, hard-disk drives are one of the highest energy-consuming components of a computing system [Greenawalt 1994; Zheng et al. 2003]. A recent study shows that storage devices account for almost 27% of the total energy consumption of a computing system [Maximum Institution 2002]. Emerging high-speed disks with high-power needs exacerbate this problem. Hence, reducing the power consumption of hard disks is a reasonable approach to achieving long battery life for real-time embedded systems. In recent years, processor energy conservation for real-time and non–real-time embedded systems [Xie and Qin 2008; Zong et al. 2008] and parallel computing systems [Liu et al. 2008; Ruan et al. 2007; Zong et al. 2007] has been extensively studied. However, energy conservation for disk storage components (see, e.g., Nijim et al. [2008] and Ruan et al. [2009]) in embedded real-time systems remains an open problem. The lack of such energy conservation technologies becomes critical because, without it, reducing energy consumption of storage systems for real-time embedded applications is highly unlikely.

In this study, we aim to develop a novel energy-management strategy for real-time embedded storage systems. Our final goal is to combine our energy-management strategy with kernel disk scheduling mechanisms to conserve the energy consumed by storage systems. The conventional implementations of real-time disk schedulers typically use Linux block device drivers. Kim et al. [2003] investigated the possibility that the Disk Internal Scheduler (DIS) could possibly reorder real-time requests, but we assume that the hardware disk scheduler preservers the order of requests by the kernel. This assumption should be reasonable considering the fact that DIS can be turned off at the expense of disk performance. Our goal is to maximize energy, so we propose that the DIS can be turned off to trade energy efficiency for a small hit to performance when modern disk schedulers are used. The proposed approach can minimize the energy consumption of real-time embedded disks, while making the best effort to meet timing constraints of real-time disk requests.

The rest of this article is organized as follows. In Section 2 we summarize the related work. Section 3 describes a model of energy-management in real-time embedded disks. In Section 4, we propose the energy-management strategy. Section 5 presents the experimental results, and Section 6 concludes the article.

2. RELATED WORK

Many excellent studies exist on the subject of storage systems. Previous techniques used to improve the performance of storage systems include caching and buffering [Forney et al. 2001; Huber et al. 1995; Ma et al. 2002], parallel file systems [Cho et al. 1997; Ligon and Ross 1996; Preslan et al. 1999], load balancing [Qin et al. 2005; Scheuermann et al. 1998], and disk striping [Bordawekar et al. 1994; Salem and Garcia-Molina 1986]. Disk-scheduling mechanisms play an equally important role in bridging the performance gap between CPUs and disks [Coffman and Hofri 1990; Jacobson and Wilkes 1991; Seltzer et al 1990; Yu et al. 1993]. Although the shortest seek time first (SSTF) algorithm is efficient in minimizing seek times, it is starvation-bound and unfair in nature. The SCAN scheduling algorithm is effective at tackling the unfairness problem while optimizing seek times [Denning 1967]. Reist and Daniel [1987] proposed a parameterized generalization of the SSTF and SCAN algorithms. These disk-scheduling algorithms are inadequate for meeting the demands of disk requests with timing constraints.

Many data-intensive applications are real time in nature in the sense that disk requests must be completed before specified deadlines [Reuther and Pohlack 2003]. To fulfil the real-time requirements, Seltzer et al. [1990] developed the SCAN-EDF algorithm, which can process disk requests in a timely manner. While some disk schedulers were implemented for a mixed-media dataset, a mixture of data accessed by multimedia applications and best-effort applications [Balafoutis et al. 2003], other disk-scheduling algorithms were proposed to provide quality-of-service guarantees for different classes of applications [Bruno et al. 1999; Reddy and Wyllie 1999; Shenoy and Vin 1998]. Our approach differs from the existing disk-scheduling algorithms in that ours can effectively conserve energy by combining an energy management strategy with real-time disk-scheduling algorithms. Our strategy can be readily integrated into any existing disk scheduler to minimize the energy consumption of real-time embedded storage systems.

Research has been carried out concerning disk energy management in the context of storage servers [Colarelli and Grunwald 2002; Hong and Potkonjak 1996]. Most research in disk energy-management has focused on the issue of when disks should be put to sleep to reduce power consumption while maintaining high performance [Douglis et al. 1994; Helmbold et al. 2000; Weissel et al. 2002]. Gurumurthi et al. [2003] proposed an approach to reduce the disk energy consumption by making use of multispeed disks with smaller spin-up and spin-down times. Carrera et al. [2003] developed an energy conservation technique to save energy by combining laptop disks and server disks. Manzanares et al. [2008] designed a prefetching mechanism to reduce energy

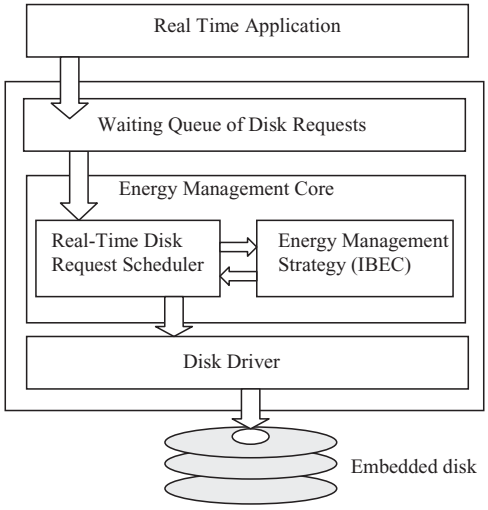

Fig. 1. Architecture of energy management in real-time storage systems.

consumption in parallel disk systems. Very recently, we implemented a distributed energy-efficient scheduling algorithm for real-time data-intensive applications running on Data Grids [Liu et al. 2008]. To measure the effectiveness of different dynamic energy-management policies developed for hard disks, Lu et al. [2000] implemented filter drivers that are able to record disk accesses and also analyze the performance impact of the energy-management overhead. These energy management schemes are focused on non–real-time applications and are unable to guarantee deadlines of real-time disk requests. The major difference between our approach and the prior disk energy conservation techniques is that ours can reduce energy consumption of disks while meeting deadlines of disk requests. Cheng and Goddard [2006] propose an online energy-efficient I/O scheduling algorithm for hard real-time systems. Although closely related, our work focuses on disk systems with soft real-time deadlines. Li et al. [2005] investigated using slack times to conserve energy consumption of memory and dynamic RPM disk systems. Our work also leverages deadlines containing slack time, but with low-power and commercially available disk systems.

3. MODEL OF ENERGY MANAGEMENT

3.1 Architecture

First, we build an abstract energy-management model for real-time embedded disk systems. The model depicted in Figure 1 is based on the concept of an energy-management strategy, which is combined with a real-time disk scheduler. The energy-management strategy is designed to manipulate the hard-disk power states and delay the execution of requests in a waiting queue of requests. While the real-time disk scheduler implements generic logic and timing

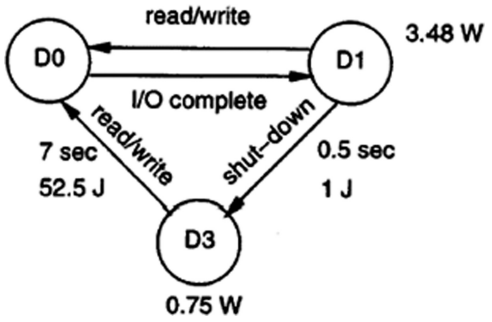

Fig. 2. Power-state model of the IBM DTTA 350-640 HDD [Benini et al. 2000].

mechanisms for scheduling and waiting, the disk driver is responsible for controlling access to a real-time embedded disk [Nijim et al. 2006].

The energy-management strategy was implemented in conjunction with a real-time disk-scheduling algorithm Earliest Deadline First, or EDF. By the virtue of the chosen implementation architecture, our strategy can be readily integrated with any real-time disk-scheduling policy. In this study, we focus on a storage system with a single disk, although the proposed strategy can be extended to storage systems with parallel and distributed disks. There are two main approaches to extending this work to parallel disk systems. The first and easiest is to just duplicate the energy-management core for each disk added to the disk system. To take advantage of the added bandwidth of the parallel disks, the kernel or the user would have to manage the placement of data in the disks. The second approach is to present one energy-management core to the kernel but attach many disks to the energy-management core. The second approach requires careful planning on how to manage the data on the disks, and we plan to save this for future research.

Energy management can be accomplished in a variety of ways in both real-time and non–real-time systems. As of now, most energy-aware hardware components support multiple power states [Benini et al. 2000]. For example, a CPU may implement a feature such that when it is idle, its clock-rate is reduced by a large factor, which reduces the core voltage. This, in turn, reduces power consumption and thermal dissipation. In this study, however, we are only concerned with hard-disks, which attempt to manage energy in a similar way. Most disks currently in deployment only explicitly support two operating power states, an active state and a sleep state. When in the active state, the disk is fully operational. When in the sleep state, the motor turning the disk platter is disabled, thus conserving energy.

Transitioning between these states is a process, which consumes a substantial amount of time and energy, especially when transitioning from the sleep state to the active state [Benini et al. 2000]. Given the high cost associated with transitioning from one state to another, it is easy to see that minimizing the number of power state transitions such that disks are kept sleeping as long as possible can substantially reduce energy consumption of disks.

3.2 Modeling Disk Requests and Energy Consumption

Each real-time disk request submitted to a disk system specifies its timing constraint in the form of a deadline, by which the request must be completed by the disk system. Note that deadlines of disk requests can be derived from real-time applications issuing the disk I/O operations. A disk request r is modelled by three parameters, $r = (a, s, t)$, where a is the disk address, s is the data size measured in KB, and d is the specified deadline. To model disk power consumption, we use a finite state machine representation built around performance data gleaned from an IBM DTTA 350-640 hard-disk drive. Let $R = \{r_1, r_2, \ldots r_n\}$ be a list of disk requests issued to a real-time embedded disk system. Let T_{on} and T_{off} denote the time intervals in which the disk system is active and inactive, respectively. We refer to T_{tr} as the time required entering and exiting the inactive state. The total energy consumed by the disk system can be computed as

$$\begin{aligned} E_{total} &= E_{on} + E_{off} + E_{tr} \\ &= T_{on}P_{on} + T_{off}P_{off} + T_{tr}P_{tr}, \end{aligned} \tag{1}$$

where P_{on} and P_{off} are the power of the disk in the active and inactive state, respectively, and P_{tr} is the state transition power.

The transition time T_{tr} and transition power P_{tr} can be computed by the following two equations.

$$T_{tr} = N_{on,off}T_{on,off} + N_{off,on}T_{off,on}, \tag{2}$$

$$\begin{aligned} P_{tr} &= f_{on,off}P_{on,off} + f_{off,on}P_{off,on} \\ f_{on,off} &= \frac{T_{on,off}}{T_{on,off} + T_{off,on}},\ f_{off,on} = \frac{T_{off,on}}{T_{on,off} + T_{off,on}}, \end{aligned} \tag{3}$$

where $T_{on,off}$ and $T_{off,on}$ are the required times to enter and exit the inactive state, $P_{on,off}$ and $P_{off,on}$ are the power values when the disk enters and exits the inactive state. In Equation (2), $N_{on,off}$ and $N_{off,on}$ are the number of times the disk enters and exits the inactive state. Without loss of generality, we assume that $N_{on,off}$ and $N_{off,on}$ are identical. Thus, Equation (2) can be simplified as

$$\begin{aligned} T_{tr} &= N_{tr}(T_{on,off} + N_{off,on}), \\ N_{tr} &= N_{on,off} = T_{off,on}. \end{aligned} \tag{4}$$

In general, T_{on} in Equation (1) is the sum of the total serve and total idle times (See Eq. 5).

$$T_{on} = T_{serve} + T_{idle}, \tag{5}$$

where the total serving time is the sum of the serving time of each disk request. Thus, we have

$$\begin{aligned} T_{serve} &= \sum_{i=1}^{n} T_s(i), \\ T_s(i) &= T_{seek}(i) + T_{rot}(i) + T_{trans}(i), \end{aligned} \tag{6}$$

where T_{seek} is the amount of time spent seeking the desired cylinder, T_{rot} is the rotational delay, and T_{trans} is the amount of time spent actually reading from or writing to the disk.

We can quantitatively compute the energy saved by the energy-management strategy using the following equation:

$$\begin{aligned} E_{save} &= (T_{on} + T_{off} + T_{tr}P_{on}) - E_{total} \\ &= (T_{on} + T_{off} + T_{tr}P_{on}) - \left(T_{on}P_{on} + T_{off}P_{off} + T_{tr}P_{tr}\right) \\ &= (P_{on} - P_{off}T_{off}) + (P_{on} - P_{tr}T_{tr}). \end{aligned} \tag{7}$$

Our energy-management strategy seeks to save energy by minimizing the disk system's total energy consumption (see Equation (1)). Therefore, we can obtain the following problem formulation, where f_i is the finish time of the ith disk request.

$$\begin{aligned} &\text{Minimize } E_{on} + E_{off} + E_{tr} \\ &\text{Subject to } \forall r_i \in R : f_i \leq d_i. \end{aligned} \tag{8}$$

4. THE ENERGY-MANAGEMENT STRATEGY

This section presents our energy-management strategy leveraging I/O Burstiness for Energy Conservation (IBEC). It is assumed that the energy overhead of running IBEC is negligible when compared to the processing times of disk requests. This assumption is reasonable because recent studies show that energy-management strategies spend less than 1% of computation time on power managers [Lu et al. 2000].

The IBEC algorithm is based on the idea that by delaying the execution of some requests, it is possible to allow the disk to remain in a "sleep" state for a longer duration of time. It also processes larger contiguous blocks of requests when the disk is active, thus making better use of the disk when it is fully powered. The implications of this are twofold. First, we improve the efficiency of the disk by reducing the amount of time that it sits idle in the "active" state. Second, we also reduce the total number of power state transitions that the disk will make when servicing a given request-stream.

This is important because transitioning from the sleep state to the active state is expensive both in terms of power consumed and the amount of time that it takes to complete the transition. By reducing the number of these transitions, it is possible to substantially improve the power-consumption characteristics of the system. If a request arrives while the disk is in the sleep state, and we determine that the request's deadline does not require immediately waking the disk, then we cache the request for later processing.

In fact, under the IBEC algorithm, if the disk is in the sleep state, we do not wake it until we determine that we must do so in order to guarantee the deadline of a(the) waiting disk request(s). Note that it is probable that while we are waiting to wake the disk to service our waiting request, other requests will continue to arrive. These requests are also queued until the disk is transitioned to the active state, at which point all waiting requests are serviced at once. This is the mechanism by which IBEC reduces the number of state transitions. It also attempts to create contiguous blocks of requests to service out of a potentially

arbitrarily sporadic stream of requests. Also, note that as more requests end up queued and consequently delayed, it is necessary to not just ensure that we guarantee the deadline of the first request in the waiting queue. We must also ensure that by delaying the first request, we are not causing the deadlines of subsequent requests to become unrealizable. Note that these deadlines are for soft real-time applications and that missed deadlines are tolerable. Thus, it may be the case that we will need to transition to the active state some length of time before it is absolutely imperative to do so to meet the deadline of the first request, such that we can still hope to guarantee the deadlines of subsequent requests. This imposes additional requirements on the algorithm. First, it becomes necessary to re-evaluate the threshold, at which we must transition the disk to the active state whenever a new request is added to the waiting queue. Second, there must be a mechanism for estimating the amount of time spent in serving a given request, given what we currently know about the state of the system. The first criteria is fairly trivial, the second is a bit less so; however, the estimation does not need to be exact, so long as it is always greater than or equal to the worst-case service time of the given request.

It is obviously better to be as close to exact as possible, but as long as our estimation is never less than the actual value, we can assume that we are not causing deadlines to be missed. It is possible to produce scenarios in which it is not possible to avoid missing a deadline when applying this energy-management policy. As we will see, this sort of situation actually occurs only rarely in practice. A simple way to estimate the worst-case service time of a request, while remaining reasonably accurate within the bounds of the system, is to simply assume that the request will require seeking the read/write head entirely across the disk. We also assume that once the head is in the correct position, we will have to wait for one full revolution of the hard-disk. Assuming that the disk we are working with performs consistently in terms of its read and write speeds, which for the purposes of our simulation it does, we then have a request service time estimate. This estimate is not only reasonably cheap to compute, but it should also always exceed the actual service time of any real disk request without being so far off from the actual value as to render such an estimation pointless. It should also be noted that by itself, IBEC is an energy-management policy, not a scheduling algorithm. It is meant to be implemented on top of, or in conjunction with, a separate scheduling algorithm, which should be a real-time scheduling algorithm, such as EDF or LLF. As requests arrive, they are first scheduled according to the real-time scheduling algorithm being used and then processed by the IBEC energy-management policy. A high-level algorithmic description of IBEC is given in Figure 3. (Note that this description assumes that any arriving requests have already been properly scheduled by the associated scheduling algorithm).

Step 7 (see Figure 4) verifies that deadlines can still be guaranteed as new requests are added to the waiting queue. Step 7 is responsible for estimating slack times based on known arrival times and I/O service times. The IBEC strategy (see Figure 3) makes use of the estimated slack times to improve disk energy-management. Generally speaking, as more slack times are approximated by Step 7, the energy-efficiency potential of the IBEC algorithm rises.

```
1.  if there are no requests active or pending then
2.    the disk enters the inactive state;
3.  else
4.      if the disk is active then
5.        dispatch a request in the queue with the earliest deadline;
6.      else /* the disk is inactive */
7.      while deadlines of all the waiting requests can be guaranteed do
8.        Cache any requests that have arrived;
9.        the disk enters the active state again to meet timing constraints;
10.       dispatch any waiting requests;
```

Fig. 3. The IBEC energy-management strategy.

```
1.  while count <  requests.size
2.    estimate=requests[count].estimate + requests[count+1].estimate
3.    if estimate > request[count+1].arrive_time + request[count+1].deadline
4.      if enough slack time available
5.        use available slack time
6.        update slack time
7.      else not enough slack time available
8.         dispatch requests
9.      end if
10.   end if
11.   count++
12. end while
```

Fig. 4. IBEC Step 7 details.

To support the presentation of Step 7, let us describe a way of calculating I/O slack times. Disk request r_i is characterized by four parameters, $r_i = (o_i, a_i, l_i, d_i)$, where o_i indicates that the request is a read or write, a_i is the disk address, l_i is the data size measured in kilobytes (KB), and d_i is the specified deadline. Let es_i and s_i denote the earliest start time and I/O service time of request r_i, we can calculate the slack time sl_i of r_i, as shown in Equation (9):

$$\begin{aligned} sl_i &= d_i - (es_i + s_i) \\ &= d_i - \left(\rho_j + \sum_{r_k \in R, d_k \le d_i} s_k + s_i\right). \end{aligned} \tag{9}$$

The earliest start time es_i in Equation (9) can be computed by $\rho_j + \sum_{r_k \in R, d_k \le d_i} s_k + s_i$, where ρ_i represents the remaining I/O service time a disk request currently being processed, and $\sum_{r_k \in R, d_k \le d_i} s_k$ is the I/O service time of requests (e.g., r_k) whose deadlines are earlier than or equal to that of r_i. In other words, the earliest start time es_i of request r_i is a sum of the remaining service time of the request being processed and the aggregated service times of the tasks with earlier deadlines.

For a given disk request r_i, we can derive its I/O service time s_i using Equation (9) as follows,

$$s_i = T_{seek}(r_i) + T_{rot}(r_i) + \frac{l_i}{B_{disk}}, \tag{10}$$

where $T_{seek}(r_i)$ and $T_{rot}(r_i)$ are the seek time and rotational latency, $\frac{l_i}{B_{disk}}$is the data transfer time that largely depends on the data size l_i of the request and the disk bandwidth B_{disk}. The disk bandwidth used in our simulations is 35MB/s for a read operation and 25MB/s for a write operation.

It is assumed that disk requests are randomly distributed on the sectors of given cylinders. Thus, rotational latency $T_{rot}(r_i)$ in Equation (10) can be derived from R_{PS} (revolutions per second) using the following equation. Note that R_{PS} is 120 for our experimental evaluation of IBEC.

$$T_{rot}(r_i) = (1/R_{PS})/3. \tag{11}$$

The seek time $T_{seek}(r_i)$ of request r_i can be approximated as a function of the number of cylinders (i.e., N_C) the disk has to travel. In our experiments, the number of cylinders to be traveled is fixed at 6,400.

$$T_{seek} = 0.06 * \sqrt{N_C}. \tag{12}$$

Now, we are in a position to outline the following three properties of the IBEC algorithm.

Property 1. If a newly arrived disk request r_i has a feasible schedule under IBEC, the deadline of r_i must be guaranteed. Thus, the slack time of r_i must be larger than or equal to zero. The condition below must be met in accordance with Equation (9):

$$\rho_j + \sum_{r_k \in R, d_k \leq d_i} s_k + s_i \leq d_i.$$

Property 2. Let us consider a newly arrived disk request r_i with deadline d_i and a set (i.e., R) of pending disk requests that have been previously scheduled. The deadlines of all the disk requests in R must be guaranteed. Since the requests whose deadlines are earlier than d_i are not affected by the newly arrived request r_i, the requests whose deadlines are later than d_i must be processed before their specified deadlines. Thus, the IBEC algorithm needs to meet the following condition

$$\forall r_k \in R, d_k > d_i : es_k + s_k \leq d_k.$$

Property 3. If the disk is in the sleep state, the start time of an accepted disk request r_i is delayed as late as possible under the condition that all the accepted disk requests must be completed before their deadlines. The basic idea in the implementation of Step 7 (see Figure 4) is to estimate the I/O service time of two subsequent requests using Equation (10). If this estimate is greater than the second requests deadline plus its arrival time (see also Property 1), there exists a scheduling issue. Specifically, if there is enough slack time (see Equation (9)) available, the slack time is used (see Property 3) and the available

slack time is adjusted accordingly. If enough slack time is not available for any one of the requests, all the requests will be dispatched to minimize the performance penalty. Step 7 has a time complexity of $O(n^2)$, where n is the number of requests. This can be derived from Figure 4. The while loop runs $O(n)$ times, but the update slack time operation has a complexity of $O(n)$. This is due to the fact that updating the slack time updates the time when the first request will be serviced, which affects all subsequent requests.

Please note that it is desirable to allow the disk to sit idle for some threshold of time before sleeping it, as opposed to immediately sleeping the disk as soon as there are no requests active or pending. This is due to the relatively large amount of power consumed in transitioning from the sleep state back to the active state, which creates a situation in which it could conceivably be more efficient to allow the disk to sit idle for a brief period of time in hopes that more requests will arrive. The exact threshold at which it is no longer beneficial to allow the drive to sit idle is a function of the disk's power consumption characteristics as well as the relative distribution of requests in the request-stream. This threshold can be computed if we know details about our request-stream in advance, or if we have some way of making a best-guess based on an observation of the request-stream at runtime [Benini et al. 2000]. This, however, is beyond the scope of both this research and the IBEC algorithm in its current incarnation.

5. EXPERIMENTAL RESULTS

In this section, we evaluate the performance of the IBEC algorithm using synthetically generated traces as well as real-world application traces. To compare the impact of our algorithm on energy savings and guarantee ratio, we compare IBEC with three existing scheduling algorithms, Earliest Deadline First (EDF), Delayed Power EDF (DP-EDF), and Power-Aware EDF (PA-EDF). We added existing DPM techniques to the EDF scheduling algorithms to come up with DP-EDF and PA-EDF to compare against our IBEC strategy [Benini et al. 2000]. EDF is a well-known real-time scheduling scheme with no power-awareness at all. The three baseline algorithms are briefly described in the following text. IBEC and DP-EDF require an idle threshold to be defined, which controls the amount of idle time to pass before sleeping a disk. In our experimental results, this idle threshold is set to 100ms.

(1) EDF: The disk request with the earliest deadline is always executed first.
(2) DP-EDF: Switch a disk to the sleep state whenever it has been idle for a certain amount of time and wake it up immediately whenever a request arrives. All arrived requests will be processed according to EDF order.
(3) PA-EDF: Switch a disk to the sleep state whenever it is idle and wake it up immediately whenever a request arrives. All arrived requests will be processed in an EDF order.

The first goal of the performance evaluation is to examine the energy consumption and guarantee ratio of IBEC as compared to the baseline algorithms when the average deadline or requests is varied. Second, we will investigate

Table I. Simulation Parameters

Parameter	Fixed Value	Varied Value
Minimal deadline	20ms	1, 10, 20, 50, 100, 250, 1,000, 2,000, 5,000, 10,000, 15,000ms
Maximal deadline	100ms	10, 50, 100, 250, 1,000, 2,000, 5,000, 15,000, 20,000, 35,000ms
Request arrival rate	0.5	0.05, 0.15, 0.25, 0.35, 0.45, 0.5, 0.55, 0.65, 0.75, 0.85, 0.95, 1.0
Request arrival pattern	Normal	Normal, Clustered, Sparse
Minimal request size	24KB	24, 128, 512, 1,024, 4,096, 16,384, 32,168, 128,672, 514,688, 1,029,376, 2,058,752, 8,235,008, 16,470,016KB
Maximal request size	32,168KB	512, 1,024, 4,096, 16,384, 32,168, 128,672, 514,688, 10,29,376, 2,058,752, 8,235,008, 16,470,016, 65,880,064KB

the impact the request distribution pattern has on power consumption and guarantee ratio. Third, we evaluate how the power consumption and guarantee ratio of IBEC are impacted when the request arrival rate is varied. Last but not least, we validate the results from the synthetic real-time requests by running three real world real-time applications with IBEC. All of the disk parameters used in the simulations are based on Figure 2.

5.1 Simulation Set-up

Before presenting the empirical results in detail, we present the simulation model as follows. Table I summarizes the key configuration parameters of the simulated disk scheduling system used in our experiments. The fixed value parameters remain constant when we are attempting to quantify the impact of changing the varied value on the IBEC strategy. For example, in Figures 7 and 8, the fixed parameters are the minimal deadline of 20ms, maximal deadline of 100ms, request arrival rate of 0.5, the minimal request size is 24KB and the maximal request size is 32,168KB. The Varied Value is the request arrival pattern and it is a Normal, Clustered, or a Sparse request distribution pattern. The experiments in Figures 7 and 8, attempt to study the impact that the arrival pattern has on the IBEC strategy.

5.2 Impact of Average Deadline

The goal of this experiment is twofold: (i) to compare the proposed IBEC algorithm against the three alternatives and (ii) to understand the impact of changing the average request deadline has on IBEC. The average deadline is found by generating a deadline for each request between the min and max deadline parameter and then taking the average of all the requests deadlines. For example the min deadline and max deadline used for the first experiment are 50ms and 100ms, respectively, and for the second experiment, 100ms and 250ms, with the last experiment using 15,000ms and 35,000ms.

Figure 5 shows the energy consumption of these four algorithms when the average request deadline varies from 75ms to 25,000ms. We observe from Figure 5 that each of the four algorithms consume the same amount of energy when the

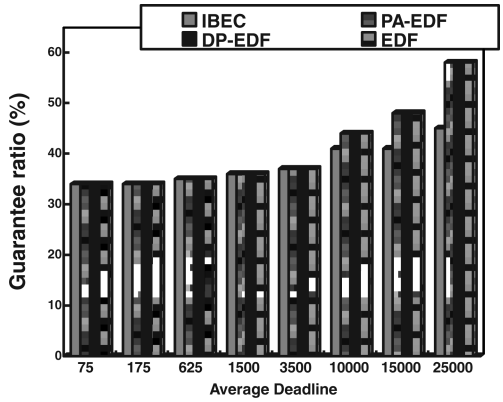

Fig. 5. Energy factor for different average request deadlines.

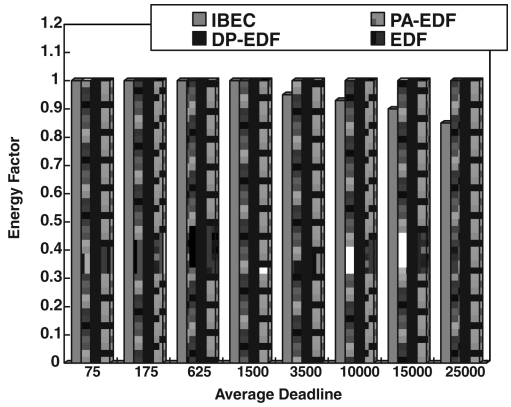

Fig. 6. Guarantee ratio for different average request deadlines.

average request deadline is less than 3,500ms. This is because the hard-disk has to be kept active to service the arriving disk requests, which have very tight deadlines. In other words, there is no opportunity for IBEC to conserve power. Therefore, IBEC gracefully degrades to existing power-aware scheduling algorithms, like DP-EDF and PA-EDF. When the average request deadline is equal to or larger than 3,500ms, however, IBEC starts to conserve more energy than the three baseline algorithms. We attribute the EF improvement of IBEC over the three baseline algorithms to the fact that IBEC judiciously employs loose deadlines to conserve energy. The improvement of IBEC over the three existing schemes in terms of EF grows larger as the average deadline is continually increased. Taking an average of the EF over the varying average deadlines, IBEC can save 10.8% of the energy consumed as compared to the three baseline algorithms.

Figure 6 plots the GR of the four algorithms when the deadline is increased from 75 to 25,000ms. It reveals that IBEC performs exactly the same, with respect to GR, as the baseline approaches when the deadline is less than 1,000ms.

This is caused by a combination of a high workload and tight deadlines. This forces IBEC to only concentrate on guaranteeing arrival requests timing constraints, which have a higher priority than the power conservation goal. Interestingly, the GR of IBEC suddenly drops off when the deadline is 10,000ms. In fact, this is an artifact of our specific implementation of the IBEC algorithm. In order to keep simulation times manageable and to approximate a real system, we limited the maximum number of requests that IBEC would ensure their deadline constraints. This was also intended to model real-world computing systems, where no infinite amount of time is available to re-evaluate the schedulability of a queue. In particular, when the length of the waiting queue of requests is larger than 1,000, our implementation of the IBEC algorithm will no longer guarantee the schedulability of requests after the 1,000th. The 10,000ms deadline allows the number of requests to grow larger than 1,000, thus the schedulability of the requests are no longer guaranteed and the GR drops. This number was chosen because of the $O(n^2)$ time complexity of the IBEC mechanism that discovers if deadlines can still be met. With n representing the size of the disk requests in the waiting queue, large waiting queues can negatively impact performance. The limit of 1,000 requests keeps IBEC running in a reasonable amount of time, while still maintaining energy savings.

5.3 Impact of Request Arrival Pattern

This section is focused on the performance impact of the request arrival pattern. Specifically, we evaluate the performance of the four algorithms where the disk request arrival pattern follows a Normal request arrival pattern, a Sparse request arrival pattern, and a Clustered request arrival pattern. Request arrival pattern is defined as the method for generating time stamps for the requests. For the normal request arrival pattern, a request is generated based on the arrival rate and the default time step, which is 1ms for our experiments. The generator advances the time step during every iteration of its execution and writes a request only if the arrival rate is larger than a uniform random number generated at each iteration. The sparse request arrival pattern follows a similar pattern, except it adds a random uniformly generated extra time, from zero to the sparse idle threshold, to the time step when a request is to be generated. The clustered request arrival pattern is also similar to the normal request arrival pattern, except it writes a cluster of requests when required instead of one request. The cluster size falls between a maximum and minimum value and these values are depicted in Figures 7 and 8.

The normal request arrival pattern is our standard pattern, which models applications that write or read a request based on a uniform random number at fixed intervals. The other two request arrival patterns follow this request arrival pattern, except they either vary the time step or vary the number of requests written at a given time step. The sparse request arrival pattern adds a uniform random time to the time step, so the intervals are no longer fixed. This makes the workload lighter as compared to the normal distribution. The clustered request arrival pattern represents bursty traffic, since it generates a cluster of requests at the fixed intervals. The letters on the x-axis denote

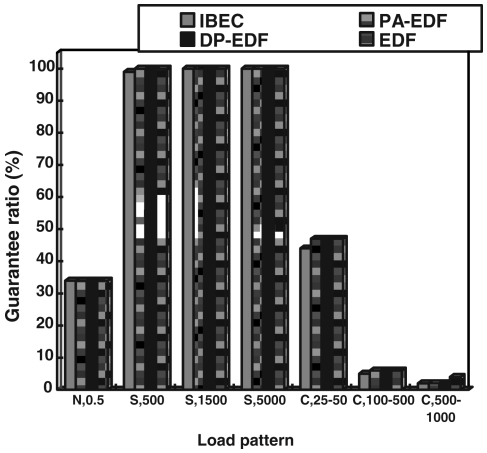

Fig. 7. Energy factor for different workload distribution.

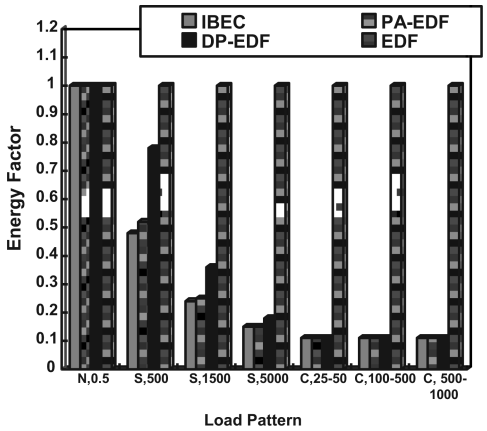

Fig. 8. Guarantee ratio for different workload distributions.

the names of the request arrival pattern being used. For example, "N" means normal, "S" means sparse, and "C" means clustered. The numbers following the letters denote the parameters that were specified to the corresponding request arrival patterns. For instance, "N, 0.5" denotes a normal request arrival pattern with an arrival rate of 0.5, "S, 500" indicates sparse request arrival pattern with a sparse idle threshold of 500ms, and "C, 25-50" indicates a clustered request arrival pattern mode with between 25 and 50 requests occurring per cluster.

From Figure 7, we can make three important observations. First, all algorithms perform identically in power consumption under the Normal distribution. Second, the three power-aware algorithms noticeably outperform the EDF scheme, which has no power-awareness at all, when the Sparse distributions were applied. This is because the Sparse distribution produces a relatively large time interval between two continuous disk requests, which in turn gives

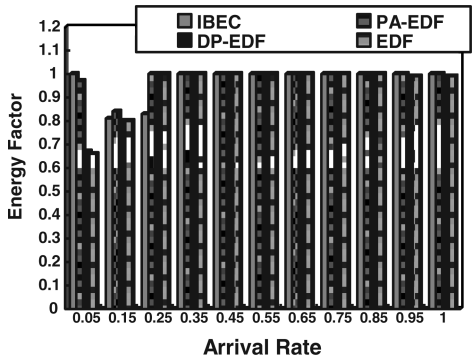

Fig. 9. Energy factor for different arrival rates.

the three power-aware algorithms many chances to switch the hard-disk to the sleep mode to save energy. Furthermore, IBEC slightly outperforms DP-EDF and PA-EDF, two naïve power-aware algorithms. The rationale behind this phenomenon is that IBEC can make the most use of the slack time of each arriving request. To put it in another way, IBEC only wakes up the disk at the last second from which all arrived requests' deadlines can still be met, while DP-EDF only lets the disk sleep for a fixed period of time no matter whether a request is waiting for service or not. Third, for clustered workloads, IBEC and the two naïve power-aware techniques perform comparably, and they both significantly perform better than EDF. This is due to the fact that IBEC, DP-EDF, and PA-EDF can put the disk to the sleep status completely between clusters of requests. Thus, the three power-aware algorithms can substantially save power compared with EDF. The reason why IBEC ties with DP-EDF and PA-EDF is that the performance improvement of IBEC in terms of power consumption depends on slack times of arriving requests rather than the overall arrival patterns.

The results reported in Figure 8 reveal that all of the four algorithms deliver a 100% guarantee ratio under the Sparse distribution. The reason for this is that the average request deadline is generally much shorter than the sparse idle threshold, which means that even though IBEC aggressively slows down the processing pace of disk requests, their deadlines can still be satisfied. When we applied a Cluster distribution pattern, the performance of the four algorithms goes down when the parameters of the Cluster distribution increase. This is because a large number of requests arrived during a cluster of incoming requests. Consequently, all the algorithms can only guarantee the deadlines of a small part of them.

5.4 Impact of the Request Arrival Rate

To examine the relationship of GR and EF of the four algorithms to the request arrival rate we varied the arrival rate from 0.05 to 1 using a Normal distribution. The first observation drawn from Figure 9 is that all of the four algorithms have almost the same low performance in energy factor when arrival rate is larger than 0.25. This scenario represents heavier workloads, which keep the

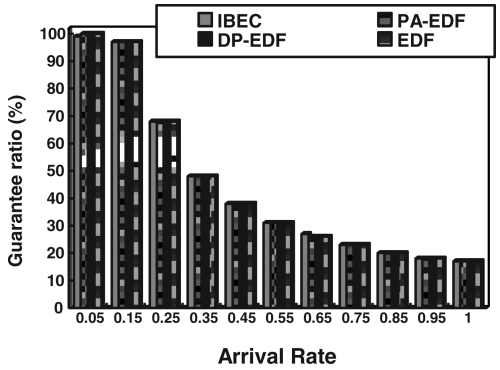

Fig. 10. Guarantee ratio for different arrival rates.

Table II. Impact on System Energy

Disk Power Consumption %	IBEC Energy Savings %	System Energy Savings %
10%	10%, 35%, 60%	1%, 4%, 6%
15%	10%, 35%, 60%	2%, 5%, 9%
27%	10%, 35%, 60%	3%, 10%, 16%

disk busy all the time and make it nearly impossible to conserve power. There is an interesting "anomaly" that occurs in the first two experimental cases. The only non–power-aware algorithm, EDF, consumes the least amount of power. Indeed, this snapshot demonstrates how high the overhead of transitioning from the sleep state to the active state can be. The work-stream is so heavy that it just does not allow the disk to be left in the sleep state long enough to offset the overhead of transitioning back to the active state. It is also worth noting that at these two arrival rates, IBEC outperforms the other two naïve power-ware algorithms in terms of EF by 20%. In terms of GR, all of the four algorithms performed identically as depicted by Figure 10. More importantly, IBEC performs identically to other scheduling algorithms, implying that IBEC can maintain a comparable performance in GR while noticeably improving power consumption.

Table II illustrates the energy impact that IBEC may have on the overall computing system using simulated energy savings and the maximum overall disk energy consumption from [Zedlewski et al. 2003]. The disk power consumption column represents the total percentage of system energy that the disk system consumes. This allows us to estimate the total system energy savings IBEC may have on the entire computing system.

5.5 Real-World Applications

To validate the results from the synthetic simulations, we applied our IBEC algorithm to three real-world applications. Table III briefly introduces the three applications. These real-world applications are I/O-intensive parallel applications that do not have explicit real-time deadlines. We modified the traces to include real-time deadlines. There are some very positive observations that can

Table III. Real-World Applications

Application	Definition
Data-mining	Extracts association rules from retail data
Factorization	Compute the dense LU decomposition of an out-of-core
Search activity	Partial match and approximate searches

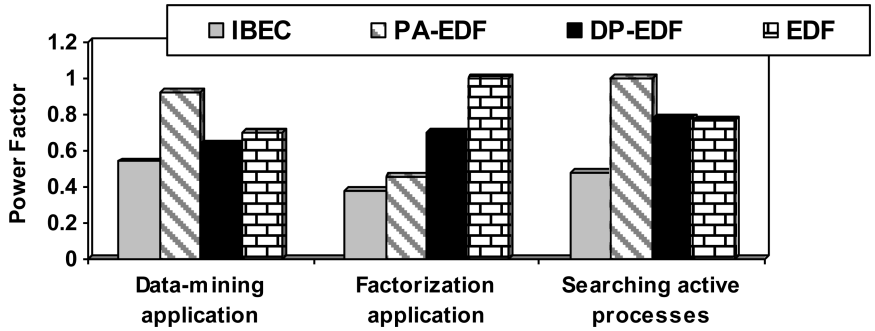

Fig. 11. Power consumption for data mining, factorization, and searching active processes applications.

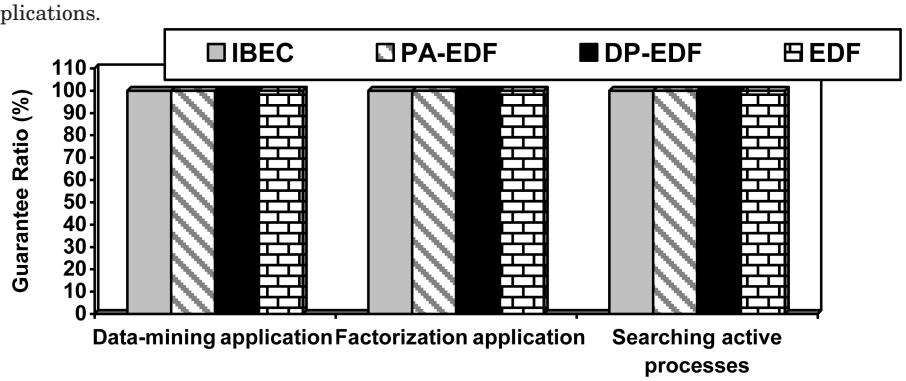

Fig. 12. Guarantee ratio for data mining, factorization, and searching active processes applications.

be drawn from Figures 11 and 12. For all three applications, IBEC is able to offer GRs that are comparable with those provided by other algorithms, while consistently delivering reduced power consumption over the baseline algorithms. In some cases, the improvement in terms of EF is marginal, but in most situations it is significant. Table IV summarizes the power consumption improvements of IBEC over the three baseline algorithms. The most desirable feature of IBEC is that the power savings are consistent across every test conducted. We can therefore have confidence that IBEC is a valid mechanism of improving the power consumption of a system without compromising scheduling performance.

These results are promising because they show that IBEC can be reliably used to consistently conserve at least a modicum of energy. The IBEC algorithm is also a very effective way to conserve energy if it is deployed in a system where a schedulable stream of disk requests can always be generated.

Table IV. Summary of Power Consumption Improvements

Algorithms	Dmine	LU	Pgrep
PA-EDF	37.8%	11.1%	54%
DP-EDF	9.6%	42.8%	42.5%
EDF	13.8%	60%	39.5%

6. SUMMARY

In this article, we developed an energy-management strategy (IBEC) leveraging I/O burstiness of real-time embedded storage system applications. To compare the performance of the IBEC strategy against three existing approaches (PA-EDF, DP-EDF, and EDF), we conducted extensive simulation experiments using synthetic workload conditions as well as real-world data-intensive applications. The results show that compared with the three alternative strategies, IBEC substantially reduces energy consumption by up to 60% while maintaining a comparable guarantee ratio.

In our future work, we will consider supporting multispeed disks where energy can be conserved by dynamically adjusting disk speed toward energy saving. We will investigate a way of applying IBEC into parallel I/O systems with multispeed disks. This extension will further improve energy efficiency of parallel disk systems.

ACKNOWLEDGMENTS

We are grateful to the anonymous referees for their insightful suggestions and comments for improving the quality of this article.

REFERENCES

AVITZOUR, D. 2004. Novel scene calibration procedure for video surveillance systems. *IEEE Trans. Aerospace and Electr. Syst. 40*, 3, 1105–1110.

ALGHAMDI, M., XIE, T., AND QIN, X. 2005. PARM: A power-aware message scheduling algorithm for real-time wireless networks. In *Proceedings of the Workshop of Wireless Multimedia Networking and Performance Modeling*. ACM, New York.

BALAFOUTIS, E., NERJES, G., MUTH, P., PATERAKIS, M., WEIKUM, G., AND TRAIANTAFILLOU, P. 2003. Clustered scheduling algorithms for mixed-media disk workloads in a multimedia server. *J. Cluster Comput. 6*, 1, 75–86.

BENINI, L., BOGLIOLO, A., AND MICHELI, G. D. 2000. A survey of design techniques for system-level dynamic power management. *IEEE Trans.VLSI Syst. 8*, 3, 299–316.

BISSON, T. AND BRANDT S. 2004. Adaptive disk spin-down algorithms in practice. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*. ACM, New York.

BORDAWEKAR, R., THAKUR, R., AND CHOUDHARY, A. 1994. Efficient compilation of out-of-core data parallel programs. Tech. Rep. SCCS-662, NPAC.

BRUNO, J., GABBER, E., OZDEN, B., AND SILBERSCHATZ, A. 1999. Disk scheduling algorithms with quality of service guarantees. In *Proceedings of the IEEE Conference on Multimedia Computing Systems*. IEEE, Los Alamitos, CA.

CARRERA, E. V., PINHEIRO, E., AND BIANCHINI, R. 2003. Conserving disk energy in network servers. In *Proceedings of the International Conference on Supercomputing*. ACM, New York.

CHANG, C., MOON, B., ACHARYA, A., SHOCK, C., SUSSMAN, A., AND SALTZ. J. 1997. Titan: A high-performance remote-sensing database. In *Proceedings of the 13th International Conference on Data Engineering*. IEEE, Los Alamitos, CA.

CHENG, H. AND GODDARD, S. 2006. EEDS_NR: An online energy-efficient I/O device scheduling algorithm for hard real-time systems with non-preemptible resources. In *Proceedings of the 18th Euromicro Conference on Real-Time Systems*. IEEE, Los Alamitos, CA.

Cho, Y., Winslett, M., Subramaniam, M., Chen, Y., Kuo, S., and Seamons, K. E. 1997. Exploiting local data in parallel array I/O on a practical network of workstations. In *Proceedings of the 5th Workshop on I/O in Parallel and Distributed Systems.* ACM, New York.

Coffman, J. R. and Hofri, M. Queueing models of secondary storage devices. *Stochastic Analysis of Computer and Communication Systems,* Ed. Hideaki Takagi, North-Holland, 1990.

Colarelli, D. and Grunwald, D. 2002. Massive arrays of idle disks for storage archives. In *Proceedings of the International Conference on Super-Computing.* ACM, New York.

Denning, P. J. 1967. Effects of scheduling on file memory operations. In *Proceedings of AFIPS Conference.*

Douglis, F., Krishnan, P., and Marsh, B. 1994. Thwarting the power-hunger disk. In *Proceedings of the Winter USENIX Conference.* USENIX, Berkeley, CA.

Forney, B., Arpaci-Dusseau, A. C., and Arpaci-Dusseau, R. H. 2002. Storage-Aware caching: Revisiting caching for heterogeneous storage systems. In *Proceedings of the International Symposium on File and Storage Technology.* USENIX, Berkeley, CA.

Greenawalt, P. M. 1994. Modelling power management for hard disks. In *Proceedings of the International Workshop Modelling, Analysis, and Simulation on Computer and Telecom. Systems.* IEEE, Los Alamitos, CA.

Gurumurthi, S., Sivasubramaniam, A., Kandemir, M., and Fanke, H. 2003. DRPM: Dynamic speed control for power management in server class disks. In *Proceedings of the International Symposium on Computer Architecture.* ACM, New York.

Helmbold, D. P., Long, D. D. E., Sconyers, T. L., and Sherrod, B. 2000. Adaptive disk spin-down for mobile computers. *Mob. Networks Appl. 5*, 4, 285–297.

Hong, I. and Potkonjak, M. 1996. Power optimization in disk-based real-time application specific system. In *Proceedings of the International Conference Computer-Aided Design.* ACM, New York, San Jose, CA.

Huber, J., Elford, C. L., Reed, D. A., Chien, A. A., and Blume, D. S. 1995. PPFS: a high performance portable parallel file system. In *Proceedings of the 9th ACM International Conference Super-Computing.* ACM, New York.

Jacobson, D. and Wilkes, J. 1991. Disk scheduling algorithms based on rotational position. Tech. rep. HPL-CSP-91-7.

Kim, K., Hwang, J., Lim, S., Cho, J., and Park, K. 2003. A real-time disk scheduler for multimedia integrated server considering the disk internal scheduler. In *Proceedings of the International Parallel and Distributed Symposium.* IEEE, Los Alamitos, CA.

Li, X., Li, Z., Zhou, Y., and Adve, S. 2005. Performance directed energy management for main memory and disks. *ACM Trans. Storage* 346–380.

Ligon, W. B. and Ross, R. B. 1996. Implementation and performance of a parallel file system for high-performance distributed applications. In *Proceedings of the IEEE International Symposium on High Performance Distributed Computing.* IEEE, Los Alamitos, CA.

Liu, C., Qin, X., Kulkarni, S., Wang, C. J., Li, S., Manzanares, A., and Baskiyar, S. 2008. Distributed energy-efficient scheduling for data-intensive applications with deadline constraints on data grids. In *Proceedings 27th IEEE International Performance Computing and Communications Conference.* IEEE, Los Alamitos, CA.

Lu, Y., Chung, E. Y., Šimunić, T., Benini, L., and de Micheli, G. 2000. Quantitative comparison of power management algorithms. In *Proceedings of the Design Automation and Test in Europe.* IEEE, Los Alamitos, CA.

Ma, X., Winslett, M., Lee, J., and Yu, S. 2002. Faster collective output through active buffering. In *Proceedings of the International Symposium on Parallel and Distributed Processing,* Ft. Lauderdale, FL.

Manzanares, A., Bellam, K., and Qin, X. 2008. A perfecting scheme for energy conservation in parallel disk systems. In *Proceedings of the NSF Next Generation Software Program Workshop.* IEEE, Los Alamitos, CA.

Maximum Institution. 2002. Power, Heat, and Sledgehammer. Maximum Institution Inc.

Nijim, M., Qin, X., Xie, T., and Alghamdi, M. 2006. Awards: An adaptive write scheme for secure local disk systems. In *Proceedings of the 25th IEEE International Performance Computing and Communication Conference.* IEEE, Los Alamitos, CA.

NIJIM, M., MANZANARES, A., AND QIN, X. 2008. An adaptive energy-conserving strategy for parallel disk systems. In *Proceedings of the 12th IEEE International Symposium on Distributed Simulation and Real-Time Applications.* IEEE, Los Alamitos, CA.

PRESLAN, K. W., BARRY, A. P., BRASSOW, J. E., ERICKSON, G. M., NYGAARD, E., SABOL, C. J., SOLTIS, S. R., D., TEIGLAND, D. C., AND O'KEEFE, M. T. 1999. 64-bit, shared disk file system for Linux. In *Proceedings of the NASA Goddard Conference on Mass Storage System.* IEEE, Los Alamitos, CA.

QIN, X., JIANG, H., ZHU, Y., AND SWANSON. D. R. 2006. Improving the performance of I/O-intensive applications on clusters of workstations. *J. Cluster Comput. 9*, 3, 297–311.

REIST, R. AND DANIEL, S. 1987. A continuum of disk scheduling algorithms. *ACM Trans. Comput. Syst.* 77–92.

REUTHER, L. AND POHLACK, M. 2003. Rotational-position-aware real-time disk scheduling using a dynamic active subset. In *Proceedings of the IEEE Real-Time System Symposium.* IEEE, Los Alamitos, CA.

RUAN, X.-J., MANZANARES, A., BELLAM, K., AND QIN, X. 2009. DARAW: A new write buffer to improve parallel I/O energy-efficiency. In *Proceedings of the 24th Annual ACM Symposium on Applied Computing.* ACM, New York.

RUAN, X.-J., QIN, X., NIJIM, M., ZONG, Z.-L., AND BELLAM, K. 2007. An energy-efficient scheduling algorithm using dynamic voltage scaling for parallel applications on clusters. In *Proceedings of the 16th IEEE International Conference on Computer Communications and Networks.* IEEE, Los Alamitos, CA.

SALEM, K. AND GARCIA-MOLINA, H. 1986. Disk striping. In *Proceedings of the International Conference Data Engineering.* IEEE, Los Alamitos, CA.

SCHEUERMANN, P., WEIKUM, AND G., ZABBACK, P. 1998. Data partitioning and load balancing in parallel disk systems. *VLDB J.* 48–66.

SELTZER, M., CHEN, P., AND J. OUSTERHOUT, J. 1990. Disk scheduling revisited. In *Proceedings of the USENIX Technical Conference.* USENIX, Berkeley, CA.

SHENOY, P. AND VIN, H. 1998. Cello: A disk scheduling framework for next generation operating systems. In *Proceedings of the ACM SigMetrics.* ACM, New York.

SUMNER, T. AND MARLINO, M. 2004. Digital libraries and educational practice: A case for new models. In *Proceedings of the ACM/IEEE Conference Digital Libraries.* ACM, New York, 170–178.

TANAKA, T. 1993. Configurations of the solar wind flow and magnetic field around the planets with no magnetic field: Calculation by a new MHD. *J. of Geophys. Res.* 17251–17262.

WEISSEL, A., BEUTEL, B., AND BELLOSA, F. 2002. Cooperative I/O: A novel I/O semantics for energy-aware applications. In *Proceedings of the Symposium on Operating Systems Design and Implementation.* USENIX, Berkeley, CA.

WIJAYARATNE, R. AND REDDY, A. L. N. 1999. Integrated QoS management for disk I/O. In *Proceedings of the IEEE Conference on Multimedia Computing Systems.* IEEE, Los Alamitos, CA.

XIE, T. AND QIN, X. 2008. An energy-delay tunable task allocation strategy for collaborative applications in networked embedded systems. *IEEE Trans. Comput. 57*, 3, 329–343.

YU, P. S., CHEN, M. S., AND KANDLUR, D. D. 1993. Grouped sweeping scheduling for DASD-based multimedia storage management. *ACM Multimedia Syst. 1*, 3, 99–109.

ZEDLEWSKI, J., SOBTI, S., GARG, N., ZHENG, F., KRISHNAMURTHY, A., WANG, R. 2003. Modelling hard-disk power consumption. In *Proceedings of the USENIX Conference File and Storage Technologies.* USENIX, Berkeley, CA.

ZHENG, F., GARG, N., SOBTI, S., ZHANG, C., JOSEPH, R., KRISHNAMURTHY, A., AND WANG, R. 2003. Considering the energy consumption of mobile storage alternatives. In *Proceedings of the International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems.* IEEE, Los Alamitos, CA.

ZONG, Z.-L., NIJIM, M., AND QIN, X. 2008. Energy-efficient scheduling for parallel applications on mobile clusters. *Cluster Comput. J. Networks, Softw. Tools Appl. 11*, 1, 91–113.

ZONG, Z.-L., QIN, X., NIJIM, M., RUAN, X.-J., BELLAM, K., AND ALGHAMDI, M. 2007. Energy-efficient scheduling for parallel applications running on heterogeneous clusters. In *Proceedings of the 36th International Conference Parallel Processing (ICPP).* IEEE, Los Alamitos, CA.

Received February 2006; revised September 2008, February 2009; accepted March 2009