System-Level Power Management Using Online Learning

Gaurav Dhiman and Tajana Šimunić Rosing, *Member, IEEE*

***Abstract*—In this paper, we propose a novel online-learning algorithm for system-level power management. We formulate both dynamic power management (DPM) and dynamic voltage-frequency scaling problems as one of workload characterization and selection and solve them using our algorithm. The selection is done among a set of experts, which refers to a set of DPM policies and voltage-frequency settings, leveraging the fact that different experts outperform each other under different workloads and device leakage characteristics. The online-learning algorithm adapts to changes in the characteristics and guarantees fast convergence to the best-performing expert. In our evaluation, we perform experiments on a hard disk drive (HDD) and Intel PXA27x core (CPU) with real-life workloads. Our results show that our algorithm adapts really well and achieves an overall performance comparable to the best-performing expert at any point in time, with energy savings as high as 61% and 49% for HDD and CPU, respectively. Moreover, it is extremely lightweight and has negligible overhead.**

***Index Terms*—Dynamic voltage frequency scaling, energy-performance trade-off, online learning, power management.**

I. Introduction

POWER consumption is a key issue in the design of computing systems today. While battery-driven systems need to meet an ever-increasing demand for performance with a longer battery life, high-performance systems contend with issues of heating. Dynamic power management (DPM) and dynamic voltage-frequency scaling (DVFS) are the two most popular techniques for dynamically reducing system power dissipation. DPM achieves this by selective shutdown of system components that are idle, while the key idea behind DVFS techniques is to dynamically scale the supply voltage/frequency level of the device. Reduction in voltage/frequency level is beneficial, since it reduces the overall power consumption [1]. DPM can be employed for any system component with multiple power states, while DVFS is useful only for components that support multiple speed and voltage levels (like CPU). A number of modern processors such as Intel XScale [2], AMD Opteron [3], etc., are equipped with DVFS capability. In existing literature, however, the design of DPM and DVFS policies for general-purpose systems has been treated as separate problems. In this paper, we target both the problems with the objective of achieving system-wide energy efficiency.

A number of heuristic and stochastic policies have been proposed in the past with their design varying in terms of how they take the decision to perform shutdown for DPM. While simpler DPM policies like timeout and predictive policies do it heuristically with no performance guarantees, more sophisticated stochastic policies guarantee optimality for stationary workloads. There is no single policy solution that guarantees optimality under varying workload conditions. We propose a novel setup for DPM, where we maintain a set of DPM policies (suited for different workloads) and design a control algorithm that selects the best suited one for the current workload. In such a setup, the DPM problem reduces to one of accurate *characterization* and *selection*, where the best-suited policy is selected based on the characterization of the current workload.

For devices like CPU that support both DPM and DVFS, it is essential to understand the interplay between the two, since the energy savings based on DVFS come at the cost of increased execution time, which implies greater leakage energy consumption and shortened idle periods for applying DPM. This impact, as we show later on, depends on the nature of the executing workload in terms of its CPU and memory intensiveness and the leakage power characteristics. Therefore, the problem of performing DPM-aware DVFS can also be viewed as one of accurate *characterization* and *selection*, where the best-suited voltage-frequency setting (hereon referred to as v-f setting) is selected based on the characterization of CPU leakage and the executing workload.

Instead of proposing a new policy for DPM and DVFS, we apply online learning [4] to select among a set of possible policies and v-f settings. The online-learning algorithm (referred to as *controller*) has a set of *experts* (DPM policies/v-f settings) to choose from and selects an expert that has the best chance to perform well based on the controller's characterization of the current workload. The selection takes into account energy savings, performance delay, and the user-specified energy-performance tradeoff (referred to as e/p tradeoff). The algorithm is guaranteed to converge to the best-performing expert in the set, thus delivering performance at least as good as the best expert in the set, across different workloads.

We implement the controller for a server and laptop hard disk drive (HDD), and Intel PXA27x CPU. The controller chooses among a set of policies representing state of the art in DPM, and v-f settings available on the Intel PXA27x CPU. In our experiments, the controller achieved as high as 61% and 49% reduction for HDD and CPU, respectively, at negligible overhead.

Manuscript received May 21, 2008; revised November 4, 2008. Current version published April 22, 2009. This work was supported in part by HPWREN under NSF Grants 0087344 and 0426879 (http://hpwren.ucsd.edu), by the Center of Networked Systems (http://cns.ucsd.edu), by Cisco Systems, and by Sun Microsystems. This paper was recommended by Associate Editor P. Eles.

The authors are with the Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA 92093-0407 USA (e-mail: gdhiman@cs.ucsd.edu; tajana@ucsd.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2009.2015740

The rest of this paper is organized as follows. In Section II, we discuss and compare the previous work for DPM and DVFS. In Section III, we formulate both the DPM and DVFS problems as one of workload characterization and selection and then describe the design of our online-learning algorithm which is used to solve them. In Section IV, we explain the implementation level details on how workload characterization and expert selection is actually performed for DPM and DVFS. We then describe our experimental setup and results in Section V before concluding in Section VI.

II. Previous Work

Since system-level power management is a critical design issue, a lot of research effort has been devoted to explore different approaches for performing DPM and DVFS in the past. In this section, we discuss the previous work and highlight the contributions of our work to the existing state of the art.

A. DPM

The existing DPM policies can be broadly classified into timeout, predictive, and stochastic policies. In a timeout policy, the device is put to sleep if it is idle for more than a specified timeout period [5], [6]. For instance, in [5], the device is put to sleep if it is idle for more than $T_{\rm be}$ (break-even time). $T_{\rm be}$ of a low power state is the minimum length of the idle period, which compensates for the cost associated with entering it. In contrast, predictive policies [7]–[9] predict the duration of upcoming idle period and make the shutdown decision as soon as the device goes idle. Such heuristic policies tend to be easy to implement but do not offer any guarantee on energy and performance delay.

Stochastic policies model the request arrival and device power state changes as stochastic processes. Minimizing power consumption and performance delay then becomes a stochastic optimization problem. For instance, in [10], Paleologo *et al.* assume the arrival of requests as a geometric distribution and model power management as a discrete-time Markov decision process. This model is subsequently improved in [11] and [12] using more complex MDP models to characterize real-life workloads. However, stochastic policies offer optimality only for stationary workloads, thus having limited adaptability. The work in [13] and [14] extends the stochastic model to handle nonstationary workloads by switching between a set of precalculated optimal stochastic policies. However, these approaches compromise optimality by switching heuristically and are also quite complex.

There are a few existing approaches which apply machine learning to power management. In [15], supervised learning techniques like nearest neighbor, decision trees, etc., are used to perform DPM, which rely on offline training. This makes them unsuitable for dynamic workload and systems that we are targeting. In [16], a reinforcement-learning-based policy is used for performing midlevel power management in wireless network cards. Unlike online learning, reinforcement learning is more suited for probabilistic environments, where it is not possible to evaluate performance of all the experts. As a result, the bounds given by it are also probabilistic in nature, compared to deterministic bounds we have provided with online learning. Furthermore, our system model is different from this work, since instead of designing a new policy, we rather select among a set of existing policies.

B. DVFS

Previous DVFS-related work may be broadly divided into three categories. The first category of techniques targets systems where the task arrival times, workload, and deadlines are known in advance [17], [18]. DVFS is performed at task level in order to reduce energy consumption while meeting hard timing constraints. The second category of techniques requires either application or compiler support for performing DVFS [19]–[21]. The third category comprises system-level DVFS techniques that target general-purpose systems that have no hard-task deadlines, and expect no support from application/compiler level. Our work for DVFS belongs to this category. The works done in [22]–[24] monitor the system workload in terms of CPU utilization at regular intervals and perform DVFS based on their estimate of CPU utilization for the next interval. These approaches, however, do not take the characteristics of the running tasks into account, which, as we show in Section IV-B, determine the potential benefits of performing DVFS. In contrast, the works done in [25]–[27] characterize the running tasks at runtime and accordingly make the voltage scaling decisions. They use dynamic runtime statistics such as cache hit/miss ratio, memory access counts, etc., obtained from the hardware performance counters to perform task characterization. However, the policy in [25] is not flexible since it operates for a static performance loss (10%), while those in [26] and [27] present results only in a single-task environment. Moreover, none of these techniques give strategies on how to adapt DVFS policies to different leakage characteristics. The prior algorithms that take leakage into account [28], [29] are targeted toward real-time systems and assume precise knowledge of task deadlines and characteristics in advance.

To summarize, the primary contributions of our work are as follows: 1) a controller based on an online-learning algorithm which guarantees convergence to the best-suited DPM policy and v-f setting (DVFS) under both single-task as well as multi-task scenarios; 2) a scalable strategy for adapting DPM/DVFS decisions across devices with different leakage characteristics; and 3) a lightweight implementation of the proposed algorithm that has negligible runtime overhead.

III. Design

In this section, we first describe how both DPM and DVFS can be formulated as a problem of accurate workload characterization and selection. The selection is done among a set of DPM policies (e.g., fixed timeout, TISMDP, etc.) or allowable v-f settings available on the processor or both depending upon the problem we are targeting. Without the loss of generality, we refer to these policies/v-f settings as *experts*. We then elaborate on our algorithm, which we employ to perform this control activity of workload characterization and expert selection.

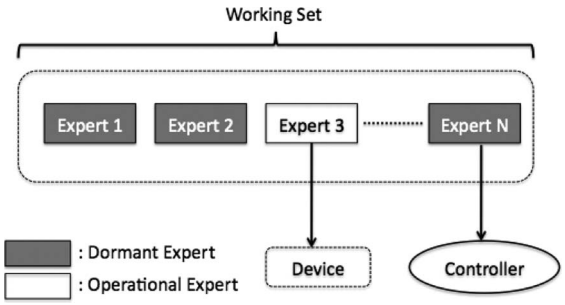

Fig. 1. System model.

TABLE I
ALGORITHM CONTROLLER

Parameters: $\beta \in [0, 1]$
Initialization:
-Initial weight vector $\mathbf{w}^1 \in [0, 1]^N$, such that $\sum_{i=1}^{N} w_i^1 = 1$
-DPM/DVFS specific initialization
For operative periods $t = 1, 2 \ldots$
1: Choose expert with highest probability factor in r^t, where $r^t = \frac{w^t}{\sum_{i=1}^{N} w_i^t}$
2: Operative period starts → operational expert takes control of the device
3: Operative period ends → DPM/DVFS specific evaluation of experts
4: For each expert i, set new weight factor to be: $w_i^{t+1} = w_i^t \cdot \beta^{l_i^t}$

A. System Model

DPM problem is fundamentally a decision problem in which the policy has to decide whether to perform a shutdown for a given idle period. As described in Section II-A, different DPM policies use different mechanisms to decide this. However, as we show in Section V, DPM policies outperform each other under different workloads, devices, and user e/p tradeoff preferences. This observation motivates the use of multiple DPM policy experts, where the best-suited expert is selected as a function of the current workload characteristics and e/p tradeoff.

DVFS problem is intuitively a problem of selection among the given v-f setting experts of the device (e.g., CPU). A lower v-f setting can prolong the execution time of a task, hence shortening the upcoming idle period durations, which has a direct impact on the energy savings possible due to DPM. Longer execution times also cause extra leakage energy consumption, which can offset power savings due to DVFS (as explained in detail in Section IV-B). An ideal DVFS policy must understand this impact and hence perform v-f setting selection with the objective of reducing the overall energy consumption. The primary elements, which govern the choice of v-f settings, are the characteristics of the executing workload in terms of its CPU/memory intensiveness and the device leakage power characteristics. This observation transforms the DVFS problem into one of accurate modeling of the characteristics of the executing task and device leakage, which, in turn, drives the expert selection.

To summarize, both DPM and DVFS problems can be solved by selection of appropriate expert based on the current workload characterization. For DPM, the characterization is in terms of the distribution of idle period durations, while for DVFS, it is in terms of the CPU/memory intensiveness of the executing task. With this background, we now present our system model, which consists of three primary entities as shown in Fig. 1: 1) *controller*: the core algorithm we employ to perform the workload characterization and expert selection activity for both DPM and DVFS; 2) *working set*: it is the set of experts that the controller selects from. An expert can be any DPM policy (for DPM) or any allowable v-f setting (for DVFS); and 3) *device*: device is the entity whose power is being managed by the controller.

We invoke the controller on an *event*, which we refer to as the *controller event*. For DPM, this event is the idle period, i.e., we run the controller whenever an idle period occurs. For DVFS, it is the scheduler tick of the operating system. Scheduler tick is a periodic timer interrupt, which is used by the OS scheduler to make scheduling decisions. Hence, it is the finest granularity at which system updates are performed.

As shown in Fig. 1, the experts can be in one of the two possible states: *dormant* or *operational*. By default, all the experts are in the dormant state and are referred to as the *dormant experts*. When a controller event occurs, the controller on the basis of its model of the current workload selects an expert that has the highest probability to perform well. This selected expert is referred to as the *operational expert*, which can either be a DPM policy or a v-f setting depending upon the event (idle period or scheduler tick). The amount of time for which the expert stays in the operational state is referred to as the *operative period*, after which it returns to its default dormant state. The operative period for an operational expert in case of DPM is the length of the idle period, while in case of DVFS, it is the length of the scheduler quantum. In Fig. 1, this implies that Expert 3 has been selected by the controller as the operational expert for the current operative period.

It may be noted that for DPM, the controller has a form of a meta-policy, where it performs selection among multiple DPM policies, while for DVFS, it is a policy that selects among multiple v-f settings. There are two primary reasons we did not implement a meta-policy for DVFS-like DPM. First, different DVFS policies operate under different setups, which might be difficult to put together on a given CPU. For instance, the policy in [26] operates on every scheduler tick, while, the one in [27] operates at 100 million instruction intervals, which might or might not coincide with a scheduler tick. As a result, there is no common controller event, where it can evaluate these two policies and select an operational policy. Second, different DVFS policies also make assumptions about the systems they operate on, with help from applications, compilers, etc. [19], [30], which might not hold on other systems.

B. Controller

We adapt the online allocation algorithm of Freund and Schapire [4] to the problem of DPM/DVFS. A big advantage of the algorithm is that it provides a theoretical guarantee on convergence to the best-suited expert in the working set. We present an analysis of the theoretical bound in Section III-C.

Table I contains the pseudocode for the algorithm. The controller has N experts to choose from; we number these $i = 1, 2, \ldots, N$. The experts can be any DPM policy (for DPM)

or any valid v-f setting (for DVFS). The algorithm associates and maintains a weight vector $\mathbf{w}^t = \langle w_1^t, w_2^t, \ldots, w_N^t \rangle$, where w_i^t is a weight factor corresponding to expert i for operative period t. The value of weight factor, at any point in time, reflects the performance of the expert, with a higher value indicating a better performance. All of the weights of the initial vector $\mathbf{w}^1$ sum to one, as shown in Table I. In our implementation, we assign equal weights to all the experts at initialization.

To perform expert selection, the controller maintains a probability vector $\mathbf{r}^t = \langle r_1^t, r_2^t, \ldots, r_N^t \rangle$, where $0 \leq r_i^t \leq 1$, that consists of probability factors associated with each expert for operative period t. It is obtained by normalizing the weight vector as shown in the following equation:

$$\mathbf{r}^t = \frac{\mathbf{w}^t}{\sum_{i=1}^{N} w_i^t}. \tag{1}$$

At any point in time, the best-performing expert, in accordance with the current workload, has the highest probability factor among all the experts. Thus, the controller simply selects the expert with the highest probability factor as the operational expert for the upcoming operative period. If the probability factor of multiple experts is equal, then it randomly selects one of them with a uniform probability (step 1 in Table I).

When the operative period starts, the operational expert takes control of the device (step 2 in Table I). For DPM, the operational DPM expert takes the shutdown decision. For DVFS, the v-f setting corresponding to the operational DVFS expert is applied to the CPU. When the operative period ends, the controller does an evaluation of all the experts in the working set (step 3 in Table I). The key objective of performing evaluation is to figure out how suitable each expert was for the just concluded operative period. The details on how the controller actually estimates this suitability are DPM/DVFS specific and are provided in the next section. The end result of this evaluation is a loss factor (l_i^t) corresponding to each expert i, which indicates how unsuitable it was for the previous operative period. A higher value indicates higher unsuitability and vice versa.

The final step in the algorithm involves updating the weight factors for each expert on the basis of the loss they have incurred

$$w_i^{t+1} = w_i^t \cdot \beta^{l_i^t}. \tag{2}$$

Thus, the weight factors corresponding to experts with higher loss are penalized more by this simple multiplicative rule. The value of constant β can be set between zero and one. The criterion for selecting its appropriate value is explained in [4]. This rule gives higher probability of selecting the better performing experts in the next operative period. Once the weights are updated, we are again ready to select the operational expert for next operative period by calculating the new probability vector $\mathbf{r}^t$ using step 1 in Table I.

C. Performance Bound of Controller

From the previous discussions, we know that l_i^t is the loss incurred by each expert for the operative period t. Hence, the average loss incurred by our scheme for a given operative period t in a system with N experts is

$$\sum_{i=1}^{N} r_i^t l_i^t = \mathbf{r}^t \cdot \mathbf{l}^t. \tag{3}$$

The goal of the algorithm is to minimize its cumulative loss relative to the loss incurred by the best expert. That is, the controller attempts to minimize the net loss

$$L_G - min_i L_i$$

where $L_G = \sum_{t=1}^{T} \mathbf{r}^t \cdot \mathbf{l}^t$ is the total loss incurred by controller and $L_i = \sum_{t=1}^{T} l_i^t$ is individual expert i's cumulative loss over T operative periods. It can be shown [4] that net loss of the algorithm is bounded by $O(\sqrt{T \ln N})$ or that the average net loss per period decreases at the rate $O(\sqrt{\ln N/T})$. Thus, as T increases, the difference decreases to zero. This guarantees that the performance of the controller is close to that of the best-performing expert for any workload.

IV. Implementation Details

The previous section gave an overview of how DPM and DVFS can be modeled as workload characterization and expert selection problems that can be solved using the controller algorithm. In this section, we provide implementation details required for accomplishing this solution. We break this section into two parts, with the first part discussing details pertaining to devices that support only DPM and the second part considering devices that support both DPM and DVFS (specifically CPU). Finally, we show how controller can explicitly be adapted to different leakage regimes.

A. Devices With Only DPM

In this section, we discuss controller implementation with respect to devices that support only DPM such as HDD, memory, network card, etc.

Expert Selection: As described before, the controller maintains a weight vector $\mathbf{w}^t$ (Section III-B) and its normalized version probability vector $\mathbf{r}^t$ (1) for the experts. At any point in time, the best-performing expert has the highest probability factor among all the experts, and hence, for performing selection, the controller simply selects the expert with the highest probability factor as the operational expert for the next idle period.

The controller evaluates the performance of all the experts at the *end* of the idle period. The evaluation takes into account energy savings, performance delay, and user-specified e/p tradeoff for this update. The energy savings and performance delay caused by the operational expert can be easily calculated since the length of that idle period is known. The dormant experts are evaluated on the basis of how they would have performed had they been the operational experts. We evaluate *loss* with respect to an ideal oracle policy that has zero delay and maximum possible energy savings. Since this loss evaluation takes place at the end of idle period, when we already know its length, we can easily estimate the performance of the ideal oracle policy as well. The value of loss factor (l_i^t) for each expert is influenced

by the relative importance of energy savings and performance delay as expressed by factor α ($0 \leq \alpha \leq 1$), which is specified by the user. For this purpose, we break it down into two components: l_{ie}^t and l_{ip}^t, which correspond to energy and performance loss, respectively, for expert i. The loss factor is then given by

$$l_i^t = \alpha \cdot l_{ie}^t + (1-\alpha) \cdot l_{ip}^t. \tag{4}$$

In our implementation, we determine the energy loss l_{ie}^t by comparing the length of the idle period with the sleep time. If it is less than $T_{\rm be}$ (break-even time, defined in Section II), then we do not save energy and thus $l_{ie}^t = 1$. For the values of sleep time $T_{{\rm sleep}_i}$ of an expert i greater than $T_{\rm be}$ and of idle period $T_{\rm idle}$, we use the following equation:

$$l_{ie}^t = 1 - T_{{\rm sleep}_i}/T_{\rm idle}. \tag{5}$$

Calculation of performance loss l_{ip}^t is based on whether the device sleeps or not. If the expert makes the device sleep, $l_{ip}^t = 1$ since we incur performance delay upon wakeup, otherwise it is set to zero.

Once the loss factor is available for each expert, the corresponding weight factor is updated using (2) [we use $\beta = 0.75$ in (2)]. Once all the weight factors have been updated, the probability factors for all the experts are updated using (1). Then, the operative expert for the next idle period is simply the expert with the highest current probability factor. At any given point in time, the current value of the weight/probability factors is a result of the successive application of (2) in the previous idle periods. This means that the weight/probability vector actually characterizes the workload or, more precisely, its idle period distribution in the form of suitability of the different DPM experts for it. By regularly updating it at the end of every idle period, this suitability is updated to keep up with changes in the workload characteristics.

B. Devices With DPM and DVFS (CPU)

For devices that support both DPM and DVFS (e.g., CPU), the controller must determine how to perform a tradeoff between the two. This is important since DVFS impacts DPM by potentially reducing the idle period durations. We will show in this section that the impact is dominated by two factors, CPU leakage characteristics and CPU/memory intensiveness of the executing task. To understand this, we first present a simple energy model based on the analysis done in [29]. Then, by using it, we show how task and CPU leakage characteristics affect energy savings due to DVFS and DPM.

Energy Model: Consider the following equation, which approximates the power consumption in a CMOS circuit:

$$P = D + L, \qquad D = C_L V^2 f; \quad L = I_L V \tag{6}$$

where C_L is the load capacitance, V is the supply voltage, I_L is the leakage current, and f is the operational frequency. The first term corresponds to the dynamic power-consumption component of the total power consumption (referred to as D), while the second term corresponds to the leakage power consumption (referred to as L).

Let the maximum frequency of the device be $f_{\max}$, and let the corresponding voltage be $V_{\max}$. The device would consume the maximum power, i.e., $P_{\max}$ at this v-f setting. We define P_n, D_n, and L_n as the normalized power-consumption values, i.e.,

$$P_n = \frac{P}{P_{\max}} = D_n + L_n = \frac{D}{P_{\max}} + \frac{L}{P_{\max}}. \tag{7}$$

We next define λ and ρ as the % contribution of dynamic and leakage power to the total power consumption at the highest v-f setting

$$\lambda = \frac{D_{\max}}{P_{\max}} \quad \rho = \frac{L_{\max}}{P_{\max}}. \tag{8}$$

If we further define V_n and f_n as the normalized voltage and frequency levels, then D_n and L_n can be rewritten as

$$D_n = \lambda \frac{D}{D_{\max}} = \lambda \frac{C_L V^2 f}{C_L V_{\max}^2 f_{\max}^2} = \lambda V_n^2 f_n \tag{9}$$

$$L_n = \rho \frac{L}{L_{\max}} = \rho \frac{I_L V}{I_L V_{\max}} = \rho V_n. \tag{10}$$

Combining (9) and (10), we get

$$P_n = \lambda V_n^2 f_n + \rho V_n. \tag{11}$$

We next define T_n as the execution time of a task at v-f setting f_n normalized against the execution time at the maximum v-f setting ($f_n = 1$), $T_{\max}$, i.e.,

$$T_n = \frac{T}{T_{\max}}. \tag{12}$$

The normalized energy consumption of an executing task as a function of the normalized voltage, frequency, and execution time is then

$$E_n(V_n, f_n, T_n) = \lambda V_n^2 f_n T_n + \rho V_n T_n. \tag{13}$$

For any given processor, λ and ρ can be considered to be constants. Given the selected v-f setting (V_n/f_n) and the execution time (T_n) for a task at that setting, we can estimate the CPU energy consumption using (13).

Task Characterization: The execution time of a task can be broken down into two components: 1) $T_{\rm cpu}$—corresponding to the time during which the execution is CPU bound, and hence, no stalls occur, and 2) $T_{\rm stall}$—corresponding to the time during which CPU stalls because of memory accesses (cache misses), dependencies, etc. If we define $T_{\max}$, $T_{{\rm cpu}_{\max}}$, and $T_{{\rm stall}_{\max}}$ as the durations at the maximum v-f setting ($f_n = 1$), and $T_{\rm ncpu}$ and $T_{\rm nstall}$ as normalized versions of $T_{\rm cpu}$ and $T_{\rm stall}$, respectively, at f_n, then

$$\begin{aligned} T &= T_{\rm stall} + T_{\rm cpu} \\ T_{\max} &= T_{{\rm stall}_{\max}} + T_{{\rm cpu}_{\max}} \\ T_n &= \frac{T}{T_{\max}} = T_{\rm nstall} + T_{\rm ncpu}. \end{aligned} \tag{14}$$

During $T_{\rm stall}$, the CPU is waiting for the cause of stall to be resolved. For instance, when there is a cache miss, CPU is stalled waiting for the memory access to complete. Thus, duration of $T_{\rm stall}$ is independent of the frequency setting of the CPU, since the CPU is not executing instructions. This means

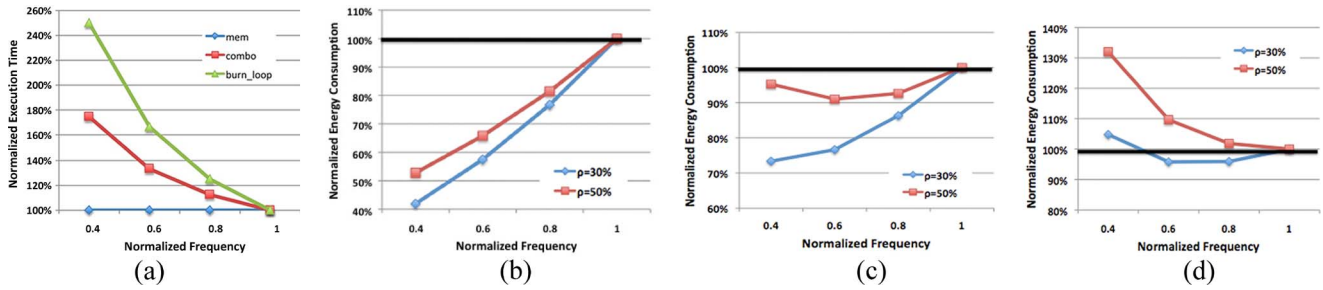

Fig. 2. Execution time and energy consumption estimates. For energy estimates, the black line indicates the baseline or the energy consumption at $f_n = 1$. The region below the baseline indicates energy savings, while the region above indicates higher energy consumption or energy loss. (a) T_n estimates. (b) mem ($\mu \approx 0$). (c) combo ($\mu \approx 0.5$). (d) burn_loop ($\mu \approx 1$).

that T_{nstall} is constant across all the v-f settings and hence also equal to $T_{\text{nstall}_{\max}}$, the normalized duration at the maximum v-f setting ($f_n = 1$)

$$T_{\text{nstall}} = T_{\text{nstall}_{\max}} = \frac{T_{\text{stall}_{\max}}}{T_{\max}}. \tag{15}$$

In contrast, during T_{cpu}, the execution is cache intensive, and thus, its duration depends on the number of cycles it takes to access the cache and CPU registers. The duration of this cycle is directly proportional to the frequency of the CPU, and hence, a reduction in frequency results in a proportional increase in T_{ncpu}

$$T_{\text{ncpu}} = \frac{T_{\text{ncpu}_{\max}}}{f_n}, \qquad \text{where } T_{\text{ncpu}_{\max}} = \frac{T_{\text{cpu}_{\max}}}{T_{\max}}. \tag{16}$$

Combining (14)–(16), we get

$$T_n = T_{\text{nstall}_{\max}} + \frac{T_{\text{ncpu}_{\max}}}{f_n}. \tag{17}$$

If we define $T_{\text{ncpu}_{\max}} = \mu$, then $T_{\text{nstall}_{\max}} = (1 - \mu)$ since T_n at the maximum v-f setting ($f_n = 1$) is one. The factor μ indicates the degree of CPU and cache intensiveness of a task, with a high value indicating high CPU intensiveness and a low value indicating otherwise. Substituting these values in (17), we get

$$T_n = (1 - \mu) + \frac{\mu}{f_n}. \tag{18}$$

On the basis of this discussion, we define three tasks with different characteristics: 1) *burn_loop*—highly CPU and cache intensive ($\mu \approx 1$); 2) *mem*—highly stall intensive due to memory accesses ($\mu \approx 0$); and 3) *combo*—combination of the previous two ($\mu \approx 0.5$). Fig. 2(a) shows T_n estimates based on (18) for these three tasks across four different f_n values (100%, 80%, 60%, and 40%). It illustrates that T_n for highly memory/stall intensive task *mem* is fairly insensitive to changes in f_n, while for CPU intensive *burn_loop*, it increases in proportion to the decrease in f_n. To verify these estimates, we implemented three benchmarks with such characteristics and ran them on an Intel PXA27x CPU [31] at 520 MHz/$f_n = 100\%$, 416 MHz/$f_n = 80\%$, 312 MHz/$f_n = 60\%$, and 208 MHz/$f_n = 40\%$. We found the estimated values of T_n to be on an average within 1% of the actual measured values. This shows that our analysis of T_n estimation is accurate, and emphasizes the importance of task characteristics for modeling its execution times.

Impact of Task and Leakage Characteristics: To understand the impact of task (μ) and CPU leakage (ρ) characteristics on the energy savings at different f_n values, we next estimate the energy consumption (E_n) of these three tasks using (13). Fig. 2(b)–(d) shows E_n for benchmarks *mem*, *combo*, and *burn_loop* at different f_n values for platforms with different leakage percentage (ρ) values (30% and 50%). For a modern day processor like Intel PXA27x, ρ is around 30% and is expected to rise further in future CPUs according to industry estimates [32]. We confirmed this value of ρ for PXA27x CPU by measuring the current flowing into the processor employing a 1.25-Msamples/s data acquisition system (DAQ) at different v-f settings. We also measured the energy consumption of these benchmarks on PXA27x CPU ($\rho \approx 30\%$) and found them to be on an average within 1% of the theoretically estimated values for $\rho \approx 30\%$ using (13). This confirmed the validity of our analysis and estimates. We used V_n values of (80%, 86.6%, 93%, and 100%), which correspond to the v-f settings on the PXA27x CPU, for these estimates (refer to Table VII).

Fig. 2(b)–(d) shows the following. First, increasing the fraction of leakage (ρ) value results in a reduction in energy savings consistently across all the tasks. The reason for this is that DVFS brings just a linear decrease in leakage power, compared to cubic in active power [refer to (6)]. Hence, its effectiveness begins to diminish with increasing ρ. Second, the energy savings decrease due to the increased execution times of the task. For *mem*, which has no performance penalty across different f_n values, the gain in energy savings is significant with decreasing frequency because of the lower voltage. Task *burn_loop* incurs the highest performance penalty, which manifests in low energy savings at $\rho = 30\%$ and higher energy consumption than the baseline ($f_n = 100\%$) at higher ρ values. This means that it is no longer energy efficient to run *burn_loop* and *combo* at lower v-f settings. It is better to run such tasks at the highest v-f setting (i.e., no DVFS) and then switch to DPM when they become inactive. This indicates that task and leakage characteristics determine the tradeoff between DVFS and DPM, and hence, it is important to take both into account. Based on these observations, we now discuss how the controller incorporates estimation of these characteristics into its design.

Expert Selection: The controller maintains two weight vectors: one for the DPM experts (policies) and second for the DVFS experts (v-f settings). During the idle periods, when the idle task is scheduled by the OS, the controller selects among the DPM experts as explained in Section IV-A. During active

execution periods, the controller selects among the DVFS experts based on the characteristics of the executing task. Since, in a multitasking environment, tasks with differing characteristics can be runnable at the same time, the controller maintains a per-task weight vector. These weight vectors get initialized when the task gets created.

Workload characterization for DVFS involves accurately measuring the degree of CPU intensiveness or μ [refer to (18)] of the executing workload. In order to estimate μ, we use the concept of *cycles per instruction (CPI) stack*, which breaks down processor execution time into a baseline CPI plus a number of miss event CPI components like cache and translation lookaside buffer (TLB) misses, etc. [33]. The following equation represents the average CPI in terms of its' CPI stack components:

$$\mathrm{CPI_{avg}} = \mathrm{CPI_{base}} + \mathrm{CPI_{cache}} + \mathrm{CPI_{tlb}} + \mathrm{CPI_{branch}} + \mathrm{CPI_{stall}}. \tag{19}$$

CPI stacks give insight into the CPU intensiveness of the currently executing task. For instance, a high value for $\mathrm{CPI_{base}}/\mathrm{CPI_{avg}}$ ratio indicates that CPU is busy executing instructions for majority of the time, thus indicating a higher μ and vice versa.

In order to dynamically characterize executing workload, we construct its CPI stack at runtime. For the PXA27x processor, we do this by using the performance monitoring unit (PMU). The PMU is an independent hardware unit with four 32-b performance counters that can be used to monitor any four out of 20 unique events available simultaneously. We monitor the number of instructions executed (INSTR), data cache misses (DCACHE), number of cycles instruction cache could not deliver instruction (ICACHE), and processor stall cycles due to dependence (STALL). We also get the total number of clock cycles (CCNT) elapsed since the PMU was started in order to calculate the CPI components

$$\begin{aligned} \mathrm{CPI_{avg}} &= \mathrm{CCNT/INSTR} \\ \mathrm{CPI_{dcache}} &= (\mathrm{DCACHE} \times \mathrm{PEN})/\mathrm{INSTR} \\ \mathrm{CPI_{icache}} &= \mathrm{ICACHE/INSTR} \\ \mathrm{CPI_{stall}} &= \mathrm{STALL/INSTR}. \end{aligned} \tag{20}$$

In this equation, $\mathrm{CPI_{cache}}$ has been broken down into $\mathrm{CPI_{icache}}$ and $\mathrm{CPI_{dcache}}$ and PEN is the average miss penalty for a data cache miss (we used PEN $= 75$ cycles at 520 MHz in our experiments). Note that $\mathrm{CPI_{tlb}}$ and $\mathrm{CPI_{branch}}$ are missing. This is because we can monitor only four events at a time, and in our experiments, we found the events being monitored to be more indicative of the task characteristics. Hence, we can estimate $\mathrm{CPI_{base}}$ as follows:

$$\mathrm{CPI_{base}} = \mathrm{CPI_{avg}} - \mathrm{CPI_{icache}} - \mathrm{CPI_{dcache}} - \mathrm{CPI_{stall}}. \tag{21}$$

Finally, we estimate μ as a ratio of $\mathrm{CPI_{base}}$ to $\mathrm{CPI_{avg}}$ in the following equation:

$$\mu = \mathrm{CPI_{base}}/\mathrm{CPI_{avg}}. \tag{22}$$

With CPU intensiveness (μ) of the task estimated, we now describe the loss evaluation stage for the DVFS experts. To evaluate the loss factor of each expert, we define a data structure called μ-mapper, which maps the suitability of v-f settings to the characteristics of the task as a function of the CPU leakage characteristics. For instance, for CPU with $\rho = 30\%$, Fig. 2(b)–(d) shows that the best-suited frequency scales linearly with μ of a task. Based on this observation, Fig. 3(a) shows a μ-mapper for a CPU with five experts, where the frequencies increase in equal steps from 0 to Expert 5. The μ-mapper divides the domain of μ $(0 \leq \mu \leq 1)$ into intervals for each expert and sequentially maps each interval to the successive experts. This captures the fact that higher frequency experts are better suited for tasks with higher μ. The mean of each interval, μ-mean, is associated with their respective expert and is used for performing weight update. If CPU has higher leakage component (e.g., $\rho = 50\%$), then the μ-mapper is changed accordingly. For instance, if we compare the energy results in Fig. 2(b)–(d) for $\rho = 50\%$ and $\rho = 30\%$, we can see that the energy consumption of *combo* for $\rho = 50\%$ is similar to that of *burn_loop* at $\rho = 30\%$. This means that for CPUs with such high leakage, workloads with medium CPU intensiveness need to be run at higher v-f settings, while highly CPU intensive tasks, such as *burn_loop*, must run only at the highest v-f setting. Thus, to derive a μ-mapper for such a CPU, we will map only 0–0.5 of μ's sample space to the available v-f settings. Fig. 3(b) shows the corresponding μ-mapper. This way, μ-mapper provides the controller with the critical information on CPU leakage characteristics and allows it to adapt seamlessly across different CPUs without any modifications to the core algorithm.

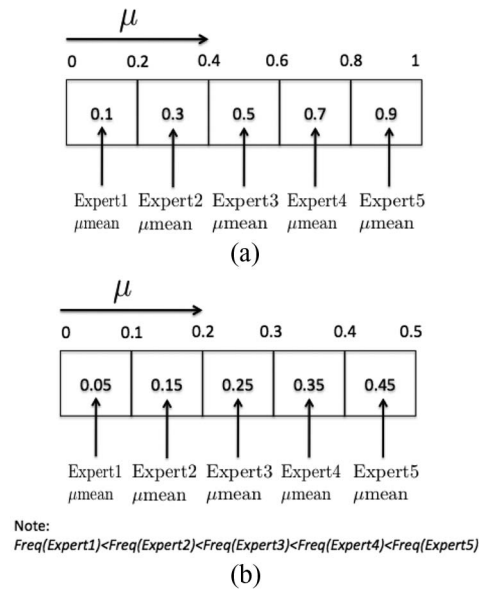

Fig. 3. μ-mappers. (a) $\rho = 30\%$. (b) $\rho = 50\%$.

Given a μ-mapper, the loss factors can be easily evaluated by comparing μ to the μ-mean of each expert. This calculation takes both the energy savings and performance delay into account by breaking this loss factor into two components, the energy loss (l_{ie}^t) and the performance loss (l_{ip}^t). If $\mu <$ μ-mean for an expert, then the task is more stall intensive with respect to the given expert and hence can afford to run slower. At the same time, it means that this expert would cause no performance

TABLE II
LOSS EVALUATION METHODOLOGY

	Energy Loss (l_{ie}^t)	Perf Loss (l_{ip}^t)
$\mu > \mu$-mean	0	($\mu - \mu$-mean)
$\mu < \mu$-mean	(μ-mean $-\mu$)	0
Total Loss (l_i^t) = $\alpha \cdot l_{ie}^t + (1-\alpha) \cdot l_{ip}^t$		

TABLE III
DEVICE AND WORKLOAD CHARACTERISTICS FOR HDD

Device	Trace	Duration (sec)	$\bar{t}_{idle}$	$\sigma_{\bar{t}_{idle}}$	P_{on} (W)	P_{sleep} (W)	T_{be} (sec)
HP	HP-1	32311	30	91	1.6	0.4	4.2
	HP-2	35375	25	86			
	HP-3	29994	36	119			
Laptop	Laptop-1	18017	17	111	0.95	0.13	5.2
	Laptop-2	7528	11	31			

$\bar{t}_{idle}$: Average Idle Period Duration (in sec)

delay for the current task, since it corresponds to a higher frequency than required. Similarly, there is a performance loss, but there is no energy loss when $\mu > \mu$-mean. Table II summarizes how we evaluate the loss factor. The α factor in Table II is similar to the one used for DPM (4), a user-defined value that determines the e/p tradeoff. Once the loss factors are evaluated for each expert, the controller updates the weights of all the experts using (2). The controller then restarts the PMU so that μ for the upcoming scheduler quantum can be evaluated at its conclusion.

The task weight vector accurately characterizes the task it represents, since it encapsulates all the updates based on previous μ values. Moreover, the weight updates are based on both the task as well as the CPU leakage characteristics (μ-mapper). This ensures that the controller understands when it is beneficial to perform aggressive DVFS or not from overall energy efficiency point of view (as described in the previous paragraphs).

V. EXPERIMENTS AND RESULTS

We performed our experiments using the following devices: an HDD and Intel PXA27x core (CPU) with real-life workloads. For HDD, we used the controller with just DPM enabled because of lack of DVFS functionality, while for CPU, we used controller with both DPM and DVFS enabled. The results indicate that our controller is capable of dynamically adapting while delivering sizable (as high as 60%) energy savings over a range of e/p tradeoff settings.

A. HDD (DPM)

We evaluated our online-learning-based DPM algorithm by studying both server and laptop HDDs. We used two sets of workload traces: 1) originally collected on an HP server [34] (referred to as HP traces) and 2) traces collected on a laptop hard disk (referred to as laptop traces) [12]. The characteristics of workloads selected are described in Table III. This is a broad range of workload characteristics. For example, HP-1 and HP-3 traces have very different distribution of idle time durations in terms of both average value and standard deviation ($\bar{t}_{\rm idle}$ and $\sigma_{\bar{t}_{\rm idle}}$, respectively). We consider the HDD to be idle after 1 s of inactivity. This threshold is based on the observation that across all the workloads, many idle periods are smaller than 1 s. These idle periods incur performance delay without contributing much to the energy savings. Table III also shows the device characteristics in terms of $P_{\rm on}$ and $P_{\rm sleep}$, which refer to the powers consumed when the devices are on and in the sleep state, respectively. $T_{\rm be}$ refers to the break-even time. We run the workload traces described in Table III and record the performance in terms of energy savings and performance delays for both the individual experts as well as the controller.

TABLE IV
WORKING-SET CHARACTERISTICS FOR HDD

Expert	Characteristics
Fixed Timeout	Timeout = $3T_{be}$
Adaptive Timeout [35]	Initial Timeout = T_{be} Adjustment = $+1T_{be}/-1T_{be}$
Exponential Predictive [8]	$I_{n+1} = \alpha \cdot i_n + (1-\alpha) \cdot I_n$ with $\alpha = 0.5$
TISMDP [12]	Optimized for delay constraint of 2.3% on HP-3 Trace

For our working set, we select fixed timeout, adaptive timeout [35], exponential predictive [8], and TISMDP [12] policies, representing different classes of state-of-the-art DPM policies. While fixed and adaptive timeout policies represent the timeout class, exponential predictive policy represents the predictive class and TISMDP represents the stochastic class of policies. Table IV describes the precise characteristics of the DPM policy experts employed for the experiments. The fixed timeout employs a timeout equal to three times the break-even time or $T_{\rm be}$ (see Section II for definition). The adaptive timeout policy uses $T_{\rm be}$ as the initial timeout with an adjustment factor of $+1T_{\rm be}/-1T_{\rm be}$, depending on whether the previous idle period resulted in energy savings or not. Exponential predictive policy is implemented as described in [8] without pre-wake-up. It predicts the length of the upcoming idle period (I_{n+1}) using the actual (i_n) and predicted (I_n) lengths of the previous idle period. TISMDP policy is optimized for a given delay (2.3%) on the HP-3 trace. The main idea we are trying to show is that given a set of experts, the controller always converges to select the best-performing expert at any point in time.

Table V shows the results achieved in terms of energy savings and performance delay for the individual experts on the HDD traces. The %energy indicates the amount of energy saved relative to the case where we do not have any DPM policy, while the %delay shows the amount of performance delay caused relative to the total time frame because of power management. The first row of the table gives the results for the oracle policy. It is an ideal offline policy which knows the workload in advance and hence always takes the optimal decision for each idle period. Consequently, its performance delay is 0%. The results highlighted in black show where we get the best energy savings, while the results highlighted in gray show the case where we get the least performance delay. We can notice that the HP trace predictive policy does well in terms of saving energy. For instance, on HP-1, it achieves around 58% energy savings. It performs equally well for the other workloads as well. However, predictive policy is also the worst in terms of causing performance delay, since it is extremely aggressive in turning

TABLE V
ENERGY SAVINGS/PERFORMANCE DELAY RESULTS FOR HDD WITH INDIVIDUAL EXPERTS

Expert	HP-1 Trace		HP-2 Trace		HP-3 Trace	
	%delay	%energy	%delay	%energy	%delay	%energy
Oracle	0	68.68	0	66.58	0	71.38
Timeout	3.95	44.4	4.08	41.28	3.23	50.44
Ad Timeout	6.61	57.2	7.35	54.12	5.27	60.63
TISMDP	2.9	38.7	2.7	36.5	2.29	44.6
Predictive	7.32	58.2	8.32	54.9	5.87	60.7

(gray shade indicates min perf delay and black indicates max energy savings)

(a)

Expert	Laptop-1 Trace		Laptop-2 Trace	
	%delay	%energy	%delay	%energy
Oracle	0	76	0	65
Timeout	1	59	1.5	45
Ad Timeout	2	65	3.3	53
TISMDP	0.7	47	0.9	32
Predictive	4.1	42	6.7	30.2

(b)

off the HDD and thus incurs delay while waking up. In contrast, TISMDP causes the least performance delay and consequently fetches the least energy savings. It can be observed in Table IV that TISMDP was optimized for 2.3% delay on HP-3 workload and the results achieved confirm this. However, the figure is not the same for HP-1 and HP-2 workloads, which confirms that it is optimal for stationary workloads and does not adapt with changing workloads. Fixed timeout performs reasonably well on both the accounts, while adaptive timeout is quite close to predictive in terms of energy savings.

Similarly, Table V(b) shows the results with individual experts for the laptop traces. We can observe that for these traces, the predictive expert performs the worst in terms of both energy savings as well as performance delay. This is in contrast to HP traces, where it did the best in terms of energy savings. This happens due to the smaller idle periods of laptop traces (Table III) and lack of correlation between successive idle period durations, which causes the predictive expert to cause many wrong shutdowns (where $T_{\text{idle}} < T_{\text{be}}$). These results highlight that different classes of policies, depending upon their characteristics, deliver different levels of performance across different workloads. However, depending upon the application requirements or user preferences, one might want the overall performance to be more delay sensitive or more energy sensitive. The problem with just having a single DPM policy is that it does not offer the flexibility to control this behavior. The controller offers exactly this flexibility.

Table VI shows results achieved on the same traces using the controller with different e/p tradeoff (α) settings. As explained in Section IV-A, α value indicates the desired e/p tradeoff setting. A high value indicates higher preference to energy savings, a low value indicates higher preference to performance, and a medium value indicates a reasonable ratio of both. In our experiments, we tested with values of α ranging from around 0.3 (low) to 0.7 (high). We used values of α around 0.5 for the medium value. As we increase the value of α, we get higher energy savings, and for lower values of α, we get low performance delay. For instance, on HP-2 workload, we get 50.1% energy savings for high α, which is quite close to that achieved by predictive and adaptive timeout policies. In

TABLE VI
ENERGY SAVINGS/PERFORMANCE DELAY RESULTS FOR HDD WITH CONTROLLER

Preference	HP-1 Trace		HP-2 Trace		HP-3 Trace	
	%delay	%energy	%delay	%energy	%delay	%energy
Low α	2.95	39.27	2.8	36.6	2.5	45.9
Med α	4.17	47.2	4	43.4	3.33	53.2
High α	5.76	55.2	6.21	50.1	4.53	57.9

(a)

Preference	Laptop-1 Trace		Laptop-2 Trace	
	%delay	%energy	%delay	%energy
Low α	0.7	49	1	33.5
Med α	0.8	54	1.3	41
High α	1	61	1.7	48

(b)

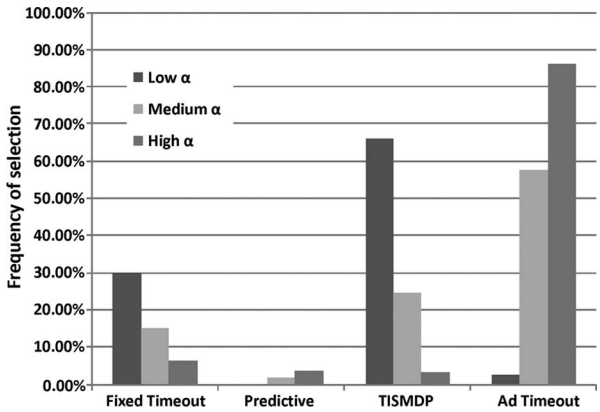

Fig. 4. Selection frequency of experts for HP-3 trace.

contrast for low α, we get performance delay that is comparable to that of TISMDP. Remember that we limit the values of α between 0.3 and 0.7. For even higher values (close to one), we achieve energy savings even closer to that of adaptive timeout and predictive policies. Similarly, for the laptop traces, we observe results converging to that of TISMDP for low α and to that of adaptive timeout for high α.

Fig. 4 shows how the frequency of selection of experts changes with α on HP-3 trace. For higher value of α, adaptive timeout expert is selected most often since it achieves close to highest energy savings (60.6%) at a lower performance delay than predictive expert. For lower values of α, TISMDP expert is selected with higher frequency since it is conservative in turning off the HDD and thus offers lower performance delays. For the medium value of α, we can see that it selects among all the experts to deliver a performance which offers a reasonable e/p tradeoff. Hence, α factor offers us a simple yet powerful control knob to obtain the desired e/p tradeoff.

We next show that our controller can form a Pareto optimal curve over a set of experts. The experts consist of TISMDP, predictive, four adaptive timeout (AT0-3), and four fixed-timeout (FT0-3) policies. We run the controller at six different α values (ranging from low of around 0.3 to high of around 0.7) to get different e/p tradeoffs. Fig. 5 shows a line connecting the e/p tradeoff points offered by the controller and the e/p tradeoff points offered by the individual experts. The line divides the e/p space into two parts: 1) the part above the line represents e/p tradeoff points that are better than those offered by the

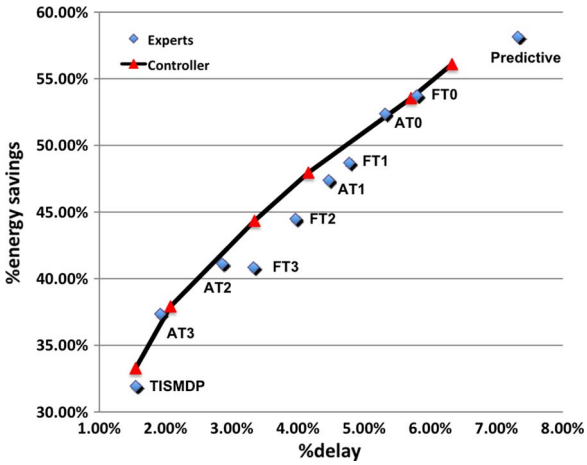

Fig. 5. Controller comparison with multiple experts for HP-1 trace.

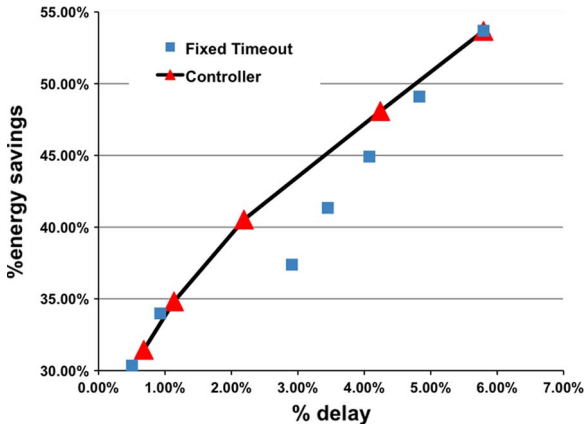

Fig. 6. Controller comparison with fixed-timeout experts for HP-1 trace.

controller, since they either offer lower performance delay for same energy savings or vice versa, and 2) the part below the line represents e/p tradeoff points that are inferior to those of controller. Fig. 5 illustrates that the e/p tradeoff points of individual experts either lie on this line or below it. Therefore, based on the e/p preference (α), the performance of the controller either converges to that of the best expert or is even superior to any of the experts. Better results than any single expert are possible due to the fast convergence property of the controller to the best-performing experts over different phases of the workload.

Selection With Fixed-Timeout Policies: We next test our controller with a working set of eight simple fixed-timeout policies that have timeout values ranging from $T_{\rm be}$ to as large as 50 s for HDD. A timeout of $T_{\rm be}$ guarantees that the energy consumption is not greater than a factor of two when compared to an ideal offline policy [5]. Timeout of 50 s represents a conservative policy to keep the performance delay low.

Fig. 6 shows the performance of these individual timeout policies against the performance of the controller corresponding to the HP-1 workload. The line in Fig. 6 shows e/p points for the controller algorithm with five different values for α. Just like Fig. 5, all the e/p tradeoff points for the fixed-timeout experts are either close to the line or below it, showing that our controller can achieve pareto optimality. For high values of α, we get energy savings as high as 54%, which is very close to that achieved by controller using a working set of more sophisticated policies (see Table V). For a low value of α, it achieves a very low performance delay of just 0.5%. This clearly shows that the controller can achieve competitive e/p tradeoffs just with a set of simple fixed-timeout policies.

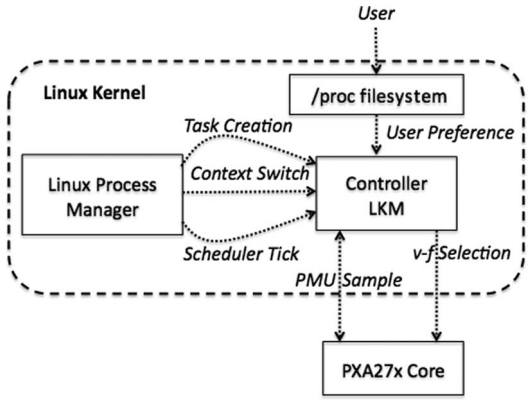

Fig. 7. CPU controller system-level implementation.

B. CPU (DPM + DVFS)

In this section, we present results for the controller with both DPM and DVFS enabled. We use the CPU of the Intel PXA27x platform (running Linux 2.6.9) as the testbed for these experiments. Fig. 7 shows details of our system-level implementation of the controller. We implement the controller as a Linux loadable kernel module (LKM). As soon as the LKM is loaded, it performs both the DPM and DVFS specific initializations. For DPM, it initializes the weight and probability vectors corresponding to the DPM policy experts. For DVFS, it scans the available v-f setting experts and calculates the corresponding μ-means based on the μ-mapper shown in Fig. 3(a). This is based on our experiments with PXA27x (refer to Section IV-B) that indicate $\rho \approx 30\%$.

As shown in Fig. 7, the LKM is closely knit to the Linux process manager. To isolate the characterization in terms of CPU/memory intensiveness on a per-task basis and preserve them across context switches, the Linux task data structure *task_struct* is modified to include the weight vector. This is required for DVFS but not for DPM since in that case, we model the inactivity of the system, and when the system gets inactive, the only task that runs is the idle thread. The controller always has a pointer to the current task data structure, and this ensures that the update and selection of experts are occurring for the current task only. The LKM also exposes a */proc* interface to the user, which can be used to specify the e/p tradeoff (α). The LKM receives notifications from the process manager on the following three events: 1) *Task creation*—The process manager provides LKM with the pointer to the new *task_struct*, and the LKM initializes the per-task DVFS weight vector; 2) *scheduler tick*—this acts as a controller event for DVFS; and 3) *context switch*—on this notification, the LKM checks if the next thread that is getting scheduled is the idle thread. If it is the case, then it switches to the lowest v-f setting in the working set and selects a DPM expert since it indicates the beginning of an idle period.

TABLE VII
DEVICE CHARACTERISTICS: CPU

P_{on}	P_{idle}	$P_{standby}$	P_{tr}	T_{tr}	T_{be}
747mW	129mW	1.7mW	747mW	12ms	12ms

(a)

Freq (MHz)	Voltage (V)
208	1.2
312	1.3
416	1.4
520	1.5

(b)

The Linux 2.6.9 kernel has a scheduler quantum of 10 ms. This means that the scheduler runs every 10 ms, irrespective of whether there are any schedulable threads in the system or not. This is clearly energy inefficient since it makes 10 ms the upper bound on the maximum achievable idle time for the CPU. This renders most of the low power modes of the PXA27x CPU (standby, sleep, etc.) unusable, since they have larger break-even times. To solve this problem, dynamic tick support has been added to the latest kernel version (2.6.24) [36]. This allows reprogramming of scheduler tick when there are no schedulable threads in the system, hence achieving longer idle periods. However, there is no publicly available support for the port of this kernel version on the PXA27x platform, with the latest being 2.6.9 [37]. The absence of dynamic tick support made practical experiments of DPM for CPU impossible because of the periodic scheduler ticks. Hence, we performed experiments for DPM based on simulations on traces of real-life workloads collected using the LKM. The LKM calculates the idle periods as durations for which the idle thread stays scheduled at a stretch. The approach is very similar to the one used in [38].

Hence, in our current setup, we run the workloads with and without DVFS enabled and calculate CPU energy savings by current measurements using a 1.25-Msamples/s DAQ. The energy savings and performance delay values are compared to a system which performs the following: 1) It runs at the highest v-f setting (1.5 V/520 MHz in our case) when active and 2) switches to idle mode and the lowest v-f setting (1.2 V/208 MHz) when idle. While running the workloads, the LKM also generates a trace of the idle period distribution. This allows us to estimate the energy savings due of DPM offline using the characteristics listed in Table VII(a). The DPM working set comprises nine fixed-timeout policies with timeouts ranging from $T_{\rm be}$ to $9T_{\rm be}$. We have shown in Section V-A on HDD that a working set of fixed-timeout policies performed nearly as well as the one with more sophisticated policies. For DVFS, we use working set of v-f setting experts listed in Table VII(b).

Workloads for CPU can be divided into two categories: *idle dominated* and *computationally intensive*. Idle-dominated workloads are the ones for which the idle thread is scheduled most of the time. All user-interface-related workloads like word processing, web surfing, etc., belong to this category. The computationally intensive workloads are the ones for which the idle thread never gets scheduled. This category includes tasks like decompressing, running intensive encryption/decryption algorithms, etc. Typical real-life applications are a mix of both. For instance, the workload for a user using GUI-based archiving application would be mostly idle dominated when he is browsing the various options and computationally intensive in bursts when he enters password for authentication or actually compresses/decompresses some file. In the next two sections, we present the results for these kinds of workloads separately.

TABLE VIII
ENERGY SAVINGS/PERFORMANCE DELAY RESULTS FOR CPU ON IDLE-DOMINATED WORKLOADS

Preference	editor		www	
	%delay	%energy	%delay	%energy
Low α	0.41	17	0.31	19
Med α	0.81	20	0.85	22
High α	1.56	24	1.15	25
Oracle	0	26	0	27

TABLE IX
FREQUENCY OF SELECTION OF EXPERTS (%)

Expert	editor			www		
	Low α	Med α	High α	Low α	Med α	High α
T_{be}	1.4	37	84	10.9	65	100
$2.T_{be}$	3.7	29.7	14.3	0.8	13.7	0
$3.T_{be}$	1.4	7	1.8	2	1.6	0
$4.T_{be}$	0.6	2.4	0	0.4	0.4	0
$5.T_{be}$	4.7	1	0	0	0	0
$6.T_{be}$	3.5	0.6	0	0	0	0
$7.T_{be}$	25.5	5.5	0	2.4	2.4	0
$8.T_{be}$	0	0	0	17.3	12.5	0
$9.T_{be}$	59.1	16.7	0	66.1	4.4	0

Idle-Dominated Workloads: For these experiments, we used two real-life workloads: *editor* and *www*. The *editor* workload involves the use of *vi* editor for writing and reviewing data files and the use of some basic shell commands (ls, etc.). The *www* workload involves general web surfing (search, reading articles, etc.) on a wireless interface using the lynx web browser. We analyzed the results for the following three configurations of the controller: only DPM, only DVFS, and both DPM and DVFS.

For the first case, we disabled DVFS and kept DPM enabled. Table VIII shows the results achieved for with the two workloads for different values of α. The %energy numbers indicate the energy savings baselined against the case where the CPU is placed in idle mode (which halts its clock supply) and the lowest v-f setting as soon as it gets idle. The %delay is the overhead incurred because of power management. We again used values around 0.3, 0.5, and 0.7 for low, medium, and high values of α. The last row shows the results for the oracle policy, which shows the maximum achievable energy savings for these workloads using DPM. We can see in Table VIII that for both workloads, with increasing value of α, the energy savings increase. For high α, the energy savings are around 25% for both *editor* and *www*, which is very close to the energy savings achieved by the oracle policy.

Table IX shows the frequency of selection of experts for the different α settings for both the *editor* and *www* workloads. We can observe that for low α, $9T_{\rm be}$ is the most selected expert since it offers the least possible delay across all the experts. As we move toward higher values of α, the frequency of selection of smaller timeout experts increases since they offer higher energy savings. In fact, for *www* workload, $T_{\rm be}$ expert gets selected for all idle periods. The table also shows that the nature

TABLE X
ENERGY SAVINGS/PERFORMANCE DELAY RESULTS FOR CPU ON COMPUTATIONALLY INTENSIVE WORKLOADS

Benchmarks	Low α		Med α		High α	
	%delay	%energy	%delay	%energy	%delay	%energy
qsort	6	17	17	32	25	41
djpeg	7	21	15	37	27	45
dgzip	15	30	21	42	28	49
bf	6	11	16	28	25	40
burn_loop	0	0	10	2.5	25	5

(a)

Benchmarks	Low α		Med α		High α	
	%delay	%energy	%delay	%energy	%delay	%energy
qsort+djpeg	6	17	15	33	25	41
dgzip+djpeg	13	24	19	40	27	49
qsort+dgzip	7	20	18	35	26	42
dgzip+bf	11	17	20	31	26	45

(b)

of workload affects the expert selection even for the same value of α. For instance, for medium value, T_{be} expert gets selected for 37% with the *editor* workload, but for 65% with the *www* workload.

Next, we experiment with the controller with DVFS enabled and DPM disabled for different values of α. The energy savings are negligible for both *editor* and *www* across all the values of α since they spend most of their time in the idle thread. As described earlier, our baseline CPU switches to the idle mode and the lowest v-f setting available in the working set (1.2 V/208 MHz) on getting idle. Thus, there are no additional savings possible because of DVFS during idle periods. The energy savings for the active periods due to DVFS are negligible, since they are very small compared to the idle periods. The performance delay is almost 0% across all α values as the active periods are not long enough to observe significant performance loss.

In the last set of experiments, we enable both DPM and DVFS. In this case, we observe that the energy savings and performance delay results converge to those in Table VIII, i.e., the case with just DPM enabled. The reason for this is that we now switch to DPM as soon as the idle thread is scheduled, where these workloads spend most of their time. This shows that DPM is very effective for idle-dominated workloads.

Computationally Intensive Workloads: We experimented with a number of computationally intensive workloads in both single and multitasking environments. For such workloads, the idle thread is not scheduled, and thus, the energy savings are exclusively due to DVFS. The chosen workloads include common UNIX utility gzip for decompression (dgzip) and three benchmarks taken from an open-source benchmark suite mibench [39]: bf (blowfish)—a symmetric block cipher; djpeg—decoding a jpeg image file; qsort—sorting a large array of strings in ascending order. All these workloads have a mix of CPU- and memory-intensive phases. We have also added results for the synthetic workload *burn_loop* (Section IV-B) to see how the controller performs for highly CPU-intensive workloads, which benefit the least from DVFS. The results achieved under both single- and multitask environments for these benchmarks are illustrated in Table X. We discuss them separately.

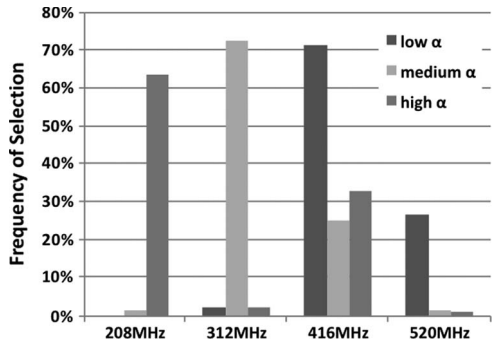

Fig. 8. Frequency of selection of experts for qsort.

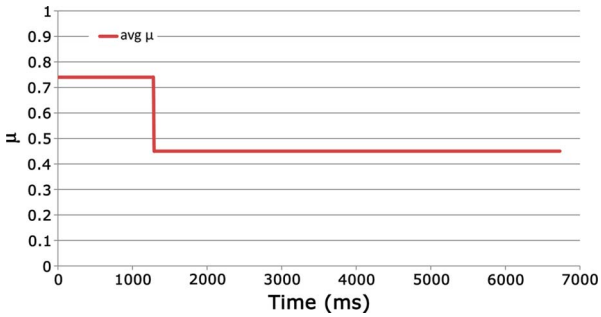

Fig. 9. Average μ for qsort.

Single-Task Environment: Table X(a) displays the results we achieved for each individual benchmark in a single-task environment. From the results, we can observe that as we increase the value of α, we get higher energy savings. For lower values of α, we get lower performance delay. For instance, with qsort, the delay is just 6% for a low value of α, while the energy savings are as high as 41% for a high value.

Fig. 8 shows the frequency of selection of different experts for qsort according to the selected value of α. For higher value of α, the 208-MHz expert is selected for 65% of the time, while the rest of the time, 416-MHz expert is chosen. This suggests that qsort has both CPU- and memory-intensive phases. Fig. 9 shows the average μ of qsort along its execution timeline and illustrates these phases. We can observe that around the first 20% of its execution, qsort is consistently very CPU intensive (high μ), for which the controller selects the 416-MHz expert. Beyond that, it varies, but the average is on the lower side (around 0.45), for which the controller mostly selects the 208-MHz expert. Thus, the controller can quickly and accurately identify CPU-/memory-intensive phases in the workload and adapt at runtime.

We perform offline analysis to evaluate the maximum achievable energy savings for these benchmarks for the given working set by running all the individual benchmarks statically at 208 MHz/1.2 V. This is in line with our observation in Section IV-B [Fig. 2(b)–(d)] that for $\rho \approx 30\%$, the energy savings and performance delay increase with decreasing v-f settings for all kinds of tasks. Table XI displays the %energy and %delay for the benchmarks at this setting. Comparing these results with Table X(a), we can see that for high α, the energy savings for all the benchmarks are on an average within 8% of the maximum possible at much lower overhead. For instance,

TABLE XI
ENERGY SAVINGS/PERFORMANCE DELAY AT 208 MHz/1.2 V

Benchmark	%delay	%energy
qsort	56	48
dgzip	34	54
djpeg	33	54
bf	40	51
burn_loop	150	10

for qsort, with the controller at high α, we get 41% energy savings at delay of 25% compared to 48% savings and 56% delay at 208 MHz/1.2 V. Thus, the controller almost gets the same energy savings at much lower performance delay because it runs the highly CPU intensive phase of qsort (Fig. 9) at 416 MHz (as discussed in the preceding paragraph). For *burn_loop*, the controller is able to identify the high CPU intensiveness and hence runs at 416 MHz/1.4 V even for high α to achieve within 5% of maximum energy savings at significantly lower performance overhead.

Multitask Environment: We next experimented with the benchmarks in a multitasking environment to verify our per-task characterization by spawning two threads running different benchmarks simultaneously. Table X(b) presents the results we achieved for multitasking for different values of α. For djpeg+dgzip, the results are roughly an average of the individual results in Table X(a). This happens because the duration of execution of both tasks is equal. The average values of the delay and savings indicate that both the tasks run with the same expert selection over their execution time frame as in the single-task case across all the α settings. This shows that per-task characterization of the controller is accurately preserved across context switches. For qsort+djpeg, the results for all the values of α correspond very closely to the results of qsort in Table X(a). Since the total time of execution of qsort benchmark is roughly four times the duration of djpeg benchmark, the results converge to that of individual qsort benchmark results. However, the djpeg benchmark runs exactly twice longer with qsort than alone. This shows that accurate preservation of per-task characteristics enables the controller to select the same set of experts for djpeg as it does when djpeg runs alone in the system, hence keeping its effective run time the same. We observed similar results for the qsort+dgzip combination.

Task Characteristics and Leakage Awareness: Since the controller incorporates task characteristics and leakage awareness, it knows when DVFS is beneficial for a task from overall system energy efficiency and performance point of view. This awareness becomes more critical for future generation processors with higher leakage, where running at lower frequency could result in higher energy consumption [see *burn_loop* results in Fig. 2(d)]. To verify how controller would adapt to such platforms, we derived a new μ-mapper for our v-f setting experts based on Fig. 3(b) to simulate a platform with $\rho = 50\%$. We then ran the benchmarks under different α settings. For *burn_loop*, we observed that the controller now selected the 1.5-V/520-MHz expert consistently for the high α settings, since the new μ-mapper now incorporates the knowledge of high leakage power consumption of the CPU. This is in contrast to the results with our old μ-mapper ($\rho \approx 30\%$), where it selected 1.4-V/416-MHz expert for high α. We observed similar results for other benchmarks as well. By incorporating CPU leakage characteristics, controller is also able to balance DVFS and DPM for better overall energy efficiency. This makes the controller scalable and adaptable across CPUs with varying leakage characteristics.

C. Overhead

For HDD, the controller causes overhead in terms of both energy and time to perform the evaluation of experts. In our experiments, we measured the average controller overhead at 0.0001% of the total time frame for HDD, which is negligible relative to the overall time frame. For CPU, the controller adds overhead to the system, since it processes the three events delivered to it by the Linux process manager as discussed in Fig. 7. We used lmbench [40] for measuring the overhead caused by the LKM. The lat_proc and lat_ctx tests measure the overhead added to process creation and context switch times. For lat_proc, the overhead was 0%, while for lat_ctx, it was around 3%. The overhead is negligible because the event processing functions in the controller are extremely fast and lightweight. The controller itself is implemented in fixed-point arithmetic and is very lightweight. The use of weights obviates the need for storing μ estimates, thereby avoiding a potential overhead. For instance, in [26], a regression-based approach is used for workload characterization, which maintains a queue of 25 most recent estimates of CPI_{avg} and MPI_{avg} samples. In tests with lmbench, they increase the context switch time by a factor of two, while our overhead is negligible.

In terms of memory overhead, the controller adds an array of unsigned long long variables, whose size is equal to the number of experts in the working set, to the *task_struct* of Linux. The LKM uses unsigned long long variables to simulate decimal values for weight/probability factors since we do not use floating point inside the kernel. Thus, overall, the overhead of controller on a running system is negligible, which makes its deployment in real systems practical.

VI. CONCLUSION

In this paper, we presented a novel online-learning algorithm that solves both DPM and DVFS problems. We presented a formulation of both DPM and DVFS as one of workload characterization and expert selection and used the algorithm to solve it. The advantage of using an online-learning algorithm is that it provides a theoretical guarantee on convergence to the best-performing expert. The general formulation of the controller makes it applicable to any system component with support for power management. We performed experiments on two different HDDs and an Intel PXA27x CPU under varying real-life workloads in both single-task and multitask scenarios. Our results indicate that our algorithm adapts really well to changing workload characteristics and achieves an overall performance comparable to the best-performing expert at any point in time. Moreover, the algorithm incorporates leakage awareness, which allows it to adapt seamlessly to changing CPU leakage characteristics and also understand the tradeoff between DPM and

DVFS. Moreover, it is extremely lightweight and has almost negligible overhead in terms of performance and energy.

REFERENCES

[1] A. P. Chandrakasan and R. Brodersen, "Minimizing power consumption in digital CMOS circuits," *Proc. IEEE*, vol. 83, no. 4, pp. 498–523, Apr. 1995.

[2] M. Gillespie, *Power Management in the Intel PXA27x Series Application Processors*. [Online]. Available: http://www.intel.com

[3] *Mobile AMD Athlon 4 Processor Model 6 CPGA Data Sheet*, 2001. [Online]. Available: http://www.amd.com

[4] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, Aug. 1997. special issue for EuroCOLT'95.

[5] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. Owicki, "Competitive randomized algorithms for non-uniform problems," in *Proc. 1st Annu. ACM-SIAM SODA*, 1990, pp. 301–309.

[6] R. Golding, P. Bosch, and J. Wilkes, "Idleness is not sloth," Hewlett Packard Lab., Palo Alto, CA, Tech. Rep. HPL-96-140, Oct. 4, 1996.

[7] M. B. Srivastava, A. P. Chandrakasan, and R. W. Brodersen, "Predictive system shutdown and other architectural techniques for energy efficient programmable computation," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 4, no. 1, pp. 42–55, Mar. 1996.

[8] C.-H. Hwang and A. C.-H. Wu, "A predictive system shutdown method for energy saving of event-driven computation," in *Proc. ICCAD*, 1997, pp. 28–32.

[9] E.-Y. Chung, L. Benini, and G. D. Micheli, "Dynamic power management using adaptive learning tree," in *Proc. ICCAD*, 1999, pp. 274–279.

[10] G. A. Paleologo, L. Benini, A. Bogliolo, and G. D. Micheli, "Policy optimization for dynamic power management," in *Proc. DAC*, 1998, pp. 182–187.

[11] Q. Qiu and M. Pedram, "Dynamic power management based on continuous-time Markov decision processes," in *Proc. DAC*, 1999, pp. 555–561.

[12] T. Simunic, L. Benini, P. W. Glynn, and G. D. Micheli, "Event-driven power management," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 7, pp. 840–857, Jul. 2001.

[13] E.-Y. Chung, L. Benini, A. Bogliolo, Y.-H. Lu, and G. D. Micheli, "Dynamic power management for nonstationary service requests," *IEEE Trans. Comput.*, vol. 51, no. 11, pp. 1345–1361, Nov. 2002.

[14] Z. Ren, B. H. Krogh, and R. Marculescu, "Hierarchical adaptive dynamic power management," *IEEE Trans. Comput.*, vol. 54, no. 4, pp. 409–420, Apr. 2005.

[15] G. Theocharous, S. Mannor, N. Shah, P. Gandhi, B. Kveton, S. Siddiqi, and C.-H. Yu, "Machine learning for adaptive power management," *Intel Technol. J.*, vol. 10, pp. 299–312, Nov. 2006.

[16] C. Steinbach, *A reinforcement-learning approach to power management*, 2002. AI Technical Report, M.Eng Thesis, Artificial Intelligence Laboratory, MIT.

[17] G. Quan and X. Hu, "Minimum energy fixed-priority scheduling for variable voltage processor," in *Proc. DATE*, 2002, p. 782.

[18] Y. Zhu and F. Mueller, "Feedback EDF scheduling of real-time tasks exploiting dynamic voltage scaling," *Real-Time Syst.*, vol. 31, no. 1–3, pp. 33–63, Dec. 2005.

[19] A. Azevedo, I. Issenin, R. Cornea, R. Gupta, N. Dutt, A. Veidenbaum, and A. Nicolau, "Profile-based dynamic voltage scheduling using program checkpoints," in *Proc. DATE*, 2002, p. 168.

[20] P. Yang, C. Wong, P. Marchal, F. Catthoor, D. Desmet, D. Verkest, and R. Lauwereins, "Energy-aware runtime scheduling for embedded-multiprocessor SOCs," *IEEE Des. Test Comput.*, vol. 18, no. 5, pp. 46–58, Sep./Oct. 2001.

[21] E.-Y. Chung, G. D. Micheli, and L. Benini, "Contents provider-assisted dynamic voltage scaling for low energy multimedia applications," in *Proc. ISLPED*, 2002, pp. 42–47.

[22] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," in *Proc. OSDI*, 1994, p. 2.

[23] K. Govil, E. Chan, and H. Wasserman, "Comparing algorithm for dynamic speed-setting of a low-power CPU," in *Proc. MobiCom*, 1995, pp. 13–25.

[24] A. Varma, B. Ganesh, M. Sen, S. R. Choudhury, L. Srinivasan, and J. Bruce, "A control-theoretic approach to dynamic voltage scheduling," in *Proc. CASES*, 2003, pp. 255–266.

[25] A. Weissel and F. Bellosa, "Process cruise control: Event-driven clock scaling for dynamic power management," in *Proc. CASES*, 2002, pp. 238–246.

[26] K. Choi, R. Soma, and M. Pedram, "Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 1, pp. 18–28, Jan. 2005.

[27] C. Isci, G. Contreras, and M. Martonosi, "Live, runtime phase monitoring and prediction on real systems with application to dynamic power management," in *Proc. MICRO*, 2006, pp. 359–370.

[28] R. Jejurikar, C. Pereira, and R. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems," in *Proc. DAC*, 2004, pp. 275–280.

[29] P. J. de Langen and B. H. H. Juurlink, "Leakage-aware multiprocessor scheduling for low power," in *Proc. IPDPS*, 2006, p. 60.

[30] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in *Proc. ISLPED*, Aug. 10–12, 1998, pp. 197–202.

[31] *Intel XScale Core Developer's Manual* . [Online]. Available: http://download.intel.com/design/intelxscale/27347302.pdf

[32] [Online]. Available: http://www.itrs.net/Links/2005ITRS/ExecSum2005.pdf

[33] S. Eyerman, L. Eeckhout, T. Karkhanis, and J. E. Smith, "A performance counter architecture for computing accurate CPI components," *Oper. Syst. Rev.*, vol. 40, no. 5, pp. 175–184, Dec. 2006.

[34] C. Ruemmler and J. Wilkes, "UNIX disk access patterns," in *Proc. USENIX Winter*, 1993, pp. 405–420.

[35] F. Douglis, P. Krishnan, and B. N. Bershad, "Adaptive disk spin-down policies for mobile computers," in *Proc. MLICS*, 1995, pp. 121–137.

[36] [Online]. Available: http://lwn.net/Articles/223185/

[37] [Online]. Available: http://pxa linux.sourceforge.net/

[38] L. Benini, A. Bogliolo, S. Cavallucci, and B. Ricco, "Monitoring system activity for OS-directed dynamic power management," in *Proc. ISLPED*, 1998, pp. 185–190.

[39] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Proc. IEEE 4th Annu. Workshop Workload Characterization*, Dec. 2001, pp. 3–14.

[40] [Online]. Available: http://www.bitmover.com/lmbench/

Gaurav Dhiman received the B.Tech. degree in computer science and engineering from the Indian Institute of Technology, Roorkee, India, in 2002 and the M.S degree in computer science from the University of California, San Diego (UCSD), La Jolla, in 2007, where he is currently working toward the Ph.D. degree in computer science with the Department of Computer Science and Engineering.

Prior to coming to UCSD, he was with STMicroelectronics, India, from 2002 to 2005 and has done internships with Qualcomm (summer 2006) and Sun Microsystems (summer 2007 and summer 2008) in the field of low-power systems. His research interests are in operating systems and computer architecture with specific focus on power management.

Tajana Šimunić Rosing (S'97–M'01) received the M.S. degree in electrical engineering, with a thesis on high-speed interconnect and driver-receiver circuit design, from the University of Arizona, Tucson, and the Ph.D. degree, with a dissertation on dynamic management of power consumption, in 2001 from Stanford University, Stanford, CA, concurrently with the M.S. degree in engineering management.

She is currently an Assistant Professor with the Department of Computer Science and Engineering, University of California, San Diego, La Jolla. Prior to this, she was a Full-Time Researcher with HP Laboratories, while working part time with Stanford University, where she was involved with leading research of a number of graduate students and taught graduate-level classes. Prior to pursuing the Ph.D. degree, she was a Senior Design Engineer with the Altera Corporation. Her research interests include energy-efficient computing and embedded and wireless systems.

Dr. Rosing has served at a number of Technical Paper Committees and is currently an Associate Editor of IEEE TRANSACTIONS ON MOBILE COMPUTING. In the past, she has been an Associate Editor of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS.