2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium

DFR-EDF: A Unified Energy Management Framework for Real-Time Systems*

Vinay Devadas Hakan Aydin
Department of Computer Science
George Mason University
Fairfax, VA 22030
{vdevadas, aydin}@cs.gmu.edu

Abstract

Dynamic Voltage Scaling (DVS) and Dynamic Power Management (DPM) techniques form the basis of numerous energy management schemes proposed for real-time embedded systems. DVS targets reducing the dynamic CPU energy consumption, while DPM attempts to reduce the energy consumption of idle devices by putting them to low-power states over sufficiently long intervals. It is imperative that the system-wide *energy management schemes efficiently integrate DVS and DPM while exploiting the subtle trade-off dimensions. In this paper, we develop and propose a unified framework for periodic real-time tasks where DVS and DPM are judiciously combined. The framework, called* DFR-EDF*, assumes a general system-level energy model and includes both static and dynamic (online) components. The static part is based on the extension of the recently proposed* Device Forbidden Regions (DFRs) *approach to Earliest-Deadline-First (EDF) scheduling. The online component integrates the predictive DPM techniques and offers a generalized slack reclaiming mechanism that can be used by DVS and DPM simultaneously. Our experimental evaluation indicates significant gains of DFR-EDF at the* system-level *compared to the state-of-the-art solutions. Finally, this research effort makes another contribution by formally showing that optimally solving the DPM problem in periodic real-time execution settings is NP-Hard in the strong sense, even in the absence of DVS.*

1 Introduction

Effective energy management for computing systems remains an elusive problem, despite the significant insight that the research community gained in the last decade through numerous research studies and projects. For real-time embedded systems, energy-awareness is now a prime design and operational objective that must be achieved without sacrificing the critical *temporal predictability* guarantees. *Dynamic Voltage Scaling* (DVS) and *Dynamic Power Management* (DPM) are two well-known techniques for energy management. With DVS [18], the CPU clock frequency and supply voltage can be adjusted dynamically on-the-fly. Due to the convex relationship between the CPU power consumption and processor frequency, DVS helps to significantly reduce processor dynamic energy consumption at the cost of increased task response times [2, 11, 12, 13]. On the other hand, DPM was proposed to reduce the device (primarily I/O device and main memory) energy consumption by transitioning devices to *low-power (sleep)* states when not in use [4, 5, 10, 17]. While it has an intuitive appeal, a primary challenge with DPM is to ensure that the non-trivial energy overheads associated with device state transitions do not offset the energy savings obtained during the device idle intervals [4, 5, 10].

While DVS and DPM are both important and each alone presents non-trivial difficulties in real-time settings, a main motivation for this research effort is the need for *unified* frameworks that contain *both techniques as essential components*. The growing awareness for *system-level energy minimization* [1, 3, 6, 20, 22] as opposed to component (e.g. CPU or device) level techniques provides another inspiration for our work. As we elaborate in Section 1.1, the system-level solutions are far fewer compared to extensive DVS- or DPM-only energy management literature. Moreover, most of the existing system-level solutions either ignore the DPM aspect or attempt to integrate it with DVS through simple run-time heuristics. Yet, the proper integration of DVS and DPM techniques poses several challenges, even for a *single* real-time application as recently demonstrated in [6]. This is due to the fact that aggressive DVS schemes lead to short device idle intervals that limit the effectiveness of DPM solutions – similarly, solutions with DPM as the primary focus may lead to excessive CPU power consumption. Our primary objective in this paper is to develop a unified energy management framework for deadline-driven periodic real-time applications by exploiting the interplay between DVS and DPM with both static and dynamic (run-time) solution components.

1.1 Related Work

DVS has attracted significant attention in the research community; in particular, the problem of minimizing energy consumption through DVS while still meeting the deadlines (the *RT-DVS* problem) has been addressed extensively over the past decade for various task models [2, 11, 12, 13]. More recently, it was reported that aggressive use of DVS tech-

*Supported by US National Science Foundation grants CNS-072047 and CNS-546244 (CAREER Award)

1080-1812/10 $26.00 © 2010 IEEE
DOI 10.1109/RTAS.2010.32

niques may lead to increased device usage times and energy consumption, thereby increasing *overall system energy*. This essentially led to the introduction of *energy-efficient frequency scaling* concept [1, 8, 20, 21, 22]. Despite some differences in their task/energy models, all these solutions essentially identify an *energy-efficient* (or, *critical*) CPU frequency below which the system consumes more energy per execution cycle. The frequency is never scaled below that threshold at run-time. However, all these system-level studies, while taking device energy consumption during task executions into consideration, effectively ignore the DPM dimension to a large extent.

This is because, in practice, device transitions incur significant time and energy overheads. As a result, for every device, there exists a minimum idle interval length (called the device *break-even time*) that justifies the device state transitions for energy saving [4, 5, 10]. The break-even time of I/O devices can be significant and comparable to task execution times [4, 10]. If the device is re-activated before the break-even time, then the overall energy would increase; as a result, it is more energy-efficient to keep the device simply in active state for short idle intervals. Thus, accurately *predicting* the length of idle intervals is crucial to perform effective DPM at run-time. In real-time systems, the length of device idle intervals are typically estimated using prediction based mechanisms. This is often achieved by estimating the earliest time an idle device will be needed by any task at run-time. Exploiting activation patterns of periodic tasks is a primary tool in existing predictive techniques [3, 15, 16, 17]. The problem of energy minimization for real-time systems through DPM (the *RT-DPM problem*) has been addressed in several works [4, 15, 16, 17]. More recently, the work in [5] proposed a novel *RT-DPM* approach based on *Device Forbidden Regions (DFR)* concept. With *DFRs*, through a pre-processing phase, the system plans for putting devices to sleep states for long intervals *in advance*. The DFRs are enforced at run-time with pre-determined periods. Further, the DFR approach can be easily combined with simple predictive schemes [5]. The framework in [5] was developed for systems using Fixed-Priority (Rate-Monotonic) Scheduling.

The research studies that analyze and/or exploit the DVS/DPM trade-offs are relatively few. In [6], an exact characterization of the interplay of DVS and DPM was presented. Despite its novelty and precision, its limitation comes from the simple application model with a single frame-based task. In [3], the authors proposed a system energy management scheme *SYS-EDF* with both DVS and DPM components for periodic real-time tasks and EDF scheduling. The DVS component of *SYS-EDF* is based on the concept of energy-efficient frequency scaling while the DPM component uses next device usage time predictions. In [14], assuming that CPU frequency can take any value to guarantee feasibility, the authors theoretically investigated the problem of minimizing processor energy consumption while taking into account both dynamic CPU power and CPU state transition overheads.

Contributions of this research. In this paper, we develop a unified framework (called *DFR-EDF*) that considers the trade-offs between the effective DPM and DVS policies to minimize overall system energy for periodic real-time tasks. We assume a general energy model where the CPU and device power consumptions, as well as device break-times and transition overheads (in terms of both energy and time) are considered. Our solution judicially combines both DVS and DPM components and includes a *static* and a *dynamic* part.

A critical building block for our framework is the *Device Forbidden Regions (DFRs)* concept [5] that is formally extended to EDF scheduling for the first time in this paper. EDF, by offering 100% CPU utilization and admitting simple utilization-based feasibility tests, provides an important leverage mechanism for our unified framework. We develop an efficient static feasibility test for *DFR-EDF* (Section 3.2), prove its correctness, and show how this test can be used to derive system-wide energy-efficient *DFR* parameters as well as processing frequencies for DVS *simultaneously* at the *static (design)* phase (Section 3.3). We also present our *dynamic* schemes that extend the duration of device idle intervals by combining DFRs and predictive techniques (Section 4.1) and exploit the run-time slack for *both* DVS and DPM, simultaneously (Section 4.3). Our experimental evaluation with realistic processor and device specifications indicates that our scheme outperforms state-of-art schemes by margins of up to 27% on *overall system energy* (Section 5). Further, our evaluation shows that unlike existing schemes, *DFR-EDF* maintains a *robust performance* for various system profiles regardless of whether the CPU or device energy is dominant. Finally, another contribution of this research effort is to show that the open problem of optimally solving the DPM problem for periodic real-time tasks (even without DVS) is NP-Hard in the strong sense (Section 3).

While inspired by our previous work [5] that introduced the DFR concept and showed its applicability in fixed-priority settings, the present work has a number of conceptual novelties *beyond* extending the DFR framework to dynamic-priority settings efficiently (which is a non-trivial challenge in its own). The focus of [5] was primarily the DPM-related issues: although it included a run-time DVS component, the DPM and DVS solutions were *disjoint*. Specifically, the DFR parameters were obtained assuming maximum frequency and the question of assigning optimal reduced frequency with DVS was left open. In contrast, the *DFR-EDF* framework determines the ideal DFR parameters and CPU operating frequency (namely, the best *power management configuration*) through a search that considers the impact on device and CPU energy simultaneously. Also, in [5] the dynamic slack was unconditionally used for frequency reduction. In this work through a *generalized reclaiming mechanism* the run-time slack can be used for both DVS and DPM, whenever possible.

2 System Model and Assumptions

2.1 Processor and Task Model

We consider a DVS-capable uni-processor system with k discrete CPU frequency values $f_1, \ldots, f_k$ ($f_i < f_{i+1}$). All frequency values are normalized with respect to f_k; i.e. $f_k = 1.0$. The workload consists of set of n independent periodic tasks $\psi = \{T_1 \ldots T_n\}$. Each periodic task T_i is represented by the pair (C_i, P_i), where C_i denotes the worst-case execution time under maximum processor frequency f_k and P_i denotes the period. The relative deadlines of the tasks are equal to their periods. The worst-case execution time of task T_i at frequency f is assumed to be $(\frac{C_i}{f})$. The *base utilization* of the task set ψ is given by $U_{tot} = \sum_{i=1}^{n} \frac{C_i}{P_i}$. At any time, tasks eligible for execution are scheduled by preemptive Earliest-Deadline-First (EDF) scheduling policy. We assume $U_{tot} \leq 1$ and hence the task set is schedulable by EDF at the maximum CPU frequency [9].

2.2 Device Model

The system has a set of m devices represented by $\mathcal{D} = \{D_1 \ldots D_m\}$. The set of devices used by a given task T_i during its execution is denoted by γ_i. Each device is assumed to have at least two power states: an *active* (working) and a low-power *sleep* state. In *sleep* state, a device cannot process any requests but consumes less power. P_a^i and P_s^i represent the power consumption of D_i in *active* and *sleep* states, respectively. The transitions between active and sleep states involve overheads both in terms of time and energy. The periodic nature of the applications implies that any device put to *sleep* state will be eventually re-activated. Thus, for convenience, the energy overhead of transitioning D_i once from *active* to *sleep* and back from *sleep* to *active* is captured under a single variable E_{sw}^i. Similarly, T_{sw}^i indicates the total transition delay between *active* and *sleep* states.

Due to the constraints imposed by device transition delays, D_i cannot be transitioned between *active* and *sleep* states over an interval of length smaller than T_{sw}^i. Further, in view of the energy transition overheads, the minimum length of idle interval over which transitioning a device D_i saves energy (as opposed to keeping D_i continuously in active state) is given by $\frac{E_{sw}^i - P_s^i \cdot T_{sw}^i}{P_a^i - P_s^i}$. Thus, the *break-even time* B_i of device D_i is expressed as [4, 5, 10] $B_i = max(T_{sw}^i, \frac{E_{sw}^i - P_s^i \cdot T_{sw}^i}{P_a^i - P_s^i})$.

In accordance with the previous RT-DPM research [3, 4, 5, 15, 17], throughout this paper we assume *inter-task device scheduling*: all devices in γ_i should be in *active* state when task T_i executes. As the exact times at which a running task generates a request for a device cannot be known in advance and device state transition delays are often significant, the inter-task device scheduling paradigm is considered realistic for energy modeling and minimization objectives in real-time systems research [3, 4, 5, 17].

2.3 Energy Model

System energy consumption can be divided into static energy and dynamic energy components. The static power is needed for purposes such as keeping the clock running, maintaining the basic circuits and keeping the devices in *sleep* states. Due to the periodic nature of real-time tasks and the significant delays involved in completely turning off CPU and other components, we assume that static energy is not manageable and focus on minimizing the system-level dynamic energy.

All the devices and CPU contribute to the overall dynamic energy consumption which is expressed as:

$$E_{system} = E_{cpu} + \sum_{i=1}^{m} E_{device}^i$$

where E_{cpu} is the energy consumed by the processor while executing the task set ψ. The processor power is modeled as a convex function of its clock frequency (i.e. $P(f) = \alpha f^3$, where α is the switching capacitance). E_{device}^i corresponds to the overall energy consumption due to a specific device D_i and includes three components: (1) The energy consumed by D_i when *active* and in use by tasks. This component depends on the execution times of tasks using D_i. (2) The energy overhead involved in transitioning D_i between *active* and *sleep* states during execution. (3) The energy consumed by D_i when *active* and not in use. Obviously, a device not in use may be forced to remain in *active* state when the estimated length of the idle interval is shorter than its break-even time.

3 DFR-EDF: Fundamentals and Static Analysis

In this section, we present the basics of the *device forbidden region* (DFR) based DPM methodology and then derive a sufficient schedulability condition for DFR and EDF scheduling. Following this, we show how to combine (and obtain energy-efficient configuration parameters for) DPM and DVS.

But first, we address a fundamental problem: *How hard is the problem of minimizing the system-level energy for periodic real-time applications?* Theorem 1, whose proof is presented in [7] due to space constraints, indicates that solving the problem of minimizing device energy consumption for general periodic tasks (even in the absence of DVS) is intractable, closing an open problem[1]. We first formally define the device energy minimization problem for real-time tasks.

[1] In [10] the authors indicate that the problem of minimizing device energy consumption in the absence of deadlines is NP-Hard only in *ordinary* sense. Further, in [5] it was shown that generating a feasible schedule for periodic tasks where every device idle interval is greater than its respective break-even time is NP-Hard in the strong sense. While the result in [5] hints to the inherent challenges involved in the problem, it does not imply that device energy minimization is NP-Hard as a schedule with the mentioned property does not necessarily minimize the total device energy consumption.

RT-DPM: Given a set of real-time tasks and a set of devices with known energy characteristics, find the feasible schedule that minimizes the total device energy consumption.

Theorem 1 *RT-DPM for general periodic tasks is NP-Hard in the strong sense.*

This result indicates that even a pseudo-polynomial time optimal algorithm for the problem is unlikely, unless $P = NP$.

3.1 Device Forbidden Regions

In [5], a novel DPM methodology based on the concept of *device forbidden regions* was introduced. The key to that approach is to *plan in advance for long device idle intervals* called *forbidden regions (FR)* that will be enforced at run-time. The forbidden region associated with device D_i is denoted by FR_i; when FR_i is enforced ("activated") at run-time by the operating system, the corresponding device is put to *sleep* state.

Each forbidden region FR_i is characterized by a length (or, duration) Δ_i, and a minimum separation time (or, period) Π_i that are determined through *static analysis*. The length of each FR is guaranteed to be greater than the break-even time of the associated device (i.e. $\Delta_i > B_i$). Thus, when FR_i is enforced at run-time, the state transition of D_i is guaranteed to be *energy-efficient*. However, no tasks using D_i can execute for the duration of FR_i. Specifically, with FRs, a job is *eligible* for execution only when it is released *and* there are no enforced FRs in the system affecting its devices. Among all eligible jobs, the one with the highest-priority is scheduled.

Since jobs may be prevented from execution due to run-time enforcement of DFRs, the static analysis for DFR components must include a *schedulability test* to guarantee the feasibility of the real-time workload. A viable strategy is to treat each FR as a *high priority task* which *only* interferes with the execution of other tasks using the device associated with the FR. This approach allows to preserve the feasibility as long as two consecutive enforcements of the same FR at run-time are separated by a *minimum* distance equal to its period. In other words, FR run-time enforcements need not be strictly periodic and can be delayed if necessary.

The above feature is also important because it allows to *combine* the run-time enforcements of FRs with known simple prediction-based DPM mechanisms. Hence, DFR-based DPM has also a mechanism to predict inherent device idle interval lengths (for example, by using information about next release times of periodic task instances). At run-time, when a device can be transitioned to *sleep* state using the prediction mechanism, an FR is not enforced (but *postponed*), saving its *bandwidth* for future usage. Further, whenever possible, FRs are enforced in conjunction with *existing* idle intervals to create long contiguous device sleep periods, resulting in additional energy savings.

Now we present an example to illustrate the potential benefits of employing DFR-based DPM policies, in conjunction with EDF. We are given two periodic tasks T_1 and T_2 with the following parameters: $C_1 = C_2 = 250$, $P_1 = 1200$, $P_2 = 1500$. T_1 uses the device D_1 with the break-even time $B_1 = 1000$, while T_2 uses D_2 with $B_2 = 1260$.

Figure 1(a) shows the EDF schedule without DFRs during the first hyperperiod. In the schedule, the maximum length of the device idle interval for D_1 and D_2 are 950 and 1250, respectively. Notice that, in this case, even an *ideal* predictive DPM policy would not be able to transition any of the devices at run-time, since the *exact* lengths of idle intervals for all devices are smaller than their respective break-even times.

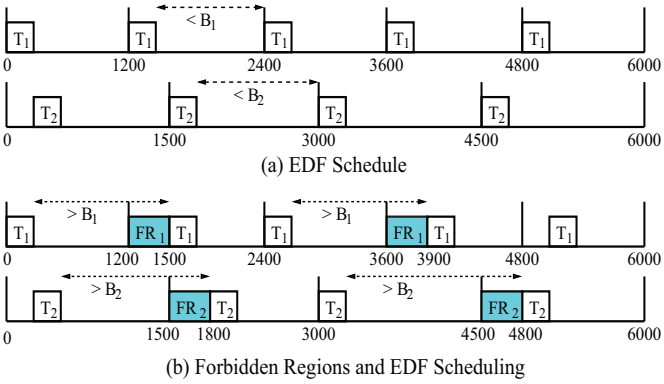


Figure 1. Motivational example for DFR-based DPM

Figure 1(b) shows the schedule with FRs and EDF scheduling for the same task set. In this case, a separate forbidden region is assigned to each device: FR_1 with duration $\Delta_1 = 300$ and period $\Pi_1 = 2400$; FR_2 with duration $\Delta_2 = 300$ and period $\Pi_2 = 3000$. The enforced forbidden regions are shown with dark colors in the schedule. Observe how the enforcement of FR_1 at $t = 1200$ is exactly aligned with the end time of an *existing* device idle interval $[250, 1200]$ for D_1. This effectively creates a long contiguous idle interval $[250, 1500]$ for D_1 during which it can be put to *sleep* state. Similar patterns can be seen in the schedule at times $t = 1500$, 3600 and 4500. Note that two consecutive FR enforcements are separated by a temporal distance at least equal to their periods. As seen in Figure 1(b) FRs and their run-time enforcements help create long sleep intervals for both devices without affecting the feasibility.

3.2 EDF Schedulability Analysis with DFRs

While the potential benefits of DFRs for enhancing the effectiveness of DPM are clear, it is imperative to make sure that all task instances meet their deadlines under various DFR activation patterns. In this subsection, we derive a *sufficient* schedulability condition for a set of periodic tasks scheduled with preemptive EDF, in the presence of device forbidden regions with given duration (Δ) and period (Π) parameters. Later in Section 3.3 we deal with determining Δ and Π parameters.

Definition 1 Υ_k *represents* all *the forbidden regions associated with* all *the devices used by the* k smallest period tasks.

Formally, if tasks are sorted in non-decreasing order of periods, Υ_k represents the set containing all the forbidden regions associated with devices $\cup_{i=1}^{k}\gamma_i$.

Theorem 2 *Given a set of periodic tasks $\psi = \{T_1 \dots T_n\}$ arranged in non-decreasing order of periods and a set of forbidden regions $\phi = \{(\Delta_1, \Pi_1) \dots (\Delta_m, \Pi_m)\}$, the periodic task set ψ can be scheduled by EDF in feasible manner if,*

$$\forall k_{k=1\dots n} \sum_{i \in \Upsilon_k} (\frac{\Delta_i}{\Pi_i} + \frac{\Delta_i}{P_k}) + \sum_{j=1}^{k} \frac{C_j}{P_j} \leq 1$$

The proof of Theorem 2 can be found in Appendix.

Complexity: The schedulability test has n iterations and each iteration takes $O(m+n)$-time. Therefore, the overall complexity is $O(mn + n^2)$. Since the number of devices is typically much smaller than the number of tasks, the complexity of the static feasibility test can be considered as $O(n^2)$.

In DVS settings with variable processing frequency, we have the following corollary as a consequence of Theorem 2.

Corollary 1 *Given a set of periodic tasks $\psi = \{T_1 \dots T_n\}$ arranged in non-decreasing order of periods, and a set of forbidden regions $\phi = \{(\Delta_1, \Pi_1) \dots (\Delta_m, \Pi_m)\}$, the periodic task set ψ can be scheduled by EDF in a feasible manner at the processing frequency f if,*

$$\forall k_{k=1\dots n} \sum_{i \in \Upsilon_k} (\frac{\Delta_i}{\Pi_i} + \frac{\Delta_i}{P_k}) + \sum_{j=1}^{k} \frac{C_j}{f \cdot P_j} \leq 1$$

3.3 Determining System Parameters for Effective Integration of DVS and DPM

To integrate DVS and DPM, DFR-EDF relies on a *power management configuration* $\mathcal{C}$ uniquely determined by a processing frequency f and a set of forbidden regions $\phi = \{(\Delta_1, \Pi_1) \dots (\Delta_m, \Pi_m)\}$. In Section 3.2, we provided an efficient test to decide whether the task set ψ is feasible under EDF with a given forbidden region set ϕ. Obviously, it is equally important to *determine* the power management configuration $\mathcal{C}$ that yields best energy savings, as there are typically many configurations that preserve the feasibility.

However, such a decision is not trivial. Typically, DVS and DPM components tend to favor power management configurations with *opposing* features. Reducing the processing frequency (to favor DVS) will scale up task execution times and hence seriously constrain the DPM opportunities. On the other hand, configurations that favor DPM will create long idle intervals that require high CPU frequencies and will also need to take into account the device transition overheads. In fact, in the light of Theorem 1 (which indicates that RT-DPM is intractable even in the absence of DVS), one can safely state that an *exact and efficient* solution is unlikely. The *DFR-EDF* framework takes a more direct approach: it exploits several features of forbidden regions and known properties of DVS solutions to quantify the energy savings one can *expect*, given a power management configuration $\mathcal{C}$. Hence, the static phase of *DFR-EDF* includes an iterative procedure to decide on the run-time power configuration.

An inspection of Corollary 1 reveals that as more FRs are assigned to the system, the minimum frequency that guarantees feasibility will typically increase. Thus, there is a tradeoff between assigning additional FRs to decrease device energy consumption and the resulting increase in processor energy consumption. *DFR-EDF* incrementally assigns FRs to the system. At each step, it evaluates the *expected* benefit (in terms of energy savings) of assigning an FR for another system device separately, and commits to the one which appears most promising. The process of assigning new FRs to the power management configuration stops when it is expected that by doing so the overall energy consumption will increase (due to, typically, excessive CPU energy consumption).

To formally describe this procedure, we introduce the notation ΔE^i_{sys} for the *expected* change in *system* energy consumption as a consequence of adding a *new* forbidden region, FR_i, to an already assigned forbidden region set ϕ. Let ΔE^i_{device} and ΔE^i_{cpu} denote the expected decrease in device energy and the expected increase in processor energy respectively, due to the *additional* forbidden region FR_i during a hyperperiod. ΔE^i_{sys} can be expressed as:

$$\Delta E^i_{sys} = \Delta E^i_{device} - \Delta E^i_{cpu}$$

If $\Delta E^i_{sys} > 0$ then the system is *expected* to benefit from the *additional* forbidden region FR_i. On the other hand, if $\Delta E^i_{sys} < 0$ then the system energy is likely to increase due to the addition of FR_i.

We now show how to quantify ΔE^i_{device} and ΔE^i_{cpu}. As a consequence of forbidden region enforcements, the devices are transitioned to *sleep* states. Each FR_i enforcement provides a potential device energy saving of $\Delta_i \cdot (P^i_a - P^i_s)$. During a hyperperiod H, there can be at most $\lfloor \frac{H}{\Pi_i} \rfloor$ FR_i enforcements. As such, ΔE^i_{device} can be approximated as,

$$\Delta E^i_{device} = \lfloor \frac{H}{\Pi_i} \rfloor \cdot \Delta_i \cdot (P^i_a - P^i_s)$$

Let f_1 and f_2 denote the minimum frequencies that guarantee the feasibility of task set ψ with FR sets ϕ and $(\phi \cup FR_i)$, respectively. Let P_{cpu} denote the power consumption of the processor at maximum frequency. Recall that U_{tot} represents task set utilization under *maximum processor frequency* ($f_k = 1$). As a consequence of adding FR_i, the system is forced to switch to f_2 from frequency f_1. Clearly, $f_2 \geq f_1$. Thus, ΔE^i_{cpu} can be quantified as,

$$\Delta E^i_{cpu} = H \cdot U_{tot} \cdot P_{cpu} \cdot (f_2^2 - f_1^2)$$

We assume the existence of a function *MinSchedulableFreq()* that determines the *minimum* CPU frequency f that guarantees the feasibility of task set ψ with FR set ϕ. *MinSchedulableFreq()* can be easily implemented in n iterations

by running the test provided in Corollary 1 and recording the minimum feasible frequency[2] at each step. The maximum frequency value recorded through n iterations is the minimum value that guarantees the feasibility (as using a lower frequency would violate the feasibility condition for at least one iteration). If this maximum frequency exceeds $f_k = 1$ then ψ is not schedulable with ϕ and *MinSchedulableFreq()* returns $null$. Note that *MinSchedulableFreq()* has the same complexity as the feasibility test provided in Section 3.2 ($O(n^2)$).

Finally, for a given FR_i, the range of possible Δ_i and Π_i values should be provided to make the algorithm's search component complete. Clearly, Δ_i must be no shorter than the corresponding device break-even time and cannot exceed the maximum laxity of a task using the corresponding device. Hence, $B_i \leq \Delta_i \leq \min_{j|D_i \in \gamma_j} (P_j - C_j)$. Also, the ratio $\frac{\Delta_i}{\Pi_i}$ must not exceed $(1 - U_{D_i})$, where U_{D_i} is the total utilization of tasks using device D_i. Thus, $\frac{\Delta_i}{1-U_{D_i}} \leq \Pi_i \leq H$.

DetermineEMPRoutine():

- Set $\mathcal{Z} = \{D_1, \ldots, D_m\}$
- Set $\phi = \emptyset$
- Set $f = MinSchedulableFreq(\psi, \phi)$
- Repeat
  - Set $\Delta E^i_{sys} = -1, \forall D_i \in \mathcal{Z}$
  - For each device D_i in $\mathcal{Z}$, compute through *MinSchedulableFreq()* a tentative (Δ_i,Π_i) pair and a processor frequency f_i that provides the maximum ΔE^i_{sys} while committing to the current FRs in set ϕ and maintaining the feasibility.
  - Let i^* represent the device index that yields $\max_{i \in \mathcal{Z}}(\Delta E^i_{sys})$
  - if $(\Delta E^{i^*}_{sys} > 0)$
    - Set $\phi = \phi \cup (\Delta_{i^*}, \Pi_{i^*})$
    - Set $Z = Z - D_{i^*}$
    - Set $f = f_{i^*}$
- Until $(\mathcal{Z} = \emptyset \ OR \ \Delta E^{i^*}_{sys} \leq 0)$
- return (f, ϕ) as the power management configuration $\mathcal{C}$

Figure 2. Determining Energy-Minimal System Parameters

DetermineEMPRoutine() given in Figure 2 determines the power management configuration $\mathcal{C}$ that consists of a set of FRs and CPU frequency f. The procedure starts with an empty FR set and incrementally assigns FRs one at a time. At each stage, for each device D_i not associated with an FR (set $\mathcal{Z}$), it scans the range of Δ_i and Π_i at a fixed number of points and determines the (Δ_i, Π_i) pair and processor frequency f_i that maximizes ΔE^i_{sys} using the *MinSchedulableFreq()* routine. Our experience with extensive simulations involving wide range of parameters show that scanning 20-30 equi-distant candidate Δ_i and Π_i values is sufficient to determine system parameters that yield significant energy savings. Further, since *MinSchedulableFreq()* has the complexity $O(n^2)$, scanning a fixed number of candidate values for each D_i can be afforded at the static phase.

Following this, the most promising FR and corresponding processor frequency combination (i.e. the one with the largest *positive* ΔE^i_{sys}) is recorded as the new best configuration. This iterative process stops either when all devices are assigned with FRs, no more FRs can be assigned due to feasibility, or assigning additional FRs does not improve the system energy (i.e. $\Delta E^i_{sys} \leq 0$ for all devices in $\mathcal{Z}$).

Complexity: *DetermineEMPRoutine()* has at most m iterations. Each iteration invokes the schedulability test a constant number of times for every device not associated with an FR. Thus, the complexity of each iteration is $O(m(mn + n^2))$ making the overall complexity $O(m^2(mn + n^2))$. The number of devices in the system is typically small compared to the number of tasks. Consequently, in practice the complexity can be approximated as $O(n^2)$, which is affordable for a routine invoked only once in static analysis phase.

4 Online Components

In Section 3, we provided the details of the static component of *DFR-EDF*, which generates a power management configuration $\mathcal{C}$ with forbidden regions and a CPU frequency. In this section, we present the details of its *online (dynamic)* component that includes mechanisms to further increase the length of device idle intervals and reclaim slack that may arise from early completions for *both* DVS and DPM.

4.1 Enforcement of Device Idle Intervals

The performance of the DPM component can be further improved by incorporating run-time prediction mechanisms, allowing enforcement of idle intervals even longer than the statically-determined FR durations. To see this, first observe that the pre-determined period Π_i of a given forbidden region FR_i gives the *minimum* separation time of two consecutive enforcements of FR_i: *delaying* the next enforcement of FR_i cannot have a negative impact on feasibility. This gives a powerful opportunity to enhance the DPM effectiveness at run-time. For example, if the online DPM component can efficiently conclude that an *active* device D_i can be put to *sleep* state for more than B_i time units *without enforcing an* FR_i or causing deadline miss, that would allow the system to *postpone* the next activation of FR_i, saving its bandwidth for future use. That is, the ability to accurately predict the inherent device idle intervals in the schedule is still important for the *DFR-EDF* framework. In this subsection we show how this can be achieved with low run-time overhead.

[2] The frequency levels that are below the energy-efficient frequency level [1] are not considered.

At time t, let C_k^r denote the remaining worst-case execution time of job J_k under the frequency assigned by the static routine. Let $\mathcal{HP}_k$ denote the set of jobs in the ready queue having higher priority than *any unfinished* job of T_k. In the worst-case feasible schedule (where all jobs present their worst-case workload), a given task T_k cannot start to execute until time $t + \sum_{j \in \mathcal{HP}_k} C_j^r$. Further, if an instance of T_k is not currently ready, then one needs to wait *at least* until its earliest *next* release time $R_k(t)$. Hence, in either case, at time t, T_k's next dispatch time cannot occur earlier than $max(R_k(t), t + \sum_{j \in \mathcal{HP}_k} C_j^r)$. Thus, at time t, D_i used by T_k can be put to *sleep* state for a maximum of $\mathcal{V}_i^k = max(R_k - t, \sum_{j \in \mathcal{HP}_k} C_j^r)$ time units without causing a deadline miss for T_k.

Interestingly, the *DFR-EDF* framework creates further opportunities to put a device to *sleep* state by exploiting the run-time information about idle intervals *currently enforced for other devices*. Specifically, note that when a device D_j is explicitly put to *sleep* state (e.g. through an *FR*), a task T_k using both D_j and another device D_i is *guaranteed* not to generate a request for D_i as well, while D_j remains in *sleep* state. This follows from the inter-task device scheduling paradigm, where all the devices of a given task must be ready before it can start to execute. In other words, the existence of a task T_k using both D_i *and* D_j during its execution, makes the *sleep* intervals of D_i and D_j *inter-dependent* at run-time.

Let n_j denote the end of a currently enforced idle interval for device D_j. At time t, a task T_k is guaranteed not to use any device $D_i \in \gamma_k$ for *at least* $(n_j - t)$ time units, if there exists another device $D_j \in \gamma_k$ for which an idle interval is already enforced until time n_j. Considering all other devices, we can see that $D_i \in \gamma_k$ cannot be used by T_k for a maximum of $\mathcal{W}_i^k = max(n_j - t),\ j | D_j \in (\gamma_k - D_i)$ time units.

Combining the two factors, namely the interference of high priority jobs and impact of already enforced sleep intervals we find that at time t, $D_i \in \gamma_k$ can be safely put to *sleep* state for $\delta_i^k(t) = max(\mathcal{V}_i^k, \mathcal{W}_i^k)$ time units without causing a deadline miss for T_k. By iterating over all tasks using D_i and taking the minimum, we can determine the *maximum* time D_i can be put to *sleep* state without affecting feasibility.

Proposition 1 *A given device D_i can be put to sleep state at time t for $\delta_i(t) = min(\delta_i^k(t)),\ k | D_i \in \gamma_k$ time units without hurting system feasibility.*

$\delta_i(t)$ can be determined in time $O(n \log n)$. Due to space constraints, the details of the run-time complexity, and the exact pseudo-code for deciding on device transitions at run-time are presented in [7].

4.2 Dynamic Frequency Scaling

Many DVS schemes adopt a *nominal* (default) frequency f_{nom} [2, 11, 13] that is statically computed. In *DFR-EDF* framework, f_{nom} is determined using the *DetemineEMPRoutine()* described in Section 3.3. It has been well-established that aggressive slowdown will affect system energy negatively and hence, at run-time, the frequency should not be lowered below a certain energy-efficient threshold [1, 3, 8]. In *DFR-EDF*, we adopt the system energy-efficient scaling technique given in [3]. In that technique, an energy-efficient frequency threshold is calculated *dynamically* based on the power characteristics of the processor, *active* devices and devices in *sleep* state, at job dispatch times. The processor frequency is never reduced below this threshold.

4.3 Generalized Slack Reclaiming

As many real-time tasks complete early at run-time, detecting and reclaiming unused CPU time (*slack*) has been a major tool for dynamic DVS schemes [2, 11, 19]. These schemes typically incorporate mechanisms to decide on the amount of slack that can be reclaimed without compromising the feasibility, before the CPU frequency is reduced. Dynamic slack can be also exploited by DPM techniques to increase the length of device idle intervals by delaying task executions [4]. *DFR-EDF* has a built-in mechanism that can be used to keep track of and reclaim dynamic slack, for DVS or DPM. However, possibly the most novel aspect of *DFR-EDF*'s generalized reclaiming mechanism is that, whenever possible, it allows the use of the *same* dynamic slack for *both* DVS and DPM (i.e. for both lowering the processor frequency and increasing the idle intervals of devices).

We first give a motivational example to illustrate this original aspect. Figure 3 shows three ready jobs with decreasing priority at time t. Each job requires a worst-case execution time of 10 units under their nominal frequency assumed to be $f_k = 1$. Jobs J_1 and J_3 use device D_1 with break-even time 12 units, while J_2 uses no device. D_1 has no associated FR and is in *active* state at time t.

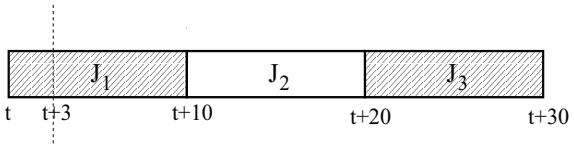


Figure 3. Reclaiming for both DVS and DPM

Assume J_1 completes early at time $t_1 = t + 3$, creating a dynamic slack of 7 units. At time t_1, J_2 will be dispatched and D_1 is no longer in use. Thus, the system needs to make a decision on whether or not to transition D_1 and decide on the processor frequency for executing J_2. Notice that at time t_1, if the dynamic slack is ignored, the device idle interval length of D_1 is predicted to be 10 units (the worst-case execution time of J_2) which is smaller than the break-even time of D_1. Hence, D_1 would be kept in *active* state by a naive predictive DPM policy. However, J_3 can start as late as $t + 20$ without violating system feasibility. Thus, by taking into account the dynamic slack of 7 units, one can transition D_1 at time t_1, as delaying its usage until $t + 20$ for 17 units (which is greater

than the break-even time of D_1) would still keep the system feasible. Also, J_2 (dispatched at time t_1) can use the same dynamic slack of 7 units to lower its dispatch frequency to $\frac{10}{17}$. Thus, the dynamic slack of 7 units from J_1's early completion has supported both DVS and DPM.

We now give the details of *DFR-EDF's* generalized slack monitoring and reclaiming mechanism. To keep track of unused run-times we adopt a technique similar to that used in Dual Speed Dynamic Reclaiming Algorithm (DSDR) [19]. When a job completes, its remaining (unused) run-time is added to a data structure called the *slack-queue*. Each element in the *slack-queue* has two components, one indicating the remaining run-time of the job and the other indicating its deadline. The *slack-queue* is maintained in non-decreasing order of deadlines. At time t, the *remaining* run-time of job J_i is denoted by rrt_i. At the release time of J_i, rrt_i is set to $\frac{C_i}{f_{nom}}$. At time t, when J_i is being dispatched, let $\mathcal{H}$ denote the set of elements in *slack-queue* with deadlines no greater than that of J_i. The slack available to J_i at time t due to early completions is given by $slack_i = \sum_{j \in \mathcal{H}} rrt_j$. Note that the remaining run-times of jobs and hence the *slack-queue* change with time and need to be updated accordingly. We refer to [19] for rules to update job run-times and *slack-queue*.

Slack Reclaiming for DVS: Recall from Section 4.1 that C_i^r represents the remaining worst-case execution time of J_i under f_{nom}. Further, from Section 4.2, when a job is being dispatched at time t, the frequency is not reduced below the energy-efficient frequency threshold (f_{thres}). Thus, J_i with available slack $slack_i$ is dispatched at frequency:

$$f = max(\frac{C_i^r}{slack_i + rrt_i}, f_{thres})$$

Slack Reclaiming for DPM: The slack that is available for a specific job at dispatch time can be also used to improve the estimation of the maximum time a device can idle without compromising the feasibility. Specifically, recall from Section 4.1 that $\mathcal{V}_i^k$ was defined as the maximum time D_i can remain in *sleep* state at time t without causing a deadline miss for T_k and was computed through the total remaining workload of *ready* jobs in $\mathcal{HP}_k$ (jobs with higher priority than *any* unfinished job of T_k). Notice that the very same principle can be applied to re-define $\mathcal{V}_i^k$ in a more precise manner: since in a pessimistic scenario the job of T_k *would have* to be delayed until all high priority jobs complete with their worst-case workload, delaying T_k *during* the *unused* run-times of such completed jobs (i.e. effectively keeping D_i in *sleep* state) would not hurt its feasibility. Let $slack_k$ denote the sum of remaining run-times from *all* completed jobs that is *available* to the earliest unfinished job instance of T_k. We update $\mathcal{V}_i^k$ as:

$$\mathcal{V}_i^k = max(R_k(t) - t, slack_k + \sum_{j \in \mathcal{HP}_k} C_j^r)$$

The run-time complexity of maintaining the *slack-queue* is $O(n)$ [19]. Further, the slack for *all* unfinished job instances at time t can be computed in $O(n \log n)$ time. Hence, the overall complexity is dominated by the DPM component with a run-time complexity of $O(n \log n)$ for each device (Our technical report [7] contains full details).

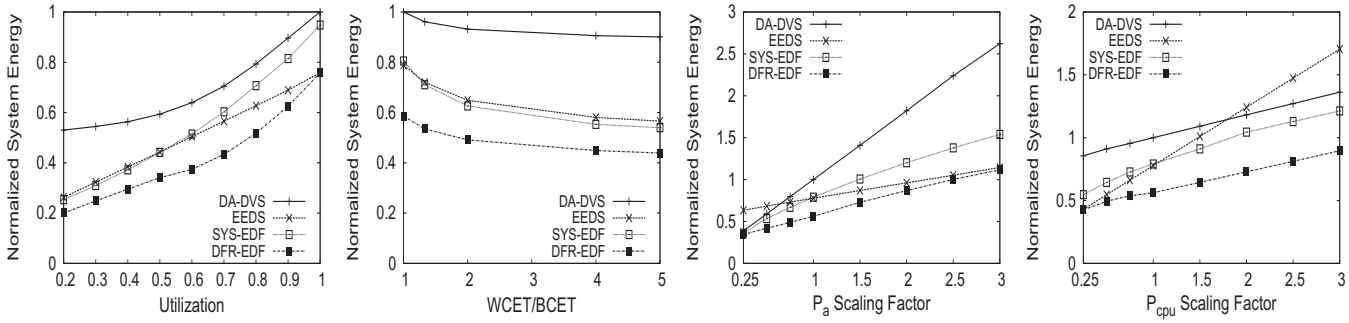
5 Experimental Evaluation

In this section, we experimentally evaluate the performance of *DFR-EDF*. We constructed a discrete-event simulator in C programming language. We generated 1000 synthetic task sets each containing 20 tasks, simulated their execution and recorded the system energy consumption. The task periods were chosen randomly in the range $[25ms, 1300ms]$. CPU power is modeled according to Intel Xscale architecture specifications [23]. Following [4], we considered settings where each task uses 0-2 devices randomly selected from a list that includes *IBM Microdrive, SST Flash, Realtek Ethernet Chip* and *SimpleTech Flash Card*. The device specifications are adopted from [4].

In addition to *DFR-EDF*, we implemented three state-of-the-art energy management algorithms in our simulator:

- *SYS-EDF* is a system-level energy management scheme with both DVS and DPM components [3]. *SYS-EDF* performs DVS using the concept of energy-efficient scaling and has a simple prediction-based DPM component that is applied at run-time.
- *DA-DVS* (Device-Aware DVS) represents the *DVS-only* schemes that exploit the concept of energy-efficient scaling. We adopted the algorithm from [1] which gives the *optimal* task level slowdown factors by taking into account device energy-efficient frequency thresholds. *DA-DVS* does not have a DPM component.
- *EEDS* (Energy-Efficient Device Scheduling) scheme, adopted from [4], is a state-of-the-art *DPM-only* scheme for dynamic priority systems and EDF scheduling. There is no DVS component in *EEDS* and it is designed for systems where the device power dominates over that of the CPU.

Figure 4(a) shows the relative performance of these schemes as a function of system utilization with worst-case workloads. All energy values are normalized with respect to the energy consumption of *DA-DVS* at 100% system utilization. *DFR-EDF* provides clear gains over all schemes throughout the entire spectrum. *DA-DVS* performs significantly poorly compared to other schemes since it does not have a DPM component. Among the two schemes with both DVS and DPM components, the performance of *SYS-EDF* quickly degrades at high utilization values, whereas *DFR-EDF* maintains its high performance. This is because, with increasing utilization, the system has to use high frequencies and hence becomes increasingly dependent on DPM (rather than DVS) for energy management. Further, the DPM policy of *SYS-EDF* is a relatively simple lookahead-based prediction scheme. But *DFR-EDF's* sophisticated DPM component helps to minimize device energy at high utilization



a. Impact of System Utilization b. Impact of Workload Variability c. Impact of Device Power d. Impact of CPU Power

Figure 4. Simulation Results

values which translate to significant system energy savings. In fact, notice that even *EEDS* with its more comprehensive DPM approach is able to outperform *SYS-EDF* at increasing utilization values when the latter's processor energy gains become less significant. By judicially combining DVS and DPM, *DFR-EDF* outperforms *SYS-EDF* and *EEDS* by margins of up to 27% and 24%, respectively.

Figure 4(b) shows the relative performance of schemes under variability in the actual workload. This variability is controlled by modifying the worst-case to best-case execution time ratio, $\frac{WCET}{BCET}$. The actual workload of each job is determined randomly at its arrival time, according to a uniform probability distribution between $BCET$ and $WCET$. Clearly, the higher the $\frac{WCET}{BCET}$ ratio, the larger the dynamic slack that can be used for additional energy savings. In these experiments, the system utilization is fixed at 60% and all energy values are normalized with respect to energy consumption of *DA-DVS* at $\frac{WCET}{BCET} = 1$. Notice that with increasing $\frac{WCET}{BCET}$ ratio the energy consumption of all schemes decreases, but relative improvements after a certain threshold become marginal. This is because with ample slack, with DVS, tasks tend to run at energy-efficient frequency thresholds. Similarly, for DPM, the idle interval of each device is naturally bounded by the periods of tasks using it. As explained in Section 4.3, whenever possible *DFR-EDF* utilizes the same dynamic slack for both DVS and DPM.

Figures 4(c) and 4(d) show the impact of varying power characteristics of system components, respectively. In these experiments the system utilization is fixed at 60% and $\frac{WCET}{BCET} = 1$. In Figure 4(c), for each task set and device usage pattern, we multiply the device active power (P_a) of all devices by a certain scaling factor and recompute the device break-even times while keeping the processor power the same. That is, the higher the scaling factor, the more dominant the device power. Similarly, in Figure 4(d), we multiply the processor power consumption at the maximum frequency (P_{cpu}) while keeping the device power characteristics the same. In both experiments, for each scaling factor we measure the energy consumed by all schemes. All energy values are normalized with respect to scaling factor of one (i.e. the original device/processor parameters as in [4, 23]).

With decreasing P_a scaling factors or increasing P_{cpu} scaling factors, processor energy consumption becomes more dominant and thus the role of DVS proves more crucial compared to that of DPM in minimizing the system energy. Thus, at low P_a scaling factors and high P_{cpu} scaling factors *EEDS* performs worse compared to all other schemes. In fact, there are regions in Figures 4(c) and 4(d) where *EEDS* performs worse than *DA-DVS* which does not have a DPM component. Similarly, with increasing P_a scaling factors or decreasing P_{cpu} scaling factors, device energy consumption becomes more dominant and thus DPM becomes critical for minimizing system energy. Thus, in these regions, *EEDS* outperforms both *SYS-EDF* and *DA-DVS*. However, due its inherent design features that exploit DVS and DPM in a synergistic way, *DFR-EDF maintains a robust performance throughout all scaling factors* – unlike other schemes, *DFR-EDF* does not suffer from excessive degradations in settings where CPU or device power dominates.

6 Conclusion

In this paper, we proposed the *DFR-EDF* framework for system-wide energy management of periodic real-time tasks. We provided a sufficient and efficient feasibility test for a set of periodic real-time tasks in the presence of device forbidden regions under preemptive EDF policy. Then, by using this test, we provided an algorithm to determine the DFR configuration and processing frequency for efficient integration of DPM and DVS in a single framework. We also provided online components to integrate predictive DPM policies and use slack reclaiming for both DVS and DPM simultaneously at run-time. Simulation results indicate that DFR-EDF offers significant energy gains over state-of-the-art energy management schemes. A by-product of this research effort is the formal demonstration that the *RT-DPM* problem for periodic real-time tasks is NP-Hard in the strong sense.

References

[1] H. Aydin, V. Devadas, and D. Zhu. System-level Energy Management for Periodic Real-Time Tasks. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS),* 2006.

[2] H. Aydin, R. Melhem, D. Mossé, and P. Mejia-Alvarez. Power-aware Scheduling for Periodic Real-time Tasks. In *IEEE Transactions on Computers*, vol. 53, no. 10, pp. 584-600, May 2004.

[3] H. Cheng and S. Goddard. Integrated Device Scheduling and Processor Voltage Scaling for System-wide Energy Conservation. In *Proc. of Int'l Wrkshp on Power-Aware Real-Time Computing (PARC)*, 2005.

[4] H. Cheng and S. Goddard. Online Energy-Aware I/O Device Scheduling for Hard Real-Time Systems. In *Proc. of Design Automation and Test in Europe (DATE)*, 2006.

[5] V. Devadas and H. Aydin. Real-Time Dynamic Power Management through Device Forbidden Regions. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS)*, 2008.

[6] V. Devadas and H. Aydin. On the Interplay of Dynamic Voltage Scaling and Dynamic Power Management. In *Proc. of the ACM international conference on Embedded software (EMSOFT)*, 2008.

[7] V. Devadas and H. Aydin. DFR-EDF: A Unified Energy Management Framework for Real-time Systems. *Technical Report.* Computer Science Department, George Mason University, October 2009. Available online at http://www.cs.gmu.edu/~aydin/tr-2009-23.pdf

[8] R. Jejurikar and R. Gupta. Dynamic Voltage Scaling for System-Wide Energy Minimization in Real-Time embedded systems. In *Proc. of Int'l Symp. on Low Power Electronics and Design (ISLPED)*, 2004.

[9] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in Hard Real-time Environment. *Journal of ACM* vol. 20. no. 1, pp. 46-61, January 1973.

[10] Y.H. Lu, L.Benini and G.D. Michele. Power-Aware Operating Systems for Interactive Systems. In *IEEE Transactions on Very large Scale Integration Systems*, vol. 10, no. 2, pp. 119-134, 2002.

[11] P. Pillai and K. G. Shin. Real-time Dynamic Voltage Scaling for Low-power Embedded Operating Systems. In *Proc. of the ACM Symposium on Operating Systems Principles (SOSP)*, 2001.

[12] A. Qadi, S. Goddard, and S. Farritor. A Dynamic Voltage Scaling Algorithm for Sporadic Tasks. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS),* 2003.

[13] S. Saewong and R. Rajkumar. Practical Voltage-Scaling for Fixed-Priority Real-time Systems. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2003.

[14] S. Irani, S. Shukla and R. Gupta. Algorithms for power savings. In *Proc. the ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 2003

[15] V. Swaminathan, K. Chakrabarty and S. S. Iyengar. Dynamic I/O Power Management for Hard Real-Time Systems.In *Proc. of the International Conference on Hardware-Software Co-design and System Synthesis (CODES)*, 2001.

[16] V. Swaminathan and K. Chakrabarty. Energy Conscious Deterministic I/O device Scheduling in Hard Real-Time Systems.In *Proc. of the International Conference in Computer Aided Design (ICCAD)*, 2003.

[17] V. Swaminathan and K. Chakrabarty. Pruning-Based, Energy-Optimal Deterministic I/O Scheduling for Hard Real-Time Systems.In *ACM Trans. on Emb. Computing Systems,* Vol. 4, No. 1, pp 141-167, 2005.

[18] M. Weiser, B. Welch, A. Demers and S. Shenker. Scheduling for Reduced CPU energy. In *USENIX Symposium on Operating Systems Design and Implementation*, 1994.

[19] F. Zhang and Chanson. Processor Voltage Scheduling for Real-Time tasks with Non-Preemptible Sections. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS),* 2002.

[20] X. Zhong and C.-Z. Xu. System-wide Energy Minimization for Real-Time Tasks: Lower Bound and Approximation. In *Proc. of the Int'l Conference on Computer-Aided Design (ICCAD)*, 2006.

[21] X. Zhong and C.-Z. Xu. Frequency-Aware Energy Optimization for Real-Time Periodic and Aperiodic Tasks. In *Proc. of the Conf. on Languages, Compilers and Tools for Embedded Systems (LCTES)*, 2007.

[22] J. Zhuo and C. Chakrabarti. System-level Energy-efficient Dynamic Task Scheduling. In *Proc. of the Conference on Design Automation (DAC)*, 2005.

[23] R. Xu, C. Xi, R. Melhem and D. Mosse. Practical PACE for Embedded Systems. In *Proc. of the ACM International Conference on Embedded Software (EMSOFT)*, 2004.

APPENDIX: Proof of Theorem 2

We will prove the theorem by contradiction. Assume there is a deadline miss and the following holds:

$$\forall k_{k=1\ldots n} \sum_{i \in \Upsilon_k} (\frac{\Delta_i}{\Pi_i} + \frac{\Delta_i}{P_k}) + \sum_{j=1}^{k} \frac{C_j}{P_j} \leq 1$$

Let t_d be the first time a job misses its deadline in the EDF schedule. Let t_s be the last time before t_d such that there are no pending job execution requests with arrival times before t_s and deadlines at or before t_d. t_s is well defined as no requests can arrive before $t = 0$.

At time t, where $t_s \leq t \leq t_d$, let $\mathcal{A}(t)$ denote the set of ready jobs with pending execution requirements whose arrival times are in interval $[t_s, t]$ and deadlines in interval $[t_s, t_d]$. By choice of t_s, $\mathcal{A}(t)$ is a non-empty set throughout the interval $[t_s, t_d]$. Hence, there is always a pending job with deadline no greater than t_d in this interval.

Let $t_d - t_s = X$. Let k be the largest index satisfying the condition $P_k \leq X$. At any time t, $t_s \leq t \leq t_d$, the job instances in $\mathcal{A}(t)$ belong to a subset of tasks in $\{T_1 \ldots T_k\}$ (since tasks are ordered according to periods). Υ_k denotes the set of forbidden regions affecting one or more tasks in $\{T_1 \ldots T_k\}$.

As a result, at any time t in the interval $[t_s, t_d]$ either a job in $\mathcal{A}(t)$ should be executing *or all* the jobs in $\mathcal{A}(t)$ should be prevented from execution (blocked) by one or more FRs in Υ_k. Therefore, the entire interval $[t_s, t_d]$ can be seen as a sequence of intervals where in each interval either a job in $\mathcal{A}(t)$ runs *or* all jobs in $\mathcal{A}(t)$ are blocked by one or more FRs in Υ_k.

Let α_1 denote total length of interval in $[t_s, t_d]$ where *all* jobs in $\mathcal{A}(t)$ are blocked due to FR enforcements. α_1 is bounded by the total length of FRs in Υ_k when they are activated with their minimum separation times (Π values) in $[t_s, t_d]$. Thus,

$$\alpha_1 \leq \sum_{i \in \Upsilon_k} \lceil \frac{X}{\Pi_i} \rceil \cdot \Delta_i$$

Let α_2 denote the length of intervals in $[t_s, t_d]$ where a job in $\mathcal{A}(t)$ is executed. α_2 is bounded by the total execution time of jobs in $\mathcal{A}(t)$, $t_s \leq t \leq t_d$. Thus,

$$\alpha_2 \leq \sum_{j=1}^{k} \lfloor \frac{X}{P_j} \rfloor \cdot C_j$$

Since a job misses its deadline at time t_d we must have,

$$\alpha_1 + \alpha_2 > X$$

$$\sum_{i \in \Upsilon_k} \lceil \frac{X}{\Pi_i} \rceil \cdot \Delta_i + \sum_{j=1}^{k} \lfloor \frac{X}{P_j} \rfloor \cdot C_j > X$$

Since $\lceil Y \rceil \leq \lfloor Y \rfloor + 1$ and $\lfloor Y \rfloor \leq Y$,

$$\sum_{i \in \Upsilon_k} (\frac{X}{\Pi_i} + 1) \cdot \Delta_i + \sum_{j=1}^{k} \frac{X}{P_j} \cdot C_j > X$$

$$\sum_{i \in \Upsilon_k} (\frac{\Delta_i}{\Pi_i} + \frac{\Delta_i}{X}) + \sum_{j=1}^{k} \frac{C_j}{P_j} > 1$$

By choice of k, $P_1 \leq P_2 \ldots \leq P_k \leq X$. Thus,

$$\sum_{i \in \Upsilon_k} (\frac{\Delta_i}{\Pi_i} + \frac{\Delta_i}{P_k}) + \sum_{j=1}^{k} \frac{C_j}{P_j} > 1$$

Giving a contradiction and proving the theorem.