Energy-Constrained Scheduling for Weakly-Hard Real-Time Systems

Tarek A. AlEnawy and Hakan Aydin
Computer Science Department
George Mason University
Fairfax, VA 22030
{thassan1,aydin}@cs.gmu.edu

Abstract

In this paper, we explore performance optimization problems for real-time systems that have to rely on a fixed energy budget during an operation/mission. We adopt the weakly-hard real-time scheduling paradigm to ensure a predictable performance for all the tasks: Our aim is to minimize the number of dynamic failures (in terms of (m,k)-firm deadline constraints) while remaining within the energy budget. We prove that this problem is NP-Hard in the strong sense even for an ideal DVS architecture with continuous speed spectrum. We propose techniques to statically compute the speed of the CPU in order to meet the (m,k)-firm deadline constraints. We present on-line speed adjustment algorithms to exploit the slack time of skipped and completed jobs. Through extensive simulations, we show how the performance can be significantly improved by selectively dispatching jobs by considering their energy costs as well as their contribution to the system performance.

1. Introduction

With the proliferation of wireless and portable computing and communication devices that rely on battery power, *energy-aware* system design has recently received considerable attention by the research community. The *Dynamic Voltage Scaling (DVS)* technique [2, 8, 20], which is based on reducing the supply voltage and clock frequency of the CPU on-the-fly, proved to be a powerful energy management tool for both general purpose and real-time (RT) systems. While it is possible to obtain significant energy savings through DVS, the CPU frequency/speed reduction results in an increase in response times. Early work on the so-called 'RT-DVS' problem focuses on providing feasibility guarantees with the reduced CPU speed to maximize the energy savings, for various task/system models and on-line/off-line scheduling algorithms [2, 18, 20, 21, 26, 27]. Thus, these studies can be seen as part of the broader *energy-aware* computing research, where the aim is to achieve the performance objectives with minimum energy consumption.

Energy-constrained real-time systems. More recently, the research community started to consider a different type of problem. In the *energy-constrained* settings, the energy is more than an important design dimension; *it is a hard constraint* [17]. The problem is to make sure that the system, with a finite energy budget, remains functional and delivers an acceptable performance during a well-defined *operation/mission time*. In the specific case of real-time operation, if the system's energy budget does not allow the execution of all the jobs even with DVS, the best strategy may be to *skip* a few jobs in a controlled fashion without running out of energy in the middle of the mission. In fact, deliberately skipping some jobs reduces the workload, and at the same time, it allows the use of a lower CPU speed to meet the deadlines of the other (non-skipped) jobs to compound the energy savings. Thus, **energy-constrained RT scheduling** aims to **maximize the system performance within the limits of available energy [1, 24, 25].** In [1], it is shown that the problem of deciding the *feasibility* of a task set on a CPU with limited energy budget and discrete speed levels is NP-Hard.

Most research on energy-constrained RT scheduling follows the *reward-based* task model, where it is assumed that the timely completion of each job will accrue a certain *reward/value*, capturing the job's contribution to the overall system's utility. Thus, existing studies focus on maximizing the system reward within available energy [1, 14, 24, 25]. Though the reward-based framework is applicable in settings where the reward of each job is accurately and a priori quantifiable, this information may not be always available to the designer. In this paper, we explore an alternative and more practical solution, through the adoption of the *weakly-hard real-time scheduling* paradigm, which is a widely-used approach to deal with *overloaded* (but not power-aware/variable-speed) RT systems.

Weakly-hard real-time systems research is motivated by the observation that for many real-time applications (which are periodic in nature) some deadline misses are acceptable as long as they are spaced distantly/evenly. As indicated in [11, 12, 15], prevalent examples include multimedia processing, real-time communication and embedded control applications. A number of closely-related models were proposed by several research groups over the years. In the *skip* model, a task's tolerance to deadline misses is characterized by the skip parameter *s*: in any *s* consecutive instances of the task at most one can miss its deadline [7, 15]. The more general *(m,k)*-firm deadline model [11, 22, 23] requires that each task meet at least *m* deadlines in every consecutive *k* instances. If this constraint is violated in any time window, the system is said to exhibit a *dynamic failure* (implying possible degradation in system performance or quality-of-service). Figure 1 shows a sample schedule for a (2,3)-firm real-time task over six consecutive periods. The third instance is skipped, but the task meets its (2,3)-firm deadline constraint in the first execution window that encompasses the first three task periods, thanks to the timely completion of the first and second task instances. The same holds for the second execution window. However, when the fifth instance is skipped, the task experiences a dynamic failure in the third window.

In the Dynamic Window Constrained Scheduling (DWCS) model motivated by the real-time packet scheduling applications, a given task needs to complete at least *m* instances before their deadlines in every *non-overlapping* window of *k* instances [19, 28, 29]. In the schedule of Figure 1, with DWCS, there would be only two execution windows: Window 1 would comprise the first three task periods, while Window 2 would include the last three. Observe that a schedule satisfying the *(m,k)*-constraints (in the original sense) also satisfies the DWCS model, but the reverse is not true. Finally, Bernat and Burns proposed the generic weakly-hard RT scheduling model [5, 6], which encompasses several alternative *(m,k)*-firm task models. In this paper, we adopt the (*m*,*k*)-firm deadline model used in [11, 22, 23] since it is the most commonly used framework.

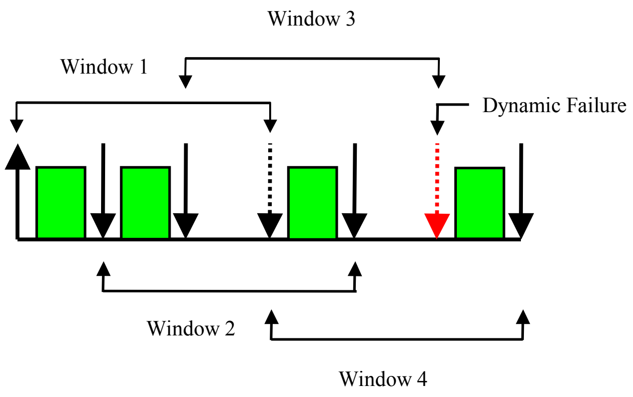

Figure 1. Dynamic failures in a schedule of a (2,3)-firm RT task

This work. The motivation of this research effort is to recognize the applicability of the weakly-hard real-time task model to energy-constrained applications. As opposed to the reward-based framework where each task/job has a statically-assigned utility value, the weakly-hard real-time task model provides greater flexibility in expressing *firm* deadline requirements: The performance will remain acceptable as long as each task meets its *(m,k)*-firm deadline requirement, regardless of the specific jobs selected for execution. Moreover, it is known that the performance requirements of audio, video, multimedia and RT communication tasks are best expressed by the *(m,k)*-firm deadline constraints [6,29]. Since these applications are considered essential for emerging battery-operated (and hence, energy-constrained) devices, the adoption of the weakly-hard RT system model should prove very useful in practice. Despite the potential prospects of the weakly-hard RT task model, a number of technical challenges need to be overcome. To start with, choosing an appropriate CPU speed to avoid or minimize the dynamic failures is a non-trivial problem. In addition, devising *efficient on-line* mechanisms to exploit the slack time of skipped and/or completed jobs is crucial to be able to meet the hard energy constraint of the system. The main **contributions** of this paper are as follows:

- We formally characterize the energy-constrained weakly-hard RT scheduling problem and show that it is NP-Hard in the strong sense. The result remains valid even when one assumes that the task set is trivially schedulable when the energy constraint is excluded. In other words, the introduction of the hard energy constraint has a deep impact on the complexity of the problem.
- We derive a sufficient (but not necessary) condition that characterizes the solution of the problem. For scarce-energy settings where this condition does not hold, we develop a solution framework that consists of:
    - Carefully selecting a *mandatory* workload among all the jobs by considering their impact on overall system performance in terms of minimizing the dynamic failures,
    - Computing a safe CPU speed to complete all the mandatory jobs before their deadlines, and
    - Applying dynamic/on-line speed adjustment techniques to aggressively reclaim the slack-time of skipped and completed jobs to boost the energy savings.
- We present two main classes of algorithms: the *greedy* schemes a priori select the mandatory workload for execution and attempt to minimize the number of dynamic failures by on-line speed adjustment techniques. The *energy-density* schemes go one step further by prioritizing the mandatory jobs by considering their 'energy cost' and contributions to the overall system performance metric.
- We present simulation results evaluating the performance of all the algorithms presented in the paper, over a wide range of system parameters. We report that the energy-density schemes exhibit a superior advantage, especially when the system's energy budget is significantly low.

2. System Model

2.1 Power and energy consumption model

The target platform of this study is a single processor system whose only power source is a battery. In order to remain functional, the system's energy level must remain in a 'safe' energy range [E_{min}, E_{max}] at all times, where E_{max} refers to the maximum battery capacity, and E_{min} denotes the minimum energy threshold below which safe operation is not guaranteed [25]. Thus, the system has *limited* initial amount of energy, referred to in this paper as *energy budget* E_{budget} , whose value (when the battery is fully charged) is $E_{budget} = E_{max} - E_{min}$. We further assume that the system has to remain functional throughout a time interval [0,X]; that is the total CPU energy consumption cannot exceed E_{budget} for X time units (in other words, the system is subject to a *hard* energy constraint). Throughout this paper, we refer to X as the system's *mission time*. We also assume that battery re-charging is not possible during the mission.

We further assume that the system has DVS capability, where the processor speed (frequency) and supply voltage can be dynamically adjusted. We distinguish two modes: *execution mode* and *stand-by mode*. In the stand-by mode, the CPU does not execute any tasks, and consumes only the stand-by power denoted by g^{stb}. The CPU switches to stand-by mode if it is idle. In the execution mode, the CPU speed can vary between a lower bound (S_{min}) and an upper bound (S_{max}). In this case, the power consumed is a function of the CPU speed/frequency; we denote this *dynamic* power consumption by *g(S)*. In accordance with [2,20,26], we assume that $g(S) \propto S^3$. For convenience, we normalize the speed values with respect to S_{max}; i.e. $S_{max} = 1.0$. In any time interval [t_1,t_2], the total energy consumption is the integral of power consumption function, which includes the stand-by and dynamic power consumption components.. We also assume that time and energy overheads due to CPU speed changes are negligible.

We adopt an *inter-task DVS* model; that is, we assume that the CPU speed can be changed only at task completion or preemption points, following [2, 20, 26].

2.2 Task model

We consider a set of n independent periodic RT tasks $\Gamma = \{T_1, T_2, \ldots, T_n\}$. Each task $T_i = (C_i, P_i, m_i, k_i)$ is characterized by four parameters: the worst-case execution time C_i under maximum speed[1], a period P_i and the parameters m_i and k_i which specify the task's tolerance to deadline misses. Tasks are assumed to be (m_i,k_i)-firm; that is at least m_i instances of task T_i must meet their deadlines in every window of k_i consecutive instances ($m_i \leq k_i$). If this constraint is violated in any such window, task T_i is said to exhibit a *dynamic failure*. We further assume that the relative deadline D_i is equal to the period P_i . $T_{i,j}$ denotes the jth instance of task T_i . We use the terms hyperperiod and frame interchangeably to refer to the least common multiple of all task periods: $P = lcm(P_1, P_2,\ldots, P_n)$. We assume preemptive scheduling, and that the preemption and speed change overheads can be incorporated in C_i if necessary. The process descriptor of T_i is augmented to include two fields related to the CPU speed: a nominal speed S_i^{nom}, which is the default speed assigned to the task when it is about to be dispatched, and an actual speed S_i, which is the speed at which the task is being executed at the specific time instant. Under a constant speed S, the execution time of task T_i is C_i /S. The utilization of task T_i under CPU speed S is given by $u_i(S) = C_i /(P_i\, S)$. Note that under maximum CPU speed (i.e. S = 1.0), u_i(1.0) = C_i /P_i. The aggregate utilization of the task set (under maximum speed) is given by $U_{tot} = \sum_i C_i /P_i$. In this paper, we assume that the execution time scales linearly with the CPU speed, ignoring memory stall effects. Note that this is a conservative but safe assumption since it *overestimates* the new execution time when the CPU speed is reduced [27]. Hence, it does not hurt the schedulability analysis.

We adopt the Earliest-Deadline-First (EDF) scheduling policy, which is known to be optimal from the feasibility viewpoint [16]. The goal of this paper is to study the feasibility and performance maximization problems in the context of energy-constrained systems (with *limited CPU energy budget*). Consequently, we assume that the task set *would be* feasible (schedulable) with EDF when executed at the highest CPU speed (i.e. if $\sum_i C_i /P_i \leq 1$), in the absence of energy constraints.

3. Energy-Constrained Weakly-Hard Real-Time Scheduling Problem

In this section, we formulate the scheduling problem for energy-constrained weakly-hard RT systems and discuss its computational complexity. We also explore feasibility conditions, define the system performance metric, and give an overview of our general solution approach.

Consider the following problem: **Given a weakly-hard real-time system that must remain operational during a mission interval [0,*X*], can *all* tasks meet their (*m*,*k*)-constraints (i.e. avoid dynamic failures) while remaining within the system's limited energy budget?** Unfortunately, this problem is computationally intractable even assuming an ideal DVS model with continuous CPU speed:

Theorem 1. *The problem of deciding whether it is possible to meet all the* (*m*,*k*)*-constraints of a set of periodic real-time tasks within a limited energy budget is NP-Hard in the strong sense on a DVS-enabled CPU with continuous speed.*

Proof: See Appendix A.

[1] C_i can also be considered as the worst-case number of processor cycles required by task T_i.

We would like to connect this result to the related intractability results that already appeared in the literature. First, in [1], it is shown that the problem of deciding if *all* the deadlines of a set of real-time tasks can be met subject to a limited energy budget (i.e. feasibility under energy constraint) is NP-Hard in the weak sense on a DVS-enabled CPU with *discrete* speed levels. At first, this last problem may appear as a special case of our problem where we set *m*=*k* (under this condition the (*m*,*k*)-firm model reduces to the traditional hard real-time model). However, note that Theorem 1 presents a stronger result by showing that the problem is NP-Hard *in the strong sense* even for *continuous* CPU speed. Second, Quan and Hu proved that deciding whether it is possible to meet all the *(m,k)*-constraints on an *overloaded* (but constant-speed, non-energy-constrained) CPU is NP-Hard in the strong sense [22]. In our settings, the system is *not* overloaded from the computational demand point of view (in fact, all tasks can meet their deadlines under maximum speed). However, energy is the hard constraint and the result above implies that deciding if it is possible to avoid dynamic failures on a DVS-enabled, energy-constrained CPU is equally hard.

Although the problem stated in Theorem 1 is NP-Hard in the general case, it is possible to decide some special instances easily. Consider the specific execution pattern in which the first m_i instances of task T_i in every interval $[qk_iP_i , (q+1)k_iP_i]$ (where q is a non-negative integer) are dispatched and the remaining k_i - m_i instances are *skipped* as illustrated in Figure 2. It is clear that this execution pattern meets the (*m*,*k*)-constraints provided that a suitable speed is used to meet the deadlines of *all* the dispatched jobs. A possible CPU speed to use in this context is the one corresponding to the total task utilization (under maximum speed) $S_u = U_{tot} = \sum_i C_i /P_i$: this speed is known to minimize the total energy consumption of the periodic task set without missing any deadlines [2, 20]. Let E_{limit} denote the energy needed to execute solely the dispatched jobs in the pattern discussed above using the speed S_u.

Proposition 1. *If* $E_{budget} \geq E_{limit}$, *then* ***all*** *the* (m,k)*-constraints can be met by using the speed* $S_u = U_{tot}$.

Proof: By using the speed $S_u = U_{tot}$, it is possible to guarantee that *all* deadlines in the task set will be met a priori [2, 20]. Also recall that E_{limit} represents the energy needed at this specific speed to execute the selected jobs in the pattern discussed above, which readily satisfies all the (*m*,*k*)-constraints. Therefore, by adopting this pattern and using the speed S_u, all (*m*,*k*)-constraints can be met. Note that Proposition 1 provides only a sufficient, but not necessary condition, for meeting the (*m*,*k*)-constraints.

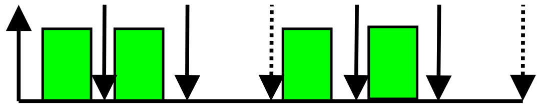

Figure 2. The "deeply-red" execution pattern of a (2,3)-firm RT task

Thus, if If $E_{budget} > E_{limit,}$ one can find a trivial solution that guarantees all the *(m,k)* constraints, and the techniques proposed in [1] can be used to further improve the system performance. **In the remainder of this paper, we focus on the settings where $E_{budget} < E_{limit}$ and we explore strategies for maximizing the system performance in these settings.** While developing our solutions we adopt the execution pattern, shown in Figure 2, according to which the first m_i instances in every non-overlapping window of k_i instances of any task T_i are *selected*, and the remaining $k_i - m_i$ instances are *skipped*. This pattern has been adopted in the weakly-hard real-time scheduling literature and is referred to as the "deeply-red" execution pattern [7, 15]. In this context, we distinguish two types of jobs: *mandatory* and *optional*.

Definition 1. *An instance* $T_{i,j}$ *of a task* T_i *is* mandatory *if and only if* $1 \leq j \bmod k_i \leq m_i$*; otherwise it is* optional *(for the case of* $m_i < k_i$*). If* $m_i = k_i$ *then all instances are mandatory.*

It is now possible to formally define E_{limit} as follows.

Definition 2. E_{limit} *is the minimum energy needed to execute all the mandatory jobs (under worst-case conditions) using the speed* $S_u = U_{tot}$.

Let us denote by M_i the number of mandatory (non-skipped) jobs of task T_i required to meet the (m,k)-constraints in $[0,X]$, according to the 'deeply-red' execution pattern. Since the execution time of each mandatory job of task T_i under the CPU speed S_u is C_i / S_u, the total amount of time the system stays in execution mode is $\sum_i M_i C_i / S_u$. Considering the stand-by and execution mode power functions and the duration of the mission X, the sum of dynamic and stand-by (i.e. the total) energy consumption can be computed as:

$$E_{limit} = g(S_u) \cdot \sum_i M_i C_i / S_u + g^{stb} \cdot (X - \sum_i M_i C_i / S_u) \qquad (1)$$

Observe that by adopting a 'deeply-red' pattern, the selected mandatory jobs are executed *early* in the schedule. This helps on-line dynamic slack reclamation since in the common case of early completions [9], the arising *slack-time* can be used to further reduce the CPU speed for the subsequent jobs and increase the energy savings. Moreover, in our severely energy-constrained problem settings, we opt to skip all the optional jobs to save energy for the mandatory jobs which are critical to meet the (m,k)-constraints .

Since it is not possible to provide a priori guarantees to meet all the (m,k)-constraints through an efficient procedure, we attempt to optimize the system performance subject to the limited available energy. We measure the system performance during the mission through the ***dynamic failure ratio (DFR)***, which is defined as:

$$DFR = \sum_{i=1}^{n} w_i DF_i \Big/ DF^{\max} \qquad (2)$$

where DF_i is the number of dynamic failures of task T_i during the mission interval $[0,X]$, DF^{max} is the total number of dynamic failures that all tasks can experience during the same interval, and w_i is the (optional) weight ($0 \leq w_i \leq 1$) indicating the relative impact of T_i's dynamic failures on overall system performance. Appropriate weight values can be chosen by the designer if additional information is available about the application semantics. It is relatively easy to evaluate DF_i: observe that the first dynamic failure of the task T_i can occur only at time $k_i P_i$, since this marks the end of the execution window of the first k_i jobs of task T_i. In addition, we may have an additional dynamic failure at every period boundary (deadline) of T_i after $t = k_i P_i$, by considering the last k_i jobs of the (sliding) execution window. This can be easily kept track of by the operating system. Following the same reasoning, the maximum number of dynamic failures that the task T_i may experience, namely DF_i^{max}, can be computed as $Max\ \{\lfloor X / P_i \rfloor - k_i + 1, 0\}$. Summing up DF_i^{max} values over all tasks yields the parameter DF^{max}.

In Section 4 we propose solutions to *minimize* the dynamic failure ratio *DFR* subject to the energy budget through the following strategies:

- We compute a *single* initial (nominal) speed (which is lower than $S_u = U_{tot}$) by considering *solely* the workload of the mandatory jobs. Hence, the energy constraint can be addressed more effectively.
- We greedily reclaim slack time resulting from skipped optional jobs and early completions of mandatory jobs, without jeopardizing the deadlines of already selected ones. We use dynamic CPU speed slow-down to further reduce the energy consumption and execute additional jobs, thus improving the dynamic failure ratio.

4. Our Solutions

The algorithms that we present in this paper generally belong to one of the two classes: The *greedy* schemes (Section 4.1) take a direct approach and attempt to execute all the *mandatory* jobs as long as the system's energy budget allows. On the other hand, the *energy-density* schemes (Section 4.2), effectively prioritize the mandatory jobs based on their energy demands and the number of (m,k)-constraints they help to meet.

4.1 Greedy schemes

The greedy schemes start by selecting *all* the mandatory jobs of *all* tasks for execution, relying on the fact that completing these jobs prior to their deadlines will enable the system to meet the (m,k)-constraints. However, as discussed above (Section 3), one needs to also determine the CPU speed at which the mandatory jobs will be executed. In addition, it is extremely crucial for the system to exploit *any* run-time opportunity to reduce the CPU speed dynamically (for example by using available slack time due to the early completions).

4.1.1 Nominal Speed Considerations and Reclaiming the Slack of Skipped Jobs

Our solutions entail pre-computing a global *nominal* (or *default)* speed for all the tasks statically, and applying the dynamic reclamation / speed adjustment techniques on-line whenever possible. At dispatch time, the speed of each job of T_i is first set to the nominal speed S^{nom}. However, its *actual* speed S_i may be even lower after the application of the dynamic slack reclamation techniques.

The nominal speed must be carefully chosen to guarantee the deadlines of the mandatory jobs. One such safe nominal speed that can be computed in linear-time is $S^{nom} = S_u = U_{tot} = \sum_i C_i / P_i$. In fact, Proposition 1 (Section 3) implies that the deadlines of all the mandatory jobs, consequently all the (m,k)-constraints, will be met with the speed S_u, provided that $E_{budget} \geq E_{limit}$. However, note that S_u is based on somewhat conservative assumptions: it guarantees that all the deadlines, including those of (skipped) optional jobs, will be met. At run-time, it is possible to further reduce the actual CPU speed, and consequently reduce the energy consumption, by observing that the schedule has idle intervals due to the optional jobs that are skipped. In fact, one can consider the skipped (optional) jobs as actually dispatched jobs with zero actual execution time. Thus, it is possible to use this slack time for *dynamic* slow-down making it possible to improve the system's performance even when $E_{budget} < E_{limit}$.

Example 1. Consider the weakly-hard real-time task set shown in Table 1. The system must remain operational for a mission time of $X = 60$ time units. Assume that, in the execution mode the CPU power consumption as a function of speed is $g(S) = S^3$, while the stand-by power is $g^{stb} = 0.025$.

	C_i	P_i	U_i	m_i	k_i
T_1	6	60	0.1	1	1
T_2	9	30	0.3	1	2
T_3	6	10	0.6	1	2

Table 1. Parameters for the task set in Example 1

Observe that for this task set $S_u = U_{tot} = 1.0$. If we select all the mandatory jobs according to the deeply-red pattern and execute them with the speed $S_u = 1.0$, then we obtain the schedule shown in Figure 3. The maximum number of dynamic failures that can occur during the mission is $DF^{max} = 7$, and the total energy consumption of all the mandatory jobs is $E_{limit} = 33.7$. If $E_{budget} = 23$ (68% of E_{limit}), then only $T_{2,1}$ and $T_{3,1}$ can complete, resulting in a total of 5 dynamic failures.

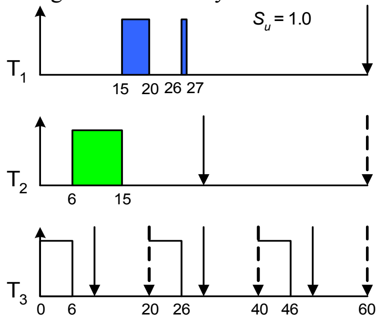

Figure 3. A schedule obtained using greedy scheme for Example 1 using nominal speed S_u *without* reclamation and $E_{budget} = E_{limit}$

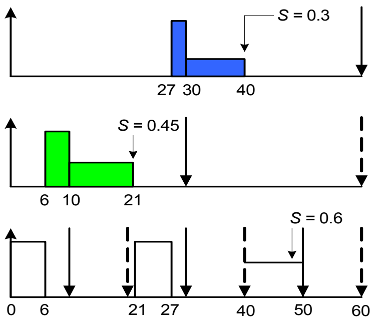

Figure 4. A schedule obtained using greedy scheme for Example 1 using nominal speed S_u *with* reclamation and $E_{budget} = 68\%$ *of* E_{limit}

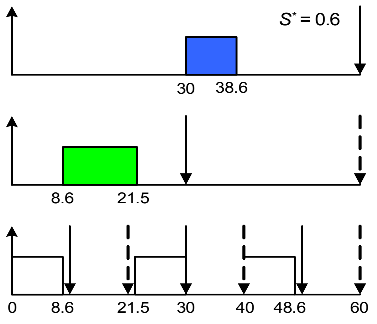

Figure 5. A schedule obtained using greedy scheme for Example 1 using nominal speed S^* *without* reclamation and $E_{budget} = 50\%$ *of* E_{limit}

Looking back at the schedule shown in Figure 3, we see that the slack time of $T_{3,2}$, which is skipped, can be effectively used to reduce the CPU speed while $T_{2,1}$ executes. Similarly, at $t = 30$, the speed of T_1 (whose start time is now delayed due to the increased response time of T_2) can be reduced to effectively use the entire CPU time in the interval [30,40]. The specific dynamic reclamation algorithm in use is DRA-OTE [1, 2] which is shown to be effective for various DVS settings. However, other algorithms (e.g. Pillai-Shin speed reduction technique [20]) can be also used to exploit the slack-time of skipped jobs with $S^{nom} = S_u$. The resulting schedule, shown in Figure 4, illustrates the fact that *all* the (m,k)-constraints can be met with $E_{budget} = 23$, which is 68% of E_{limit}. In this schedule, $T_{1,1}$ runs at speed $S = 0.3$ in [30,40], $T_{2,1}$ runs at speed $S = 0.45$ in [10,21], and $T_{3,5}$ runs at speed $S = 0.6$ in [40,50] thanks to dynamic speed slow-down.

Note that this form of slack reclamation is possible even if all the selected mandatory jobs take their worst-case execution time since the optional jobs are skipped in our framework. At this point, we can make the following important observation: Skipping the optional jobs helps in meeting the (m,k)-constraints of the energy-constrained systems in two ways. First, it reduces the effective task workload, thereby saving energy for the *mandatory* workload whose timely completion will help to meet the weakly-hard timing constraints in a predictable way. Second, it creates additional opportunities for dynamic CPU speed slow-down at run-time by the slack time it creates through the skipped optional jobs.

A natural question to consider is whether it is possible to compute a *better (lower)* nominal speed for the mandatory jobs, using the knowledge that the optional jobs will be skipped. This will effectively enable the designer to compute a uniform speed for the mandatory jobs, by incorporating the slack-time of the skipped jobs by static analysis. In fact, this question can be answered positively, as shown below.

Definition 3. *Given a weakly-hard real-time task set whose optional jobs are skipped at run-time, the* ***processor demand*** of the mandatory workload *in the interval* [t_1, t_2] *is defined as* $D(t_1, t_2) = \Sigma_i M_i(t_1, t_2) \cdot C_i$, *where* $M_i(t_1, t_2)$ *is the number of mandatory jobs of the task* T_i *released at or after* t_1, *and having deadlines less than or equal to* t_2.

Theorem 2. *All the* (m,k)*-constraints of a weakly-hard real-time task set will be met, if the mandatory jobs are executed with the speed* $S^* = \max_{L \geq 0} \{D(0,L)/L\}$.

Proof: See Appendix B.

Note that this result can be also seen as an extension of the utilization bound proposed by Caccamo and Buttazzo for the skip model in [7] to the more general (m,k) model.

Returning to the motivational example, one can verify that the processor demand of the mandatory workload is maximum in the interval [0,30], and S^* is evaluated as 21/30 = 0.7. Running all the mandatory instances with this statically computed speed yields the schedule shown in Figure 5, that satisfies all the (m,k)-constraints with an energy budget as low as 16.49 (50% of E_{limit}). Note that this improvement is achieved without considering any further dynamic adjustments (for example, T_1 may be able to use the interval [38.6,40] with the simple One-Task-Extension technique [2]).

When evaluating S^*, it is sufficient to consider L values that correspond to period boundaries up to the hyperperiod P =

$lcm(P_1, P_2, \dots, P_n)$. Nevertheless, despite the possible improvements on the energy, the computation of S^* may take pseudo-polynomial time since the value of the mandatory processor demand needs to be evaluated at every period boundary. Observe that when running the task set with the speed S^*, we no longer reclaim the slack time of skipped jobs, since they are incorporated statically while evaluating the speed S^*.

Note that the greedy schemes have aggressive nature since they select for execution all the mandatory jobs of all tasks, regardless of the available energy. Aggressively selecting all mandatory jobs can then deplete the system's energy before the end of the mission. To ensure that the system remains operational until the end of the mission when greedy schemes are used, we perform a simple check: When a job is about to be dispatched, it is executed only if the system will have enough energy to complete the mission even if the job were to present its worst-case workload; otherwise the job is skipped.

4.1.2 Reclaiming gain time due to early completions

If, at run-time, the mandatory jobs take less than their worst-case execution requirements, then it is possible to exploit the unused CPU time to improve the energy consumption by performing dynamic speed reduction. In fact, this slow-down can take place regardless of the choice of the nominal speed (S_u or S^*). We perform dynamic speed slow-down by using the *Dynamic Reclaiming Algorithm* (*DRA*) [2]. Originally proposed for periodic hard real-time task sets [2], DRA has been also recently adapted to the scheduling of aperiodic/periodic task sets [3], and energy-constrained reward-based periodic task sets [1].

DRA detects early task completions by comparing the actual schedule to the static optimal schedule. In this schedule, all the jobs run at the same speed, namely the nominal speed S^{nom}, through which all the (selected) jobs will be able to meet their deadlines even under a worst-case workload. DRA determines the amount of CPU time that a dispatched job can *safely* use to slow down its speed. This additional CPU time is referred to as the *earliness* and is used to calculate the new (lower) speed of a dispatched job. A main feature of the scheme is to calculate the *earliness* quickly, and without affecting the feasibility of already selected tasks. The earliness is computed in such a way to allow the low-priority tasks to use the slack-time of (completed) high-priority tasks. The exact formula for calculating the earliness and determining the reduced speed, as well as the details of the DRA can be found in [2].

4.2 Energy-Density Schemes

The greedy schemes discussed in Section 4.1 are based on the implicit assumption that all the mandatory jobs will be executed. Though they attempt to improve the energy-efficiency of the system through the careful selection of the nominal speed and dynamic reclamation, they do *not* consider the specific energy budget of the system. Albeit natural, in scarce-energy settings where some dynamic failures are unavoidable, this approach may not be the best. In fact, it may be more reasonable to carefully select the mandatory jobs to be executed by considering the energy budget, mission time and the energy demands of individual jobs. The system performance can be improved by giving preference (in energy allocation) to tasks that have the lowest energy requirements and that can potentially cause a large number of dynamic failures, if skipped altogether. Hence, we give preference to tasks based on a metric we refer to as the *energy density (ED)*:

Definition 4. *The* ***energy density*** ED_i *of the task* T_i *is the* weighted *ratio of the energy consumption* E_i *of the* mandatory *instances of the task* T_i *divided by the number of dynamic failures* DF_i^{max} *that this task can cause during the mission time (that is,* $ED_i = E_i / w_i \, DF_i^{max}$*).*

In Definition 4, recall that w_i is a weight indicating the relative impact of T_i's dynamic failures on the overall system performance. To ensure that important tasks are given priority in allocation of the energy needed to meet their constraints, the designer can assign large weights to those tasks. Under a constant speed, the energy demand E_i of the mandatory jobs of a given task is proportional to $C_i \; m_i \; / \; P_i \; k_i$, since the energy consumption is proportional to the utilization, and to the ratio of jobs selected for execution.

Thus, we sort the tasks according to the energy density parameter ED_i. The energy budget is allocated incrementally starting with the mandatory jobs of the task with the lowest energy density. Then, we keep augmenting the set of selected tasks by considering the energy densities until the energy budget is depleted. Naturally, at every step, the energy needed is calculated using a nominal speed based on the subset of currently selected tasks Γ_s. This is in sharp contrast with the greedy schemes, where the *entire* mandatory workload is considered when selecting a nominal speed. Note that the two nominal speed computation techniques, one based on the utilization and the other one based on the processor demand analysis, are still applicable. However, the speed will now be determined based on the selected workload, potentially creating opportunities for adopting a lower CPU speed.

It is still possible to perform slack time reclamation and dynamic speed slow-down. However, to guarantee the deadlines of the selected tasks Γ_s only the slack time of tasks in Γ_s can be reclaimed, since the nominal speed in this case is determined based on the workload of Γ_s. For example, if the nominal speed is set to the total utilization of tasks in Γ_s, one can safely reclaim the slack time due to skipped or completed instances of tasks in Γ_s. If the nominal speed is computed using the mandatory processor demand of selected task sets (using Theorem 2), the slack time of skipped jobs is already incorporated to the statically computed nominal speed, and we can reclaim only the slack time resulting from the early completions of mandatory tasks in Γ_s.

In the context of the energy-density schemes, if there is sufficient excess energy accumulated at run-time due to dynamic speed slow-down, then this energy can be used to select additional mandatory instances of tasks that have been previously marked as unselected. By exploiting this excess energy, one can significantly improve the system performance. We refer to this mechanism as *job promotion*. However, job promotion must be used with care to guarantee the deadlines of the currently selected jobs. Recall that the nominal speed at which selected tasks run is based on the workload Γ_s, and not the total workload corresponding to the entire task set Γ. Consequently, if additional tasks are added to Γ_s through job promotion, the previously calculated nominal speed may be no longer safe. To avoid this problem, we use the excess energy for job promotion only at the beginning of a new frame (hyperperiod). In other words, the set of selected tasks Γ_s is determined for the first frame, and the corresponding nominal speed S^{nom} is calculated subject to the system energy budget. Then, throughout the frame, the excess energy resulting from dynamic slack reclamation and speed slow-down is accumulated. At the beginning of the next frame the available energy (including the excess energy) is used to recalculate Γ_s and S^{nom}. The same procedure is repeated at every frame boundary. Note that the job promotion occurring at the frame (hyperperiod) boundaries is reminiscent of the re-chargeable energy-aware system framework of Rusu et al. [25].

5. Experimental Results

We experimentally evaluated our proposed schemes using a discrete-event simulator implemented in C++. In our simulation experiments we measured the dynamic failure ratio *DFR* as a function of three parameters:

- System energy budget E_{budget} is represented as a percentage of the total energy E_{limit} required to execute all the mandatory jobs. E_{limit} is calculated assuming that all tasks run at the nominal speed S_u (the lowest speed guaranteed to meet all deadlines without considering energy budget) throughout the mission time. E_{budget} is a measure of how energy-constrained the system is, and as it decreases it becomes more important to carefully manage the available energy to reduce the dynamic failure ratio.
- Execution time ratio *ER* is the ratio of best-case execution time to worst-case execution time. It controls the variability of the actual workload compared to the worst-case.
- Total system utilization $U_{tot} = \sum_i C_i / P_i$.

The workload in our experiments comprised 15 tasks all of which had the same (2,3)-constraints. We generated 1000 generic task sets, and then for each task set we ran 100 experiments to generate actual execution requirements for each task instance (job). We varied the above three parameters over the following ranges: *U* ranged from 0.1 to 1.0 in increments of 0.1, E_{budget} (as a percentage of E_{limit}) ranged from 10% to 100% in increments of 10%, *ER* ranged from 0.1 to 1.0 in increments of 0.1. The mission time *X* was 4 times the hyperperiod *P*. Task periods were generated through a uniform probability distribution, and the ratio of maximum period to minimum period was 20. Similarly, the actual execution time *AET* of any given task under maximum speed was generated uniformly between *ER*WCET* and *WCET*, where *WCET* is the worst-case execution time under maximum speed. We assumed that all tasks have equal weights. We assumed that the CPU power consumption varies as a cubic function of the CPU speed [2,3,18,26,27] and that S_{min} = 0.1. We underline that the experiments with discrete speed models and different (m,k) parameters yield similar patterns, but those results are not included due to the space limitations.

5.1 Experimental results for greedy schemes

The greedy schemes operate by initially selecting all the mandatory jobs of all tasks for execution. We implemented four different greedy schemes that differ in the nominal speed used and whether dynamic reclamation is adopted.

- Static–S_u: Mandatory jobs run with speed S_u and slack-time reclamation is *not* performed.
- Static–S^*: Mandatory jobs run with speed S^* and slack-time reclamation is *not* performed.
- Dynamic–S_u: Mandatory jobs run with speed S_u and slack-time is reclaimed.
- Dynamic–S^*: Mandatory jobs run with speed S^* and slack-time is reclaimed.

We compared the performance of all four schemes through simulations under both worst-case and actual workload conditions. For the sake of completeness, we compare the performance of our proposed solutions against a well-known scheduling algorithm that has been traditionally used for scheduling (*m*,*k*)-firm RT tasks, namely the Distance-Based Priority (DBP) algorithm [11]. DBP is a (non-power-aware) priority-based scheduling heuristic that gives higher priority to tasks that are closer to the state of dynamic failure. To use DBP in such settings, one must carefully choose a nominal CPU speed to reduce the energy consumption without causing a significant number of deadline misses. We performed extensive simulations to evaluate the performance of DBP under various nominal speed assignments, but we opted to omit these results due to space limitation. In this paper, we include the results for DBP with the CPU speed = 0.5, which yielded consistently the best results for the simulation settings under consideration.

5.1.1 Results under worst-case workload conditions

If the worst-case workload conditions do occur at run-time, the reclamation of slack time due to early completions is not applicable. However, it is still possible to reclaim slack time of skipped optional jobs if the nominal speed S_u is used. Recall that reclaiming slack time of skipped optional jobs is not performed with the nominal speed S^* (in order to preserve the timing constraints of mandatory jobs as discussed in Section 4.1).

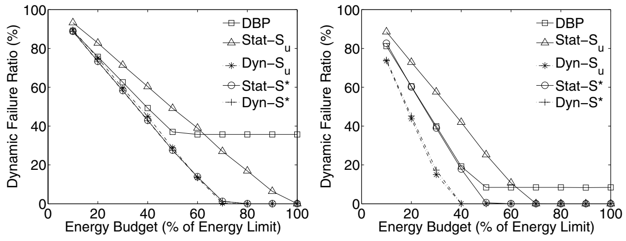

Figure 6. Effect of energy budget on greedy schemes under worst-case (left) and actual (right) workload conditions

Figure 6 (left) presents the effect of E_{budget} (at U_{tot} = 0.7) on the dynamic failure ratio *DFR* for all four greedy schemes and DBP under worst-case workload (i.e. *ER* = 1.0) for (2,3)-firm constraints. As expected, for all the schemes, increasing E_{budget} tends to decrease the dynamic failure ratio. At E_{budget} = 100% the greedy schemes do not exhibit any dynamic failures because the system has sufficient energy to meet all the (*m*,*k*)-constraints. On the other hand, even with E_{budget} = 100% DBP yields a dynamic failure ratio of about 38% due to its sub-optimal priority assignment. Among the greedy schemes, Static–S^* and Dynamic–S^* have identical performance (since reclamation is not possible for worst-case workloads), and are the best among all greedy schemes. The next best scheme is Dynamic–S_u with comparable performance thanks to its ability to reclaim slack time of optional jobs. Static–S_u has the worst performance among all greedy schemes since it uses a conservative speed and does not reclaim slack time of skipped jobs.

5.1.2 Results under actual workload conditions (effects of reclamation)

In most cases, the actual workload is usually significantly lower than the worst-case, providing room for improving system performance through dynamic reclamation of slack time. Under such conditions, in addition to reclamation of slack time of skipped optional jobs, another form of reclamation can be exploited. Namely, slack time resulting from early completions of selected mandatory jobs can be reclaimed. This can occur regardless of the choice of the nominal speed without compromising the deadlines of selected jobs.

Figure 6 (right) shows the effect of E_{budget} (at U_{tot} = 0.7 and ER = 0.4) on the dynamic failure ratio DFR for all four greedy schemes and DBP under actual workload for (2,3)-firm constraints. Comparing the performance of all schemes to worst-case workload conditions presented in Figure 6(left), one notices a clear improvement in performance under actual workload since the effective energy consumption is now significantly lower. There is a change in the order of best performing schemes, however. The two best performing schemes are the ones that employ reclamation, namely Dynamic–S^* and Dynamic–S_u. *The results indicate that the use of reclamation is more important than the choice of the nominal speed for greedy schemes*, since Dynamic–S_u outperforms Static–S^* which uses a lower nominal speed but without reclamation.

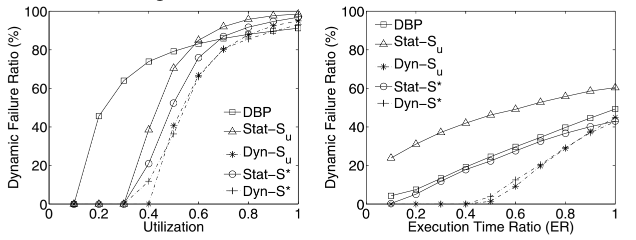

Figure 7. Effect of utilization (left) and execution time ratio (right) under actual workload on greedy schemes

Figure 7 (left) shows the effect of utilization for $E_{budget} = E_{limit}(U_{tot}{=}0.3)$ and ER = 0.4 under actual workload conditions. Unlike the other figures where E_{budget} is recalculated as a percentage of E_{limit} (which is also a function of the total utilization U_{tot}), in this set of experiments E_{budget} is set to a fixed value, namely, to the energy required to meet all the mandatory deadlines when $U_{tot} = 0.3$ (i.e. $E_{budget} = E_{limit}(U_{tot}{=}0.3)$). When $U_{tot} \leq 0.3$ the system has enough energy budget to meet all mandatory deadlines (i.e. $E_{budget} \geq E_{limit}$) and our greedy schemes yield zero dynamic failure ratio, while we observe again DBP's unpredictable performance. As U_{tot} increases from 0.3 to 1.0, the system becomes effectively more energy-constrained and the performance of all schemes degrades resulting in higher dynamic failures. The top performing schemes are Dyn–S^* and Dyn–S_u followed by Static–S^*.

The effect of execution time ratio ER (i.e. workload variability) on the dynamic failure ratio DFR is shown in Figure 7 (right) for U_{tot} = 0.7, and E_{budget} = 40%. The lower the value of ER, the more significant the performance improvement due to the reclamation. Therefore, at small ER values ($ER \leq 0.4$), the schemes that use reclamation, namely Dynamic–S^* and Dynamic–S_u, practically avoid all dynamic failures and outperform other schemes. However, as ER increases the performance gap shrinks.

5.2 Experimental results for energy-density schemes

The energy-density schemes carefully select the mandatory jobs to be executed by considering the energy budget, mission time, and the energy demands of individual jobs. We implemented four energy-density schemes that differ in the selection of the nominal speed and the use of dynamic reclamation. All four schemes use job promotion at frame boundaries to utilize the excess energy accumulated in the previous frame. Specifically:

- ED–S_u: Nominal speed computed through the utilization of the selected tasks, no reclamation
- ED–S^*: Nominal speed computed through the processor demand of the selected tasks, no reclamation
- EDR–S_u: Nominal speed computed through the utilization of the selected tasks, uses reclamation
- EDR–S^*: Nominal speed computed through the processor demand of the selected tasks, uses reclamation

In this section we present experimental results for energy-density schemes under both worst-case and actual workload conditions. For comparison, we also include the best performing greedy scheme, Dynamic–S^*.

5.2.1 Results under worst-case workload conditions

Under worst-case workload conditions, the only possible form of slack time reclamation is the one due to the skipped optional jobs. Figure 8 (left) presents the effect of E_{budget} (at U_{tot} = 0.7) on the dynamic failure ratio DFR for all four energy-density schemes under worst-case workload (i.e. ER = 1.0) for (2,3)-firm constraints. The order of schemes based on performance is similar to that of the greedy schemes (Figure 6, left). The best performing energy-density schemes are EDR–S* and ED–S^* (which have identical performance since reclamation is not possible under worst-case workload when $S_{nom} = S^*$). ED–S_u has the worst performance among the energy-density schemes. One important observation is that the performance of the energy-density schemes is significantly better than the greedy schemes, represented by Dyn–S^* in this figure, especially at low to medium E_{budget} values, thanks to the low CPU speed they are able to adopt. However, when E_{budget} is at least 70%, the performance of Dyn–S^* improves significantly and becomes comparable to the energy-density schemes, since in this region the system is less energy-constrained and aggressively selecting all mandatory tasks for execution helps.

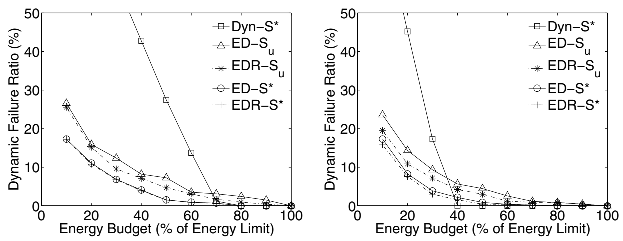

Figure 8. Effect of energy budget on energy-density schemes under worst-case (left) and actual (right) workload conditions

5.2.2 Results under actual workload conditions (effect of reclamation)

Figure 8 (right) shows the effect of E_{budget} (at U_{tot} = 0.7 and ER = 0.4) on the dynamic failure ratio DFR for all four energy-density schemes under actual workload for (2,3)-firm constraints. There is a noticeable performance improvement, particularly for the schemes EDR–S* and EDR–S_u that use reclamation, over the results under worst-case workload conditions presented in (Figure 8, left), since the effective workload is now significantly lower. There is also a change in the order of best performing schemes. Unlike the case of the greedy schemes (Figure 6, right), the two best performing schemes are EDR–S* and ED–S^* which use the nominal speed S^*. Thus, *for energy-density schemes the use of a low nominal speed has a more significant impact on performance than the use of dynamic slack reclamation.*

Figure 9 (left) shows the effect of utilization for $E_{budget} = E_{limit}(U_{tot}=0.3)$ and $ER = 0.4$ under actual workload conditions. Similar to Figure 7(left) for the greedy schemes, E_{budget} here is set to a fixed value, namely the energy required to meet all the mandatory deadlines when $U_{tot} = 0.3$ (i.e. $E_{limit}(U_{tot}=0.3)$). When $U_{tot} \leq 0.3$ the system has enough energy to meet all mandatory deadlines (i.e. $E_{budget} \geq E_{limit}$) and there are no dynamic failures. As U_{tot} increases from 0.3 to 1.0, the system becomes effectively more energy-constrained and the performances of all the schemes degrade resulting in higher dynamic failure ratios. The top performing schemes are EDR–S* and ED–S^* followed by EDR–S_u. All energy-density schemes significantly outperform the best greedy scheme.

The effect of execution time ratio *ER* (i.e. workload variability) on the dynamic failure ratio *DFR* is shown in Figure 9 (right) for $U_{tot} = 0.7$ and $E_{budget} = 40\%$. As *ER* gets smaller, the effective workload to be executed decreases, and so does the energy needed to execute this workload, resulting in reduced dynamic failures. However, this improvement in performance is less emphasized in the results of the energy-density schemes, when compared to the greedy schemes, since *DFR* is already rather low with energy-density schemes. Note also that the energy-density schemes significantly outperform the best greedy scheme for medium to large *ER* values (i.e. $ER > 0.6$). Only when the effective workload is small (i.e. $ER < 0.5$) does the greedy scheme perform competitively.

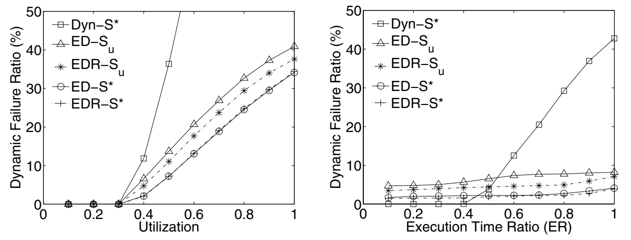

Figure 9. Effect of utilization (left) and execution time ratio (right) under actual workload on energy-density schemes

6. Conclusion

To the best of our knowledge, this research effort is the first to explore the energy-constrained scheduling problem for weakly-hard real-time systems. We showed that the problem is NP-Hard in the strong sense even assuming continuous CPU speed. We provided a comprehensive framework that consists in selecting a mandatory workload and an appropriate CPU speed to meet the *(m,k)*-firm deadlines using the processor demand analysis. We presented two classes of solutions. The “greedy” algorithms attempt to select the entire mandatory workload a priori, while “energy-density” algorithms further prioritize the jobs in the mandatory workload by considering their energy cost and the total number of *(m,k)*-firm deadlines they help to meet. We further explored the effects of the nominal speed selection and on-line slack reclamation techniques. Our results indicate that the energy-density algorithms dominate over the greedy algorithms when the system is significantly energy-constrained.

References

[1] T. A. AlEnawy and H. Aydin. On Energy-Constrained Real-Time Scheduling. *Proceedings of the 16th EuroMicro Conference on Real-Time Systems (ECRTS'04),* Catania, Italy, June 2004.

[2] H. Aydin, R. Melhem, D. Mosse and P.M. Alvarez. Power-aware Scheduling for Periodic Real-time Tasks. *IEEE Transactions on Computers*, vol. 53, no. 5, pp. 584-600, May 2004.

[3] H. Aydin and Q. Yang. Energy-Responsiveness Tradeoffs for Real-Time Systems with Mixed Workload. *Proceedings of the 10th IEEE Real-time and Embedded Technology and Applications Symposium (RTAS'04)*, Toronto, Canada, May 2004.

[4] S. Baruah, R. Howell, and L. Rosier. Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-time Tasks on One Processor. *Real-Time Systems*, vol 2, no. 4, pp. 301 – 324, Nov 1990.

[5] G. Bernat and A. Burns. Combining (n,m)-hard deadlines and dual priority scheduling. *Proceedings of the 18th IEEE Real-Time System Symposium (RTSS'97),* San Francisco, CA, Dec. 1997.

[6] G. Bernat, A. Burns, and A. Llamosi. Weakly Hard Real-time Systems. *IEEE Transactions on Computers,* vol. 50, no. 4, pp. 308 – 321, Apr. 2001.

[7] M. Caccamo and G. C. Buttazzo. Exploiting Skips in Periodic Tasks for Enhancing Aperiodic Responsiveness. *Proceedings of the 18th IEEE Real-Time System Symposium (RTSS'97)*, San Francisco, CA, Dec. 1997.

[8]http://developer.intel.com/design/intelxscale/benchmarks.htm

[9] R. Ernest and W. Ye. Embedded Program Timing Analysis Based on Path Clustering and Architecture Classification. *International Conference on Computer-Aided Design* (*ICCAD'97*), 1997.

[10] M.R. Garey and D.S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. 1979.

[11] M. Hamdaoui and P. Ramanathan. A dynamic priority assignment technique for streams with (m,k)-firm deadlines. *IEEE Transactions on Computers*, vol. 44, no. 5, pp. 1443 – 1451, Dec. 1995.

[12] D. Isovic and G. Fohler. Quality aware MPEG-2 stream adaptation in resource constrained systems. *Proceedings of the 16th EuroMicro Conference on Real-Time Systems (ECRTS'04),* Catania, Italy, June 2004.

[13] K. Jeffay and D. L. Stone. Accounting for Interrupt Handling Costs in Dynamic Priority Task Systems. *Proceedings of IEEE Real-Time Systems Symposium (RTSS'93)*,Raleigh-Durham, NC, Dec. 1993.

[14] D. Kang, S. Crago, and J. Suh. A Fast Resource Synthesis Technique for Energy-Efficient Real-time Systems. *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02)*, Austin, TX, Dec. 2002.

[15] G. Koren and D. Shasha. Skip-Over: algorithms and complexity for overloaded systems that allow skips. *Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS'95)*, Pisa, Italy, Dec. 1995.

[16] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real time environment, *Journal of the ACM*, vol. 20, no 1, pp. 46 – 61, Jan. 1973.

[17] J. Liu, P.H. Chou, N. Bagherzadeh, and F. Kurdahi. Power-aware scheduling under timing constraints for mission-critical embedded systems. ***Proceedings of the 39th*** *Design Automation Conference*, Las Vegas, NV, June 2001.

[18] Y. Liu and A. Mok. An Integrated Approach for Applying Dynamic Voltage Scaling to Hard Real-Time Systems. *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'03)*, Washington D.C., May 2003.

[19] A.K. Mok and W. Wang. Window-constrained real-time periodic task scheduling. *Proceedings of the 21st IEEE Real-Time Systems Symposium (RTSS'00)*, Orlando, FL, Nov. 2000.

[20] P. Pillai and K.G. Shin. Real-time dynamic voltage scaling for low power embedded operating systems. *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01)*, Banff, Canada, Oct. 2001.

[21] A. Qadi, S. Goddard, and S. Farritor. A Dynamic Voltage Scaling Algorithm for Sporadic Tasks. *Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS'03)*, Cancun, Mexico, Dec. 2003.

[22] G. Quan and X. Hu. Enhanced fixed-priority scheduling with (m,k)-firm guarantee. *Proceedings of the 21st IEEE Real-Time Systems Symposium (RTSS'00)*, Orlando, FL, Nov. 2000.
[23] P. Ramanathan. Overload management in real-time control applications using (m, k)-firm guarantee. *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 6, pp. 549 – 559, June 1999.
[24] C. Rusu, R. Melhem and D. Mosse. Maximizing the system value while satisfying time and energy constraints. *Proceedings of 23rd IEEE the Real-Time Systems Symposium (RTSS'02)*, Austin, TX, Dec. 2002.
[25] C. Rusu, R. Melhem, and D. Mosse. Multi-Version Scheduling in Rechargeable Energy-Aware Real-time Systems. *Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS'03)*, Porto, July 2003.
[26] S. Saewong and R. Rajkumar. Practical Voltage-Scaling for Fixed-Priority Real-time Systems. *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'03)*, Washington D.C., May 2003.
[27] K. Seth, A. Anantaraman, F. Mueller and E. Rotenberg. FAST: Frequency-Aware Static Timing Analysis. *Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS'03)*, Cancun, Mexico, Dec. 2003.
[28] R. West and C. Poellabauer. Analysis of a window-constrained scheduler for real-time and best-effort packet streams. *Proceedings of the 21st IEEE Real-Time Systems Symposium (RTSS'00)*, Orlando, FL, Nov. 2000.
[29] R. West, Y. Zhang, K. Schwan, and C. Poellabauer. Dynamic window-constrained scheduling of real-time streams in media servers. *IEEE Transactions on Computers*, vol. 53, no. 6, pp. 744 – 759, June 2004.

Appendix A (Proof of Theorem 1)

We first formally state the energy-constrained weakly-hard scheduling problem.

WEAK-EC: Consider a set Γ of n periodic weakly-hard real-time tasks for which a feasible schedule exists when executed with the maximum CPU speed in the absence of hard energy constraint. Is it possible to satisfy the (m,k)-constraints during a mission time X, with a limited energy budget E_{budget}, and a DVS-enabled CPU?

Consider the problem **3-PARTITION** which is known to be NP-Hard in the strong sense [10]:

3-PARTITION: Given $3k$ integers $a_1, a_2, \ldots, a_{3k}$ and two additional integers k, B, such that $\Sigma_i\, a_i = kB$ and $B/4 < a_i < B/2\ \forall i$; is it possible to partition these $3k$ integers into k groups in such a way that the sum of elements in each group is exactly B?

Suppose that WEAK-EC admits a polynomial-time solution. We will show that, using this solution, it is possible to transform any given instance of 3-PARTITION to an instance of WEAK-EC in polynomial-time and provide an answer. Given an instance of 3-PARTITION, we construct the corresponding instance of WEAK-EC as follows: We have $3k$ periodic real-time tasks $T_1,\ldots, T_{3k}$; each having (1,k)-firm deadline constraint and period B. T_i's worst-case execution time C_i is given as a_i / k. The mission time $X = kB$ and $E_{budget} = B / k^2$.

First, observe that, in the non-energy-constrained case (with the maximum CPU speed) each instance of every task will be able to meet its deadline (since $U_{tot} = \Sigma_i\ a_i / kB = 1.0$), trivially satisfying the (1,k)-constraints. For the energy-constrained case, notice that a *necessary* condition to meet the (1,k)-constraints of all the tasks during the mission time is to be able to execute exactly *one instance* of each task in the interval $[0,kB]$ (since there are k invocations of each task during the mission interval). A *lower bound* on the energy needed to complete exactly one instance of each task during the mission time can be computed by considering that using a constant speed S to execute a given workload in an interval minimizes the energy consumption. Thus, this constant speed S can be computed as: $S = \Sigma_i\ C_i / kB = \Sigma_i\ a_i / k^2B = 1 / k$.

In fact, using this (optimal) constant speed will be necessary for this particular instant of WEAK-EC, since assuming a power consumption of $g(S) = S^3$ and $g^{stb} = 0$, we will need a minimum energy of $\Sigma_i\ (a_i / kS)\ S^3 = B / k^2$, which happens to be the same as E_{budget} in this problem instance. But with speed $S = 1 / k$, the execution time of each task T_i becomes exactly a_i. Thus, the instance of WEAK-EC admits a positive answer if and only if it is possible to schedule exactly one instance of each task T_i (with the increased execution time a_i and period/relative deadline B) in interval $[0,kB]$. This last statement, on the other hand, is true, if and only if the original 3-PARTITION instance admits a YES answer (since $\Sigma_i\ a_i = kB$); proving the claim.

The reader should note that it is straightforward to re-write the proof for other types of power consumption functions; it suffices to choose an energy budget to force the system to adopt a continuous speed of $1/k$ with the corresponding power function. ■

Appendix B (Proof of Theorem 2)

A fundamental result in real-time scheduling theory establishes the necessary and sufficient condition for feasibility through the processor demand function.

Theorem 3 [4, 13]. *A set of preemptive real-time jobs* Γ *can be scheduled (by EDF) if and only if* $D(t_1,t_2) \le t_2 - t_1$ *for all intervals* $[t_1,t_2]$, *where* $D(t_1,t_2)$ *denotes the total execution time of jobs in* Γ *that arrive at or after* t_1*, and having deadlines less than or equal to* t_2 *(also known as the "*processor demand*" function).*

Observe that in a weakly-hard periodic RT task set where all the optional jobs are skipped and the mandatory jobs are executed with constant speed S, $D(t_1,t_2)$ is simply $\Sigma_i\, M_i(t_1,t_2)\cdot C_i / S$.

Corollary 1. *The mandatory jobs of a weakly-hard RT task set will meet their deadlines by EDF and with the CPU speed* S *if and only if:* $\Sigma_i\, M_i(t_1,t_2)\cdot C_i \le (t_2 - t_1)\, S$, *for all intervals* $[t_1,t_2]$.

A well-known result of the RT scheduling theory indicates that we can restrict our attention to the intervals starting at a task release time, and ending at a task deadline [4]. Moreover, we can prove the following:

Proposition 2. *In a schedule where the mandatory jobs of task* T_i *are dispatched according to the "deeply-red" pattern:*
$M_i(0,L) \ge M_i(t_1, t_1 + L)\ \forall t_1, L$.

Combining Corollary 1 and Proposition 2, we get Proposition 3, implying Theorem 2:

Proposition 3. *The mandatory jobs of a weakly-hard RT task set will meet their deadlines by EDF, if they are executed with the speed* S*, such that:* $S \ge \max_{L \ge 0} \{ \Sigma_i\, M_i(0,L)\cdot C_i / L \}$.

Proof of Proposition 2:

In interval $[0,L]$ there are exactly $y = \lfloor L / P_i \rfloor$ instances of T_i. Note that the total number of T_i's instances in $[t_1, t_1{+}L]$ cannot exceed y in any way. Thus, for a given t_1 and L, locate the smallest the $L_2 \ge L$ such that there are exactly y instances in the interval $[t_1, t_1 + L_2]$. We will show that $M_i(0,L) \ge M_i(t_1,t_1 + L_2)$, implying that $M_i(0,L) \ge M_i(t_1,t_1 + L)$ since $M_i(t_1,t_1 + L_2) \ge M_i(t_1,t_1 + L)$ (considering that $L_2 \ge L$).

Suppose that the task T_i is supposed to meet m_i deadlines in every k_i consecutive invocations. Then, y can be written as $y = q.k_i + r$ in a unique way, where q and r are integers and $0 \le r \le k_i - 1$. The following two properties are instrumental in the remainder of the proof.

Property 1. *In a deeply-red schedule, there are exactly* m_i *mandatory jobs among any consecutive* k_i *jobs of* T_i.

Property 2. *Among any consecutive* $y = q.k_i + r$ *instances of* T_i *, the number of mandatory jobs cannot exceed* $q.m_i + r$.

We now finalize the proof by distinguishing two cases for r:

- $r \le m_i$: In this case, in interval $[0,L]$ we will first have $q.m_i$ mandatory jobs in interval $(0, q.k_i.P_i)$ followed by r mandatory jobs. Thus, in this case: $M_i(0,L) = q.m_i + r \ge M_i(t_1,t_1 + L_2)$ (from Property 2)
- $r > m_i$: In this case, the interval $[0,L]$ ends with some optional jobs. Hence, $M_i(0,L) = q.m_i + m_i$. Further, since $\lfloor L_2 / P_i \rfloor = y = q.k_i + r$, $M_i(t_1,t_1 + L_2) \le q.m_i + r$ (Property 2). This implies that $M_i(0,L) \ge M_i(t_1,t_1 + L_2)$ (since $r < k_i \le m_i$), proving the claim. ■