On the Interplay of Voltage/Frequency Scaling and Device Power Management for Frame-Based Real-Time Embedded Applications

Vinay Devadas and Hakan Aydin, *Member*, *IEEE*

Abstract—Voltage/Frequency Scaling (VFS) and Device Power Management (DPM) are two popular techniques commonly employed to save energy in real-time embedded systems. VFS policies aim at reducing the CPU energy, while DPM-based solutions involve putting the system components (e.g., memory or I/O devices) to low-power/sleep states at runtime, when sufficiently long idle intervals can be predicted. Despite numerous research papers that tackled the energy minimization problem using VFS or DPM separately, the interactions of these two popular techniques are not yet well understood. In this paper, we undertake an exact analysis of the problem for a real-time embedded application running on a VFS-enabled CPU and using multiple devices. Specifically, by adopting a generalized system-level energy model, we characterize the variations in different components of the system energy as a function of the CPU processing frequency. Then, we propose a provably optimal and efficient algorithm to determine the optimal CPU frequency as well as device state transition decisions to minimize the system-level energy. We also extend our solution to deal with workload variability. The experimental evaluations confirm that substantial energy savings can be obtained through our solution that combines VFS and DPM optimally under the given task and energy models.

Index Terms—Real-time systems, energy management, voltage/frequency scaling, device power management.

1 Introduction

Many embedded devices are battery operated and hence, have limited energy supply. Due to the growing demand for smaller devices with longer battery life, energy management has become one of the major goals in embedded systems research. Over the past decade, the research community has made significant progress in the area of low-power system design [13], [20]. On the industry side, the Advanced Configuration and Power Interface (ACPI) standard has moved power management to the operating system level by providing system calls for predictive shutdown of system components [37]. Many applications running on power-limited systems (such as embedded controllers) are subject to timing constraints. As a result, the real-time and energy-aware operation is a highly desirable and sometimes critical feature of an embedded system.

Voltage/Frequency Scaling (VFS) [31] is a popular and widely used technique for power management in real-time embedded systems. With VFS, the processor can operate at different voltage and frequency levels. Since the CPU power consumption increases in a convex fashion with the frequency, VFS helps to significantly reduce the CPU dynamic energy consumption. In real-time systems, preserving the temporal correctness (the system *feasibility*) is of paramount importance [17]. Hence, in VFS settings, utmost care must be exercised to avoid deadline misses. The problem of minimizing the energy consumption while satisfying the timing constraints has been extensively studied in recent past for various task/system models [2], [21], [22], [24].

Device Power Management (DPM) is another commonly used energy management technique, aiming at reducing device energy consumption [4], [6], [7], [10]. Typical devices have an active state in which they process requests and at least one low-power sleep state. DPM involves transitioning devices to low-power states when not in use so as to reduce the device energy consumption. Memory modules and I/O devices which consume significant energy have been the primary targets of DPM. However, nontrivial time and energy overheads are associated with each device state transition. As a consequence, transitioning devices to low-power states is energy efficient only when the device idle interval is guaranteed to be greater than a certain threshold (frequently called the *device break-even time*).

One of the primary difficulties associated with the use of DPM is to decide *when* to switch a device to a low-power state. DPM techniques can be classified as stochastic, predictive, and timeout-based [4]. In real-time systems, the predictive DPM techniques are commonly used. The predictive techniques involve making accurate predictions about the *next usage time* of idle devices. As such, predicting the next device usage time is of critical importance in real-time systems. Underestimations may lead to inefficient energy management (as devices would not be put to low-power states) and overpredictions may lead to potential deadline misses (due to the nontrivial transition delays).

• *The authors are with the Department of Computer Science, George Mason University, 4400 University Drive, MSN 4A5, Fairfax, VA 22030. E-mail: vdevadas@gmu.edu, aydin@cs.gmu.edu.*

Manuscript received 20 July 2009; revised 19 Feb. 2010; accepted 8 Sept. 2010; published online 6 Dec. 2010.
Recommended for acceptance by G. Lipari.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2009-07-0333.
Digital Object Identifier no. 10.1109/TC.2010.248.

Several offline and online solutions have been proposed for real-time DPM under different application/device settings [7], [10], [27], [28], [29].

While VFS and DPM are popular techniques targeting energy minimization in CPU and external devices, respectively, a comprehensive *system-level* energy management policy is likely to use *both* VFS and DPM. However, integrating VFS and DPM in a single framework poses several challenges. With VFS, low processor frequencies lead to low CPU dynamic energy consumption figures. However, this also results in elongated task execution times and shortened device idle intervals. This not only forces devices to remain in active state for longer periods but also limits the possibility of transitioning devices to sleep states (as shortened idle intervals will tend to be smaller than the device break-even times). On the other hand, running the processor at higher frequencies reduces the device energy and creates more opportunities for transitioning devices to sleep states, *at the cost of increased CPU energy and transition energy*. Thus, there is an intriguing trade-off spectrum covering the benefit/cost spaces of VFS and DPM techniques.

Recently, a number of research efforts addressed system-wide energy consumption issues for real-time embedded systems. In [14], the concept of *critical frequency* (or, *energy-efficient frequency*) was introduced. This stems from the observation that lowering the processor frequency below a certain threshold can have negative effects on the system-wide energy consumption. The energy-efficient frequency is calculated by considering both the device energy and CPU energy consumed during task executions. Each task can potentially have a unique energy-efficient frequency, depending on the devices it uses during its execution. In [14], the authors provide a single policy to manage both processor leakage energy and device energy. In [36], the authors propose a dynamic task scheduling algorithm using the concept of critical frequency which minimizes the system-wide energy consumption.

A number of research studies [1], [3], [8], [25] revisited and extended VFS solutions after observing that task execution times do not scale linearly with the CPU frequency. In fact, off-chip access times (e.g., main memory and I/O device access latencies) are mostly independent of the CPU frequency. Hence, the task execution time has a frequency-dependent (on-chip) component that scales linearly with CPU frequency and a frequency-independent (off-chip) component that does not depend on CPU frequency.

In [1], a generalized power model that takes into account several factors such as on-chip/off-chip workload ratios, effective switching capacitance, and frequency-dependent and frequency-independent power components is considered. For this generalized power model, the authors show how to derive the task-level energy-efficient frequencies and propose an $O(n^3)$ algorithm to optimally solve the system-wide energy minimization problem for n periodic hard real-time tasks. In [34], the authors address the energy minimization problem assuming a VFS processor with limited number of frequency values.

While the concept of critical frequency helps mitigate the negative impacts of VFS on the system-wide energy, one major drawback of most system-level real-time energy management research efforts is that they either assume negligible device transition overheads or provide no DPM policies. In fact, most of these studies often make the assumption that the device will be transitioned to the low-power state whenever it is not in use. However, this is typically not the case due to the nontrivial time/energy device transition overheads. A recent research effort that combines both VFS and DPM in the same framework is given in [6], where the authors propose a practical system-level energy management heuristic called SYS-EDF for periodic real-time tasks and discrete model VFS capable processor. SYS-EDF is a combination of a DPM policy Conservative Energy-Efficient Device Scheduling (CEEDS) and a VFS scheme based on energy-efficient scaling.

Despite all these efforts, **an extremely important but mostly unexplored issue is the analysis of exact interplay of VFS and DPM in real-time embedded systems**. Obviously, a straightforward integration of a VFS scheme using critical frequency and a DPM heuristic (such as SYS-EDF) does not exploit this interplay fully. We contend that it is imperative to formally characterize this interplay to devise optimal energy management systems in the presence of both VFS and DPM features.

Contributions of this research effort:

- We address the problem of exact characterization of *system-level* energy consumption for a frame-based real-time embedded application running in an environment with both VFS and DPM features. This characterization allows us to perform a *formal* analysis of the interplay between VFS and DPM for a real-time application that uses several devices. By using the results from this analysis, we propose an $O(m \log m)$-time algorithm (where m is the number of devices used by the application) to determine the optimal processor frequency and device transitioning decisions to minimize the system-wide energy consumption. To the best of our knowledge, this is the first research effort to not only investigate the exact interplay between VFS and DPM, but to also provide a *provably optimal* solution to the system-level energy management problem *by taking into account VFS/DPM-related issues, device transition overheads, and on-chip/off-chip workload characteristics* under the given energy model.
- We evaluate our optimal scheme, over a wide spectrum of system/application parameters and show that it yields significant energy gains when compared to the existing suboptimal approaches. Our experimental results are obtained using real device and CPU specifications.
- We also show how our optimal solution can be extended to address the workload variability. Assuming the knowledge of average-case execution time, we show how to derive CPU frequency and device transitioning decisions that minimize the *average-case system energy* under the constraint that the application's deadline must be still met. Through extensive simulations, we show how this helps achieve significant energy savings in the presence of variability in dynamic execution behavior of the application.

The remaining of this paper is organized as follows: In Section 2, we give the system model and assumptions. In Section 3, we illustrate the nontrivial challenges in the

analysis of the interplay between VFS and DPM, by focusing on the simple case where the real-time application uses only one device. In Section 4, we solve the general case of the problem for multiple devices. In Section 5, we extend our solution to show how the knowledge about average-case execution time can be exploited to reduce the average-case system energy. We conclude in Section 6.

2 System Model and Assumptions

2.1 Application Model

We consider a real-time embedded application that is invoked periodically with a period of d, at time instants $k \cdot d$, where k is a nonnegative integer. At each invocation, the application must complete its execution within the time interval $[k \cdot d, (k+1) \cdot d]$, which is referred to as a *frame*. This embedded application model is also known as a *frame-based system* in the literature [23], [32].

The system is equipped with a VFS-enabled CPU where the processing frequency f can be adjusted up to a maximum frequency f_{max}. We normalize the frequency values with respect to f_{max}. In line with studies in [1], [3], [8], and [25], the worst-case execution time (WCET) of the application under maximum frequency f_{max} is denoted by $c = x + y$, where x denotes the frequency-dependent on-chip workload which scales linearly with CPU frequency and y denotes the frequency-independent off-chip workload which does not scale with CPU frequency. Thus, at frequency f, the WCET of the application is $WCET(f) = \frac{x}{f} + y$ [1], [3], [8], [25].

We assume $c \leq d$ (i.e., the application meets its deadline when executed at f_{max}). The *slack* refers to the unused CPU time between the completion time of the application and the beginning of the next frame, at each invocation. Formally, the slack of the application at frequency f is given by $\delta(f) = d - (\frac{x}{f} + y)$.

2.2 Device Model

The real-time embedded application is assumed to use a set of m devices denoted by $\mathcal{D} = \{D_1 \dots D_m\}$ during its execution. Each device is assumed to have (at least) two states: an active state and a sleep (low-power) state. Following [7], [10], [27], [28], and [29] we assume *intertask device scheduling*. Under *intertask device scheduling* approach, all devices needed by the real-time application must be in active state at the beginning of each frame and they should remain in active state until the application completes its execution in that frame. A device can be put to sleep state when the application completes its execution (i.e., during the *slack* period). These assumptions are realistic given that the device state transitions typically involve nontrivial costs and it is fairly difficult to predict when a running application will rerequest a specific device during *execution* [7], [10], [28].

The following parameters are associated with each device D_i:

- P_a^i. The device power consumption in active state.
- P_s^i. The device power consumption in sleep state.
- T_{sd}^i and T_{wu}^i. The device state transition times (from active to sleep, and from sleep to active, respectively).
- E_{sd}^i and E_{wu}^i. The device transition energy overheads (from active to sleep, and from sleep to active, respectively).

Given that devices are associated with nonzero transition costs, the device *break-even time* B_i denotes the minimum length of idle period which justifies a device transition from active to sleep state. Let $T_{sw}^i = T_{sd}^i + T_{wu}^i$. We denote by B_{actual}^i the minimum idle interval length during which keeping D_i in active state consumes the same amount of energy as transitioning D_i from active to sleep and back from sleep to active. Thus, $B_{actual}^i = \frac{E_{sd}^i + E_{wu}^i - T_{sw}^i \cdot P_s^i}{P_a^i - P_s^i}$. In other words, B_{actual}^i characterizes the minimum idle interval length for *energy-efficient device state transitions*. Further, the device idle interval should be long enough to allow the device transitions from active to sleep, and from sleep to active states, implying that device break-even times cannot be shorter than T_{sw}^i. Hence, the device break-even time B_i is given as $B_i = max(B_{actual}^i, T_{sw}^i)$, [6], [7]. In other words,

$$B_i = max\left(\frac{E_{sd}^i + E_{wu}^i - T_{sw}^i \cdot P_s^i}{P_a^i - P_s^i}, T_{sd}^i + T_{wu}^i\right).$$

We assume that due to periodic nature of real-time execution, devices cannot be completely turned off at runtime; but they can be put to low-power (sleep) states whenever possible. As a result, a device D_i will always consume power at the rate of at least P_s^i. Thus, for simplicity, all power consumption rates for a device D_i are given in excess of P_s^i in the rest of the paper. In other words, the following transformations are applied: $P_a^i = P_a^i - P_s^i, E_{sd}^i = E_{sd}^i - (P_s^i \cdot T_{sd}^i), E_{wu}^i = E_{wu}^i - (P_s^i \cdot T_{wu}^i)$, and $P_s^i = 0$. Notice that such a transformation does not change the original value of B_i.

Observe that the devices become idle at the end of task execution and remain so until the beginning of the next frame. In each frame, a device D_i can be transitioned to sleep state at the end of task execution, only if the slack $\delta(f) \geq B_i$, where f is the processor frequency at which the application is executed. Further, if the slack at the maximum frequency $\delta(f_{max})$ is smaller than B_i, then D_i will be forced to remain in active state throughout the frame (since lower frequencies can only reduce its slack time). Since this work explores the combined effects of DPM and VFS, we will assume that $\delta(f_{max}) \geq B_i$, for all devices. Note that if this condition is not satisfied for a given device D_i, then D_i cannot be managed and its energy consumption can be considered as part of the static energy. The framework of the paper is still applicable to the remaining devices.

2.3 Energy Model

Since the embedded application is invoked in periodic fashion, we concentrate on the energy consumption over a single frame. The system energy E can be divided into static energy (E_s) and dynamic energy ($E(f)$):

$$E = E_s + E(f).$$

The static energy E_s is due to the static power which is required for purposes such as keeping the system clock running, maintaining the basic circuits, and keeping the devices in sleep states. Since the static power can only be

eliminated by completely turning off the entire system, we assume that the static energy is not manageable [6], [35]. Hence, we focus on minimizing the dynamic energy consumption $E(f)$ which is a function of the processor frequency and includes the system components such as CPU, the main memory, and I/O devices.

At the end of task execution in each frame, depending on the system slack, the devices can either be transitioned to sleep state or kept in active state. Let $\mathcal{D}_A$ denote the set of devices kept in active state throughout the frame. Devices in $\mathcal{D}_S = \mathcal{D} - \mathcal{D}_A$ are transitioned to sleep state at the end of task execution. It is assumed that each device $D_j \in \mathcal{D}_S$ is reactivated T^j_{wu} time units before the start of the next frame, to allow timely execution. Also, let P_{ind} denote the total active power of the application's devices. We denote by P_{on} the total active power of devices in $\mathcal{D}_A$ and E_{tr} denotes the total transition energy of devices in $\mathcal{D}_S$. Formally, $P_{ind} = \sum_{i|D_i \in \mathcal{D}} P^i_a$, $P_{on} = \sum_{i|D_i \in \mathcal{D}_A} P^i_a$, and $E_{tr} = \sum_{i|D_i \in \mathcal{D}_S} (E^i_{sd} + E^i_{wu})$.

Given this notation, the dynamic system energy consumption at frequency f over the duration of a frame is given as

$$E(f) = (af^3 + P_{ind}) \cdot \left(\frac{x}{f} + y\right) + P_{on} \cdot \delta(f) + E_{tr}.$$

The processor power consumption is modeled as a convex function af^3, where "a" is the switching capacitance. At frequency f, the application executes for $(\frac{x}{f} + y)$ units, during which the processor consumes $af^3 \cdot (\frac{x}{f} + y)$ units of energy. $P_{ind} \cdot (\frac{x}{f} + y)$ represents the total device energy consumption during the execution of the application. $P_{on} \cdot \delta(f)$ represents the total energy consumed over the slack period by devices remaining in active states. E_{tr} represents the transition energy overhead for devices that are transitioned to sleep state during the slack period. We note that this component of the system energy representing device transition overheads was not considered in previous system-level energy management papers [1], [14], [36].

3 Single-Device Model

In this section, we consider a simplified model where the real-time application uses a single device. Using this simple model, we illustrate several nontrivial observations that lead to the characterization of the exact interplay between VFS and DPM. We also provide an $O(1)$ algorithm to calculate the frequency that optimizes system-wide energy while taking into account the VFS/DPM interplay and device transition overheads. In Section 4, we will extend these results to the general case with multiple devices.

The exact characterization of the trade-offs between VFS and DPM is critical for system-wide energy minimization. Consider a real-time application with WCET of $c = x + y$ units and frame length of d units, using a device D_0 with break-even time B_0. By adjusting the processor frequency, the completion time of the task can be anywhere from c to d (Fig. 1). This frequency assignment has obviously serious consequences for the applicability of DPM, and hence for overall system energy. Let f_a denote the minimum frequency at which the task can still meet its deadline, $f_a = \frac{x}{d-y}$ [3]. Further, let us denote the frequency which produces a slack of exactly B_0 units by $f^* = \frac{x}{d-B_0-y}$. Note that to transition D_0, the processor has to run the real-time frame-based application at a frequency no less than f^*.

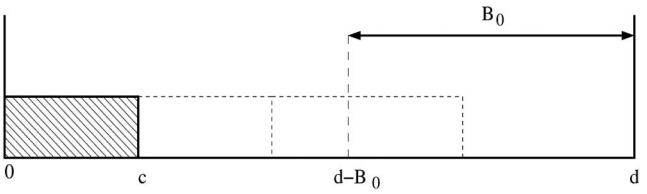

Fig. 1. The break-even time and the impact of VFS.

Fig. 2a shows the variations in device energy consumption (E_{device}) and CPU energy consumption (E_{cpu}) as a function of the processor frequency.[1] Note that E_{device} also includes device transition costs, when applicable. To start with, E_{cpu} increases with increasing frequency in a quadratic manner. However, E_{device} follows different patterns in two different regions. In Region A where $f_a \leq f < f^*$, the device D_0 cannot be transitioned (because the slack is smaller than B_0) and it is forced to remain in active state throughout the frame. As such, $E_{device} = P_a \cdot d$ is constant in Region A. In Region B where $f^* \leq f \leq f_{max}$, the device can be transitioned to sleep state. Further, as the frequency increases beyond f^* in Region B, the slack and hence, the length of the device sleep interval, increases. Thus, in Region B, the total device energy consumption during the execution of the application (E_{device}) decreases with increasing frequency.

Fig. 2b shows the variation of the system energy consumption, $E_{system} = E_{device} + E_{cpu}$, as a function of the frequency. We can see that E_{system} exhibits varying trends in Regions A and B. While E_{system} increases in Region A with increasing frequency, the local minimal of E_{system} in Region B can lie anywhere in the range $[f^*, f_{max} = 1]$. Also, as additional plots with dashed lines in Region B illustrate, the minimum value of E_{system} in that region can have quite different values.

It is worthwhile to compare these trends to the results of prior energy management studies. Region A is the spectrum where only the dynamic CPU power can be controlled. In fact, this was precisely the assumption of the early real-time VFS papers [2], [21], which effectively ignored Region B. Consequently, in Region A, the minimum frequency that guarantees system feasibility ($f = f_a$) is optimal. Region B is somewhat similar to the spectrum assumed by the recent system-level energy management papers [1], [14], [36], which considered the CPU and device energy figures at the same time. But, these papers neither accounted for energy transition overheads nor addressed the question of whether DPM is justifiable at runtime, given the length of *actual* idle intervals. As a result, these approaches ignored Region A. We can see that one really needs to consider *both* regions to analyze (and get full benefits of) VFS and DPM, simultaneously.

1. For the purpose of presentation, Fig. 2 is drawn assuming device break-even time $B = B_{actual}$. The formal analysis does not make such an assumption.

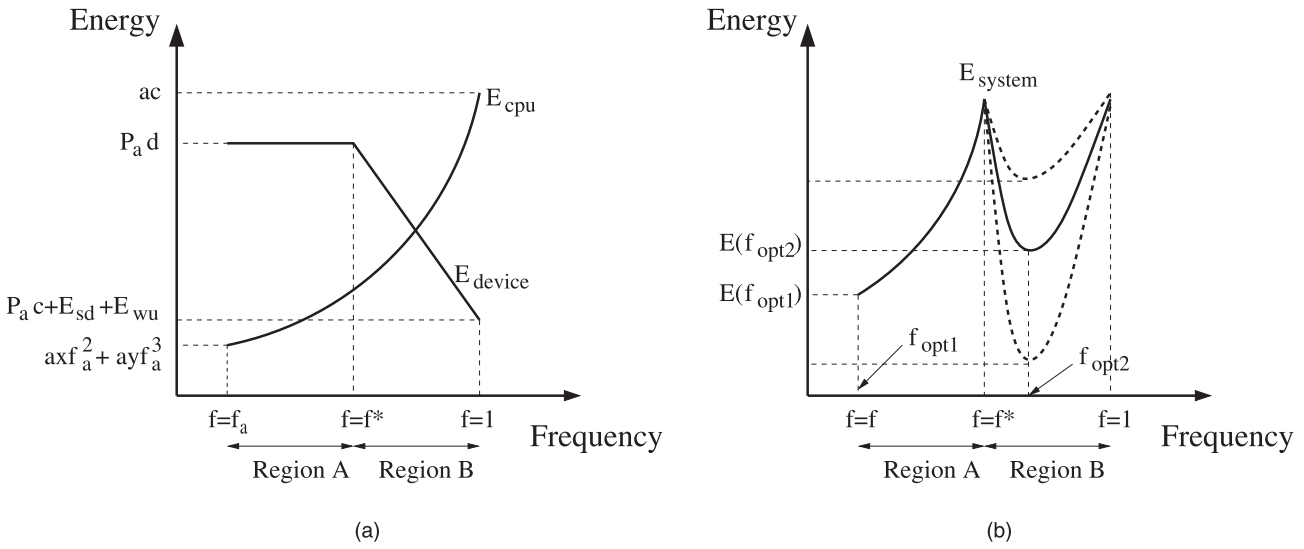

Fig. 2. Energy consumption as a function of the processor frequency (a) CPU and device energy (b) Total system energy.

The local optimal frequencies that minimize E_{system} in Regions A and B are well defined. However, *there is no a priori reason why global optimal frequency that minimizes E_{system} should lie in Region A or Region B*. Depending on the relative power consumption rates of the device and CPU, the execution time of the application, and the relative positions of f_a and f^*, the global optimal may be in either Region A or Region B. In fact, f_{opt2}, which optimizes $E_{cpu} + E_{device}$ *and* transitions the device to sleep state *(by incurring the transition energy)* may possibly consume more system-wide energy compared to the frequency f_{opt1}, which minimizes E_{cpu} and avoids device transition costs, without paying special attention to E_{device}. Consequently, exact evaluation and comparison of the local optimal values in Regions A and B are necessary in order to determine the global optimal.

3.1 System Energy Minimization in Region B

While the local optimal in Region A is straightforward to find, the one in Region B requires some elaboration. In Region B, all frequency values support energy-efficient device transitions and the device *will* be transitioned at the end of task execution. Thus, in Region B, the system energy consumption can be expressed as

$$\bar{E}(f) = \left(af^3 + P_a^0\right) \cdot \left(\frac{x}{f} + y\right) + \left(E_{sd}^0 + E_{wu}^0\right).$$

Observe that $\bar{E}(f)$ is a strictly convex function. $(E_{sd}^0 + E_{wu}^0)$ appears as a constant in $\bar{E}(f)$ and hence, the frequency f_{ee} that minimizes $\bar{E}(f)$ can be found by setting its derivative to zero. This gives the following *quartic* equation that can be solved analytically and in constant time [30]:

$$3a\frac{y}{x}f^4 + 2af^3 - P_a^0 = 0. \quad (1)$$

Further, through the *Descartes' Rule of Signs* [30], one can verify that the above equation has exactly one positive real root, which corresponds to the *energy-efficient frequency* value f_{ee}. This is, as expected, numerically equal to the energy-efficient frequency value given in [1], which solely focused on Region B, but without considering device transition energy and DPM-related issues.

Remark 1. An energy-efficient device state transition as assumed by the operation in Region B may not be possible by using the frequency f_{ee}, if f_{ee} lies outside the range $[f^*, f_{max}]$.

Remark 2. Even when f_{ee} is in the range $[f^*, f_{max}]$, in order to find the global optimal frequency, one still needs to compare the minimum energy consumption in Region B which *incurs a device transition overhead* against the minimum energy consumption in Region A which does *not* incur a transition overhead. This will be fully analyzed in Section 3.2.

Recall that a strictly convex function with one variable has a single global optimal and its second derivative is always positive. Hence, the convex nature of $\bar{E}(f)$ justifies the following two basic properties for any $\epsilon > 0$:

Property 1. $\forall f, f > f_{ee}, \bar{E}(f_{ee}) < \bar{E}(f) < \bar{E}(f + \epsilon)$.

Property 2. $\forall f, f < f_{ee}, \bar{E}(f_{ee}) < \bar{E}(f) < \bar{E}(f - \epsilon)$.

Let f_b denote the frequency that minimizes system energy in Region B. We determine f_b by considering three possible cases.

- **Case 1.** $f^* \leq f_{ee} \leq f_{max}$.
  In this case, D_0 can be transitioned to sleep state at $f = f_{ee}$ as $\delta(f_{ee}) \geq B_0$. Also, there is no other frequency which can transition D_0 *and* yield better system energy consumption in Region B. Thus, $f_b = f_{ee}$.
- **Case 2.** $f_{ee} > f_{max}$.
  From Property 2, the system energy in Region B is minimized when $f = f_{max}$. Thus, $f_b = f_{max}$.
- **Case 3.** $f_{ee} < f^*$.
  This implies $\delta(f_{ee}) < B_0$ and D_0 cannot be transitioned in energy-efficient fashion at $f = f_{ee}$. Thus, in an effort to transition the device, we have to increase frequency beyond f_{ee} and toward f^*, which

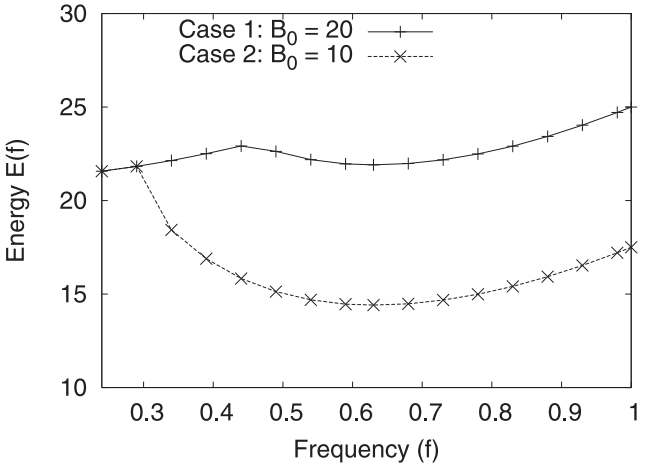

Fig. 3. Example illustrating the position of global optimal.

represents the first instance when the device can be transitioned. From Property 1, we can find that the system energy in Region B is minimized when $f = f^*$. Hence, $f_b = f^*$.

Based on the analysis above, we can write

$$f_b = max(f^*, min(f_{ee}, f_{max})).$$

Note that this formulation covers the three cases examined for energy minimization in Region B and also restricts f_b to the range $[f^*, f_{max}]$. Since $0 \leq B_0 < d$, we have $f^* \geq f_a$ and it follows that $f_b \geq f_a$. Thus, f_b preserves the system feasibility as well.

3.2 Finding the Global Optimal

In the preceding analysis, we showed that f_a and f_b are the local optimal values in Regions A and B, respectively. However, there is no a priori reason for global optimal frequency that minimizes system energy across Regions A and B to lie in one region as opposed to the other. The example below illustrates this fact.

Illustrative Example 1. Consider a real-time application with $c = 10$ and $d = 42$. For simplicity of illustration, we assume that the entire workload scales linearly with frequency (i.e., $x = c$ and $y = 0$). Let D_0 have the parameters $P_a^0 = 0.5$, $E_{sd}^0 = E_{wu}^0 = 5$, and $T_{sd}^0 = T_{wu}^0 = 10$. Assume switching capacitance $a = 1$. From the data, we can find that $B_0 = 20$ and $f_{ee} = 0.63$. Observe that $\delta(f_{ee}) > B_0$. Hence, it turns out that the device can be effectively put to sleep state and run at $f = f_{ee}$. However, we obtain $E(f_a = \frac{10}{42}) = 21.57$ and $E(f_b = f_{ee}) = 21.91$. *This shows that, for the given settings, despite the fact that the device can be transitioned at f_{ee}, from the system energy point of view it is better to keep D_0 in* active *state throughout the frame and run the application at $f = f_a$.* This is shown through Case 1 in Fig. 3. In the same example, by changing $E_{sd}^0 = E_{wu}^0 = 1.25$ and $T_{sd}^0 = T_{wu}^0 = 5$, we get $B_0 = 10$. With these new parameters, one can verify that $E(f_{ee}) < E(f_a)$ as shown in Case 2 of Fig. 3.

Thus, to determine the global optimal, it is essential to consider the local optimal values in both Regions A and B and compare them. Determining $E(f_a)$ and $E(f_b)$ and comparing them are all constant time operations. Hence, the overall complexity of the algorithm to determine the global optimal (the frequency that minimizes the total system energy) is $O(1)$.

A final observation is in order about the relative ordering of B_{actual}^0 and T_{sw}^0, whose maximum was defined as the break-even time B_0. If $B_0 = B_{actual}^0$, since $\delta(f^*) = B_0 = B_{actual}^0$, the following inequality holds:

$$E(f_a) \leq E(f^*) \leq E(f^* + \epsilon).$$

This implies that, if $f_b = f^*$, we know for sure $f = f_a$ is the global optimal and a comparison between $E(f_a)$ and $E(f_b = f^*)$ is not required. However, the above inequality may not hold when $B_0 = T_{sw}^0 > B_{actual}^0$. In this case, nothing can be said about the relative ordering of $E(f_a)$ and $E(f^*)$, as illustrated by the following example:

Illustrative Example 2. Let $c = x = 5$, $y = 0, d = 19$, $P_a^0 = 0.25$, $T_{sd}^0 = T_{wu}^0 = 5$, and $E_{sd}^0 = E_{wu}^0 = 0.625$. For the given data, $B_0 = T_{sw}^0 = 10$ and $f^* = \frac{5}{9}$. Assume $a = 1$. We can verify that $E(f^*) = 5.04 < E(f_a) = 5.096$. By setting $E_{sd}^0 = E_{wu}^0 = 1$, we can verify that B_0 and f^* still remain the same. However, in these new settings, $E(f^*) > E(f_a)$.

3.3 Experimental Evaluation

In this section, we perform an experimental evaluation using the actual device specifications taken from [7]. The CPU power consumption rate at the maximum processing frequency is modeled after Intel XScale [33]. We consider a real-time application with a frame length of 44 ms. The application is assumed to use the device IBM Microdrive during its execution. Based on the device characteristics of IBM Microdrive [7], its break-even time can be computed as 24 ms. Observe that if the worst-case execution time c of the real-time application at the maximum frequency is greater than 20 ms, then the device can never be transitioned to sleep state as there is not enough slack to justify it. Thus, when $c > 20$, the problem of system-wide energy minimization reduces to minimizing CPU energy only and running the CPU at $f = f_a$ is optimal. Hence, we only vary c from 2 to 20 ms in steps of 2 ms. For each distinct c value, we compare three schemes:

- *OPT*. Optimal scheme from Section 3.2.
- *Aggressive slow-down (AG-SD)*. This scheme runs the processor at the lowest frequency $f_a = \frac{x}{d-y}$ that can still meet the deadline [1], [3]. With *AG-SD*, the devices are never transitioned to sleep state. The frequency f_a minimizes the *CPU dynamic energy consumption only* [2], [21].
- *Device-aware slow-down (DA-SD)*. This scheme is based on the concept of energy-efficient frequency [1], [14], [36] and is adopted from [1], where an optimal solution to the system-wide energy minimization problem is developed, but by *ignoring* the DPM dimension. The energy-efficient frequency (denoted by f_{ee}) is computed as the frequency that minimizes the system energy, by considering only CPU energy and device active energy consumption. Since f_{ee} can be less than the system utilization, to preserve the feasibility, we execute the task at $f = max(f_a, f_{ee})$. If the device can be transitioned at $f = f_{ee}$ we do so; else, the device remains in active state throughout the frame.

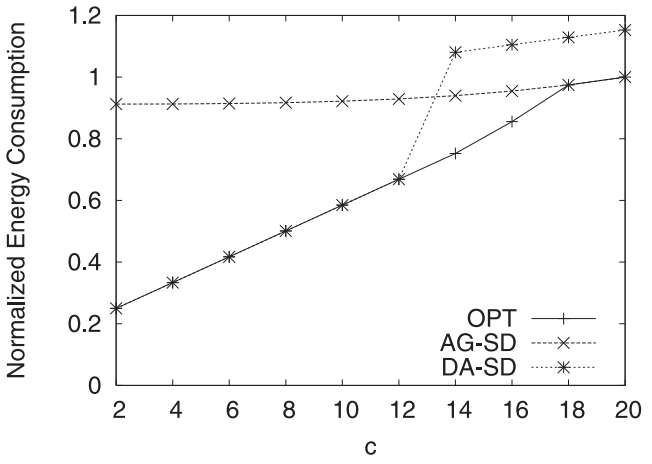

Fig. 4. Impact of the worst-case execution time c.

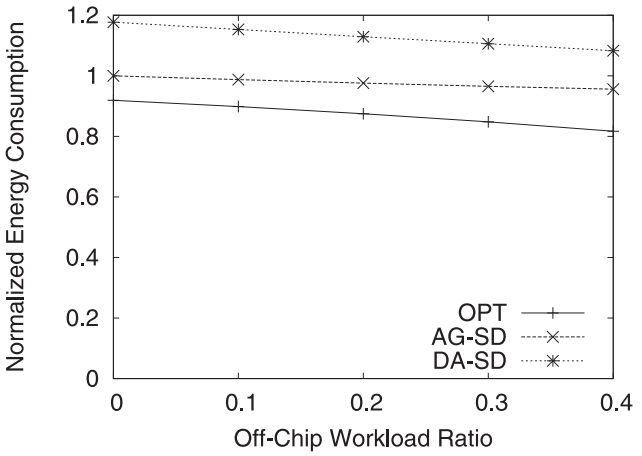

Fig. 5. Impact of the off-chip workload ratio $\frac{y}{c}$.

First, we consider the effect of varying worst-case execution time (Fig. 4). The off-chip workload ratio ($\frac{y}{c}$) is set to 0.2. The values are normalized with respect to *AG-SD* when $c = 20$. For $c \leq 18$, the system energy benefits from running the processor at frequencies higher than f_a since the gain in device energy consumption overshadows the loss in CPU energy. Hence, $f = f_b$ is optimal in this region. On the other hand, for $c > 18$, at high frequencies the loss in CPU energy starts to overshadow the gain in device energy. Consequently, running at frequencies higher than f_a starts to hurt system energy and $f = f_a$ is the optimal in this spectrum.

It can be seen that in the interval where $c \leq 12$, *OPT* follows *DA-SD* while for $c \geq 18$, *OPT* follows *AG-SD*. However, for $12 < c < 18$, it can be seen that *OPT* follows neither *AG-SD* nor *DA-SD*. During this period, $f_b = f^*$ and *OPT* represents $E(f^*)$ which is optimal in this spectrum. Notice that for this example, the optimal frequency that minimizes the system-wide energy, changes from $f_{opt} = f_{ee}$ to $f_{opt} = f^*$ and finally to $f_{opt} = f_a$ with increasing utilization values. As f_{opt} transitions from f_{ee} to f^*, the device can no longer be put to sleep state at $f = f_{ee}$. Thus, running at $f = f_{ee}$ consumes more system energy compared to $f = f_a$, explaining the sharp increase in *DA-SD* at $c = 14$. The energy optimal scheme *OPT* avoids the suboptimal performances of *AG-SD* and *DA-SD* at low and high utilization values, respectively.

We emphasize that there is an interval $[12, 18]$ where f_{opt} is neither f_a nor f_{ee}, but f^*, in the above results. Thus, the optimal scheme (*OPT*) is more than just determining at every point the better performing frequency in the set $\{f_a, f_{ee}\}$. In other words, there are regions where both of these well-known frequencies fail to minimize the system-wide energy consumption as in these regions, the optimal frequency f^* differs from both f_a and f_{ee}.

Fig. 5 shows the impact of varying off-chip workload ratio ($\frac{y}{c}$). In this experiment, the worst-case execution time of the application is set to 16 ms (i.e., $c = 16$ ms). The relative performance of the schemes is not heavily impacted by variations in off-chip workload ratio and *OPT* outperforms both *AG-SD* and *DA-SD* throughout the spectrum. With increasing off-chip workload ratio, both f_{ee} and f_a decrease [1], [3], since there are additional opportunities for VFS to further reduce the energy consumption. Thus, the normalized energy consumption values of schemes decrease with increasing off-chip workload ratio.

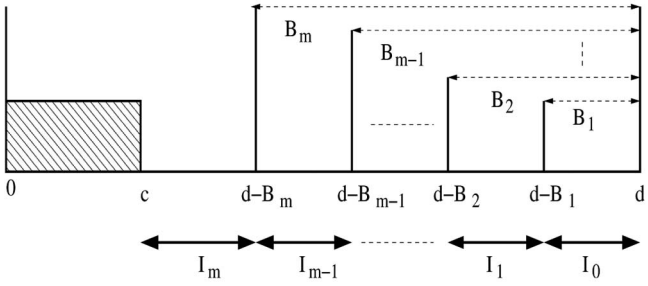

Fig. 6. The ordering of the break-even times.

4 Multiple-Device Model

In this section, we generalize our solution to the case of multiple devices. The real-time application is assumed to use m different devices $\{D_1 \dots D_m\}$ during the span of its execution. Each device D_i has its own parameters (P_a^i, P_s^i, E_{sd}^i, E_{wu}^i, T_{sd}^i, and T_{wu}^i) and is associated with a break-even time denoted by B_i. First, we formally define the problem.

Problem statement. *Given a frame-based real-time application using m different devices, determine the CPU frequency and device transitioning decisions so as to minimize the system-wide energy consumption.*

Since each device can be put to sleep state at the end of the execution, or remain in active state until the end of the frame, at first, it seems that there are 2^m possibilities that need to be examined. If true, this would imply an exponential-time algorithm. By careful analysis, we establish some important properties of the optimal solution, which enables us to develop an $O(m \log m)$-time algorithm.

Let R_{opt} denote the response time of the real-time application in the optimal solution. We know that $R_{opt} \in [c, d]$. Without loss of generality, the break-even times are arranged in nondecreasing order, i.e., $0 < B_1 < \dots < B_m < d - c$. With this ordering, we can divide $[c, d]$ into $m + 1$ intervals $\{[c, (d - B_m)] \dots [(d - B_{i+1}), (d - B_i)] \dots [(d - B_1), d]\}$ denoted by $\{I_m \dots I_0\}$, respectively (Fig. 6). If two devices have the same break-even time, $B_m = d - c$, or $B_1 = 0$, then we will have less than $m + 1$ intervals. The same analysis can then be performed on this reduced interval set.

For convenience, we divide our analysis into two steps that eventually will lead to an $O(m \log m)$-time algorithm.

- *Step*1. For each of these intervals, *assuming that R_{opt} lies in that interval*, we determine the set of devices that will be transitioned at the end of task execution which helps to evaluate the exact system-wide energy consumption function. Once we have the exact form of system-wide energy in an interval, we show how it can be minimized.
- *Step*2. Based on the analysis in *Step*1, we narrow down our analysis to at most $m+2$ cases that have to be examined. By comparing the energy consumption figures of these $m+2$ cases, we determine the optimal solution.

We now elaborate on these two steps and present our full analysis. By ordering the devices in nondecreasing order of break-even times, *Step*1 can be addressed as follows: if R_{opt} belongs to interval I_i, then devices $\{D_{i+1} \dots D_m\}$ cannot be transitioned as the idle time is smaller than their break-even times. However, all devices $\{D_1 \dots D_i\}$ *can and should be transitioned*. This is because if any device in the set $\{D_1 \dots D_i\}$ is not transitioned to sleep state at the end of task execution, then by transitioning that device, we would effectively reduce device energy consumption and hence obtain a schedule with reduced system energy consumption. Based on this, one can infer that if $R_{opt} \in I_m$, all m devices will be transitioned. Similarly, if $R_{opt} \in I_0$, then none of the devices will be transitioned.

Based on the interval to which R_{opt} belongs, we can characterize the devices that should be transitioned to sleep states at the end of task execution. With this information, we can characterize the system energy consumption function when $R_{opt} \in I_i$. *The number of devices transitioned to* sleep *states and hence the exact form of the system energy consumption remains the same as R_{opt} varies within a given interval and changes only when R_{opt} transitions between intervals*. Let $E_i(f)$ represent the system energy consumption when $R_{opt} \in I_i$. Specifically,

$$E_i(f) = \left(af^3 + \sum_{j=1}^{i} P_a^j\right)\left(\frac{x}{f} + y\right) + \sum_{j=i+1}^{m} P_a^j d + \sum_{j=1}^{i} \left(E_{sd}^j + E_{wu}^j\right).$$

Notice that in the formulation of $E_i(f)$, devices $\{D_1 \dots D_i\}$ are transitioned while devices $\{D_{i+1} \dots D_m\}$ are kept in active state throughout the frame. For uniformity, let us denote the lower and upper limits of interval I_i by LL_i and UL_i, respectively. That is, $LL_0 = d - B_1$, $UL_0 = d$, $LL_m = c$, $UL_m = d - B_m$, and $LL_i = d - B_{i+1}$; $UL_i = d - B_i$, $(i = 1 \dots m-1)$. We can formalize the problem of minimization of E_i by enforcing that the response time of the task falls in interval I_i. This leads to the following constrained convex optimization problem for I_i, denoted by OPT_i:

$$\text{minimize } E_i(f) \tag{2}$$

$$\text{subject to } -\frac{x}{f} - y + LL_i \le 0 \tag{3}$$

$$\frac{x}{f} + y - UL_i \le 0. \tag{4}$$

Constraints (3) and (4) make sure that the response time of the application does not fall outside the range of interval I_i to which R_{opt} is assumed to belong.

Proposition 1. *If the frequency f is the solution to the optimization problem OPT_i, then $f_a \le f \le f_{max}$.*

Proof. If $f > f_{max}$, from both (3) and (4) it follows that the response time at $f > f_{max}$ is in the range $[LL_i, UL_i]$. Since $c = LL_m$, this implies $\frac{x}{f_{max}+\epsilon} + y \ge c = \frac{x}{f_{max}} + y$, which is a contradiction.

Similarly, if $f < f_a$, from both (3) and (4) it follows that the response time at $f < f_a$ is in the range $[LL_i, UL_i]$. Now, since $UL_0 = d$, this implies $\frac{x}{f_a-\epsilon} + y \le d = \frac{x}{f_a} + y$, which is again a contradiction. □

Let f_i be the value that sets the derivative of $E_i(f)$ to zero. Thus, f_i is the unique positive real root of the following quartic equation:

$$3a\frac{y}{x}f^4 + 2af^3 - \sum_{j=1}^{i} P_a^j = 0.$$

Lemma 1. *If f_i satisfies conditions (3) and (4), then it is the solution to OPT_i. Else, in the solution to OPT_i either $f = \frac{x}{LL_i - y}$ or $f = \frac{x}{UL_i - y}$.*

Proof. By definition, the response time of the application in OPT_i must be in interval I_i (i.e., in the interval $[LL_i, UL_i]$). As a consequence, it follows that the frequency at which the application is executed is in the range $[\frac{x}{LL_i-y}, \frac{x}{UL_i-y}]$. In other words, if $f \in [\frac{x}{LL_i-y}, \frac{x}{UL_i-y}]$, then conditions (3) and (4) will be satisfied.

Since $E_i(f)$ is strictly convex, it is minimized at f_i. Thus, if $f_i \in [\frac{x}{LL_i-y}, \frac{x}{UL_i-y}]$, then it is the solution to OPT_i. On the other hand, if $f_i \notin [\frac{x}{LL_i-y}, \frac{x}{UL_i-y}]$, then due to convexity of $E_i(f)$ either $f = \frac{x}{LL_i-y}$ or $f = \frac{x}{UL_i-y}$ is the solution to OPT_i [19]. □

Observe that when $I_i = I_m$, f_m is the unique positive root of the equation $3a\frac{y}{x}f^4 + 2af^3 - \sum_{i=1}^{m} P_a^i = 0$. Thus, $f_m = f_{ee}$, the traditional energy-efficient frequency for a task using m devices *derived by ignoring DPM issues* [1]. Assuming that f_{ee} satisfies the response time constraints for interval I_m, an interesting observation at this point is that f_{ee} is *only* the local optimal solution for the interval I_m.

While Lemma 1 solves *Step*1 of the analysis, the following corollary connects *Step*1 and *Step*2.

Corollary 1. *If the response time under frequency f_i lies outside the interval I_i ($\forall i = 0 \dots m$), then in the optimal solution, $R_{opt} \in \{c, (d - B_m), \dots, (d - B_1), d\}$.*

Corollary 1 states that if for all the $(m+1)$ intervals, I_i, f_i do not satisfy the conditions (3) and (4) of the optimization problem OPT_i, then in the optimal solution, the response time of the application is limited to the set $\{c, (d - B_m), \dots, (d - B_1), d\}$. *That is, if the given conditions hold, in the optimal solution, the slack of the application should be exactly equal to 0, $d - c$, or one of the break-even times $\{B_i\}$.*

If $R_{opt} = d - B_i$, technically it falls in two intervals and one may tend to think that there is a need for evaluating two cases: one in which D_i is not transitioned (as part of interval I_{i-1}) and another in which D_i is transitioned (as part of interval I_i). However, as the following observation states, one of these possibilities is never worse than the other.

Observation 1. *If $R_{opt} = d - B_i$, then the device D_i can be transitioned without increasing overall energy consumption.*

Observation 1 follows from the fact that B_i is defined as $max\{B^i_{actual}, T^i_{sw}\}$. If $B_i = B^i_{actual} > T^i_{sw}$, then by definition of B^i_{actual}, transitioning or not transitioning the device results in the same energy consumption when the application completes at $t = d - B_i = d - B^i_{actual}$. On the other hand, if $B_i = T^i_{sw} > B^i_{actual}$, and the application completes at $t = d - B_i$, it leaves a slack strictly larger than B^i_{actual} and transitioning the device reduces the energy consumption. Hence, in either case, the device D_i can be transitioned without increasing the energy consumption when $R_{opt} = d - B_i$.

Observation 1 implies that there are at most $m + 2$ cases that need to be examined to determine the optimal solution. Let EC_{opt} denote the set of energy consumption values obtained by evaluating the final $m + 2$ cases. An interesting question is whether there exists a pattern among these final $m + 2$ cases that can be further exploited by convex optimization techniques. Unfortunately, the answer is negative.

Observation 2. *The relative ordering of the $(m + 2)$ values in the set EC_{opt} does not exhibit a special pattern.*

We give an example to justify the above observation.

Illustrative Example 3. Consider a real-time application with $c = x = 10$, $y = 0$, and $d = 30$. The application uses four devices $D_1(P^1_a = 0.2, B_1 = 5)$, $D_2(P^2_a = 0.15, B_2 = 10)$, $D_3(P^3_a = 0.5, B_3 = 15)$, and $D_4(P^4_a = 0.4, B_4 = 17)$. Let us assume $a = 1$ and $T_{sw} \leq B^i_{actual}$ for all devices. $B^i_{actual} = \frac{E^i_{sd}+E^i_{wu}}{P^i_a}, i = 1 \ldots 4$. In these settings, we can verify that $f_4 = 0.855, f_3 = 0.752, f_2 = 0.559$, and $f_1 = 0.464$. Notice that $\forall i, \frac{c}{f_i} \in I_i$. In interval $I_0, f = f_a$ is the best as no devices are transitioned to sleep states. With the above data, we can verify $E(f = f_a) = 38.611$, $E(f = f_1) = 38.963$, $E(f = f_2) = 38.886$, $E(f = f_3) = 38.958$, and $E(f = f_4) = 38.730$.

Notice how the interval-optimal energy consumption $E_i(f)$ first increases, next decreases, then increases before decreasing once again, as we move from the first candidate frequency f_1 to f_2, f_3, and f_4. This shows that the optimal energy consumption values of the final $m + 2$ cases, need not to have a well-defined relationship (such as convexity) which can be exploited by optimization techniques. As an implication, it turns out that it is indeed necessary to evaluate and compare the $m + 2$ candidate cases for the optimal solution.

4.1 Computing the Optimal Frequency Efficiently

Based on the above characterizations, we formulate an $O(m \log m)$ algorithm, given in Fig. 7, to find the optimal frequency for the multiple-device model. As an implication of Observation 2, it is necessary to compare the best energy consumptions obtained by assuming $R_{opt} \in I_i, i = 0 \ldots m$ to obtain the global optimal. From Lemma 1 and Observation 1, in every interval $I_i, i \neq m$, if $f = f_i$ does not satisfy the response time constraints, then it is sufficient to evaluate and compare energy consumption at $f = \frac{x}{UL_i - y}$. We start out with the assumption that the optimal solution is in I_0 and $f = f_a$ minimizes system energy (lines 4-6). The E_{best} variable holds the minimum system energy consumption value encountered so far and the f_{best} holds the corresponding frequency. In lines 7-19, we consider cases where the optimal response time of the application is assumed to belong to each of the remaining m intervals $I_1 \ldots I_m$. For each such interval, we compute f_i. Based on whether or not f_i satisfies the response time constraints, we compare energy consumption at either $f = f_i$ or $f = \frac{x}{UL_i - y}$ with E_{best}. For interval I_m, if f_m does not satisfy the response time constraints, then it is necessary to evaluate and compare energy consumption at $LL_m = c$, as c does not act as an upper limit to any interval. In lines 20-24, we perform this final comparison. At the end, f_{best} holds the optimal value of f that minimizes system energy consumption.

Function *Optimal frequency:*

```
1   P_ON = Σ_{i=1}^{m} P_a^i
2   P_OFF = 0
3   E_T = 0
4   E_0 = a x f_a^2 + a y f_a^3 + P_ON · d
5   E_best = E_0
6   f_best = f_a
7   for i = 1 to m
8       P_ON = P_ON − P_a^i
9       P_OFF = P_OFF + P_a^i
10      E_T = E_T + E_sd^i + E_wu^i
11      Set f_i to root of 3a(y/x)f^4 + 2af^3 − P_OFF = 0
12      if (x/f_i + y ∈ I_i) then f = f_i
13      else f = x/(UL_i − y) = x/(d − B_i − y)
14      E_i = (af^3 + P_OFF)(x/f + y) + P_ON · d + E_T
15      if (E_i < E_best)
16          E_best = E_i
17          f_best = f
18      end if
19  end for
20  E_{m+1} = (af^3 + P_OFF)(x/f + y) + P_ON · d + E_T
21  if (x/f_m + y ∉ I_m and E_{m+1} < E_best)
22      E_best = E_{m+1}
23      f_best = f_max
24  end if
25  return f_best
```

Fig. 7. Algorithm to compute the optimal frequency (multiple-device case).

Time complexity. Sorting the devices based on break-even times requires $O(m \log m)$ time. The algorithm performs a constant time comparison in every interval and there are at most $m + 1$ intervals. Thus, the time complexity of the algorithm is $O(m \log m)$. Since the number of devices in a system is typically small, this offline algorithm can be considered efficient.

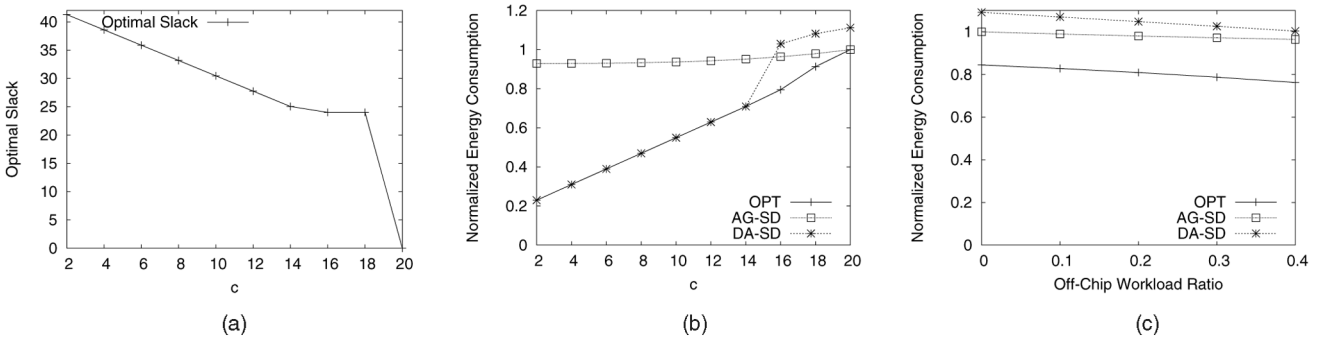

Fig. 8. Experimental evaluation for multiple-device model. (a) Optimal slack as a function of c. (b) Relative performance of schemes as a function of c. (c) Relative performance of schemes as a function of $\frac{y}{c}$.

4.2 Experimental Evaluation

The experimental methodology we follow in this section is an extension of the one described in Section 3.3. Again, the real-time application has a frame length of 44 ms. It uses three devices during its execution: IBM Microdrive ($B = 24$ ms), Realtek Ethernet Chip ($B = 20$ ms), and Simple Tech Flash Card ($B = 4$ ms). The specifications of these devices are from [7].

We consider the effect of worst-case execution time on both optimal system slack and optimal system energy consumption (Figs. 8a and 8b). The off-chip workload ratio ($\frac{y}{c}$) is set to 0.2. Fig. 8a shows the optimal slack ($d - R_{opt}$) which minimizes system-wide energy as a function of c. Fig. 8b shows the relative performance of the three schemes. The energy values are normalized with respect to *AG-SD* when $c = 20$ ms. The optimal slack decreases uniformly with increasing utilization in the range $2 \leq c \leq 14$. In this interval, $f = f_{ee}$ is optimal, and the *OPT* scheme follows *DA-SD*. For values in the range $14 < c < 20$, *OPT* is significantly different from both *DA-SD* and *AG-SD*. During this period, one or more devices cannot be transitioned at $f = f_{ee}$, which explains the sharp increase in *DA-SD* at $c = 16$. The step-like behavior of the optimal slack is also a consequence of the optimal frequency shifting from $f = f_{ee}$ to an intermediate value between f_a and f_{ee}. Note that depending on the power characteristics of devices and frame length, the sharp increase in *DA-SD* scheme may also occur at an earlier stage than the one shown in figure. In such cases, the advantages of our optimal scheme are even more pronounced.

In models minimizing the system-wide energy *while ignoring device transition overheads and DPM-related issues*, *DA-SD* scheme was shown to be optimal assuming it satisfies feasibility constraints [1]. Observe that *AG-SD* outperforms *DA-SD* in the spectrum $c \geq 16$ in Fig. 8b. Due to device transition overheads, as mentioned before, transitioning devices at $f = f_{ee}$ is not always possible. When $c = 16$, IBM Microdrive cannot be transitioned at f_{ee} and remains active throughout the frame significantly increasing device energy consumption. The CPU power consumption rate is significantly high compared to that of Flash Card and Ethernet Chip. Thus, with IBM Microdrive in active state throughout the frame, the CPU energy savings in scheme *AG-SD* dominate the device energy saving obtained by transitioning Flash Card and Ethernet Chip in *DA-SD*. This explains the reason why *AG-SD* outperforms *DA-SD*.

Finally, at $c = 20$, *OPT* follows *AG-SD*. Observe that for the devices considered $T^i_{sw} > B^i_{actual}$. As a result, the device break-even time, defined as $max(T^i_{sw}, B^i_{actual})$, is dominated by the device transition times. Thus, even at $c = 20$, the system has enough slack to *potentially* transition all three devices energy efficiently. However, when $c = 20$, there is no device transitioning decision, involving transitioning at least one device to sleep state, which can reduce device energy consumption to an extent that it overshadows the increase in CPU energy by running the processor at frequencies higher than $f = f_a$. Thus, *AG-SD* is optimal at $c = 20$.

Fig. 8c shows the impact of off-chip workload ratio with c set to 16 ms. The findings are similar to those for single device. Also, the relative gains of *OPT* seem to get better only marginally with increasing off-chip workload ratio showing that variability in off-chip workload ratio has minimal impact on relative performance of the schemes.

While the above experiments were based on device/processor parameters taken from [7] and [33], in the following experiments, we will analyze the sensitivity of the results with respect to power characteristics of the system components by scaling up and scaling down the device/processor parameter values given in [7] and [33]. Fig. 9 shows the impact of varying device, processor, and application characteristics. In these experiments, $c = 16$ ms and off-chip workload ratio is set to 0.2.

Figs. 9a and 9b show the impact of varying device and processor power characteristics, respectively. In Fig. 9a, we multiply the active power of all devices by a certain scaling factor and recompute device break-even times while keeping processor characteristics the same. On the contrary, in Fig. 9b, we multiply processor power consumption at the maximum frequency by a certain scaling factor while keeping device characteristics the same. At each scaling point, we evaluate the system energy consumption of all three schemes. All energy values in Figs. 9a and 9b are normalized with respect to scaling factor of one (i.e., the original device/processor parameters).

In Figs. 9a and 9b, it is worth observing that there is a well-defined region where *OPT*'s energy savings differ from those of *AG-SD* and *DA-SD*. At lower P_a scaling factors and higher P_{cpu} scaling factors, the processor energy consumption overshadows device energy consumption and dominates system energy. As such, in these regions, *AG-SD* outperforms *DA-SD*. On the contrary, at higher P_a scaling factors and lower P_{cpu} scaling factors device energy consumption is

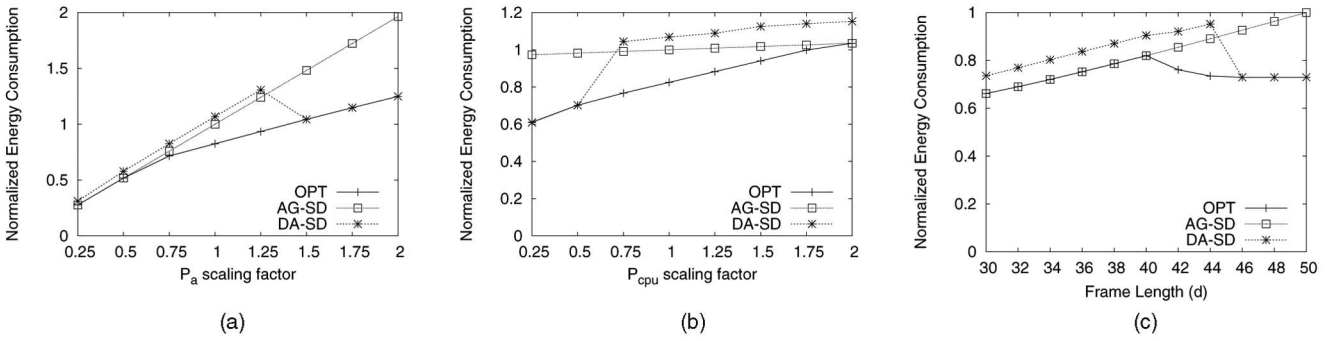

Fig. 9. Impact of variations in device, processor, and application characteristics. (a) Impact of device power scaling. (b) Impact of processor power scaling. (c) Impact of frame length.

dominant resulting in *DA-SD* outperforming *AG-SD*. Thus, *DA-SD* suffers from performance degradation in settings where CPU energy dominates, while in settings where device energy dominates the performance of *AG-SD* significantly degrades. As is evident in Figs. 9a and 9b, *OPT maintains a robust performance at all scaling factors* and is not susceptible to performance degradation due to changes in power characteristics of system components.

Fig. 9c shows the impact of varying the frame length (d). d is varied between 30 and 50 ms in steps of 2 ms. The energy values are normalized with respect to that of *AG-SD* at $d = 50$. With increasing values of d, the energy consumption of *AG-SD* increases as devices are forced to remain in active state for prolonged periods of time. For the same reason in the range $30 \leq d < 46$, the energy consumption of *DA-SD* increases with increasing values of d. In this range, there is not enough slack to create an effective and energy-efficient device transition with which *DA-SD* can outperform *AG-SD*. However, when $d = 46$, such a transition does occur and hence *DA-SD* outperforms *AG-SD* in the range $46 \leq d \leq 50$. Notice that in the region $40 < d < 46$, *OPT* differs from both *AG-SD* and *DA-SD*. Finally, the sudden decrease in energy consumption of *OPT* at $d = 42$ is due to the fact that the frame length becomes large enough to allow for transitioning *additional* devices energy efficiently.

Before concluding this section, we note that additional experimental results exploring a larger parameter space are presented in our technical report [9].

5 Workload Variability

While Section 4 has developed a provably optimal solution to minimize the system energy consumption for a *deterministic* workload by assuming worst-case execution behavior, frequently, the actual workload in real-time applications exhibits significant variability [11]. In fact, exploiting workload variability to minimize CPU energy through *reclaiming* is a heavily explored problem in VFS research [2], [21], [24]. In the presence of such variability in runtime execution behavior, the optimal solution derived in Section 4 becomes suboptimal and pessimistic. In variable workload settings, even though the application's *actual* workload may not be known precisely in advance, some stochastic information about the runtime execution behavior of the application can be determined [2], [5], [16], [12], [18], [33]. In this section, following [2] and [16], we show how information about the *average-case* execution behavior of the application, if known, can be used to extend our framework to minimize the *average-case system energy* consumption, while still providing deterministic guarantees to meet the deadline.

First, we give a motivational example to illustrate the fact that the knowledge about *average-case* execution time can provide important opportunities for system energy management. Consider a real-time application with $c = 10$, $d = 15$ and using device D_1 with the following parameters: $T_{sd} = T_{wu} = 4$, $E_{sd} = E_{wu} = 1$, and $P_a = 2$. From the data, we can derive $B_1 = 8$. Assume switching capacitance $a = 1$. Assuming the worst-case workload c, D_1 cannot be transitioned and hence the optimal frequency is $f_{wc} = \frac{c}{d} = \frac{2}{3}$. Now assume we have information that the average-case execution time of the application is six. Using this, one can verify that the *average-case energy consumption* is minimized at $f_{exp} = 1$. If the actual execution time of the application is 7, then it can be verified that $E(f_{exp}) = 23$ which is less than $E(f_{wc}) = 33.14$ by a margin of 30 percent.

Before proceeding, we underline that the hard deadline constraint will impose an absolute lower bound on CPU frequency, occasionally limiting the efficacy of energy optimization with average-case execution time. We now give the details of our approach.

Let $c' = x' + y'$ denote the average-case execution time of the real-time application (under maximum frequency), where x' and y' are the on-chip and off-chip components of the workload, respectively. Let R_{exp} represent the response time of the application in the solution that minimizes the average-case energy. Thus, by definition, we have $R_{exp} \in [c', d]$. As in Section 4, the interval $[c', d]$ can be divided into $m + 1$ intervals $I_0 \ldots I_m$. All interval values are the same as in Section 4 except for the lower limit of I_m which is c' as opposed to c (the worst-case execution time). Thus, R_{exp} belongs to one of the $m + 1$ intervals $I_0 \ldots I_m$.

Further, following the same reasoning as in Section 4, we can see that if $R_{exp} \in I_i$, in the solution that minimizes the average-case energy, all devices $\{D_1 \ldots D_i\}$ *can and must be transitioned* to sleep state. On the other hand, devices $\{D_{i+1} \ldots D_m\}$ cannot be transitioned in energy-efficient fashion and will remain in active state over the interval $[0, d]$. As such, if $R_{exp} \in I_i$, then the average-case system-wide energy consumption $E_i'(f)$ is given by

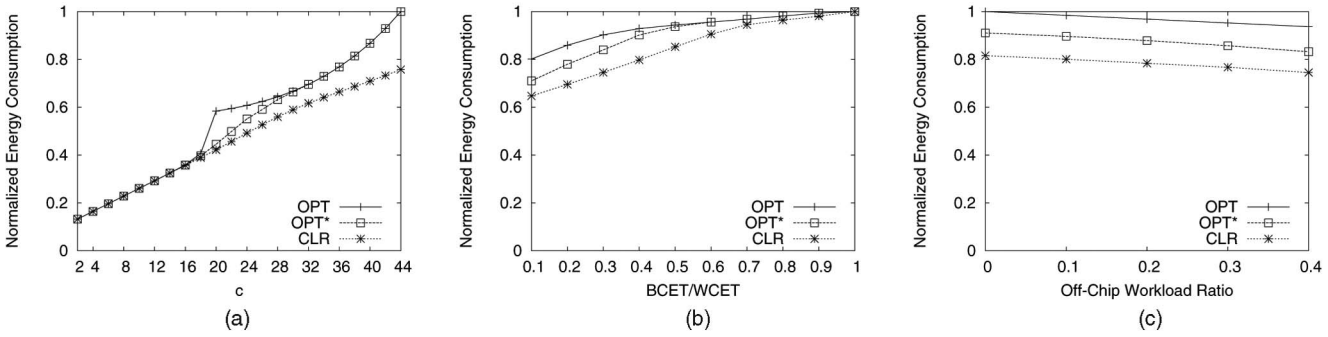

Fig. 10. Experimental evaluation under dynamic workload variability. (a) Relative performance of schemes as a function of c. (b) Relative performance of schemes as a function of $\frac{BCET}{WCET}$. (c) Relative performance of schemes as a function of $\frac{y}{c}$.

$$E_i'(f) = \left(af^3 + \sum_{j=1}^{i} P_a^j\right)\left(\frac{x'}{f} + y'\right) + \sum_{j=i+1}^{m} P_a^j d + \sum_{j=1}^{i} \left(E_{sd}^j + E_{wu}^j\right).$$

Now, we can again construct $(m+1)$ constrained optimization problems by enforcing $R_{exp} \in I_i\ i = 0 \ldots m$. Formally, we define the constrained optimization problem OPT_i^* as

$$\text{minimize } E_i'(f) \tag{5}$$

$$\text{subject to } -\frac{x'}{f} - y' + LL_i' \leq 0 \tag{6}$$

$$\frac{x'}{f} + y' - UL_i' \leq 0 \tag{7}$$

$$f_a \leq f \leq f_{max}. \tag{8}$$

LL_i' and UL_i' are the lower and upper limits of interval I_i, respectively. Recall from Section 3 that $f_a = \frac{x}{d-y}$ represents the minimum frequency which guarantees feasibility under worst-case execution workload. Though we are minimizing the average-case system energy, it is still important to meet the application deadline under worst-case execution behavior. The constraint (8) in OPT_i^* helps to meet that objective.

Let F_i denote the solution to OPT_i^*. Further, let F_i' denote the solution to the same optimization problem, but without the constraint (8). By repeating the analysis in Section 4, one can verify Propositions 2 and 4, below. Proposition 3 is a consequence of convexity.

Proposition 2. *$F_i' = f_i$ if $\frac{x'}{f_i} + y' \in I_i$ where f_i is the unique positive root of $3a\frac{y'}{x'}f^4 + 2af^3 - \sum_{j=1}^{i} P_a^j = 0$. Otherwise, either $F_i' = \frac{x'}{UL_i' - y'}$ or $F_i' = \frac{x'}{LL_i' - y'}$.*

Proposition 3. *If $f_a \leq F_i' \leq f_{max}$, then $F_i = F_i'$. Otherwise, if $F_i' < f_a$, then $F_i = f_a$ else $F_i = f_{max}$.*

Proposition 4. *A solution to the problem of minimizing the average-case energy can be obtained in $O(m \log m)$ time by comparing the OPT_i^* solutions (F_i), $i = 0 \ldots m$.*

5.1 Experimental Evaluation

To evaluate the performance of the new scheme under workload variability, we conducted a series of experiments. As in Section 4.2, we consider a real-time application with a frame length of 44 ms, using three devices: IBM Microdrive, RealTek Ethernet Chip, and Simple Tech Flash Card. We determine the actual execution time of the application using normal distribution with mean $\frac{BCET+WCET}{2}$ and standard deviation $\rho_0 = \frac{WCET-BCET}{6}$, where BCET and WCET are the best-case and worst-case execution times of the application under maximum frequency, respectively. The specific mean and standard deviation values coincide with those used in [2], [12], [15], [16], [26], and guarantee that 99.7 percent of the execution times fall in the range $[BCET, WCET]$.[2] The results presented are the average from 1,000,000 experiments. We compare three schemes:

- OPT. It is the optimal solution from Section 4 assuming worst-case execution behavior. However, when the application completes in a given frame, the amount of actual slack until the next invocation is computed and all devices that can be transitioned in energy-efficient fashion are put to low-power states.
- OPT^*. It minimizes the *average-case system energy* by assuming average-case execution time ($\frac{BCET+WCET}{2}$).
- CLR. It is the clairvoyant scheme that knows the *actual* execution times in advance and uses this information to derive optimal CPU operation frequency and device transitioning decisions. While CLR is not a practical scheme, it is used as a yardstick algorithm yielding the lower bound on system energy consumption.

Fig. 10a shows the comparison of the schemes as a function of the worst-case execution time of the application (c). Both the off-chip workload ratio and the best-case to worst-case execution time ratio are fixed at 0.2. Results are normalized with respect to OPT at $c = 44$. When c is in the range $[18, 30]$, the benefits of OPT^* over OPT are evident. In this range, the frequency determined by OPT^* is more energy-efficient toward dynamic workload variability compared to that of OPT. At low c values $[0, 18]$, there is more slack in the system and hence more DPM opportunities; thereby, all schemes perform the same. For c values in the range $[30, 44]$, the high worst-case and average-case execution times severely limit device transitions while calculating the CPU frequency. As a consequence, both OPT and OPT^* perform the same. However, CLR which uses actual workload information performs significantly better.

2. If the randomly generated execution time exceeds $WCET$, which happens with the probability 0.3 percent, that value is not considered in the experiments.

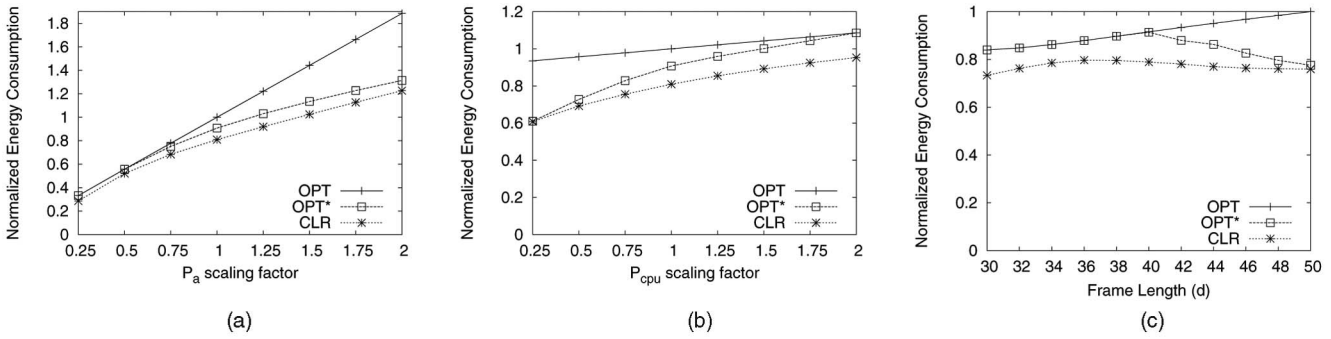

Fig. 11. Impact of variations in device, processor, and application characteristics. (a) Impact of device power scaling. (b) Impact of processor power scaling. (c) Impact of frame length.

Figs. 10b and 10c show the impact of varying $\frac{BCET}{WCET}$ ratio and $\frac{y}{c}$ ratio, respectively, at $c = 24$. In Fig. 10b, $\frac{y}{c} = 0.2$ and results are normalized with respect to OPT at $\frac{BCET}{WCET} = 1$. In Fig. 10c, $\frac{BCET}{WCET} = 0.2$ and the results are normalized with respect to OPT at $\frac{y}{c} = 0$. One can see that as $\frac{BCET}{WCET}$ ratio decreases, OPT^* performs better compared to OPT (Fig. 10b). This is because with more variations in runtime behavior of the application, the accuracy of OPT^* in estimating actual workload increases compared to OPT and thus the frequency determined by OPT^* yields lower system energy consumption. Fig. 10c indicates that varying the $\frac{y}{c}$ does not cause much difference in the relative ordering of the schemes.

Fig. 11 shows the impact of power and application characteristics. In these experiments, $c = 24$ and $\frac{BCET}{WCET} = \frac{y}{c} = 0.2$. The experiment methodology is the same as described for Fig. 9 in Section 4.2. While OPT deviates significantly from CLR at high P_a and low P_{cpu} scaling factors, OPT^* remains close to the CLR, a fact showing its robustness for various device and processor power characteristics (Figs. 11a and 11b). Similar trends can also be seen in Fig. 11c where OPT deviates significantly from CLR at high d values, while the performance of OPT^* gets closer to CLR as d increases. This is because at large frame lengths, the importance of being able to accurately estimate actual workload information (and use it to determine CPU frequency while accounting for DPM issues) translates to significant system-wide energy savings.

Finally, Fig. 12 shows the impact of the standard deviation on the performance of the schemes. Again, $c = 24$, $\frac{BCET}{WCET} = \frac{y}{c} = 0.2$, and the results are normalized with respect to OPT at standard deviation ρ_0. The x-axis shows the varying standard deviation. Specifically, a value v on the x-axis corresponds to a standard deviation of $(v \cdot \rho_0)$. At low standard deviation values, the performance of OPT^* is close to CLR and significantly better than OPT. This is because with less variance in actual execution times, OPT^* can better estimate and exploit the variability in dynamic workload while determining the CPU frequency and device transitioning decisions that minimize the average-case system energy consumption. In fact, at zero standard deviation, the actual execution time is exactly the mean $(\frac{BCET+WCET}{2})$, which is the estimate used by OPT^* for energy management decisions. Thus, at this point, the performance of CLR and OPT^* is the same. With increasing standard deviation values, the distribution tends toward uniform distribution and the performance of OPT^* gets closer to OPT due to the large variance in execution times. Additional experimental analysis can be found in our technical report [9].

6 Conclusions

In this work, we addressed the problem of system-wide energy minimization through a novel approach. Unlike prior studies, our system-level energy model considered both VFS- and DPM-related issues and accounted for device transition overheads. In addition, our model also considered variations in on-chip/off-chip workload characteristics. With this general model, we were able to characterize the exact interplay between VFS and DPM formally. By deriving useful properties from this characterization, we formulated an $O(m \log m)$-time algorithm (where m is the number of devices) to determine the CPU frequency and device transitioning decisions to minimize the system-wide energy. Our extensive experimental evaluations using real device parameters demonstrated the potential benefits of our optimal scheme. We also extended our solution to deal with workload variability and showed how to minimize average-case energy assuming the knowledge about average-case execution time. To the best of our knowledge, this is the first work formally investigating the interplay between two well-known energy management techniques for real-time embedded systems, VFS and DPM.

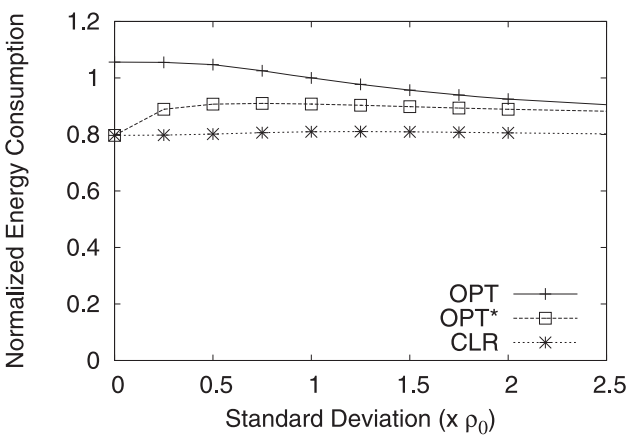

Fig. 12. Impact of the standard deviation.

ACKNOWLEDGMENTS

This work was supported by US National Science Foundation (NSF) through the Awards CNS-0720647 and NSF CAREER Award CNS-0546244.

REFERENCES

[1] H. Aydin, V. Devadas, and D. Zhu, "System-Level Energy Management for Periodic Real-Time Tasks," *Proc. 27th IEEE Real-Time Systems Symp. (RTSS),* 2006.
[2] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Power-Aware Scheduling for Periodic Real-Time Tasks," *IEEE Trans. Computers,* vol. 53, no. 10, pp. 584-600, May 2004.
[3] E. Bini, G.C. Buttazzo, and G. Lipari, "Speed Modulation in Energy-Aware Real-Time Systems," *Proc. Euromicro Conf. Real-Time Systems (ECRTS),* 2005.
[4] L. Benini, A. Bogliolo, and G.D. Micheli, "A Survey of Design Techniques for System-Level Dynamic Power Management," *IEEE Trans. Very Large Scale Integration Systems,* vol. 8, no. 3, pp. 299-316, June 2000.
[5] J.J. Chen and L. Thiele, "Expected System Energy Consumption Minimization in Leakage-Aware DVS Systems," *Proc. Int'l Symp. Low Power Electronics and Design (ISPLED),* 2008.
[6] H. Cheng and S. Goddard, "Integrated Device Scheduling and Processor Voltage Scaling for System-Wide Energy Conservation," *Proc. Int'l Workshop Power-Aware Real-Time Computing (PARC),* 2005.
[7] H. Cheng and S. Goddard, "Online Energy-Aware I/O Device Scheduling for Hard Real-Time Systems," *Proc. Conf. Design, Automation and Test in Europe (DATE),* 2006.
[8] K. Choi, R. Soma, and M. Pedram, "Fine-Grained Dynamic Voltage and Frequency Scaling for Precise Energy and Performance Trade-Off Based on the Ratio of Off-Chip Access to On-Chip Computation Times," *Proc. Conf. Design, Automation and Test in Europe (DATE),* 2004.
[9] V. Devadas and H. Aydin, "Additional Experimental Results on the Interplay of Voltage/Frequency Scaling and Device Power Management for Frame-Based Real-Time Embedded Applications," technical report, Dept. of Computer Science, George Mason Univ., http://www.cs.gmu.edu/~aydin/tr-2010-45.pdf, Sept. 2010.
[10] V. Devadas and H. Aydin, "Real-Time Dynamic Power Management through Device Forbidden Regions," *Proc. IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS),* 2008.
[11] R. Ernst and W. Ye, "Embedded Program Timing Analysis Based on Path Clustering and Architecture Classification," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design,* 1997.
[12] F. Gruian, "Hard Real-Time Scheduling for Low-Energy Using Stochastic Data and DVS Processors," *Proc. Int'l Symp. Low Power Electronics and Design (ISLPED),* 2001.
[13] P.J.M. Havinga and G.J.M. Smit, "Design Techniques for Low-Power Systems," *J. Systems Architecture,* vol. 46, no. 1, pp. 1-21, 2000.
[14] R. Jejurikar and R. Gupta, "Dynamic Voltage Scaling for System-Wide Energy Minimization in Real-Time Embedded Systems," *Proc. Int'l Symp. Low Power Electronics and Design (ISLPED),* 2004.
[15] W. Kim, D. Shin, H.S. Yun, J. Kim, and S.L. Min, "Performance Comparison of Dynamic Voltage Scaling Algorithms for Hard Real-Time Systems," *Proc. IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS),* 2002.
[16] L.-F. Leung, C.-Y. Tsui, and X.S. Hu, "Exploiting Dynamic Workload Variation in Low Energy Preemptive Task Scheduling," *Proc. Conf. Design, Automation and Test in Europe (DATE),* 2005.
[17] J. Liu, *Real Time Systems.* Prentice Hall, 2000.
[18] J. Lorch and A. Smith, "Improving Dynamic Voltage Scaling Algorithms with PACE," *Proc. ACM SIGMETRICS Int'l Conf. Measurement and Modeling of Computer Systems,* 2001.
[19] D. Luenberger, *Linear and Nonlinear Programming.* Addison-Wesley, 1984.
[20] M. Pedram, "Power Minimization in IC Design: Principles and Applications," *ACM Trans. Design Automation of Electronics Systems,* vol. 1, no. 1, pp. 3-56, 1996.
[21] P. Pillai and K.G. Shin, "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems," *Proc. ACM Symp. Operating Systems Principles (SOSP),* 2001.
[22] A. Qadi, S. Goddard, and S. Farritor, "A Dynamic Voltage Scaling Algorithm for Sporadic Tasks," *Proc. IEEE Real-Time Systems Symp. (RTSS),* 2003.
[23] C. Rusu, R. Melhem, and D. Mosse, "Maximizing Rewards for Real-Time Applications with Energy Constraints," *ACM Trans. Embedded Computing Systems,* vol. 2, no. 4, pp. 1-23, 2003.
[24] S. Saewong and R. Rajkumar, "Practical Voltage-Scaling for Fixed-Priority Real-Time Systems," *Proc. IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS),* 2003.
[25] K. Seth, A. Anantaraman, F. Mueller, and E. Rotenberg, "FAST: Frequency-Aware Static Timing Analysis," *Proc. IEEE Real-Time Systems Symp. (RTSS),* 2003.
[26] Y. Shin and K. Choi, "Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems," *Proc. Design Automation Conf. (DAC),* 1999.
[27] V. Swaminathan and K. Chakrabarty, "Energy Conscious, Deterministic I/O Device Scheduling in Hard Real-Time Systems," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 22, no. 7, pp. 847-858, July 2003.
[28] V. Swaminathan and K. Chakrabarty, "Pruning-Based, Energy-Optimal Deterministic I/O Scheduling for Hard Real-Time Systems," *ACM Trans. Embedded Computing Systems,* vol. 4, no. 1, pp. 141-167, 2005.
[29] V. Swaminathan, K. Chakrabarty, and S.S. Iyengar, "Dynamic I/O Power Management for Hard Real-Time Systems," *Proc. Int'l Conf. Hardware-Software Co-Design and System Synthesis (CODES),* 2001.
[30] H.W. Turnbull, *Theory of Equations.* Oliver and Boyd, 1947.
[31] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for Reduced CPU Energy," *Proc. USENIX Conf. Operating Systems Design and Implementation,* 1994.
[32] R. Xu, D. Mosse, and R. Melhem, "Minimizing Expected Energy Consumption in Real-Time Systems through Dynamic Voltage Scaling," *ACM Trans. Computer Systems,* vol. 25, no. 4, 2007.
[33] R. Xu, C. Xi, R. Melhem, and D. Mosse, "Practical Pace for Embedded Systems," *Proc. ACM Int'l Conf. Embedded Software (EMSOFT),* 2004.
[34] X. Zhong and C.-Z. Xu, "System-Wide Energy Minimization for Real-Time Tasks: Lower Bound and Approximation," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design (ICCAD),* 2006.
[35] D. Zhu and H. Aydin, "Energy Management for Real-Time Embedded Systems with Reliability Requirements," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design (ICCAD),* 2006.
[36] J. Zhuo and C. Chakrabarti, "System-Level Energy-Efficient Dynamic Task Scheduling," *Proc. Design Automation Conf. (DAC),* 2005.
[37] Advanced Configuration and Power Interface Standard, http://www.acpi.info/, 2011.

Vinay Devadas received the BS degree in computer science and engineering from Visvesvaraya Technological University, Bengaluru, India, in 2005, and the MS degree in computer science from George Mason University, Fairfax, Virginia, in 2007. He is currently working toward the PhD degree at the Department of Computer Science, George Mason University. His research interests include low-power computing, real-time embedded systems, and operating systems.

Hakan Aydin received the BS and MS degrees in control and computer engineering from Istanbul Technical University, in 1991 and 1994, respectively, and the PhD degree in computer science from the University of Pittsburgh in 2001. He is currently an associate professor in the Computer Science Department at George Mason University, Fairfax, Virginia. He served on the program committees of several conferences and workshops, including IEEE RTSS and RTAS. He was also the technical program committee chair of IEEE Real-time and Embedded Technology and Applications Symposium (RTAS'11). He was a recipient of the US National Science Foundation (NSF) Faculty Early Career Development (CAREER) Award in 2006. His research interests include real-time systems, low-power computing, and fault tolerance. He is a member of the IEEE and the IEEE Computer Society.