A Time Series-based Approach for Power Management in Mobile Processors and Disks *

Xiaotao Liu xiaotaol@cs.umass.edu
Prashant Shenoy shenoy@cs.umass.edu
Weibo Gong† gong@ecs.umass.edu

Department of Computer Science
University of Massachusetts Amherst
Amherst, MA 01003

†Department of Electrical and Computer Engineering
University of Massachusetts Amherst
Amherst, MA 01003

ABSTRACT

In this paper, we present a time series-based approach for managing power in mobile processors and disks that see multimedia workloads. Since multimedia applications impose soft real-time constraints, a key goal of our approach is to reduce energy consumption of multimedia applications without degrading performance. We present simple statistical techniques based on time series to dynamically compute the processor and I/O demands of multimedia applications and present techniques to dynamically vary the voltage settings and rotational speeds of mobile processors and disks, respectively. We implement our approaches in the Linux kernel running on a Sony Transmeta laptop and in a trace-driven simulator. Our experiments show that, compared to the traditional system-wide CPU voltage scaling approaches, our technique can achieve up to a 38.6% energy saving while delivering good performance to applications. Simulation results for our disk power management technique show a 20.3% reduction in energy consumption without any significant performance loss when compared to a traditional disk power management scheme.

Categories and Subject Descriptors: D.4.1 [Process Management]: Scheduling; D.4.8 [Performance]: Modelling and prediction

General Terms: Algorithms, Design, Experimentation.

Keywords: Dynamic Voltage Scaling, Power Management, Multimedia, Dynamic Rotations per Minute.

1. INTRODUCTION

Modern mobile devices such as laptops, personal digital assistants (PDAs) and cellular phones tend to have rich multimedia capabilities such as DVD drives, software video and MP3 players, and in-built cameras for image and video capture. Energy is a scarce resource in such battery-powered mobile devices. In contrast, multimedia applications such as audio-video players, multimedia capture and editing programs tend to be resource-hungry. Typically, such multimedia applications consume energy by accessing, processing and rendering large amounts of data. Further, these applications impose soft real-time constraints, and thereby impose lower bounds on the speeds at which multimedia data can be accessed, processed and rendered by these applications.

Recent mobile devices have incorporated a number of power management features to conserve battery power. For instance, modern processors such as Intel's XScale and Pentium-M, AMD's Athlon, Transmeta's Crusoe and IBM's PowerPC-405LP all incorporate dynamic voltage and frequency scaling (DVFS) capabilities. DVFS enables the CPU speed to be varied dynamically based on the workload and reduces energy consumption during periods of low utilization [14]. A number of software and hardware power management techniques have been developed to take advantage of DVFS-capable processors. For instance, *LongRun* is a hardware technique developed by Transmeta to dynamically measure system-wide processor utilization and vary the CPU speed accordingly [4].

Several software approaches for DVFS have also been developed [2, 3, 9, 11, 15]. In [2, 3], the authors propose a work-tracking heuristic that dynamically infers the periodicity of tasks and determines a CPU speed based on this period. A system-wide CPU setting is determined based on the CPU demands of individual tasks. Lorch et. al. [9] divide applications into two categories, interactive and non-interactive. For interactive applications, they assume implicit deadlines (of 50 ms for user input and 25ms for mouse events) and vary the CPU speed based on whether these deadlines are met or missed. A system-wide CPU setting is used in their approach, and non-interactive tasks are handled using a traditional interval-based scheduler. An energy-efficient soft real-time CPU scheduler for mobile multimedia applications was proposed in [15]. The technique integrates DVFS with a real-time CPU scheduling algorithm such as Earliest Deadline First (EDF). The technique makes such scheduling and scaling decisions based on the probability distribution of application cycle demands, and can use different CPU speeds for different tasks. The approach handles only periodic applications (e.g., audio and video players). Further, it assumes that applications convey their periods and the amount of the work in each period to the operating system using system calls. Techniques for varying the rotational speed of disks were studied in [6, 7]—the approach monitors the length of the disk request queue and varies disk speeds so that the response time of requests is within a prede-

*This research was supported in part by NSF grants CCR-9984030, CCR-0098060, CCR-0219520, EIA-0080119 and gifts from IBM, Microsoft and Intel.

fined range. The impact of on-disk caches on performance was not considered in their work.

In this paper, we propose a new approach for power management in mobile devices. Our approach differs from past work in several different respects.

- Our techniques are based on time series and employ simple statistical methods to predict future workloads and to compute power settings.
- Our techniques are designed to handle both processors and disks with power management capabilities. We assume a DVFS-capable processor and a disk that supports multiple rotational speeds and present techniques for varying the speeds of these components based on the workload.
- Unlike techniques that use a single system-wide setting for all tasks [2, 3, 9], our techniques can employ different CPU frequencies for different tasks. The CPU setting is modified at context-switch time based on the needs of each task. Further, unlike approaches such as [15], our approach is completely transparent to applications, and does not require any information (such as the period) from applications. Last, our approach does not make any assumptions about the nature of applications, unlike past work that either assumes periodic or interactive applications. Our time series-based approach can be applied to *any application*, regardless of its nature. Although we present experimental results only for multimedia applications in this paper due to space constraints, we refer the reader to [8] for results from other applications such as batch compilations, emacs, X Server, and interactive shell terminals.

Our *time series-based power management* (TS-PM) approach consists of two components: (i) a time series-based DVFS (TS-DVFS) that uses per-process utilizations to compute the task-specific CPU speed settings, and (ii) a time series-based dynamic rotations per minute (TS-DRPM) technique that dynamically varies disk rotational speeds based on the arrival rate, response times, and access patterns (hit ratios seen at the on-board disk caches) of disk requests.

We have implemented TS-DVFS in the Linux kernel 2.4.20-9, and have evaluated it on a Sony Vaio laptop equipped with Transmeta's Crusoe TM5600-667 processor [12]. Since disks that support variable rotational speeds are not yet commercially available, we resort to trace-driven simulations to evaluate TS-DRPM.

Our results show that:

1. *TS-DVFS* can achieve up to 38.6% energy saving when compared to the hardwired *LongRun* technology, while still delivering good performance to not only multimedia applications but also other kinds of applications.
2. *TS-DRPM* reduces the energy consumption of disks by up to 36% when compared to disks without such features, and by up to 20.3% when compared to the traditional disk power management techniques based on disk spin-down.

The rest of this paper is organized as follows. Section 2 introduces the design and algorithms employed in *TS-PM*. Sections 3 and 4 present the implementation and experimental evaluation of TS-PM, respectively. Finally, Section 5 summaries our key results.

2. TS-PM DESIGN

In this section, we present the architecture of our time series-based power management (TS-PM) approach and the specific algorithms used by our approach.

2.1 Overview

Our time series-based power management techniques assume a processor that supports dynamic voltage and frequency scaling and a hard disk that supports different rotational speeds. While DVFS-capable processors are widely available, disks that support multiple rotational speeds are not yet commercially available—we assume that such disks will be widely used in future mobile devices.

We assume that our TS-PM technique is implemented in the OS kernel and consists of three components: (i) a *profiler* that measures the current CPU and I/O demands for individual tasks as well as for the system as a whole, (ii) a *predictor* that uses a time series of recent CPU and I/O demand to predict future demands using statistical methods, and (iii) a *speed setting* strategy that uses these predictions to compute the desired CPU and disk settings as well as a *speed adapter* that maps these settings to the nearest speed actually supported by the hardware. Observe that the speed settings strategy computes an ideal speed for the processor and disk, while the speed adapter maps it to a speed actually supported by the hardware (since different processors and disks support different power settings, this separation ensures OS portability across hardware). Figure 1 depicts these components. We assume that both TS-DVFS and TS-DRPM employ these components, although the specific algorithms used for profiling, prediction and speed setting will differ for processors and disks.

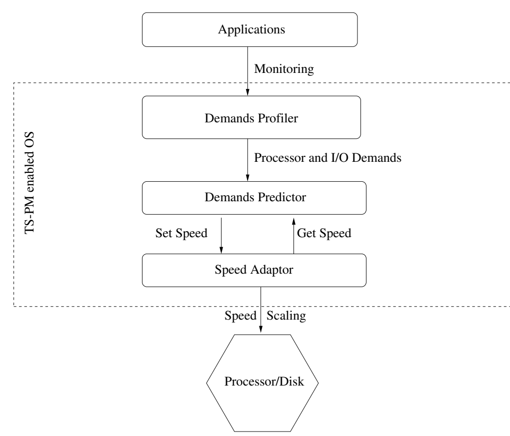

Figure 1: The architecture of a TS-PM-enabled OS kernel

2.2 Profiling Current Demands

This section provides an overview of the profiling techniques employed in TS-PM to measure processor and I/O demand.

2.2.1 Measurement of Processor Demand

Since TS-DVFS supports per-process CPU speed settings, the profiler must estimate the processor demands of individual tasks (the terms tasks and processes are used interchangeably in this paper). Typically, system-wide processor demand is measured using *system utilization*, which is given as

$$U_{sys} = \frac{T_{busy}}{T} \tag{1}$$

there T_{busy} denotes the time for which the CPU is busy during some interval T. This concept can be extended to capture the CPU demands of individual processes. *Process utilization*—the CPU utilization due to an individual process—can be defined as follows. Consider a process that executes for time e within a quantum. The CPU frequency can vary dynamically within this quantum; assume that frequency changes j times within the quantum and that the process runs at CPU frequency f_1 for time t_1 and then at frequency f_2 for time t_2 and so on within the quantum. Then the *full-speed equivalent execution time* e_{fse} for the execution e is given by

$$e_{fse} = \sum_{i=1}^{j} f_i t_i \tag{2}$$

where f_i is the CPU frequency expressed as a percentage of the maximum available frequency. Intuitively, e_{fse} represents the time for which the process would have executed in the quantum if the processor were running at full speed throughout.

To compute the process utilization, assume that the process was scheduled n times during an interval T, and the full-speed execution time of each execution is e_1, e_2, ..., e_n, respectively. Let q_i denote the time quantum that the process gets during its ith execution in that interval. Then, the process utilization u during that interval is given as:

$$u = \frac{\sum_{i=1}^{n} e_i}{\sum_{i=1}^{n} q_i} \tag{3}$$

The quantum q_i is given by formula:

$$q_i = e_i + \frac{e_i}{t_{busy}} t_{idle} \tag{4}$$

where t_{busy} refers to the length of the continuous non-idle time period in which the ith execution sits, and t_{idle} denotes the length of the first continuous idle time after the ith execution.

2.2.2 Measurement of I/O Demand

TS-DRPM measures the I/O demand of applications by measuring the response time of disk requests. Given an interval T, suppose that there are n I/O requests during this interval with response times r_1, r_2, ..., r_n, respectively. Then, the *disk utilization* in this interval is given by:

$$u = \frac{\sum_{i=1}^{n} r_i \cdot s_i}{T} \tag{5}$$

where s_i is a scaling factor based on the current disk speed, $0 < s_i \leq 1$, and $r_i \cdot s_i$ denotes the *full-speed response time*—the response time that would have been observed if the disk were to run at full speed. Due to the presence of an on-board disk cache, not all requests result in disk accesses, and hence, Equation 5 does not correctly reflect the I/O demand of applications. To measure true I/O demand, we should only consider those requests that result in misses in the on-disk cache (and result in actual disk accesses). The profiler labels each I/O request as a *hit* or a *miss* and computes the utilization by only considering misses. Our profiler uses a heuristic for this labelling—only those read requests with response times below a threshold τ are labelled as hits; the remaining reads and all writes are labelled as misses (typically τ is set to less than a millisecond for modern disks).

2.3 Predicting Future Demand

The processor utilizations of individual tasks and the disk utilization are measured by the profiler periodically, yielding a time series of their values. We can then use simple time series-based statistical models to predict future processor and I/O demands. We have experimented with a number of auto-regressive and moving average models such as AR(1), AR(2), AR(3), MA(1), and MA(2) for such predictions [1]. Our analysis using real traces of processor and I/O usage has shown that the first-order auto-regressive model AR(1) yields a good balance between prediction accuracy and computational complexity. We omit these results due to space constraints and assume a predictor based on the AR(1) model in the rest of this paper (see [8] for detailed results).

To understand how the AR(1) predictor works, consider a sequence of observations of the processor or I/O demands: u_0, u_1, u_2, ..., u_n. Given this time series, we wish to predict the demand in the $(n+1)$th interval. Let u_{n+1} denote the actual demand and let $\hat{u}_{n+1}$ denote the predicted demand.

The first-order autoregressive process (AR(1)) is given by [1]:

$$\tilde{u}_t = \phi_1 \tilde{u}_{t-1} + a_t \tag{6}$$

where a_t is some random variable with zero mean and $-1 < \phi_1 < 1$. If u_t has a non-zero mean μ, then $\tilde{u}_t = u_t - \mu$, otherwise, $\tilde{u}_t = u_t$.

Given such a process, an *AR(1) predictor* estimates the mean of u_t and the parameter ϕ_1 of the model and then predicts the next value based on these estimates. Let $\hat{\mu}$ denote the estimated mean and let $\hat{\phi}_1$ denote the estimated value of ϕ_1. The prediction $\hat{u}_{n+1}$ is given by:

$$\hat{u}_{n+1} = \hat{\mu} + \hat{\phi}_1 (u_n - \hat{\mu}) \tag{7}$$

Estimation of the mean $\hat{\mu}$ and the parameter $\hat{\phi}_1$ are important issues in the design of an AR(1) predictor.

Our predictor estimates these two parameters dynamically using recent observations. Consider a window that hold the most recent m observations of utilizations, $m \leq n$. The estimate of the mean $\hat{\mu}$ is given by:

$$\hat{\mu} = \frac{\sum_{i=0}^{m-1} u_{n-i}}{m} \tag{8}$$

The estimate of $\hat{\phi}_1$ is given by:

$$\hat{\phi}_1 = \frac{\sum_{i=0}^{m-1} (u_{n-i} - \hat{\mu})(u_{n-1-i} - \hat{\mu})}{\sum_{i=0}^{m-1} (u_{n-i} - \hat{\mu})(u_{n-i} - \hat{\mu})} \tag{9}$$

Although both TS-DVFS and TS-DRPM use the AR(1) predictor, they use it in different ways. TS-DVFS uses a different AR(1) predictor for each task in the system, while TS-DRPM uses a single predictor to predict aggregate I/O demand.

2.4 Speed Setting Strategy

This section outlines the strategies used to compute the processor and disk power settings.

2.4.1 Processor Speed Setting Strategy

The AR(1) predictor assumes a stationary process. Since application behavior tends to change over time, in reality, the time series of processor and I/O demands is non-stationary. Consequently, our predictor is imperfect and will yield prediction errors. In order to quickly respond to the prediction errors, we implement a two-level CPU speed setting strategy in TS-DVFS. The first level works at the time scale of the prediction interval T and is responsible for computing a baseline CPU frequency for the entire interval T. The second level works at the granularity of *subintervals* within T and adjusts the baseline CPU speed setting whenever prediction errors are detected.

Suppose that the prediction interval is $T = n \times 10$ ms, where n is an integer (we choose $3 \leq n \leq 5$ in our implementation). At the end of each such interval, TS-DVFS computes the baseline CPU frequency for the next interval as

$$f_{base} = \hat{u} \times f_{max} \tag{10}$$

where f_{max} is the maximum CPU frequency, and $\hat{u}$ is the processor utilization prediction for the next interval.

The interval T is further divided into m subintervals, and the length of each such subinterval is $\frac{10n}{m}$ms. Let u_j denote the observed process utilization of the application until the end of the jth subinterval and let f_j denote the frequency setting in this subinterval. Then the CPU frequency setting is adjusted as follows:

$$f_k = \begin{cases} f_{k-1} & \text{if } |u_{k-1} - u_{k-2}| \leq threshold_{u_{k-2}} \\ u_{k-1} \times f_{max} & \text{if } |u_{k-1} - u_{k-2}| > threshold_{u_{k-2}} \\ f_{base} & \text{if } k = 1 \text{ or } u_{k-1} = 0 \end{cases} \quad (11)$$

where $threshold_j$ is a predefined series of thresholds, $u_0 = \hat{u}$ and $f_0 = f_{base}$. Intuitively, the frequency setting is adjusted whenever the observed utilization in a subinterval is a threshold bigger than that in the previous subinterval (indicating a prediction error). It is left unchanged when the two utilizations are within a threshold and reset to the baseline when the utilization drops to zero.

In a real implementation, the computed f_j will be mapped to the closest available frequency which is not less than itself, and $threshold_j$ series are specifically determined for each processor platform.

2.4.2 I/O Speed Setting Strategy

Modern hard drives implement pre-fetching in the hardware to maximize disk cache performance (for instance, track buffering is a form of pre-fetching where an entire disk track is read whenever any sector on that track is requested). Any speed setting strategy should avoid choosing disk speeds that will interfere with such pre-fetching. Aggressive lowering of disk speed can impact prefetching, reduce the cache hit ratio, and severely degrade application performance. This can be especially harmful for soft real-time multimedia applications.

Our I/O speed setting strategy takes these factors into account when computing an appropriate speed setting. Specifically, our technique take into account the arrival rate during last interval, the hit ratio of the most recent n requests, and the performance slowdown at different RPM levels when computing the speed. Suppose that the arrival rate for the last T seconds is a, the hit ratio of the most recent n requests is h, and TS-DRPM predicts disk utilization for the next T seconds as $\hat{u}$. Let $R_{diff}[i]$ denote the difference in the rotational latency between the maximum RPM level and the candidate RPM level i (note that the seek time remains unchanged when changing the disk rotational speed). The performance slowdown $P_{diff}[i]$ is given by the increase in rotational latency seen by these $a \times T$ requests:

$$P_{diff}[i] = a(1-h) \times T \times R_{diff}[i] \quad (12)$$

With this, we can predict the disk utilizations under different RPM levels for the next T seconds by:

$$\hat{u}_i = \hat{u} + \frac{P_{diff}[i]}{T} \quad (13)$$

where $\hat{u}_i$ is the predicted utilization if running on RPM level i for the next T seconds.

Let $\hat{u}_{max}$ denote the utilization at the maximum RPM level. We choose the lowest RPM level that satisfies the following property:

$$\frac{\hat{u}_i - \hat{u}_{max}}{\hat{u}_{max}} \leq threshold \quad (14)$$

where $threshold$ is a predefined threshold.

3. IMPLEMENTATION AND SIMULATION

We have implemented TS-DVFS in the Linux kernel on a Sony Vaio PCG-V1CPK laptop with Transmeta Crusoe TM5600-667 processor [12]. Since DRPM-enabled disks are not yet commercially available, we implement TS-DRPM in a simulated DRPM-ready hard disk using DiskSim [5]. Next we present the details of our implementation.

3.1 Implementation of TS-DVFS

The Transmeta TM5600 processor supports five discrete frequency and voltage levels (see Table 1) and implements the *LongRun* [4] technology in hardware to dynamically vary the CPU frequency based on the observed system-wide CPU utilization. LongRun varies the CPU frequency between a user-specified maximum and minimum values—these values can be set by users by writing to two machine special registers (MSR). By default, these values are set to 300 MHZ and 677 MHz, enabling the full range of voltage scaling. LongRun can be disabled by setting the minimum and maximum register values to the same frequency (e.g., setting both to 533 MHz does not allow any leeway in changing the CPU frequency, effectively disabling LongRun). This feature can be used to implement voltage scaling in *software*—the OS can periodically determine the desired frequency and set the two registers to this value.

Freq. (MHz)	Voltage (V)	Power (W)
300	1.2	1.3
400	1.225	1.9
533	1.35	3.0
600	1.5	4.2
667	1.6	5.3

Table 1: Characteristics of the TM5600-667 processor

Our prototype of TS-DVFS is implemented as a set of modules and patches in the Linux kernel 2.4.20-9. Our implementation uses a scaling interval T of 40ms, sub-intervals of 10ms, and a window size 4 for the AR(1) predictor. Our prototyping effort involved the following issues: (i) implementation of the CPU demand profiler and predictor, (ii) modifications to the kernel CPU scheduler to support per-process DVFS settings (which are taken into account at context switch time), (iii) implementation of the CPU speed adaptor for the Transmeta processor, and (iv) determination of the $threshold_j$ values from off-line empirical experiments. The latter experiments yield a hardware-specific conversion table (see Table 2) for mapping process utilizations to a corresponding CPU frequency.

Process Utilization	Freq. (MHz)
$[0\%, 45\%)$	300
$[45\%, 60\%)$	400
$[60\%, 80\%]$	533
$(80\%, 90\%]$	600
$(90\%, 100\%]$	667

Table 2: Mapping Process Utilizations to a CPU Frequency in the Transmeta TM5600.

3.2 Simulation of TS-DRPM

To simulate TS-DRPM, we consider an IBM TravelStar 40GNX [13] laptop hard disk as the baseline and enhance it with DRPM features. We enhance the disk with five different RPM levels from 3000 to 5400 RPM with a step size of 600 RPM. The assumed power characteristics for these RPM levels are shown in 3. We implement TS-DRPM for this disk in the DiskSim simulator [5]. We assume a scaling interval T of 10s, a threshold value of 0.15, a history of the 100 most recent requests, and a window size of 2 for the AR(1) predictor. We augment DiskSim with power models to record the energy consumption of the disk at various RPM levels. Our implementation also accounts for the queuing and service delays caused by the changes in the RPM level of the disk and in the STANDY/ACTIVE/IDLE modes.

RPM	Idle Power	Seek Power	Read/Write Power
3000	0.8W	1.4W	1.3W
3600	1.1W	1.7W	1.6W
4200	1.4W	2.0W	1.9W
4800	1.7W	2.3W	2.2W
5400	2.0W	2.6W	2.5W

Table 3: Characteristics of the Simulated DRPM-Ready Hard Disk

4. EXPERIMENTAL EVALUATION

We evaluated TS-DVFS with a variety of applications and TS-DRPM with a variety of real application traces. This section presents a summary of our key results (see [8] for detailed results).

4.1 TS-DVFS Results

To evaluate TS-DVFS, we ran applications under three different configurations: (i) with DVFS disabled—the CPU always runs at the maximum available speed (denoted as FULL); (ii) using the hardwired LongRun technology; (iii) using TS-DVFS and with the LongRun technology disabled. The energy consumption of the processor during an interval T is computed as

$$energy = \sum_{i=1}^{n} p_i t_i \tag{15}$$

where n is the number of available frequency/voltage combinations on the processor, p_i denotes the power consumption of the processor when running at the ith frequency/voltage combination, and t_i represents the time spent at the ith frequency/voltage combination during the interval T. We modify the Linux kernel to record the energy consumption of the TM5600 processor using Equation 15 and Table 1.

4.1.1 Multimedia Applications

We encoded several DVD movies at different bit-rates and resolutions using Divx MPEG4 video codec and MP3 audio codec. The characteristics of two such movies are listed in Table 4. The bit-rates are depicted in the form $(a+b)$Kbps, where a is the video and b is the audio bit-rate. We recorded the energy consumed in the playback of these movies at full speed, with LongRun and with TS-DVFS.

	Res.	Length	Bit-Rate(Kbps)
Movie 1	640x272	6720s	1290.9 + 179.20
Movie 2	640x352	7168s	679.7 + 128.00

Table 4: Characteristics of MPEG 4 Videos

As shown in Figure 2, all three configuration—FULL, LongRun, and TS-DVFS—handle these movies very well. The same playback quality is observed under all three configurations: identical execution times, same frame rate, no dropped frames and no user-noticeable delays. The results also show that although LongRun already achieves significant energy savings (from 53.77% to 57.17%) compared to FULL, TS-PM can achieve additional 20.51% to 24.64% energy savings when compared to LongRun.

	Orig. Charac.		Charac. after Resc./Trans.	
Len.	Res.	Bit-Rate(Kbps)	Res.	Bit-Rate(Kbps)
610s	512x288	838.40+128.00	480x270	802.25+128.00
2270s	480x360	861.90+128.00	480x360	803.40+128.00

Table 5: Characteristics of MPEG Videos for Rescaling/Transcoding

In addition to movie playback, we studied the energy efficiency of TS-PM for transcoding. We used *mencoder*, an encoder tool in MPlayer suite [10], to perform two tasks: (i) to rescale the resolution of an MPEG4 movie, and (ii) to transcode a MPEG1 movie to MPEG4. The characteristics of these workloads are shown in Table 5. All these movies use MP3 codec as their audio codec.

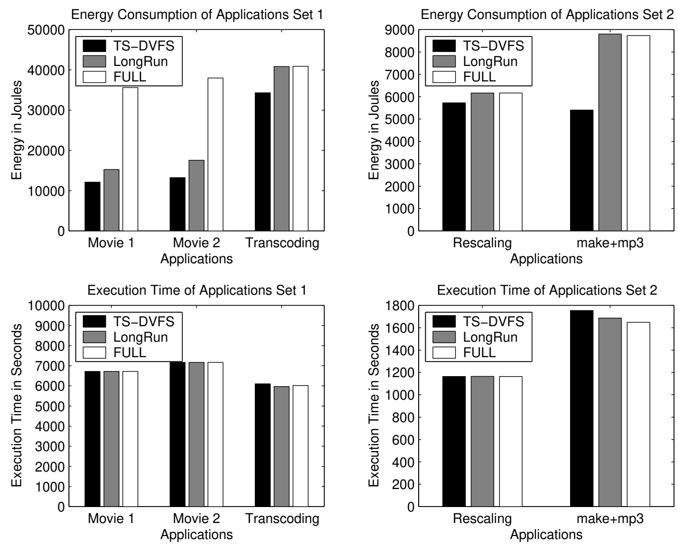

Figure 2: CPU Energy Consumption and Execution Times for Different Workloads

Our results in Figure 2 (labelled as Transcoding and Rescaling) show that LongRun is unable to extract any energy savings for these workloads. However, TS-DVFS can still achieve up to 7.08% and 14.41% energy savings for rescaling and transcoding, respectively, when compared to LongRun and Full. The data in Figure 2 also shows that TS-DVFS can extract these savings without any significant performance degradation—the observed loss in performance is only 2.73% for the transcoding workload.

4.1.2 Other Applications

We also evaluated the efficiency of TS-DVFS for a variety of other application workloads such as editors, X server, shell terminals, and build jobs [8]. We consider one such workload, namely, building the Linux kernel with background MP3 audio playback, labelled as “make+mp3” in Figure 2.

Our results in Figure 2 show that LongRun is unable to extract any energy savings when compared to FULL. In fact, Longrun incurs a 2.31% slowdown and consumes an extra 69.27 Joules when compared to FULL (the increase in energy consumption is due to the longer completion time for the build job). In contrast, TS-DVFS is able to extract 38.62% energy savings at the expense of 4.03% longer execution time when compared to LongRun.

4.2 TS-DRPM Results

To evaluate TS-DRPM, we consider four disk configurations: (i) FULL, where the disk is assumed to run at full speed with no power optimizations, (ii) TPM_{perf}, the traditional power management based on disk spin-down, where we assume a perfect arrival time predictor and transition the disk to sleep mode if the time to next request is long enough to accommodate the spin-down followed by a spin-up, (iii) SIMPLE-DRPM—the technique proposed in [7, 6]—which uses the variance in mean response times of disk requests to estimate the I/O demand, and (iv) TS-DRPM, our time series-based technique.

We instrument the Linux kernel to gather traces of disk requests from a variety of application mixes. We present results from one such workload consisting of a mix of movie playback, MP3 playback, and movie transcoding. We conduct trace driven simulations for the four configurations and determine the energy consumption of the disk and the response time CDF. Figure 3 depicts our results and indicates that TS-DRPM, TPM_{perf}, and SIMPLE-DRPM can all achieve significant energy savings when compared to FULL. TS-DRPM yields the best savings and reduces the energy consumption of the multimedia workload by 9.46%, 20.35%, and 36.01% when compared to SIMPLE-DRPM, TPM_{perf}, and FULL, respectively.

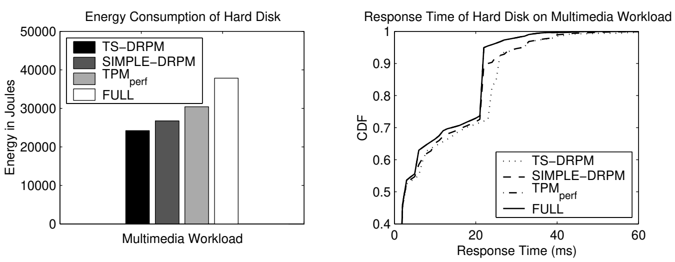

Figure 3: Energy Consumption and Response Times of Disk Requests for a Multimedia Workload

The response time CDFs indicate the performance seen by requests under the four configurations. Since the TPM_{perf} is equipped with a perfect predictor, it does not incur any performance penalty, and its CDF curve is identical to FULL. The figure shows that although TS-DRPM incurs a performance slowdown, the degradation is small when compared to FULL.

5. SUMMARY AND CONCLUSIONS

This paper proposes a new approach for power management in mobile devices. Our TS-PM approach is based on time series and employs simple statistical methods to predict future workloads and to compute power settings. TS-PM consists of two components: (i) a time series-based DVFS (TS-DVFS) that uses per-process utilizations to compute the task-specific CPU settings, and (ii) a time series-based dynamic rotations per minute (TS-DRPM) technique that dynamically varies disk rotational speeds based on the arrival rate, response times, and access patterns (hit ratios seen at the onboard disk caches) of disk requests.

We have evaluated the energy efficiency of TS-PM through implementation and simulations. Our results show that, when compared to the LongRun technology, TS-PM reduces energy consumption by 24.64%, 7.08%, 14.41%, 38.62% for movie playback, movie rescaling, transcoding and Linux builds, respectively, without any significant performance loss. The results from our simulations of TS-DRPM show that TS-PM can achieve up to 36.01% energy savings when compared to disks without any power saving features. TS-PM yields up to 20.35% savings when compared to traditional power management based on disk spin-down.

6. REFERENCES

[1] G. P. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis Forecasting and Control Third Edition*. Prentice Hall, 1994.

[2] K. Flautner and T. Mudge. Vertigo: Automatic performance-setting for linux. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI'02), Boston, MA*, pages 105–116, December 2002.

[3] K. Flautner, S. Reinhardt, and T. Mudge. Automatic performance-setting for dynamic voltage scaling. In *Proceedings of the 7th ACM International Conference on Mobile Computing and Networking (MobiCom'01), Rome, Italy*, pages 260–271, July 2001.

[4] M. Fleischmann. Longrun power management - dynamic power management for crusoe processors. Technical report, Transmeta Corporation, 2001.

[5] G. R. Ganger, B. L. Worthington, and Y. N. Patt. The disksim simulation environment - version 2.0 reference manual.

[6] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. Drpm: Dynamic speed control for power management in server class disks. In *Proceedings of the 30th IEEE Annual International Symposium on Computer Architecture (ISCA'03), San Diego, CA*, June 2003.

[7] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. Reducing disk power consumption in servers. *IEEE Computer: Special Issue on Power-aware and Temperature-aware Computing*, 36(12):59–66, December 2003.

[8] X. Liu, P. Shenoy, and W. Gong. A time series-based approach for power management in mobile processors and disks. Technical report 04-25, University of Massachusetts Amherst, 2004.

[9] J. R. Lorch and A. J. Smith. Operating system modifications for task-based speed and voltage scheduling. In *Proceedings of the 1st ACM/USENIX International Conference on Mobile Systems, Applications, and Services (MobiSys'03), San Francisco, CA*, pages 215–229, May 2003.

[10] Mplayer 0.90. http://www.mplayerhq.hu.

[11] T. Pering, T. Burd, and R. W. Broderson. Voltage scheduling on the lparm microprocessor system. In *Proceedings of the 2000 IEEE International Symposium on Low Power Electronics and Design (ISLPED'00), Rapallo, Italy*, July 2000.

[12] Crosoe tm5600 processor data sheet. Transmeta Inc., http://www.transmeta.com.

[13] Ibm hard disk - travelstart 40gnx. IBM, http://www.ibm.com.

[14] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced cpu energy. In *Proceedings of the 1st USENIX Symposium on Operating Systems Design and Implementation (OSDI'94), Monterey, CA*, pages 13–23, November 1994.

[15] W. Yuan and K. Nahrstedt. Energy-efficient soft real-time cpu scheduling for mobile multimedia systems. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03), Bolton Landing, NY*, pages 149–163, October 2003.