On Energy-Constrained Real-Time Scheduling

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

Abstract

In this paper, we explore the feasibility and performance optimization problems for real-time systems that must remain functional during an operation/mission with a fixed, initial energy budget. We show that the feasibility problem is NP-Hard in the context of systems with Dynamic Voltage Scaling (DVS) capability and discrete speed levels. Then, we focus on energy-constrained periodic task systems where the available energy budget is not sufficient to meet all the deadlines. We propose techniques to maximize the total number of deadlines met and the total reward (utility) while guaranteeing the completion of the mission and a minimum performance for each task. We consider separately: (i) systems with or without DVS capability, and (ii) off-line (static) and on-line (dynamic) solutions to select most valuable jobs for execution. We also discuss the tractability of the involved optimization problems. Our on-line algorithms combine job promotion, job demotion and speed reduction techniques to maximize the system performance while guaranteeing the completion of the mission. We evaluate our schemes through simulations and show that the on-line schemes can yield significant performance improvements over static solutions.

1. Introduction

Traditionally, the *feasibility* problem - given a task and system model, determining whether it is possible to meet all the deadlines - has always had a central position in research on hard real-time (RT) systems [10,19]. Another line of research considers *soft RT scheduling* for *overloaded* systems: typically, researchers choose to trade-off the precision (such as imprecise computation [18] and reward-based scheduling [2] approaches) or shed a part of the workload [7, 8, 9, 14,16] to provide timeliness guarantees to a subset of tasks or task instances deemed to be essential or more valuable. In [11], a framework that adjusts the rates for tasks in overloaded RT applications is presented.

Recently, we have observed extensive research activity on Power/Energy management with the proliferation of wireless, portable and embedded devices. Of particular interest to RT system research is the **Dynamic Voltage Scaling** (DVS) technique, which consists in reducing the CPU supply voltage and the clock frequency (CPU speed) to obtain energy savings. Thus, many studies investigate ways to guarantee the feasibility in the presence of DVS for various task/system models and off-line/on-line scheduling algorithms [3, 4, 5, 6, 20, 23, 24].

There exist also settings with a fundamentally different character: One can easily think of RT applications where the system, given a finite energy budget, has to remain functional for the entire duration of an *operation/mission.* In this case, the energy is obviously a *hard* constraint. Example applications include embedded control applications as well as military, space and disaster recovery missions requiring predictable RT system operation while not allowing battery re-charging until the end of the mission. If the system's energy budget does not allow executing all the jobs, then there is the risk of losing functionality - with potential detrimental effects - in the middle of the mission while trying to meet each and every deadline [1]. Thus, it may be more preferable to *skip* a few jobs in a controlled fashion without affecting the stability of the system during the mission. As such, **energy-constrained RT scheduling**, with the objective of maximizing the system performance **within the limits of available energy***,* is the topic of this study.

We note that a number of recent research studies explored related problems. In [21], Rusu et al. considered the problem of maximizing the total reward in a system with fixed energy budget when all the tasks share a common deadline. They showed that the problem of maximizing the total reward is NP-Hard with DVS even for the case when all tasks have a common deadline. The same authors extended this framework to the periodic task case, focusing on re-chargeable systems with DVS capability in [22]. They also presented a set of energy reclamation schemes that can be invoked only at hyperperiod/frame boundaries. Static scheduling of periodic tasks with *continuous* utility functions and fixed energy budget is explored in [15]. In [1], the *static* energy-constrained RT scheduling problem for systems without DVS capability is investigated.

The main **contributions** of this paper are as follows:

- We formally characterize energy-constrained RT systems and explore the nature of the feasibility problem in limited-energy settings. We show that the feasibility problem on systems with fixed energy budget and discrete speed levels is NP-Hard.
- We consider separately the objectives of maximizing the total number of deadlines met and maximizing the total reward. We also show how to extend the framework to the case where some minimum performance guarantees must

be provided for each task (in the form of a "minimum deadline meet ratio").

- We explore these performance optimization problems for systems both with and without DVS capability.
- Although we include the discussion of static analysis, a focal point of this research is efficient and dynamic energy reclamation schemes for the common case where tasks complete early, without presenting their worst-case workload. We show how excess energy arising from early completions can be re-used by dynamically selecting jobs with an eye towards improving the performance of the system. For systems with multiple speeds, we extend the dynamic reclaiming algorithm (DRA) [3] to energy-constrained settings, showing how dynamic energy harvesting through on-line speed reduction can provide additional improvements. Our reclaiming schemes are *greedy*, i.e. any slack time is put in re-use for speed reduction as soon as possible (at task completion/preemption times). This is in contrast with the nature of dynamic schemes presented in [22], where the reclamation can occur only at hyperperiod/frame boundaries. We provide experimental evaluation of our schemes through simulations. Our results show that reward gains over static schemes can be as high as 70% for systems with DVS and as high as 30% for the ones without.

2. System model and assumptions

2.1. Power and energy consumption model

We consider a single processor system that has to rely on battery power. In order to remain operational, the available energy must remain in a 'safe range' [E_{min}, E_{max}], where E_{max} denotes the maximum battery capacity and E_{min} indicates the minimum energy threshold below which the stable/safe operation is not guaranteed [22]. Thus the system has a limited energy budget $E_{budget} = E_{max} - E_{min}$. Further, we assume that the system must remain operational in the time interval [0,X]; that is, the total CPU energy consumption should not exceed E_{budget} for X time units (in other words, the system is subject to a *hard* energy constraint). We refer to X as the *mission time* throughout the paper. Re-charging the battery is not possible or feasible during the mission.

We consider systems with or without DVS capability. A system that does not support multiple speed or voltage levels can be either in *normal* (*execution) mode* or *stand-by mode.* In stand-by mode, the CPU does not execute any task, but consumes much less power. Systems with DVS capability can adjust their speed (frequency) and supply voltage when in normal mode. We assume that the DVS-enabled CPU can be in any of the $m > 1$ speed/frequency levels (or "modes") denoted by $S^1, S^2, \ldots, S^m$, where $S^{i+1} > S^i$. The speeds are normalized with respect to S^m, i.e. $S^m = 1.0$. The power consumption with speed S^i is denoted by g^i; similarly, g^{stb} indicates the power consumption in stand-by mode. Finally, in any time interval [t_1,t_2], the total energy consumption is:

$$E(t_1, t_2) = \sum_{i=1}^{m} E_i = \sum_{i=1}^{m} g^i t_i + g^{stb} t_{stb} \qquad (1)$$

where t_i and t_{stb} denote the total time the CPU speed is S^i and the system is in stand-by mode, respectively.

In current DVS-enabled systems, it is not always possible to find a speed level that exactly corresponds to a desired target speed assignment (often obtained assuming a continuous speed spectrum). We will use the notation $HS(S)$ for the quantity $min(S^i | S^i \geq S)$; informally, $HS(S)$ denotes the lowest speed level available in the system that is equal to or greater than a target speed S. Similarly $LS(S)$ will denote the highest speed level available in the system that does not exceed a target speed S; formally $LS(S) = max(S^i | S^i \leq S)$.

2.2. Task model

We consider a set of n periodic tasks $T = \{T_1, T_2, \ldots, T_n\}$. Each task T_i has a period P_i which is equal to the relative deadline of each instance (job) T_{ij}. The worst-case execution time of task T_i with speed S^k is denoted by C_i^k (we will omit the superscript and use notation C_i instead of C_i^1 for systems without DVS capability). Preemption is allowed; the preemption and the speed change overhead can be incorporated in C_i^k if necessary. We adopt the Earliest-Deadline-First (EDF) scheduling policy which is known to be optimal from the feasibility point of view [17]. The emphasis of this paper is to study the feasibility and performance maximization problems in the context of energy-constrained systems (with *limited CPU energy budget*). Therefore, we assume that the task set *would be* feasible (schedulable) with a sufficient energy budget when the CPU speed is maximum (i.e. $\sum_i C_i^m / P_i \leq 1$).

3. Energy-constrained real-time scheduling

In this section, we introduce the concept of energy-constrained RT systems, explore feasibility conditions, and present the performance metrics and our general approach.

Given an RT system that must remain operational during the mission interval [0,X], the natural question to ask is whether all deadlines can be met within the limited energy budget. To answer this question, it is necessary to know the minimum amount of CPU energy needed to complete all the tasks before their deadlines while sustaining the system's operation until the end of the mission. We refer to this threshold energy as E_{bound} in the remainder of the paper.

Definition 1. *A real-time system is* ***energy-constrained*** *if* $E_{budget} < E_{bound}$.

If the system is energy-constrained, then trying to execute *all* the jobs may result in a situation where the system runs out of energy in the middle of the mission with potentially detrimental effects (see [1] for an illustrative example). Thus, the aim should be to provide maximum predictability or utility while sustaining the system operation until the end of the mission.

The computation of E_{bound} for systems without DVS capability and those with continuous speed spectrum can be performed efficiently (see below). Once E_{bound} is computed, then it can be compared against E_{budget} to check if the system is energy-constrained or not.

If the system does not support DVS, then all tasks must be executed with the only available speed (S^1). The system can switch to stand-by mode when there are no ready tasks, thus we have:

$$E_{bound} = Xg^{stb} + (g^1 - g^{stb}) \sum_{i \in \alpha} C_i \qquad (2)$$

where α is the set of all task instances with deadlines smaller than or equal to X.

Assuming an ideal DVS architecture where the speed can be varied over a continuous spectrum, the optimal speed assignments while preserving the feasibility can be computed in polynomial-time for both aperiodic [24] and periodic [3] task models. In particular, the optimal CPU speed for periodic model is constant across all the tasks and is equal to the utilization value, U_v (see [3, 20]). Using this result, one can easily compute E_{bound} by a formula similar to (2), assuming that the power/speed relationship is known.

Unfortunately, the feasibility problem for a system with limited energy *and* discrete speed levels is intractable.

Theorem 1. *The problem of deciding whether it is possible to meet all the deadlines of a set of real-time tasks within a limited energy budget is NP-Hard for a DVS-enabled CPU with discrete speed levels.*
Proof: See Appendix A.

Corollary: Deciding if a task set is energy-constrained is NP-Hard.
Remark 1: The proof shows that the problem is intractable for the simple case of tasks sharing a common deadline; thus it is valid for both aperiodic and periodic task models.
Remark 2: Although the problem is NP-Hard in the general case, some instances can be decided easily. In the periodic task model, if E_{budget} is smaller than the energy needed to execute all the tasks with speed $LS(U_v)$, then the system is energy-constrained since some deadlines will be definitely missed. Conversely, if E_{budget} is greater than the energy needed to execute all the tasks with the speed $HS(U_v)$, then all tasks can complete without violating the energy bound.

Since the problem of deciding if a system is energy-constrained (consequently, computing E_{bound}) is intractable, we adopt a conservative (but safe) approach for periodic tasks; we take E_{bound} to be the energy required to execute all the tasks at the speed $S=HS(U_v)$. Recall that this represents the lowest speed available in the system that is greater than or equal to the optimal *continuous* speed.

In the remainder of this paper, we focus on the problem of maximizing system performance in energy-constrained settings (where $E_{budget} < E_{bound}$).

3.1. Performance objectives

We propose two metrics to maximize the performance in energy-constrained settings:

- **Objective O1 –** maximize the *total number of deadlines met* during the mission without exceeding the energy budget.
- **Objective O2 –** maximize the *total system reward* during the mission without exceeding the energy budget. In this objective we account for the fact that some tasks may be more important than others. To this aim, we associate with each task T_i a *weight* (reward) w_i that signifies its importance. An instance of task T_i contributes w_i units to the total system reward only if it completes by its deadline.

Some applications may require that a minimum number of instances of each task be successfully completed during the mission time, for the whole system to function properly. Maximizing system performance without observing this constraint cannot provide such a guarantee, since all instances of a task with small weight may be skipped all together. To this aim, we may associate with each task T_i a **minimum deadline meet ratio** M_i. We require that $n_i / N_i \geq M_i$, where n_i is the number of instances of task T_i that complete by their deadlines in $[0,X]$ and $N_i = \lfloor X / P_i \rfloor$ is the number of all the instances of the same task whose deadlines lie within the same interval. We refer to the actual instances selected to meet this minimum requirement as *mandatory jobs* and refer to all the remaining jobs as *optional jobs*. If the application at hand imposes such a constraint, then the goal would be to maximize system performance, in the context of objective *O1* or *O2*, while providing a minimum performance guarantee for each task.

3.2. Our approach

If the system is energy-constrained, then the task instances to be executed must be carefully selected in order to make best use of available energy. This selection process is guided by the performance metric under consideration for either objective O1 or O2. To this aim, we use a job selection phase in which each task instance (job) is labeled as **skipped** or **selected**. The selection process considers the parameters of the task set, as well as available energy and mission time. We associate with each job T_{ij} a label L_{ij}, where $L_{ij} = 0$ indicates that the job is skipped and $L_{ij} = 1$ indicates that it is selected for execution. **At run-time, only jobs whose labels are set to "selected" are dispatched**. Thus, the problem becomes choosing the job labels for each performance objective (O1 or O2) while making sure that the energy budget is not exceeded. Note that the labeling of jobs can be performed in static or dynamic fashion. **In systems with DVS, another major component of the problem is to determine the speed assignments of selected jobs.**

4. Solutions to energy-constrained performance optimization problems

4.1. Classification of energy-constrained RT scheduling algorithms

In general, it is possible to classify energy-constrained RT scheduling algorithms depending on the on-line/off-line nature of the solution *and* the workload information used by the scheduler.

- **Off-line (Static)** schemes select task instances before the mission starts and, thus, cannot adjust if the actual workload deviates from the expected one. **On-line (Dynamic)** schemes, on the other hand, start with an initial schedule generated off-line, and later adjust it on-line by, for example, selecting more jobs when excess energy becomes available or unselecting some jobs if the system is running short on energy.
- **Conservative schemes** only have knowledge about *worst-case* workload and, hence, allocate the energy budget to selected jobs based on worst-case execution times. On the other hand, **aggressive schemes** have the additional knowledge of *expected* workload and, hence, allocate the energy budget to selected jobs based on average-case execution times. Finally, **speculative schemes**, while not having the information about expected workload, try to estimate it dynamically by monitoring *actual* workload. By virtue of their nature, aggressive and speculative schemes are on-line schemes.

Note that an off-line scheme can only be conservative because it must guarantee the completion of the mission even under the worst-case scenario. Based on the above classification, we identified the following generic schemes:

1. **Off-line conservative scheme (OFC)** is an off-line job selection scheme that allocates available energy budget to tasks based on their worst-case execution requirements. This scheme was the focus of the study in [1].
2. **On-line conservative scheme (ONC)** is an on-line job selection scheme that allocates available energy budget to tasks based on their **worst-case** execution requirements. It also *promotes* jobs on-line (by re-labeling some skipped jobs as **selected**) if there is excess energy, thus trying to improve system reward at the expense of slightly increased overhead.
3. **On-line aggressive scheme (ONA)** is an on-line job selection scheme that allocates available energy budget to tasks based on their **expected** execution requirements.
4. **On-line speculative scheme (ONS)** is an on-line job selection scheme that **estimates average-case** requirements of tasks by monitoring actual workload, and allocates energy accordingly.

We underline that:

- The specifics of the schemes above differ for systems with or without DVS capability. We elaborate on the details of these in Sections 4.2 and 4.3.
- Both aggressive and speculative schemes risk running out of energy before the end of the mission if (most of the) tasks present their worst-case workload as opposed to the expected (average-case) workload. To ensure that the system remains functional throughout the mission, our on-line aggressive and speculative schemes -before dispatching a selected job- check whether the system has enough energy to remain operational until the end of the mission if this job were to present its worst-case workload. We execute the job only if this condition is satisfied; otherwise the job is skipped (dynamically "demoted").

4.2. Solutions for systems without DVS capability

4.2.1. Off-line conservative scheme (OFC) for systems without DVS

In [1], we explored in detail the energy-constrained RT scheduling algorithms assuming *off-line* settings and a scheduler having *only* the worst-case workload information. In other words, [1] presented the details of the OFC scheme for systems without DVS capability. Below we summarize the main results of [1], since it forms the basis for extended solutions discussed in the remaining part of this study.

The generic job selection framework that we proposed in [1] for energy-constrained real-time systems includes the following major steps:

- For each task T_i, calculate first the total number of instances N_i whose deadlines are smaller than or equal to X ($N_i = \lfloor X / P_i \rfloor$).
- Set aside enough energy to sustain the system throughout the mission and to execute the mandatory jobs required to meet the minimum deadline meet ratio, if any.
- Invoke an **auxiliary optimization module *OM***, which can be an optimal algorithm or a heuristic, to decide on the total number of instances n_i to be executed for each task T_i in interval $[0,X]$ to achieve the given objective, without exceeding the energy budget. This is the only module that needs to be modified if the performance objective changes.
- Finally, determine specific instances to be actually scheduled using an additional labeling criterion. In other words, determine the label L_{ij} (*selected* or *skipped*) for each job.

Now we discuss the details of OFC scheme for each of the performance objectives O1 and O2.

- **Objective O1: maximize total number of deadlines met**

In [1], we showed that, in the absence of DVS, the problem of maximizing the total number of deadlines met during the mission time without exceeding the energy budget *can be solved optimally and efficiently*. We presented an optimal policy, which we refer to as **Favor Shortest Job (FSJ)**. FSJ uses an optimization module *OM* that first orders tasks according to worst-case execution time per instance C_i, and then chooses for execution the maximum possible number of instances of tasks with shortest execution times until E_{budget} is exhausted. Its overall complexity is $O(n \log n)$.

Theorem 2. (from [1]) *FSJ is optimal for the problem of maximizing the number of deadlines met.*

- **Objective O2: maximize total system reward**

In [1] we also showed that, in the absence of DVS, the problem of maximizing total system reward during the mission time without exceeding the energy budget when tasks have different weights (which we refer to as the problem REWARD) is intractable.

Theorem 3. (from [1]) *REWARD is NP-Hard.*

In [1], we presented a number of fast and greedy heuristics for REWARD that give preference to tasks that have higher values of a specific (combination of) task parameter(s) such as execution time, period, and weight. We showed through simulation results that the best performing heuristic is the one that favors jobs with larger reward density values w_i/C_i (i.e. jobs with larger reward and shorter execution time). We refer to this heuristic as LRD (Larger Reward Density).

An implicit assumption of the work in [1] is that tasks always present their worst-case execution times. However, actual workloads in real-time systems exhibit large variability. This implies that using on-line schemes to reclaim excess energy can increase the system reward significantly. Now we show how to extend the work in [1] to incorporate a number of reclamation schemes, based on the on-line schemes discussed in Section 4.1. All of these schemes use LRD as the optimization module.

4.2.2. On-line solutions for systems without DVS

As an initial step, our on-line schemes reserve enough energy to sustain the system throughout the mission, and to execute the mandatory jobs required to meet the minimum deadline meet ratio, if any, for each task. To guarantee that mandatory jobs are successfully completed, the selection framework allocates enough energy to execute these jobs under worst-case workload, even for aggressive and speculative schemes. The selection scheme then uses the optimization module *OM*, *FSJ* for *O1* and *LRD* for *O2,* to select additional (optional) jobs to be executed during the mission to achieve the performance objective under consideration. Thus, we have an initial static schedule comprising, potentially, both mandatory and optional jobs.

When the mission starts, the static schedule is used as an initial schedule, which is adjusted on-line if the actual workload deviates from the expected one. We refer to the on-line adjustment of the static schedule as *reclamation*. To perform reclamation we keep track of the cumulative difference between expected energy consumption and actual one at task completions. For systems without DVS, reclamation can take two forms: either *job promotion* or *job demotion*. Job promotion occurs if there is excess energy due to some jobs completing early. In this case we re-invoke the optimization module *OM* to try to select additional (optional) future jobs, if any, for execution using the excess energy available. Note that *OM* can potentially be re-invoked at each task completion point to greedily re-use excess energy. The associated overhead of this process is $O(n)$ (recall that both LRD and FSJ are efficient algorithms) and it has the potential of significantly increasing total system reward, as we show in Section 5. Job demotion, on the other hand, occurs when the actual aggregate energy consumption is higher than the expected one (due to some jobs running longer than their expected execution times), in which case the system may become unstable before the end of the mission. To avoid this scenario, we *unselect* jobs to save energy for safe completion of the mission. Job demotion applies only to aggressive and speculative schemes. The details of the reclamation process depend on specific schemes and are discussed below.

- **ONC for non-DVS systems**
  When invoking *OM*, ONC uses worst-case task execution times (since it assumes that average-case information is not available) to allocate the energy budget to selected optional jobs. It uses job promotion, but does not need to use job demotion because of its conservative nature.
- **ONA for non-DVS systems**
  ONA uses average-case task execution times when invoking *OM*. It uses both job promotion and job demotion.
- **ONS for non-DVS systems**
  ONS invokes *OM* with worst-case task execution times when generating the static schedule. However, at run-time it estimates the expected execution time of each task using the well-known exponential averaging formula: $PET_{n+1} = 0.5(AET_n + PET_n)$, where PET_n is the predicted execution time of the n^{th} instance of a given task and AET_n its actual execution time. When invoking *OM* for job promotion, ONS uses the most recent predicted execution time for each task. It also uses job demotion to ensure mission completion, if necessary.

4.3. Solutions for systems with DVS capability

The DVS capability provides additional opportunities to save energy through CPU speed adjustment. However, it also makes the problem of energy-constrained RT scheduling harder; since, **in addition to carefully selecting jobs to achieve the performance objective, one also needs to determine the speed(s) at which the selected jobs will run**. Achieving objective *O2* with DVS (which we refer to as DVS-O2) is NP-Hard for static settings as Rusu et al. proved in [22]. We show below that even achieving the (seemingly) easier objective *O1* in the presence of DVS capability (which we refer to as DVS-O1) is intractable.

Theorem 4. *DVS-O1 is NP-Hard.*

Proof: Follows from Theorem 1: If it were possible to maximize the number of tasks that meet their deadlines in polynomial-time on a CPU with multiple speed levels, then the feasibility problem would admit an efficient solution. ■

4.3.1. Static solution for systems with DVS

Since the problem with either of the objectives O1 and O2 is NP-Hard, we present a general solution that applies to both. Just as in the case where DVS is not available, we construct an initial set of 'selected' jobs (thus, a schedule) in off-line fashion before the mission starts. However, we impose a rigorous structure on this simple schedule allowing us to perform reclamation aggressively, and enhance the performance.

In [3], Aydin et al. address the problem of minimizing energy consumption using DVS while meeting all the deadlines of a periodic task set. They adopt a continuous CPU speed spectrum and EDF scheduling policy. They show that the optimal speed S' that minimizes energy consumption, in these settings, is constant for all tasks and equal to the task set's utilization U_v under maximum CPU speed. In *static optimal* schedule, all the tasks run with constant speed S' and present their worst-case workload, meeting their deadlines.

We chose to adapt this static solution to our settings because it enables us to perform CPU-time reclamation greedily and efficiently. We denote the schedule produced by our static solution by Γ_{nom}. To obtain Γ_{nom} we start with a temporary schedule Γ_0 in which *all* instances of *all* tasks are selected and run at exactly the same speed S_{nom}, which we refer to as the *nominal speed*, set to $S_{nom} = HS(S')$. Γ_0 is a feasible schedule *and* it minimizes the energy consumption, but it does not necessarily meet the energy constraint. We use Γ_0 to obtain a schedule Γ_{nom} that meets the energy constraint. To this aim, we invoke the optimization module *OM* to select a set of jobs that achieves the performance objective while meeting the energy constraint; only these jobs are labeled as *selected* and all the others are labeled as *skipped*. All the selected jobs in Γ_{nom} run at the speed S_{nom} exactly as in Γ_0. The selected jobs in Γ_{nom} meet their deadlines since Γ_{nom} is obtained by eliminating some jobs from Γ_0, which is feasible in hard RT sense. It also meets the energy constraint since the job selection is done using *OM*, which never exceeds the energy budget, by definition. The job selection schemes we discussed in Section 4.1 are still applicable in this context.

The schedule Γ_{nom} is not optimal. In fact, computing the optimal solution is NP-Hard (Theorem 4). On the other hand, computing the total number of *selected* instances for all the tasks has a complexity of $O(n \log n)$ (the dominant term coming from the initial task sorting).. The simple structure of Γ_{nom} makes it possible to perform reclamation *greedily* during the mission at *every* task completion/preemption point, without compromising the deadlines of selected tasks (unlike the work in [22] where reclamation occurs only at hyperperiod/frame boundaries).

4.3.2. Principles of dynamic adjustment

For systems without DVS, we presented two forms of reclamation, namely job promotion and job demotion. Both techniques are still applicable to systems with DVS. In addition, we can also have *CPU-time reclamation* through *dynamic speed reduction*. When tasks complete early, we have some slack time that can be used to further increase the excess energy by reducing the execution speeds of some subsequent jobs (as long as this does not compromise the deadlines of *already selected* tasks). By reducing the execution speed of a job, we also reduce its energy consumption due to the convex nature of the relation between power and speed.

In [3], Aydin et al. propose a dynamic speed reduction scheme for periodic tasks assuming continuous speed spectrum. They refer to this scheme as *Dynamic Reclaiming Algorithm* (*DRA*). We adapt DRA to energy-constrained RT systems and to the case of discrete speeds. The extended scheme adds dynamic speed reduction to job promotion and demotion. We summarize below the principles of DRA and how it is extended in our solution.

DRA starts with a static optimal schedule where each job presents its worst-case execution time. In this schedule, all jobs run at the same speed, namely the nominal speed U_v, which minimizes the energy assuming continuous speed without affecting feasibility. DRA computes the amount of CPU time that each job can *safely* use to slow down its execution at *dispatch* time. This additional CPU time is referred to as the *earliness*. The gist of the scheme is how to calculate the *earliness* without affecting feasibility. Since it is impractical to produce the entire schedule a priori, DRA uses a data structure called the *α-queue*. The *α-queue* is effectively the ready queue of the static optimal schedule. The formula for calculating the earliness as well as the details of the DRA are given in [3, 5].

Our solution is based on DRA with the following extensions: We start with Γ_{nom} as the initial (static) schedule. However, at any time, the *α-queue* reflects the ready queue of Γ_0 which is a feasible schedule that approximates the static optimal schedule with continuous speed, assuming sufficient energy. At dispatch time, the CPU speed is reduced by computing the earliness with respect to Γ_0. Note that the skipped jobs of Γ_{nom} can be considered as the jobs in Γ_0 with *zero actual execution time*, enabling us to further reduce the speed. Based on this earliness amount, we determine the lowest speed that still guarantees the deadlines of *already selected* jobs (through HS(.) function) for the dispatched job. By running at a lower speed we further increase excess energy that can later be used for job promotion. **The speed reduction helps job promotion and is not an alternative to it (because Γ_0 is feasible in hard RT sense).** When a job is promoted, its initial speed is always S_{nom}. The fact that all the deadlines of selected tasks are met follows from (i) the feasibility of DRA (proved in [3]), and (ii) the fact that the selected jobs run at a speed that does not exceed S_{nom} at all times, consequently ensuring a workload that does not exceed the one in the static schedule Γ_{nom}.

4.3.3. On-line solutions for systems with DVS

Three reclamation and task selection schemes ONC, ONA and ONS are still applicable to DVS settings. Just as in non-DVS settings, each scheme starts with an initial off-line schedule generated before the mission starts, which can later be modified on-line through reclamation. Moreover, each scheme still invokes an optimization module *OM* chosen depending on the performance objective of interest. The information with which *OM* is invoked for each module depends on whether it is conservative (uses worst-case workload information), aggressive (uses average-case workload information), or speculative (estimates average-case workload information). In addition, in DVS settings the CPU speeds of selected tasks are reduced whenever possible (see Section 4.3.2).

5. Experimental evaluation

In order to evaluate our schemes experimentally, we implemented a discrete-event simulator. We evaluated:

- **Off-line conservative scheme (OFC):** the static solution that uses worst-case workload information, and does not perform any on-line adjustments.
- **On-line conservative scheme (ONC):** dynamically uses job promotion assuming worst-case workload.
- **On-line aggressive scheme (ONA)** that makes use of its (additional) knowledge about average workload when selecting jobs.
- **On-line speculative scheme (ONS)** that tries to estimate average-case workload by monitoring the actual workload, using both job promotion and demotion.

All the schemes invoke the optimization module *OM* that is carefully chosen to maximize system performance subject to the performance objective of choice; we use FSJ as the optimization module for O1, while LRD is used for O2. The computational complexity of OM is $O(n \log n)$ at each invocation in both cases. We applied these schemes to systems with and without DVS. In both cases, we compared the performance of our proposed schemes to a *clairvoyant* optimal scheme, which we refer to as **Bound**, which has knowledge about the *actual* workload information. This scheme provides an upper bound on the performance of our schemes. In the absence of DVS, to implement **Bound** we use a clairvoyant optimal scheme that uses FSJ for O1 and an optimization module based on dynamic programming for O2. In the case of DVS, we use a clairvoyant scheme that provides an upper bound on the performance of the (real) optimal solution, since the latter is computationally too expensive. This scheme maximizes system reward without exceeding the energy budget; however it only ensures that the completion time of the total workload does not exceed the mission time without worrying about individual deadlines. Clearly, this scheme outperforms the optimal solution in terms of total reward and provides an even more optimistic performance bound.

In our simulation experiments we measured the total system reward R as a function of three parameters:

- *System energy budget, E_{budget}*, represented as a percentage of the total energy required to meet all deadlines, E_{bound}. If the system has DVS capability, then E_{bound} is calculated assuming that all jobs run at the nominal speed (the lowest speed with which all deadlines can be met without considering energy budget) throughout the mission time.
- *Execution time ratio, ER*, which is the ratio of best-case execution time to worst-case execution time that is taken to be the same for all tasks. *ER* represents the variability of the actual workload compared to the worst-case.
- *Total system utilization, $U = \sum C_i^m / P_i$*. Note that $C_i^m = C_i$ if the system does not have DVS capability.

We generated 100 generic task sets with 30 tasks each, and then for each task set we ran 1000 experiments to generate actual execution requirements for each task. We changed the above 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_{bound}) ranged from 10% to 100% in increments of 10%, *ER* ranged from 0.1 to 1.0 in increments of 0.1. Task periods were generated according to a uniform probability distribution. The mission time X was 5 times the hyperperiod P. Similarly, the actual execution time *AET* of any given job was generated uniformly between *ER*WCET* and *WCET*, where *WCET* is its worst-case execution time. Finally, for objective O2, the weight w_i of each task was chosen according to a uniform probability distribution in the interval [1, 10]. For O1, each task weight was set to 1. We computed the average reward achieved by each scheme over this spectrum. We present all the results as a percentage of system reward improvement over the static scheme OFC.

5.1. Experimental results for systems without DVS capability

In the absence of DVS, the static solution for O1 is provided optimally by FSJ. The problem of achieving O2 is NP-Hard. In both cases, the on-line schemes do provide improvement upon OFC, through dynamic job promotion.

Figure 1 shows the reward improvement of our reclamation schemes over OFC as a function of E_{budget} at U=0.7 and ER=0.4 for objectives O1 (left) and O2 (right). When E_{budget} is low, the system is very energy-constrained and it is crucial to utilize any excess energy due to early completions to achieve the performance objective. Hence, under such conditions the difference in performance between the different schemes is significant. The on-line reclamation schemes provide improvements up to 25% over OFC. ONA and ONS outperform ONC because of their aggressive nature while selecting jobs for execution. Moreover, ONA approaches the performance of the optimal scheme by a margin of 10%. As E_{budget} increases the system becomes less energy-constrained; more task instances can be executed, increasing the overall reward. The schemes converge when E_{budget} = 100%, because the system has enough energy to meet all the deadlines and the reward reaches its maximum value. Examining Figure 1, one notices that the reward improvement for O1 (left) is slightly higher than that for O2 (right). This is due to the difference in task reward assignment between O1 and O2 and the greedy nature of our solutions. For O1 all tasks have the same reward w_i = 1, and hence dynamically promoted jobs accrue the same reward as originally selected ones. For O2, on the other hand, the reward can potentially be different for different tasks. The optimization module that consists of LRD gives preference to jobs with largest reward density return among the unselected jobs whenever it is invoked.

Figure 2 (left) shows the reward improvement vs. execution time ratio *ER* for E_{budget} = 30% and U = 0.7 for non-DVS O2. As *ER* increases the actual workload increases, compared to the worst-case. At small *ER* values, say 0.1, early completions yield more slack time, resulting in ample excess energy to be used by the reclamation schemes. Hence, at small *ER* values the use of aggressive and speculative techniques pay off and hence the benefit of ONA and ONS is greater. Figure 2 (right) shows the reward improvement vs. utilization for E_{budget}=E_{bound}(U = 0.3) and non-DVS O2. Unlike Figure 1 and Figure 2 (left) where E_{budget} is recalculated as a percentage of E_{bound}, which is also a function of the total utilization U, in this set of experiments E_{budget} is set to a fixed value, namely the energy required to meet all the deadlines when $U=0.3$ (i.e. $E_{bound}(U=0.3)$). When $U \leq 0.3$ the system has enough energy budget to meet all deadlines (i.e. $E_{budget} \geq E_{bound}$) and all heuristics yield the same reward, which is simply the maximum possible system reward. As U increases from 0.3 to 1.0, the system becomes effectively more energy-constrained and the performance of the online schemes compared to OFC improves.

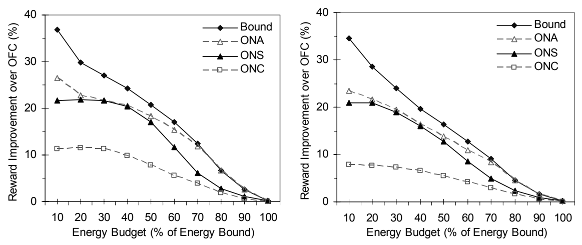

Figure 1. Effect of E_{budget} on non-DVS systems for objective O1 (left) and objective O2 (right)

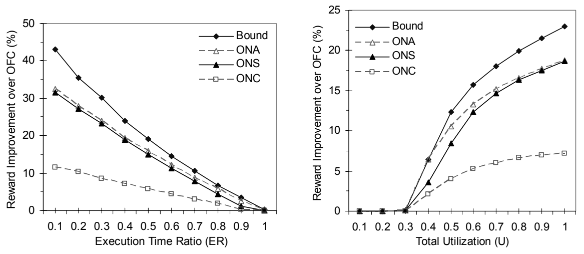

Figure 2. Effect of execution time ratio (left) and utilization (right) on non-DVS systems for objective O2

5.2. Experimental results for systems with DVS capability

If the system has DVS capability then the reclamation schemes ONC, ONA and ONS are still applicable. Further, our schemes now resort to dynamic speed reduction upon detecting early completions. This results in additional energy savings and opportunities to promote jobs during mission time, thus improving the system performance. The combined dynamic promotion/speed reduction scheme was presented in Section 4.3.2. In the next set of experiments, we use the Intel XScale specifications shown in Table 1 [12]. Here, ***Bound*** is an upper bound on the reward of the clairvoyant optimal solution.

Table 1. Intel XScale speed settings and voltages

Speed (MHz)	150	400	600	800	1000
Voltage (V)	0.75	1.0	1.3	1.6	1.8

Figure 3 shows reward improvement over OFC as a function of E_{budget} for U = 0.7 and ER = 0.4 for objectives O1 (left) and O2 (right). We observe the same general patterns as in the non-DVS case. In addition, we note that the improvement of our reclamation schemes over OFC is much higher with DVS than in the case of non-DVS. This is due to the fact that by utilizing low-speed/low-energy levels, we effectively increase excess energy that can be used to promote additional jobs.

Figure 4 (left) shows the reward improvement as a function of execution time ratio *ER* for E_{budget}=30% and U=0.7 for DVS O2. Again, the general patterns are similar to the non-DVS case. But in addition, we notice the improved performance of our reclamation schemes compared to OFC. Note also that ONS slightly outperforms ONA at high *ER* values (above 0.4) and at low E_{budget} values (from Figure 3). ONA is an aggressive scheme, but this aggressiveness may involve a cost; by underestimating the actual workload ONA can select a large number of jobs which results in subsequent demotions undertaken by the algorithm to guarantee the completion of the mission, resulting in a decrease in reward. This critical region emphasizes this shortcoming of ONA. ONS, on the other hand, is a less aggressive scheme; it starts with an initial schedule assuming worst-case workload and gradually estimates the actual workload using exponential averaging. Recall that *Bound* is an upper bound on the optimal solution since it makes the best use of available energy to complete all the workload before the end of the mission, without paying attention to individual deadlines. Interestingly, we also observe that on-line schemes yield a gain over the static scheme even when *ER=1.0.* This is due to the fact that the scheme is able to reduce the CPU speed dynamically by considering the CPU time allocated to *skipped* tasks with respect to the nominal schedule Γ_{nom} (see Section 4.3.2). Figure 4 (right) shows the reward improvement as a function of the utilization for E_{budget}=E_{bound}(U=0.3) for DVS O2.

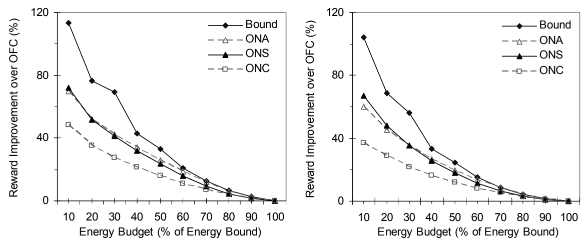

Figure 3. Effect of E_{budget} on DVS systems for objective O1 (left) and objective O2 (right)

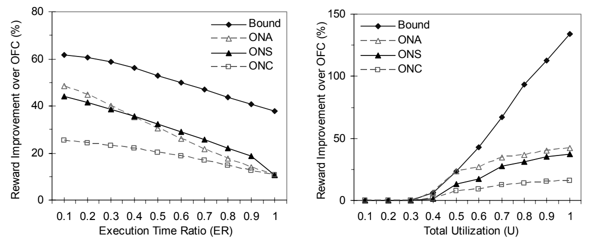

Figure 4. Effect of execution time ratio (left) and utilization (right) on DVS systems for objective O2

6. Conclusion

This research effort was aimed at addressing the fundamental RT scheduling issues in the context of systems with a given energy budget and mission time. We considered separately:

- systems both with and without DVS capability,
- the objectives of maximizing the total number of deadlines met and the total reward, and
- on-line and off-line solutions.

Our solutions are based on skipping less important jobs in a controlled manner to achieve performance objectives while guaranteeing that the system remains functional until the end of the mission. For DVS-enabled systems, we extended the Dynamic Reclaiming Algorithm (DRA) of [3] to energy-constrained settings in order to dynamically reduce CPU speed to obtain additional excess energy for selecting new jobs in a dynamic fashion. Our algorithms differ both in the type of workload information they have: conservative, aggressive, and speculative algorithms use worst-case, average-case, and estimated workload respectively. They also differ in their reclamation behavior: off-line schemes use reclamation while on-line ones do not. The simulation results show that on-line schemes can provide up to 70% and 30% reward improvement over the off-line schemes for systems with or without DVS capability, respectively.

References

[1] T. A. AlEnawy and H. Aydin. Energy-Constrained Performance Optimizations For Real-Time Operating Systems. *Workshop on Compilers and Operating Systems for Low-Power (COLP'03)*, New Orleans, LA, September 2003 (available on-line at http://cs.gmu.edu/~aydin/colp03.pdf).

[2] H. Aydin, R. Melhem, D. Mossé and P.M. Alvarez. Optimal Reward-Based Scheduling for Periodic Real-Time Tasks. *IEEE Transactions on Computers,* 50(2), February 2001.

[3] H. Aydin, R. Melhem, D. Mossé and P.M. Alvarez. Dynamic and aggressive scheduling techniques for power-aware real-time systems. *Proceedings of the Real-Time Systems Symposium (RTSS'01),* 2001.

[4] H. Aydin and Q. Yang. Energy-aware partitioning for multiprocessor real-time systems. *Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS'03), Workshop on Parallel and Distributed Real-Time Systems*, 2003.

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

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

[7] S. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. Rosier, and D. Shasha. On-line scheduling in the presence of overload. *IEEE Annual Symposium on the Foundations of Computer Science*, 1991.

[8] G. Bernat, A. Burns, A. Llamosi. Weakly Hard Real-time Systems. *IEEE Transactions on Computers,* 50(4), April 2001.

[9] G. Buttazzo, M. Spuri, and F. Sensini. Value vs. Deadline Scheduling in Overload Conditions. *Proceedings of the Real-Time System Symposium,* Pisa, Italy, 1995.

[10] G. Buttazzo. Hard Real-Time Computing Systems. Kluwer Academic Publishers, Boston, 1997

[11] G. Buttazzo, G. Lipari, and L. Abeni. Elastic Task Model for Adaptive Rate Control. *Proceedings of the IEEE Real-Time Systems Symposium*, Madrid, Spain, 1998.

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

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

[14] M. Hamdaoui and P. Ramanathan. A dynamic priority assignment technique for streams with (m,k)-firm deadlines. *IEEE Transactions on Computers*, 44(12), 1995.

[15] D. Kang, S. Crago, and J. Suh. A Fast Resource Synthesis Technique for Energy-Efficient Real-time Systems. *Proceedings of IEEE Real-Time Systems Symposium*, 2002.

[16] G. Koren, and D. Shasha. D-over. An optimal on-line scheduling algorithm for overloaded real-time systems. *Proceedings of the IEEE Real-Time Systems Symposium*, 1992.

[17] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real time environment, *Journal of the ACM*, 17(2). 1973.

[18] J. Liu, K. J. Lin, W. K. Shih, A.C. Yu, J. Y. Chung, W. Zhao. Algorithms for Scheduling Imprecise Computations. *IEEE Computer 24(5),* 1991

[19] J. Liu. *Real-Time Systems*. Prentice Hall, NJ, 2000.

[20] P. Pillai and K.G. Shin. Real-time dynamic voltage scaling for low power embedded operating systems. *Symposium on Operating Systems Principles*, 2001.

[21] C. Rusu, R. Melhem and D. Mossé. Maximizing the system value while satisfying time and energy constraints. *Proceedings of the Real-Time Systems Symposium*, 2002.

[22] C. Rusu, R. Melhem, and D. Mossé. Multi-Version Scheduling in Rechargeable Energy-Aware Real-time Systems. *Euromicro Conference on Real-Time Systems (ECRTS'03)*, Porto, 2003.

[23] S. Saewong and R. Rajkumar. Practical Voltage-Scaling for Fixed-Priority Real-time Systems. *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'03)*, May 2003.

[24] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. IEEE Annual Foundations of Computer Science, 1995.

Appendix A

In this section, we present the proof of Theorem 1. We first formally define the problem in the context of a simple task model where all the tasks have the same deadline:

EC-FEASIBILITY: Given a DVS-enabled CPU with $m > 1$ speed levels and a task set $T=\{T_1, T_2, ..., T_k\}$ with a common ready time $R = 0,$ is it possible to complete all the tasks before a common deadline $X,$ while remaining within a fixed energy budget E_{budget} ?

We will show that EC-FEASIBILITY reduces to a restricted version of PARTITION. We will denote this special case of PARTITION as 2n-PARTITION.

2n-PARTITION: Given a set of *2n* items $S = < s_1, s_2, \ldots, s_{2n} >$ where each item s_i has an integer cost z_i, is it possible to partition S into 2 subsets S' and S'' such that the total cost of items in each subset is the same and each subset contains exactly one element of every pair (s_{2i-1}, s_{2i}) $i=1 \ldots n$?

This restricted version of PARTITION is known to be NP-Hard (p.223 in [13]). Note that the solution to 2n-PARTITION requires that we consider each pair $(s_1, s_2), (s_3, s_4), \ldots, (s_{2n-1}, s_{2n})$ separately and put each element of every pair to a different subset. We will show that 2n-PARTITION reduces to EC-FEASIBILITY even for the special case where the CPU has 2 speed levels (i.e. $m = 2$) and $g^{stb} = 0$.

Given an instance of 2n-PARTITION, we can construct the corresponding instance of EC-FEASIBILITY as follows. The task set has $k = n$ tasks, where C_i^1 and C_i^2 denote the execution time of T_i at low and high CPU speed, respectively. Similarly, e_i^1 and e_i^2 denote the energy consumption of T_i at low and high CPU speed, respectively. We set $C_i^1 = e_i^2 = z_{2i-1}$ and $C_i^2 = e_i^1 = z_{2i}$ for $i = 1 \ldots n$. Further, let $X = E_{budget} = \frac{1}{2}\sum_{i=1}^{2n} z_i$.

Now, suppose that there is a polynomial-time solution to EC-FEASIBILITY. Then, given an instance of 2n-PARTITION, we can compute the answer in polynomial-time by constructing (in polynomial-time) the corresponding instance of EC-FEASIBILITY and then solving it. We claim that the answer to 2n-PARTITION instance is YES if and only if the answer to the corresponding instance of EC-FEASIBILITY is YES.

To start with, observe that the system has a fixed energy budget E_{budget} whose numerical value is equal to the common deadline X ("time budget"). Suppose that the answer to the EC-FEASIBILITY instance is YES, that is, all tasks complete within energy *and* time budget $X = E_{budget} = \frac{1}{2}\sum_{i=1}^{2n} z_i$. Then either low- or high-speed version of a each task T_i must be definitely scheduled. Observe that if the low-speed version of T_i *is* chosen, then z_{2i-1} contributes to Time Cost (total execution time) and z_{2i} contributes to Energy Cost (total energy consumption). Similarly, if the high-speed version of T_i *is* chosen, then z_{2i} contributes to Time Cost (total execution time) and z_{2i-1} contributes to Energy Cost (total energy consumption). Thus, in the corresponding instance of 2n-PARTITION, each element in the pair (s_{2i-1}, s_{2i}) can be seen as allocated to one subset ("Time Cost") while the other one is allocated to the other subset ("Energy Cost").

Now define the sets $TB = \{z_j \mid z_j$ contributes to Time Cost$\}$ and EB = $\{z_j \mid z_j$ contributes to Energy Cost$\}$. Observe that *TB* and *EB* are disjoint, and exactly one element of each pair (z_{2i-1}, z_{2i}) appears in *TB (EB).* The completion time of all the tasks, that is the sum of all elements in *TB,* must be smaller than or equal to X since the schedule is feasible. Similarly, the total energy consumption of all the tasks that is the sum of all elements in *EB* must be smaller than or equal to E_{budget} since the system does not run out of energy until the end of the mission. Since $X = E_{budget} = \frac{1}{2}\sum_{i=1}^{2n} z_i$, this shows that the sum of all the elements in *TB* and *EB* are equal, implying that the answer to the corresponding 2n-PARTITION instance is also YES.

Conversely, assume that the answer to 2n-PARTITION is YES. Then, consider the elements of S' and S''. We claim that the elements in S' imply which version of each task should be scheduled to meet the timing and energy constraints by the following reasoning: For every pair (s_{2i-1}, s_{2i}) $i = 1 \ldots n,$ one of them must be in S' while the other is in S''. The cost associated with the one in S' will denote the execution time of the version selected for T_i, while the cost associated with the one in S'' will denote the energy consumption of the version selected for T_i. Once again, observe that one element of the corresponding cost pair (z_{2i-1}, z_{2i}) contributes to Time Cost, while the other one contributes to Energy Cost. By virtue of being a solution to 2n-PARTITION instance, we have:

$$\sum_{s_i \in S'} z_i = \sum_{s_i \in S''} z_i = X = E_{budget} = \frac{1}{2}\sum_{i=1}^{2n} z_i$$, showing that all tasks complete within the common deadline and energy budget. Thus, the answer to EC-FEASIBILITY is YES. ■