Energy - Responsiveness Tradeoffs for Real-Time Systems with Mixed Workload

Hakan Aydin, Qi Yang
Computer Science Department
George Mason University
Fairfax, VA 22030
(aydin, qyang1)@cs.gmu.edu

Abstract

In this paper, we explore the performance tradeoffs for real-time systems with Dynamic Voltage Scaling (DVS) capability, when the workload includes aperiodic jobs as well as periodic tasks. As opposed to the assumptions of early works on Real-Time DVS or non-power-aware scheduling of hybrid task sets, the settings require the consideration of two often-conflicting objectives: Improving the responsiveness of aperiodic jobs and reducing the energy consumption. We propose the composite metric, Energy * Average Response Time*, as a performance measure in energy-aware scheduling of hybrid task sets. Then we develop our framework that integrates Dynamic Reclaiming Algorithm (DRA) and Total Bandwidth Server (TBS) mechanism in variable-speed settings. In addition to the static algorithm, we propose Basic Reclaiming Scheme (BRS) and Mutual Reclaiming Scheme (MRS) that enable the re-use of the system slack arising from early task completions. We also present our Bandwidth Sharing Scheme (BSS) that aggressively exploits the bandwidth reserved for TBS to further slow down the periodic tasks. We provide an experimental evaluation of our algorithms under different workloads and speed settings, and show that BSS can provide significant performance improvements when the actual variability in the workload is high.*

1 Introduction

In the early '90s, the real-time research community addressed the problem of scheduling workloads with mixed timing constraints (also known as *hybrid* task sets). Improving the *responsiveness* of aperiodic jobs that do not have explicit deadlines by minimizing their *average response time*, while meeting the hard deadlines of periodic tasks was the main objective in this line of research. Several solutions are proposed for fixed-priority systems [16, 17] and dynamic-priority systems [6, 8]. In [19], Spuri and Buttazzo proposed *Total Bandwidth Server (TBS)* and *Dynamic Priority Exchange (DPE)* schemes to be used in conjunction with Earliest-Deadline-First policy. The performances of these algorithms are shown to be close to an ideal scheduler through simulation studies, although their run-time overhead (especially that of TBS) is extremely low.

In the late '90s, a growing emphasis was put on power-aware resource management techniques. The fact that the CPU power consumption is related to the CPU supply voltage and clock frequency in a convex fashion gave rise to the *Dynamic Voltage Scaling (DVS)* technique. DVS-based scheduling techniques attempt to save energy by reducing the CPU speed *on-the-fly* at the expense of increased response time. For jobs with non-stringent timing requirements, Weiser et al. proposed schemes where the CPU utilization is monitored and the CPU speed is adjusted based on these observations in a dynamic fashion [22]. On the real-time DVS side, the pioneering work of Yao et al. [23] showed how to adjust the CPU speed to minimize the CPU energy consumption for a set of aperiodic jobs with release times and hard deadlines. The case of periodic real-time task scheduling on variable-speed processors was explored in depth in the context of both fixed-priority and dynamic-priority systems [2, 4, 9, 15, 20]. An on-line admission algorithm for joint scheduling of *hard* aperiodic jobs and periodic tasks was described by Hong et al. in [11].

However, scheduling hybrid task sets in energy-limited settings remains relatively unexplored up to this day. To start with, there is a need to extend the DVS-based techniques to address the problem of scheduling aperiodic requests with no specific deadlines without jeopardizing the deadlines of periodic (hard) real-time tasks. An initial difficulty is about the **performance metric** itself. All the hard deadlines must be met, but the criterion by which two different feasible schedules should be compared is not obvious. Clearly, one should try to save as much energy as possible, but the *responsiveness (average response time)* of soft aperiodic requests cannot be ignored altogether. In fact, the energy consumption component due to aperiodic jobs can be minimized by executing them with the minimum CPU speed[1]. However, this policy can make the *response times* unacceptably large. On the other hand, the response times can be improved by committing to the maximum CPU speed for aperiodic jobs. This, in turn, may result

[1] One can even argue that, in the absence of specific deadlines, these jobs could be *dropped* if the ultimate goal is to maximize energy savings.

in unacceptably high energy consumption due to the convex relationship between CPU speed and power consumption. The selection of the CPU speed for the *periodic tasks* raises similar considerations.

We suggest that an appropriate performance metric for hybrid task sets should incorporate *both* the energy consumption and responsiveness dimensions while meeting the hard deadlines. One plausible choice is to adopt the product *Energy * Average Response Time* as the performance metric. This metric is reminiscent of *Energy* $*$ *Delay* metric adopted in power-aware paging and sensor networks [14, 21]. Note that finding an analytical solution to the problem is not feasible, since we do not have closed formulas for average response times of soft aperiodic jobs that arrive randomly and that are scheduled in conjunction with hard periodic tasks [16, 19].

Once we decide to use this *composite* performance metric, a number of additional design issues become apparent. For instance, *reclaiming unused CPU computation time* in the common case where tasks/jobs complete earlier than the worst-case can help to reduce the energy consumption. In fact, recent studies show that such reclaiming policies can improve energy savings for *periodic workloads* by a factor of 40-50% [2, 15] while preserving the feasibility. However, the extent to which the responsiveness of aperiodic jobs will be affected by such reclaiming strategies is not clear.

In this paper, we develop a framework where the available CPU capacity is partitioned a priori between the periodic task set and a Total Bandwith Server in *variable-speed* settings. Soft aperiodic jobs are executed by TBS, whose low overhead and near-optimal performance under various load conditions was reported elsewhere [19]. In addition to this *static* approach, we present two schemes in which Dynamic Reclaiming Algorithm(DRA) [2] and TBS mechanism are integrated in order to detect and exploit early completions of *both* periodic tasks and aperiodic jobs. In *Basic Reclaiming Scheme (BRS)*, reclaiming is performed only within each type of workload, while *Mutual Reclaiming Scheme (MRS)* allows reclaiming of any slack by any task/job (periodic or aperiodic). Finally, our last solution, called *Bandwidth Sharing Scheme (BSS)*, aggressively uses the bandwidth of TBS to further slow down periodic tasks when the *actual arrival rate* of soft jobs is lower than predicted. We show that the feasibility is preserved in all these schemes.

We also provide an experimental evaluation of the proposed schemes. We analyze the performance of the algorithms on each of the *Energy*, *Average Response Time* and *Energy * Average Response Time* dimensions. We conclude that reducing the CPU speed beyond a certain threshold, while yielding significant energy savings, results in drastic degradation in responsiveness of soft aperiodic jobs and consequently in *Energy * Average Response Time* dimension. We show that reclaiming unused computation time by BRS, MRS and BSS can provide considerable benefit over the static scheme, though their relative performance is highly dependent on the *variability* in the actual workload.

2 System Model, Assumptions and Notation

Throughout the paper, we adopt preemptive Earliest-Deadline-First (EDF) [13] scheduling policy. The computational workload has two components: A set $T = \{T_1, \ldots, T_n\}$ of periodic tasks with hard deadlines, and a set of soft aperiodic jobs that arrive randomly at run-time. In accordance with previous work on scheduling hybrid task sets [8, 16, 17, 19], we assume that the worst-case resource requirements of periodic tasks are known in advance, whereas those of soft aperiodic jobs are made available only when they arrive.

All periodic tasks are assumed to be independent and simultaneously ready at $t = 0$. The relative deadline of each periodic task instance is equal to its period. We assume that aperiodic requests are executed by a *Total Bandwidth Server* [19]. Although there are other algorithms proposed for scheduling aperiodic jobs in deadline-driven systems [8, 19], we opt to use TBS because of its simplicity, low run-time overhead, and near optimal performance over a large spectrum of load values as shown in [19]. We adopt the following notation:

- P_i : The constant period of the periodic task T_i
- $T_{i,j}$: The j^{th} instance of the periodic task T_i
- C_i : The worst-case workload of T_i expressed as the worst-case number of CPU cycles required per instance
- J_i : The i^{th} aperiodic job (request)
- r_i : The arrival time of J_i
- c_i : The worst-case workload of J_i expressed as the number of CPU cycles
- u_B : The *capacity* of Total Bandwith Server (Section 3.1)
- d_i : The deadline assigned to J_i by Total Bandwidth Server (Section 3.1)

We assume a variable voltage/speed processor whose speed S (in terms of processor cycles per unit time) can vary between 0 and an upper bound S_{max}. For convenience, we normalize the CPU speed with respect to S_{max}; that is, we assume that $S_{max} = 1.0$. The power consumption of the processor under the speed S is given by the function $g(S)$. In current DVS technologies, the function $g(S)$ is assumed to be a strictly convex and increasing function on non-negative real numbers [1, 2, 9, 10, 20]. Further, it is usually represented by a polynomial of at least the second degree [2, 10]. If the CPU speed in the time interval $[t_1, t_2]$ is denoted by $S(t)$, then the *energy* consumed during this interval is $E(t_1, t_2) = \int_{t_1}^{t_2} g(S(t))dt$. Our focus is on the CPU energy consumption; low-power techniques for memory, disk and I/O subsystems, albeit important, are beyond the scope of this paper.

We assume that the process descriptor of each task/job is augmented by two extra fields, the *current speed* and the *nominal speed*. The former denotes the speed level at which the task is executing and the latter represents the "default" speed it has whenever it is dispatched by the operating system prior to any dynamic adjustment. The current and nominal speed of the periodic task T_i are denoted by S_i and $\widehat{S}_i$, respectively.

3 Preliminaries

In this section, we provide a brief review of the Total Bandwidth Server mechanism and Dynamic Reclaiming Algorithm (DRA). We also present an extension of DRA, called Extended Dynamic Reclaiming Algorithm (EDRA), that will be used in conjunction with TBS mechanism to schedule hybrid task sets on variable-speed processors in Section 4.

3.1 Total Bandwidth Server

Given a set $T = \{T_1, \ldots, T_n\}$ of periodic tasks and a *constant-speed* CPU, the *total utilization* of T is defined as $U_T = \sum_{i=1}^{n} \frac{\alpha_i}{P_i}$ where α_i is the worst-case execution time of T_i. A well-known result of real-time scheduling theory [13] states that T can be scheduled in feasible manner by EDF if and only if $U_T \leq 1.0$. In case that *aperiodic jobs* with soft deadlines are also submitted to the system at run-time, a common solution is to assign a dedicated *server process* to schedule these jobs. In Total Bandwidth Server [19] approach, an aperiodic job server with *capacity (size)* $u_B \leq 1.0$ is adopted. According to the TBS scheduling rules [19], the i^{th} aperiodic job J_i, upon its arrival at $t = r_i$, is assigned a deadline d_i and is subsequently scheduled along with other periodic tasks according to EDF scheduling policy. If the worst-case execution time of J_i is e_i, then the deadline (hence, the priority) assigned to J_i is computed as:

$$d_i = max\{r_i, d_{i-1}\} + \frac{e_i}{u_B} \tag{1}$$

where $d_0 = 0$. Notice that the deadline of J_i is a function of both the deadline of J_{i-1} and the server capacity u_B: In particular, the larger u_B, the smaller the deadline assigned to J_i, the higher its priority, and the better its response time. Spuri and Buttazzo proved that [19], in a system where aperiodic requests are served through Total Bandwidth Server with capacity u_B, all periodic tasks will meet their deadlines if and only if

$$U_T + u_B \leq 1.0 \tag{2}$$

where U_T is the total utilization of the periodic task set T.

3.2 Scheduling Periodic Real-Time Tasks with Dynamic Voltage Scaling

In DVS-enabled processors, the worst-case execution time of a task depends on the CPU speed. In particular, if all periodic tasks are executed with the speed $S \leq 1.0$, then the **effective utilization** (U_p) of the periodic task set becomes:

$$U_p = U_T(S) = \sum_{i=1}^{n} \frac{C_i}{S \cdot P_i} = \frac{U_{tot}}{S}$$

where $U_{tot} = U_T(1.0) = \sum_{i=1}^{n} \frac{C_i}{P_i}$. The quantity U_{tot} is of particular interest: It represents the periodic utilization under maximum speed $S = 1.0$, and this is the *minimum* effective utilization that the periodic task set can have (i.e. $U_T(S) \geq U_{tot}$ for $S \leq 1.0$). Moreover, it is known that [2, 15] for a purely periodic workload scheduled by EDF, the CPU speed that minimizes the total energy consumption while meeting all the deadlines (i.e. *the static optimal speed*), is constant and equal to $\bar{S} = U_{tot}$. We will call the quantity U_{tot} the *base utilization* of the periodic task set T.

However, even when one schedules all the tasks with the static optimal speed, many task instances complete without presenting their worst-case workload in practice. Thus *reclaiming* unused computation time to reduce the CPU speed while preserving feasibility was subject to numerous research papers in recent years [2, 4, 15, 20]. In [2, 4], a generic dynamic reclamation algorithm (GDRA) was proposed for power-aware scheduling of *periodic* tasks. In that algorithm, each task instance $T_{i,j}$ assumed a **nominal (default)** speed of $\widehat{S}_i = \bar{S} = U_{tot}$. At dispatch time, this nominal speed was reduced by computing the unused CPU time of already-completed tasks (called the *earliness* factor of $T_{i,j}$). Below, we provide a brief review of GDRA. However, the version we employ is slightly different from the one proposed in [2, 4] in that **the nominal speed of tasks is no longer necessarily equal to $\bar{S} = U_{tot}$. Instead, $\widehat{S}_i$ can assume any value such that** $\widehat{S}_i \geq U_{tot}$. We will later derive some useful properties of this **Extended Dynamic Reclaiming Algorithm (EDRA)**, enabling us to undertake joint scheduling of aperiodic and periodic jobs.

EDRA is based on *dynamically* comparing the actual schedule to the static schedule $\mathcal{S}^{can}$(in which each task instance runs with its nominal speed and presents its worst-case workload). To perform the comparison, a data structure (called α**-queue**) is maintained and updated at run-time. The α-queue is effectively the ready queue of $\mathcal{S}^{can}$ at time t. Specifically, at any time t, the information about each task instance $T_{i,j}$ that *would* be ready at t in $\mathcal{S}^{can}$is available in α-queue, including its identity, ready time, deadline and *remaining execution time* (denoted by $rem_{i,j}$)[2]. EDRA assumes that tasks are scheduled according to EDF* policy [2]. EDF* is the same as EDF [13], except that, among tasks whose deadlines are the same, the task with the earliest arrival time has the highest priority (FIFO policy); in case that both deadline and arrival times are equal, the task with the lowest index has the highest priority. The α-queue is also ordered according to EDF* priorities. We denote the EDF* priority-level of task T_i by d_i^* (low values denote high priorities).

The key notation and rules pertaining to EDRA are presented in Figure 1 and 2, respectively. Rules 1-3 are provide the update rules for the α-queue structure at important "events"

[2]Since there can be at most *one* ready instance of a periodic task at anytime t, we will drop the second index in $rem_{i,j}$ and other $\alpha-$queue related notation when the context is clear.

(task arrival/completion), while Rule 4 shows how the speed of a task T_x is reduced by evaluating its earliness at dispatch time. $\epsilon_x(t)$ represents the unused computation time of tasks at higher or equal priority level with respect to T_x at time t: these tasks would still have some non-zero remaining execution time in $\mathcal{S}^{can}$, but now they must have been completed because T_x is the highest priority task in the system. Observe that rem_i values are available in the α-queue.

- $\mathcal{S}^{can}$: The "canonical" schedule in which each task T_i presents its worst-case workload C_i at every instance and runs with nominal speed
- $rem_i(t)$: the remaining execution time of T_i at time t in $\mathcal{S}^{can}$
- $w_i^S(t)$: the remaining worst-case execution time of task T_i under the speed S at time t in the actual schedule
- $\epsilon_i(t)$: The earliness of task T_i at time t in the actual schedule, defined as:
  $\epsilon_i(t) = \sum_{j|d_j^*<d_i^*} rem_j(t) + rem_i(t) - w_i^{\widehat{S_i}}(t) = \sum_{j|d_j^*\le d_i^*} rem_j(t) - w_i^{\widehat{S_i}}(t)$

Figure 1. Key Notation for EDRA

Rules for EDRA

1. Initialize the α-queue to the empty-list.
2. At every event (arrival/completion), consider the head of the α-queue and decrease its rem_i value by the amount of elapsed-time since the last event. If rem_i is smaller than the time elapsed since the last event, remove the head, update the time elapsed since the last event, and repeat the update with the next element. This is done until all "elapsed time" is used.
3. At every new arrival, insert into the α-queue, in the correct EDF* order, the information regarding the new instance of task $T_{i,j}$ with $rem_i(t) = w_i^{\widehat{S_i}}$.
4. Whenever T_x is about to be dispatched at time t:
   4.1. Set $S_x = \widehat{S_x}$.
   4.2. Consult the α-queue and compute the earliness of T_x.
   4.3. Reduce the speed of task T_x by allocating an extra $\epsilon_x(t)$ time units to T_x:
   $S_x = \frac{w_x^S \cdot S_x}{w_x^S + \epsilon_x(t)}$
5. At every event of preemption or completion of a task, say T_i, decrease the value of the remaining execution time: $w_i^{S_i} = w_i^{S_i} - \Delta_t$, where Δ_t is the time elapsed since the task T_i was last dispatched.

Figure 2. Extended Dynamic Reclaiming Algorithm (EDRA)

4 Scheduling Hybrid Task Sets with DVS

4.1 Static Approach (No Reclaiming)

The first scheme we present is an extension of TBS mechanism to variable-speed settings. Given a periodic task set with base utilization U_{tot}, we execute all the periodic tasks with a (nominal) speed of $S_p \ge U_{tot}$. This yields an *effective periodic utilization* of $U_p = \frac{U_{tot}}{S_p}$. The aperiodic jobs will be served by TBS whose capacity is set to $u_B = 1 - U_p$. Unused computation times are not reclaimed and the feasibility of the resulting schedule follows trivially from Equation (2).

When we consider performance issues such as **energy consumption** and **average response time of aperiodic jobs**, an interesting *trade-off dimension* becomes apparent: The objective of reducing the energy consumption of periodic tasks suggests that we should keep S_p as low as possible, but this will decrease the TBS capacity u_B, hence will tend to increase the average response time of aperiodics. In fact, setting $S_p = U_{tot}$ would *minimize* the energy consumption of periodic tasks (while meeting their deadlines), but at the cost of setting u_B to 0, thereby not executing any aperiodic jobs at all.

Even when we decide on the speed S_p for the periodic workload and determine $u_B = 1 - \frac{U_{tot}}{S_p}$, a second problem needs to be resolved: What should be the speed S_a at which we execute the aperiodic jobs? Again, the convex relation between CPU speed and power consumption suggests that we should reduce S_a to save energy. However, this will increase the execution time of J_i, which is equal to $\frac{c_i}{S_a}$. Moreover, the deadline assigned to J_i will become (from Equation (1)):

$$d_i = max\{r_i, d_{i-1}\} + \frac{c_i}{u_B \cdot S_a}$$

That is, decreasing S_a will increase the response time of J_i and the reason is twofold: its CPU demand (in terms of execution time) increases, and its scheduling priority decreases (the deadline assigned by TBS increases).

Experimental Evaluation: In order to experimentally evaluate the effects of the workload and the speed settings (S_p and S_a) on three performance metrics *Energy, Average Response Time* and *Energy * Average Response Time* over a wide range of workload, we implemented a scheduling simulator.

In our experiments, we generated 1000 synthetic task sets, each containing 30 periodic tasks. The periods of the tasks were chosen randomly in the interval [1000, 32000]. For each task set, we simulated the execution up to LCM 20 times, where LCM is the least common multiple of $P_1, \ldots, P_{30}$. The *base utilization* U_{tot} of the periodic workload component was set to 0.3. However, we changed the *effective periodic utilization* U_p of the task set between 0.3 and 0.9, by varying the periodic speed S_p between 0.333 and 1.0. We used a quadratic power/speed function.

The capacity u_B of the Total Bandwidth Server was set to $1 - U_p$, which is the processor utilization unused by the periodic tasks. The *nominal aperiodic load* U_w (the computational demand of aperiodic jobs under $S_{max} = 1.00$) was also varied from 0 to $1 - U_p$. The interarrival times of soft aperiodic jobs were modeled by a Poisson probability distribution, with the average of $T_a = \frac{LCM}{30}$. The worst-case number of CPU cycles for each aperiodic job J_i (i.e. c_i) was generated according to a normal probability distribution with a mean of $T_a \cdot U_w$, which guarantees a nominal aperiodic load of U_w on the average. For a given nominal aperiodic load U_w, the aperiodic speed S_a was varied between 0.1 and 1.0, giving a wide range of *effective aperiodic load*.

For each task set simulated, we computed the total energy consumption, the average response time (of soft aperiodics) and the resulting *Energy * Average Response Time* value. The *actual* number of CPU cycles for each periodic task instance and aperiodic job was set to the worst-case in the simulations we report in this section. We varied the nominal aperiodic load U_w, the periodic speed S_p and the aperiodic speed S_a to observe their effect on each performance metric. When we investigated the effect of the nominal aperiodic load and aperiodic speed, the periodic speed S_p was set to 0.6, giving an effective periodic load of 0.5. Similarly, in experiments where we investigated the effect of S_p and S_a, the nominal aperiodic load U_w was set to 0.1. Recall that the *effective aperiodic load* is a function of both U_w and S_a (specifically, $\frac{U_w}{S_a}$).

• **Energy Consumption:** Figure 3 (left) shows the average energy consumption as a function of the nominal aperiodic load U_w under different aperiodic speed settings. The results are normalized with respect to the energy consumption of the task set when $U_w = 0.5$ and $S_a = 1.0$. We observe that the energy consumption increases in linear fashion with aperiodic load, and this is to be expected, since the CPU demand/energy relation is linear under a given speed. Note that there is a constant energy component due to the periodic workload (which can be clearly observed when U_w is zero). As we increase S_a, the slope of the energy consumption curve becomes steeper due to the convex relationship between the speed and energy. When we examine the effect of S_p on the energy consumption (Figure 3 (right), the results are normalized with respect to the case where S_a and S_p are both set to 1.0), we see that the increase is superlinear, despite the fact that the energy consumption of the aperiodic load is constant for a fixed S_a value.

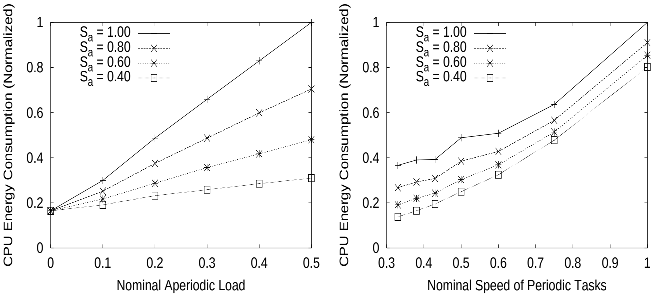

Figure 3. Energy consumption of the static scheme as a function of the aperiodic load (left) and the nominal speed of periodic tasks (right)

• **Average Response Time:** When we examine the effect of varying S_a, S_p and U_w on the average response time of soft aperiodic jobs, we get a completely different picture (Figure 4, left). The average response time increases in linear fashion with U_w, up to some threshold value for a given S_a, and beyond this, it sharply increases. The threshold value indicates the point at which the system practically enters the *overloaded* region, and the threshold is encountered much earlier for smaller S_a values (when the effective aperiodic load increases fast). Similarly, the average response time of soft aperiodic jobs increases abruptly when the speed of periodic tasks falls below a threshold, making the effective periodic utilization and the total load too high (Figure 4, right). Interestingly, when we execute the soft aperiodics with the speed $S_a = 1.0$, the increase is marginal even at low S_p values.

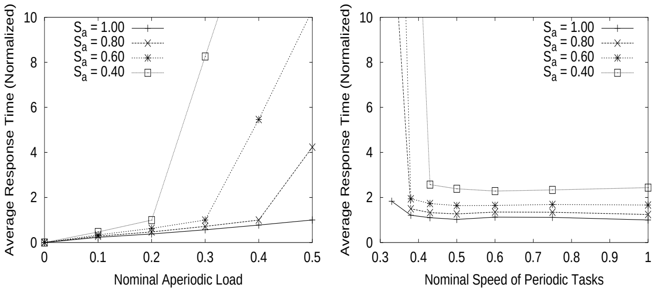

Figure 4. Average response time of the static scheme as a function of the aperiodic load (left) and the nominal speed of periodic tasks (right)

• **Energy*Average Response Time:** By examining the changes in the *Energy * Average Response Time* with varying parameters (Figure 5), we can have a better assessment of the trade-off dimensions. We observe that the trends in the metric's curve follow broadly those of the *average response time*: As the system steps into the "overloaded" region due to the decrease in either S_a or S_p, the energy savings improve, but this is shadowed by the superlinear increase in average response time and the net effect is negative. In fact, the benefits of using low S_a values is marginal even at low aperiodic load values (Figure 5, left), *which suggests the use of the maximum CPU speed for aperiodic jobs to improve the composite performance metric*. On the other hand, the metric benefits from reducing S_p *as long as the system stays in underloaded region* (Figure 5, right). However, it increases drastically when the *effective periodic utilization* exceeds a threshold value.

4.2 Basic Reclaiming Scheme (BRS)

While we explored in Section 4.1 the basic principles of DVS on a system with periodic and aperiodic workloads, a tacit assumption of the framework was that each task instance/job presents its worst-case workload when it executes. While this assumption is helpful to guarantee the timing constraints under a worst-case scenario, in practice many jobs complete early. Thus, detecting early completions and reclaiming unused CPU time by further *slowing down* other tasks has been a central issue in recent Real-Time DVS research, but in many cases in the context of purely periodic workloads [2, 4, 15, 20].

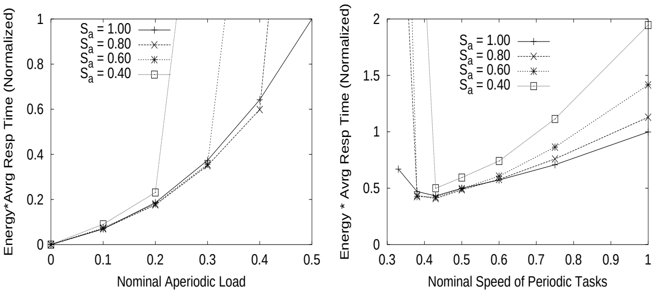

Figure 5. *Energy * Average Response Time* **performance of the static scheme as a function of the aperiodic load (left) and the nominal speed of periodic tasks (right)**

It is clear that reclaiming can provide energy benefits also for *hybrid* workloads. On the other hand, reclaiming can have *negative* effects on average response time of soft aperiodic jobs: a given (aperiodic) job can complete earlier if the slack of a periodic task is *not* reclaimed by another periodic task (reducing the potential interference to the CPU demand of the aperiodic job).

The first reclaiming scheme we present is still based on *a priori partitioning* of the CPU capacity between the periodic workload and TBS. Specifically, the periodic tasks assume a (nominal) speed of $S_p \geq U_{tot}$. The aperiodic jobs are still served by a TBS with capacity set to $u_B = (1 - U_p)$, and the CPU speed is set to S_a when an aperiodic job is executed. However, unlike the static scheme, early completions are detected and unused CPU time is reclaimed. Further, the isolation between two types of workload is *preserved* in the reclaiming mechanism: Periodic tasks can *only* reclaim the unused CPU time of *periodic* tasks, while the deadline of TBS is updated (reduced) dynamically when a soft aperiodic job completes prematurely. The process of adjusting the deadline of TBS was described in detail in [18], and it is omitted here due to space limitations. That scheme effectively enables the *reuse* of the bandwidth belonging to prematurely-completed soft aperiodic jobs, and it improves the average response time while not necessarily reducing the energy consumption.

The reclaiming mechanism for *periodic* tasks is based on EDRA presented in Section 3.2. Again, note that the choice of nominal speed values for periodic and aperiodic tasks, namely S_p and S_a can potentially have a deep impact on the energy consumption and average response time of soft aperiodics (as shown by the simulation results in Section 4.5).

We will prove the correctness of the Basic Reclaiming Scheme by resorting to *processor demand* approach.

Definition 1 *The* processor demand *of a real-time job set β in an interval $[t_1, t_2]$, denoted as $h_\beta(t_1, t_2)$, is the sum of computation times of all jobs in β with arrival times greater than or equal to t_1 and deadlines less than or equal to t_2.*

A fundamental result in the feasibility analysis of task systems scheduled by EDF is the following:

Theorem 1 (from [5, 12]) *A set of independent real-time jobs β can be scheduled (by EDF) if and only if $h_\beta(t_1, t_2) \leq t_2 - t_1$ for all intervals $[t_1, t_2]$.*

The proof of the following theorem, which establishes an upper bound on the processor demand of *periodic* task sets scheduled by EDRA, is presented in Appendix.

Theorem 2 *When scheduled by EDRA and a nominal speed S_p, the processor demand of a periodic task set T in any interval $[t_1, t_2]$ does not exceed $\frac{U_{tot}}{S_p}(t_2 - t_1)$.*

Proposition 1 *All the deadlines of a periodic task set T scheduled by BRS are met, provided that $S_p \geq U_{tot}$ and $u_B = 1 - \frac{U_{tot}}{S_p}$.*

Proof: In [19], it was proven that for a set of aperiodic jobs A scheduled by a TBS with capacity u_B:

$$h_A(t_1, t_2) \leq (t_2 - t_1)u_B \tag{3}$$

combining (2) with (3), and noting that $u_B = 1 - \frac{U_{tot}}{S_p}$, for any interval $[t_1, t_2]$ we get:

$$\begin{aligned} h_{A \cup T}(t_1, t_2) &= h_A(t_1, t_2) + h_T(t_1, t_2) \leq \\ & (t_2 - t_1)(u_B + \frac{U_{tot}}{S_p}) = (t_2 - t_1) \end{aligned}$$

Hence, the processor demand in each interval is smaller than or equal to the length of the interval. □

4.3 Mutual Reclaiming Scheme (MRS)

The second reclaiming mechanism that we propose is based on the generalization of the α-queue mechanism and EDRA to include *aperiodic jobs* in addition to periodic tasks. **This effectively results in an integrated mechanism through which periodic tasks and aperiodic jobs can mutually reclaim their unused computation times**. In other words, the restriction of the previous scheme in which a periodic task (aperiodic job) can claim only the slack of another periodic task (aperiodic job) is removed.

Just as before, upon arrival, a deadline is assigned to aperiodic jobs through TBS mechanism. However, the α-queue is also updated to include the information about the worst-case remaining execution time of the aperiodic job, its deadline, ready time and EDF* priority. At dispatch time, each periodic task or aperiodic job, inspects the $\alpha-$queue and reclaims available slack by taking into account its EDF* priority.

Note that the addition of aperiodic jobs to the $\alpha-$queue and EDRA mechanism requires that we re-visit the definitions and notations of EDRA. First, we will assume that the default speed used to execute soft aperiodic jobs (namely, S_a) is the nominal speed for any job J_i. In addition, $\mathcal{S}^{can}$ will denote the "canonical" (static) schedule in which each task T_i (aperiodic job J_i) presents its worst-case workload $C_i(c_i)$ at every instance and

- S_{n+i} : the current speed of soft aperiodic job J_i
- $\widehat{S_{n+i}}$: the nominal speed of J_i (set to S_a)
- $rem_{n+i}(t)$: the remaining execution time of J_i at time t in $\mathcal{S}^{can}$
- $w^S_{n+i}(t)$: the remaining worst-case execution time of J_i under the speed S at time t in actual schedule
- d^*_{n+i}: The EDF* priority level of J_i
- $\epsilon_{n+i}(t)$: The earliness of J_i at time t in actual schedule, defined as $\epsilon_{n+i}(t) = \sum_{a|d^*_a \le d^*_{n+i}} rem_a(t) - w^{\widehat{S_{n+i}}}_{n+i}(t)$

Figure 6. Additional $alpha$-queue related notation

runs with nominal speed S_p (S_a). Finally, to ensure a compact notation, we will use indices beyond n (the number of periodic tasks) to refer to soft aperiodic jobs (Figure 6).

Thanks to this notation, EDRA shown in Figure 2 can be used with very little change for MRS: In Step 3, at the time of the arrival of an *aperiodic job* J_i, TBS will assign a deadline according to its rules, and the information about its EDF* deadline (d^*_{n+i}) and its remaining worst-case execution time under its "nominal speed" S_a (i.e. rem_{n+i}) will be "pushed" to the $\alpha-$queue. Naturally, the rules must be re-edited to substitute the expression "task T_x or aperiodic job J_x" for "task T_x" and care must be taken to ensure that the index of the $\alpha-$queue related parameters of the job J_x are introduced as $n + x$ (see above); but we choose not to duplicate the entire algorithm because of space limitations.

The correctness of MRS can be established by explicitly writing and evaluating the processor demand equations (as done in the proof of Theorem 2); however it can be also verified by using first principles: Equation (3) basically states that the aperiodic jobs, when their deadlines are assigned by TBS, *collectively* act as a periodic task with utilization u_B *from the processor demand point of view*. This in turn, implies that the periodic task set T with *effective utilization* $\frac{U_{tot}}{S_p}$ and aperiodic jobs scheduled by TBS, *can be considered as a single periodic task set* T' with *effective utilization* $\frac{U_{tot}}{S_p} + u_B \le 1.0$ scheduled solely by DRA (since periodic tasks and aperiodic jobs use $\alpha-$queue in MRS for reclaiming). But we know that DRA can schedule all periodic task sets whose *effective* utilization does not exceed 1.0 ([2, 3]), justifying the correctness.

4.4 Bandwidth Sharing Scheme (BSS)

While BRS and MRS provide a framework to reclaim unused computation times of periodic tasks and soft aperiodic jobs without compromising the feasibility, it is possible to further refine the system's adaptation to current workload conditions. In all previous schemes a fraction of CPU capacity (u_B) was *reserved* for TBS to execute the aperiodic jobs, *regardless of the actual aperiodic load*. But if the actual aperiodic load of the system turns out to be relatively low compared to u_B, in many cases TBS will be idle and some CPU capacity will be wasted. Note that this scenario is **fundamentally different** from having soft aperiodic tasks arrive with an expected rate and then complete early (without presenting their worst-case number of CPU cycles): instead, **we consider the case where their arrival rate is significantly low**. Under this scenario, the mutual reclaiming algorithm would *not* be able to provide compensation, because the unused computation time of aperiodics that do *not* arrive would *not* be in the $\alpha-$queue. For such scenarios, it may be more efficient (from energy consumption point of view) to dynamically *use* some capacity of TBS to further reduce the speed of periodic tasks.

To achieve this, we modify MRS in the following way: when TBS is idle, and there are ready periodic tasks in the system, the algorithm will create a "ghost" job J_g, and will push its worst-case execution time rem_g to the $\alpha-$queue. However, the *actual* execution time of J_g will be zero: that is, we consider it as a job that completes as soon as it is dispatched. The net effect of this approach is that the *periodic tasks* will now *find* the "unused" computation time of J_g in the $\alpha-$queue and will be able to reclaim it by the rules of MRS.

However, a number of details need to be worked out. In particular, what will be the worst-case execution time (and hence, the deadline) of the ghost job J_g? Observe that we are effectively using the bandwidth of aperiodic jobs aggressively in order to be able to reclaim it in the $\alpha-$queue; thus, choosing a very large worst-case execution time may result in a potentially late deadline and degraded responsiveness for future aperiodic jobs. **To prevent excessive degradation in responsiveness, we choose to create the ghost job only when there are no waiting aperiodic jobs and current time t is larger than the current deadline d_c of TBS.**

Let NPD be the *earliest* deadline among the deadlines of all ready and yet-to-arrive *periodic* task instances when a ghost job J_g is created at time t. The worst-case execution time rem_g will be computed as:

$$rem_g = \lfloor u_B \cdot (NPD - 1 - t) \rfloor \tag{4}$$

Observe that this choice guarantees that the "deadline" of J_g will be *smaller* than NPD; since $d_{c+1} = t + \frac{\lfloor u_B \cdot (NPD-1-t) \rfloor}{u_B} \le NPD - 1$. In other words, we are introducing a ghost task with *the largest possible slack that could be still used by the periodic task with the earliest deadline in the future.* This is because, according to the speed adjustment rules, a periodic task T_k can use the slack of another task/job X only when $d^*_k > d^*_X$. The correctness of BSS follows from that of MRS: for every schedule generated by BSS, one can imagine a 'parallel' scenario in which whenever a 'ghost' job is created by BSS, an aperiodic job (with rem_g as above and actual execution time 0) arrives and is subsequently scheduled by MRS. Both algorithms would generate exactly the same schedules in this case. Since MRS preserves the feasibility, so does BSS.

4.5 Experimental Evaluation

We implemented all the schemes we proposed in this paper in our scheduling simulator in order to evaluate their relative performance for three different metrics under consideration. Specifically, we implemented: (a) **Static**, which does not use any advanced reclaiming scheme (except switching to power-down mode whenever there is no ready task), (b) **BRS**, which is based on using EDRA for the periodic tasks and the dynamic TBS deadline update technique of [18] for soft aperiodic jobs, (c) **MRS**, which uses the EDRA algorithm to reclaim unused computation time from $\alpha-\text{queue}$ for both periodic tasks and aperiodic jobs, and (d) **BSS**, which dynamically and aggressively uses the bandwidth of TBS to further slow down periodic tasks. The basic simulation settings are identical to those given in Section 4.1. However, one important addition is the fact that we also investigated the effect of the variability in *the actual workload* on the performance metrics *Energy, Average Response Time* and *Energy * Average Response Time*. For a given CPU speed, the variability in the actual workload is achieved by modifying the $\frac{BCET}{WCET}$ ratio (that is, the best-case to worst-case execution time ratio). We ran experiments where the actual execution time (of periodic tasks and aperiodic jobs) follows a normal probability distribution function. The mean and the standard deviation are set to $\frac{WCET+BCET}{2}$ and $\frac{WCET-BCET}{6}$ respectively, for a given $\frac{BCET}{WCET}$, as suggested in [20]. The choice of this mean and standard deviation ensures that, on the average, 99.7% of the execution times fall in the interval $[BCET, WCET]$. We varied the $\frac{BCET}{WCET}$ ratio from 0.1 to 1.0 with increments of 0.1.

Since we have four important parameters (namely, S_a, S_p, U_w and $\frac{BCET}{WCET}$), we will comment on the effect of each variable when others are kept constant. Specifically, we will focus on the case where $S_p = 0.6, S_a = 0.7, U_w = 0.1, \frac{BCET}{WCET} = 0.1$, and we will present the trends for each of the four performance metrics by modifying one of the variables while others assume the specified values. We report the performance of the schemes by normalizing with respect to that of *Static* (at $U_w = 0.1$ for the curves where the nominal aperiodic load varies, at $S_p = 1.0$ for the curves where S_p varies, at $\frac{BCET}{WCET} = 1.0$ when the actual execution times vary and at $S_a = 1.0$ when S_a varies). The experiments led us to the following conclusions:

• **Energy:** The simulation results (Figures 7 and 8) indicate that the relative energy consumption ordering of four techniques is the same throughout the entire spectrum of variables: The best technique is BSS, followed by MRS and BRS. The static scheme was consistently outperformed by other schemes. The clear domination of BSS can be explained by its aggressive use of the TBS' bandwidth to further slow down periodic tasks. Further, we observe that its relative energy performance is better at high S_a or S_p values, since being able to adopt lower speeds by using TBS' bandwidth becomes crucial at energy-consuming (high) default speeds. The improvement in energy savings increases with respect to Static as the ratio $\frac{BCET}{WCET}$ decreases, as the opportunities for reclaiming become more frequent. Observe that BSS is able to reduce the energy consumption even when $\frac{BCET}{WCET} = 1.0$ (each task takes worst-case execution time), since it exploits the unused bandwidth originally reserved for *aperiodic* jobs.

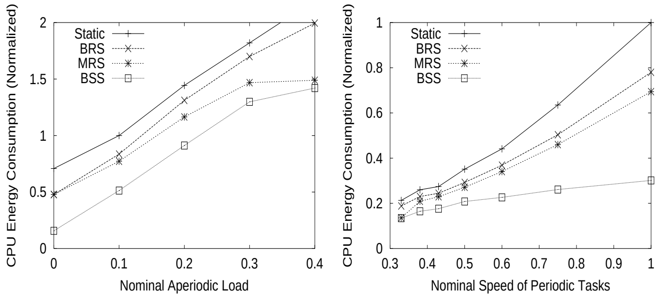

Figure 7. Energy consumption of the proposed schemes as a function of the aperiodic load (left) and the nominal speed of periodic tasks (right)

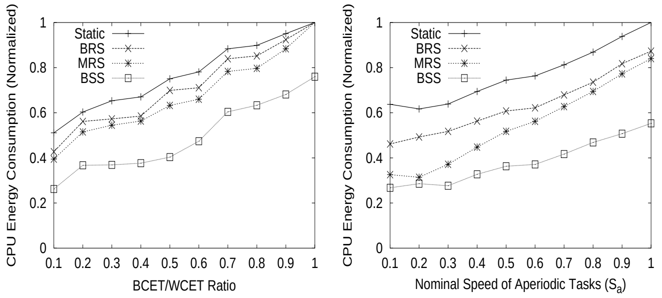

Figure 8. Energy consumption of the proposed schemes as a function of the $\frac{BCET}{WCET}$ ratio (left) and the nominal speed of aperiodic tasks (right)

• **Average Response Time:** When we consider responsiveness dimension (Figures 9 and 10), the picture is reversed once again. Static, MRS and BRS yield comparable response times, and they consistently outperform BSS. Further, at high $\frac{BCET}{WCET}$ ratio values, the BSS's responsiveness further degrades, since aggressively using the bandwidth of TBS does not usually pay off by early completions. Though the trends for Static, MRS and BRS are similar, we observe that MRS and BRS are likely to experience sharp increases in response time when the system approaches the overload conditions. This is due to the fact that reclaiming unusued computation time at high effective utilization values has a clearly distinguishable and negative impact on soft aperiodic jobs, since these jobs are often *delayed* by periodic tasks which reclaim unused CPU times.

• **Energy * Average Response Time:** The simulation results (Figures 11 and 12) indicate that important energy savings of BSS help promote the hybrid performance metric as well, *but with some noteworthy exceptions*. We observe that when $\frac{BCET}{WCET}$ ratio exceeds 0.5, the performance of BSS quickly degrades and it becomes the *worst* technique. This is due to the excessive increase observed in response time when reclaiming opportunities are few and aggressive use of bandwidth does

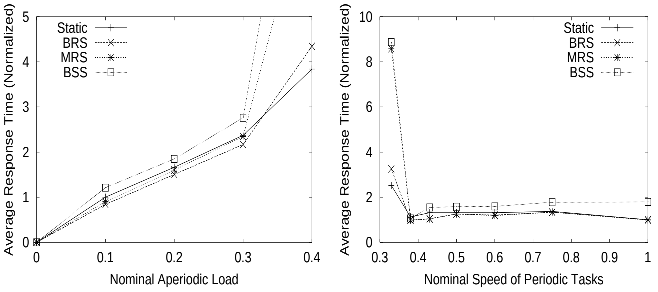

Figure 9. Average response time of the proposed schemes as a function of the aperiodic load (left) and the nominal speed of periodic tasks (right)

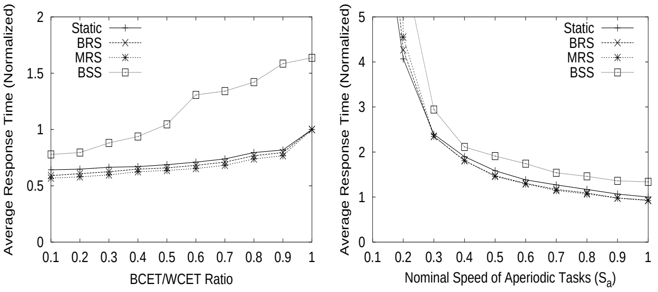

Figure 10. Average response time of the proposed schemes as a function of the $\frac{BCET}{WCET}$ ratio (left) and the nominal speed of aperiodic tasks (right)

not provide adequate compensation. At this high $\frac{BCET}{WCET}$ ratios, MRS yields the best results, because of its non-aggressive reclaiming approach. At low S_a values, the system tends to enter the "overloaded zone" rather quickly. Based on our experiences, we can propose the following rule of thumb for the choice of S_p and S_a: The total *effective* utilization of the system should not exceed 0.8 to avoid drastic and sudden degradation in *Energy * Average Response Time* performance. If the $\frac{BCET}{WCET}$ ratio is likely to be high, then using MRS or BRS is the most reasonable decision, otherwise BSS is the best technique.

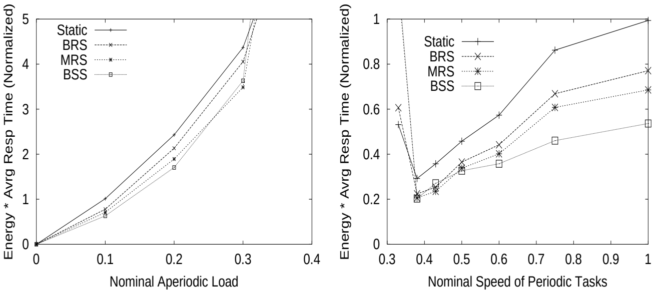

Figure 11. *Energy * Average Response Time* **performance of the proposed schemes as a function of the aperiodic load (left) and the nominal speed of periodic tasks (right)**

5 Conclusion

In this paper, we explored the energy/delay tradeoffs involved in scheduling hybrid task sets on DVS-enabled processors. We proposed *Energy * Average Response Time* as a

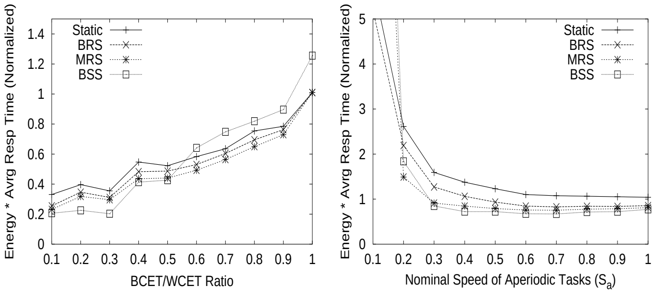

Figure 12. *Energy * Average Response Time* **performance of the proposed schemes as a function of the $\frac{BCET}{WCET}$ ratio (left) and the nominal speed of aperiodic tasks (right)**

metric that combines two important dimensions of the system performance. Our framework combines the Dynamic Reclaiming Algorithm [2] and Total Bandwidth Server [19] mechanisms in variable-speed settings. In addition to the static algorithm, we presented two schemes (Basic Reclaiming Scheme and Mutual Reclaiming Scheme) that dynamically re-use available system slack. Finally, Bandwidth Sharing Scheme allows the periodic tasks to aggressively use the bandwidth of TBS. Our experiments indicate that Bandwidth Sharing Scheme exhibits the best performance over a wide range of workload settings. However, if the actual variability in the workload is low, then the aggressive slow-down approach of Bandwidth Sharing Scheme affects the responsiveness *and* the composite metric negatively. In such scenarios, Mutual Reclaiming Scheme gives the best results.

References

[1] H. Aydin, R. Melhem, D. Mossé and P.M. Alvarez. Determining Optimal Processor Speeds for Periodic Real-Time Tasks with Different Power Characteristics. In *Proceedings of the 13th EuroMicro Conference on Real-Time Systems (ECRTS'01)*, 2001.

[2] H. Aydin, R. Melhem, D. Mossé and P.M. Alvarez. Dynamic and Aggressive Power-Aware Scheduling Techniques for Real-Time Systems. In *Proceedings of IEEE Real-time Systems Symposium (RTSS'01)*, 2001.

[3] H. Aydin. Enhancing Performance and Fault Tolerance in Reward-Based Scheduling. Ph.D. Dissertation, University of Pittsburgh, 2001. Available on-line at http:// www.cs.gmu.edu /˜aydin/dissertation.

[4] H. Aydin, R. Melhem, D. Mossé and P.M. Alvarez. Power-aware Scheduling for Periodic Real-time Tasks. *IEEE Transactions on Computers*, 53 (10), pp. 584–600, May 2004.

[5] S. Baruah, R. Howell and L. Rosier. Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-time Tasks on One Processor. In *Real-Time Systems (2)*, 1990.

[6] H. Chetto and M. Chetto. Some Results of the Earliest Deadline Scheduling Algorithm. *IEEE Transactions on Software Engineering*, 15(10), pp. 1261–1269, 1989.

[7] I. Davis, K.W. Tindell, and A. Burns. Scheduling slack time in fixed priority pre-emptive systems. In *Proceedings of the Real-Time Systems Symposium*, 1993.

[8] T.M. Ghazalie, and T.P.Baker. Aperiodic servers in deadline scheduling environment. *Real-Time Systems Journal*, 9 (1), pp.31–68,1995.
[9] F. Gruian. Hard Real-Time Scheduling using Stochastic Data and DVS Processors. In *Proceedings of International Symposium on Low-Power Electronics and Design*, 2001.
[10] I. Hong, G. Qu, M. Potkonjak and M. Srivastava. Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processors. In *Proceedings of IEEE Real-Time Systems Symposium (RTSS'98)*, 1998.
[11] I. Hong, M. Potkonjak and M. B. Srivastava. On-line Scheduling of Hard Real-Time Tasks on Variable Voltage Processor. In *Computer-Aided Design (ICCAD)'98*. .
[12] K. Jeffay and D. L. Stone. Accounting for Interrupt Handling Costs in Dynamic Priority Task Systems. In *Proceedings of IEEE Real-Time Systems Symposium (RTSS'93)*, 1993.
[13] C.L. Liu and J.W.Layland. Scheduling Algorithms for Multiprogramming in Hard Real-time Environment. *Journal of ACM* 20(1): pp.46–61, 1973.
[14] S. Lindsey, C. S. Raghavendra, and K. Sivalingam. Data Gathering in Sensor Networks using the Energy*Delay Metric. In *Proceedings of the IPDPS Workshop on Issues in Wireless Networks and Mobile Computing*, 2001.
[15] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proc. ACM Symposium on Operating Systems Principles (SOSP'01)*, 2001.
[16] S. Ramos-Thuel and J. P. Lehoczky, On-Line Scheduling of Hard Deadline Aperiodic Tasks in Fixed-Priority Systems. In *Proceedings of IEEE Real-Time Systems Symposium (RTSS'93)*, 1993.
[17] B. Sprunt, L. Sha, J. P. Lehoczky. Aperiodic Task Scheduling for Hard Real-Time Systems. *Real-Time Systems*. 1(1): 27-60, 1989.
[18] M. Spuri, G.C. Buttazzo, and F. Sensini. Robust Aperiodic Scheduling under Dynamic Priority Systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS'95)*, 1995.
[19] M. Spuri and G. Buttazzo. Scheduling Aperiodic Tasks in Dynamic Priority Systems. In *Real-Time Systems*, vol. 10, 1996.
[20] Y. Shin and K. Choi. Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems. In *Proceedings of the 36th Design Automation Conference, DAC'99*, 1999.
[21] A. Vahdat, A. Lebeck, and C. Ellis. Every Joule is precious: The case for revisiting operating system design for energy efficiency. In *SIGOPS European Workshop*, 2000.
[22] M. Weiser, B. Welch, A. Demers and S. Shenker. Scheduling for Reduced CPU energy. In *USENIX Symposium on Operating Systems Design and Implementation*, 1994.
[23] F. Yao, A. Demers and S. Shenker. A Scheduling Model for Reduced CPU Energy. *IEEE Annual Foundations of Computer Science (FOCS'95)*, 1995.

APPENDIX

Theorem 2: When scheduled by EDRA and a nominal speed S_p, the processor demand of a periodic task set T in any interval $[t_1, t_2]$ does not exceed $\frac{U_{tot}}{S_p}(t_2 - t_1)$.

Proof: First, note that the statement holds for $\mathcal{S}^{can}$ in which each task/job presents its worst-case workload and there is no reclaiming; since in this case:

$$h_T^{CAN}(t_1, t_2) = \sum_{t_1 \le r_{i,j} \le d_{i,j} \le t_2} \frac{C_{i,j}}{S_p \cdot P_i} = \sum \lfloor \frac{t_2 - t_1}{P_i} \rfloor \frac{C_i}{S_p} \le (t_2 - t_1) \sum \frac{C_i}{S_p P_i} = \frac{t_2 - t_1}{S_p} U_{tot}$$

For the case where the reclaiming is done through EDRA (with nominal speed S_p), we will show that for any interval $[t_1, t_2]$,

$$h_T^{EDRA}(t_1, t_2) \le h_T^{CAN}(t_1, t_2) \quad (5)$$

where $h_T^{EDRA}(t_1, t_2)$ is the processor demand of the periodic task set T in the actual schedule produced by EDRA, which will prove the claim.

The proof is by induction: we will show that (5) holds after each reclaiming point t_r, and for any interval $t_r \le t_1 \le t_2$. First, note that, if there is no reclamation then $h_T^{EDRA}(t_1, t_2) = h_T^{CAN}(t_1, t_2)$, and the statement holds for every interval $[t_1, t_2]$. Thus, assume that the statement holds after the first k reclaiming points.

Consider the $(k+1)^{st}$ reclaiming occurring at $t = t_i$ and affecting the speed of the periodic task T_i which is about to be dispatched. It is clear that for any interval $[t_a, t_b]$ such that $t_b \ge t_a > t_i$ the statement still holds: the CPU allocations of the instances that will arrive in the future are identical with those in $\mathcal{S}^{can}$(since these instances, by default, have the nominal speed in both $\mathcal{S}^{can}$ and actual EDRA schedule). We choose to ignore the task instances that will arrive in the future (at $t_x > t_i$) in the following processor demand evaluation, since they will make exactly the same contributions to the demand expressions in both $\mathcal{S}^{can}$ and the actual schedule.

Observe that, though the arrival time of currently ready tasks are in the past, for the sake of computing the processor demand, we can consider them as if they arrived at $t = t_i$ (with the worst-case remaining execution time equal to $w^{S_x}(t_i)$ for the actual schedule, and $rem_x(t_i)$ for $\mathcal{S}^{can}$, for any ready task T_x). Further, we can restrict our analysis to intervals $[t_1, t_2]$ such that t_2 is a task deadline [5]. In other words, to show the validity of the inequality (5), we will prove that

$$h_T^{EDRA}(t_i, d_a) \le h_T^{CAN}(t_i, d_a) \quad (6)$$

for every d_a which is the deadline of a ready task. First, observe that, for any ready task T_a at time t_i in the actual schedule, $h_T^{CAN}(t_i, d_a) = \sum_{j | d_j^* < d_a^*} rem_j(t) + rem_a(t)$.

Now, for any deadline d_a such that $t_i < d_a^* < d_i^*$, $h_T^{EDRA}(t_i, d_a)$ must be zero since T_a is clearly completed at $t = t_i$ (otherwise, T_i would not be dispatched). Thus, (6) trivially holds for such deadlines.

The second case we examine is $d_a = d_i$: in this case, $h_T^{EDRA}(t_i, d_i) = \sum_{j | d_j^* < d_i^*} rem_j(t) + rem_i(t) - w_i^{\widehat{S_i}}(t) + w_i^{\widehat{S_i}}(t)$ (due to EDRA reclaiming rule). But this is exactly $h_T^{CAN}(t_i, d_i)$, and (6) still holds.

The last possibility we consider is the case where $d_a \ge d_i$. Then:

$$h_T^{EDRA}(t_i, d_a) = h_T^{EDRA}(t_i, d_i) + \sum_{T_k \text{ is ready and } d_i^* < d_k^* \le d_a^*} w_k^{\widehat{S_k}}(t)$$

The last term in the expression is obtained by considering that tasks effectively return to their nominal speeds when preempted. And we can write:

$$h_T^{CAN}(t_i, d_a) = h_T^{CAN}(t_i, d_i) + \sum_{T_k \text{ is ready and } d_i^* < d_k^* \le d_a^*} rem_k(t)$$

But in [2, 3] it was proven that for any ready task T_i, $rem_i(t) \ge w_i^{\widehat{S_i}}(t)$ during the execution of GDRA algorithm. That proof can be repeated almost verbatim to show that the same holds for EDRA. Combining this with the fact that $h_T^{EDRA}(t_i, d_i) = h_T^{CAN}(t_i, d_i)$ (see above), the third case is also verified, proving the statement. □