# 36

# Periodic Reward-Based Scheduling and Its Application to Power-Aware Real-Time Systems

Hakan Aydin
*George Mason University*

Rami Melhem
*University of Pittsburgh*

Daniel Mossé
*University of Pittsburgh*

## 36.1 Introduction

Hard real-time systems are characterized by the utmost importance of satisfying the timing constraints at run-time. Consequences of missing the deadlines in a hard real-time system can be very serious, even catastrophic. Examples of real-time systems can be found in control systems of nuclear power plants, air traffic systems, and command-and-control applications. Thus, a vast collection of research works in *hard* real-time scheduling theory study the problem of guaranteeing the timely completion of tasks under various task and system models, namely the *feasibility* problem (see [1]). These studies necessarily consider worst-case scenarios when analyzing the problem, such as worst-case task execution times and minimum task inter-arrival times.

A tacit assumption of the hard real-time scheduling framework is that a task's output is of no value if it is not completed by the deadline. However, in a large number of emerging applications a *partial*

1-58488-397-9/$0.00+$1.50

or *approximate* result is acceptable as long as it is produced in a *timely* manner. Such applications can be found in the areas of multimedia, image and speech processing, time-dependent planning, robot control/navigation systems, medical decision making, information gathering, real-time heuristic search, and database query processing. For example, given a short amount of time, an approximate (or fuzzy) image can be produced and transmitted by a multimedia system, or a radar tracking system can compute the approximate location of the target.

*Reward-based (RB) scheduling* is based on the idea of trading precision for timeliness when the available resources are not sufficient to provide worst-case guarantees. Such a situation can also occur as a result of transient overload and/or faults. In this framework, each task is logically decomposed into two subtasks: the *mandatory* part produces a result of acceptable quality, and the *optional* part refines the result within the limits of available computing capacity. A nondecreasing *reward function*, associated with the execution of the optional part, captures its contribution to the overall system utility. The primary objective of RB scheduling is thus assuring the timely completion of mandatory parts, while maximizing a performance metric—usually the total reward accrued by the optional parts.

The apparent complexity of RB scheduling is partly due to the fact that it is a general framework encompassing both hard and soft real-time scheduling theories. In fact, a task with no optional part corresponds to a traditional hard real-time task, and a task with no mandatory part can model a soft real-time task.

## 36.2 Reward-Based Task Model, Problem Formulation, and Notation

A RB task $T_i$ comprises a mandatory part $M_i$ and an optional part $O_i$. Throughout the chapter, we will denote the worst-case execution times of $M_i$ and $O_i$ by $m_i$ and $o_i$, respectively. The mandatory part $M_i$ of a task becomes ready at task's release time $r_i$. The optional part $O_i$ becomes ready for execution only when the mandatory part $M_i$ completes. The mandatory part must complete successfully by the task's deadline, whereas the optional part may be left uncompleted by the deadline if more important/urgent objectives require so. In other words, no mandatory or optional execution can take place beyond the task's deadline $d_i$.

During its execution, $O_i$ *refines* the approximate result produced by $M_i$. To quantify the "utility" (or "accuracy") of the refinement process, we associate a reward function $R_i(t_i)$ with the execution of the optional part $O_i$, where $t_i$ is the amount of *optional* service time $T_i$ receives beyond the mandatory part.

All studies in RB scheduling [2–7] assume that the reward functions are nondecreasing, which is based on the observation that the utility of a task does not decrease by allowing it to run longer in almost every practical application.

A schedule of RB tasks is *feasible* if all the mandatory parts complete in a timely fashion. A feasible schedule is also *valid* if additional constraints that may be imposed by the problem definition, such as precedence constraints or execution without preemption, are satisfied. Furthermore, a feasible and valid schedule is *optimal* if the reward accrued maximizes a performance metric. The performance metric most often considered is to maximize the (weighted) total reward of RB tasks [2–4,6,8].

An interesting question concerns the types of reward functions which represent realistic application areas. Figure 36.1 shows the form of the most common reward function types. A *linear* reward function [3,6,8–11] models the case where the benefit to the overall system increases *uniformly* during the optional execution. The simplest case is to have *identical* linear functions, while *nonidentical* (or weighted) linear functions allow us to distinguish between the importance of optional parts of different tasks.

*Concave* reward functions [4,5,7,12] go further by addressing the case where the greatest increase/refinement in the output quality is obtained during the first portions of the optional execution. A function $f(x)$ is concave if and only if for all $x$, $y$ and $0 \leq \alpha \leq 1$, $f(\alpha x + [1-\alpha]y) \geq \alpha f(x) + (1-\alpha) f(y)$. Geometrically, this condition means that the line joining any two points of a concave curve *may not be above the curve.* Examples of concave functions are linear functions $(kx + c)$, logarithmic functions $(\ln[kx + c])$, some exponential functions $(c \cdot [1 - e^{-kx}])$, and $k$th root functions ($x^{1/k}$, where $k \geq 1$). Note that the first derivative of a nondecreasing concave function is nonincreasing (diminishing returns).

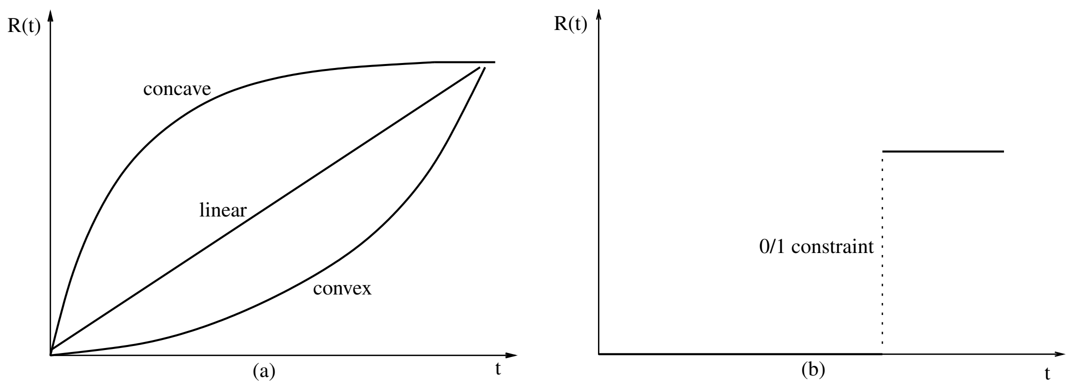

**FIGURE 36.1** Reward functions: (a) continuous concave, convex, and linear; (b) 0-1 constraint.

Having nondecreasing concave reward functions means that although the reward monotonically increases beyond $M_i$, its *rate of increase* decreases or remains constant with time. Hence, linear and concave reward functions successfully model the applications where earlier slots of optional executions are *at least* as valuable as the later ones. In fact, a large number of typical applications reported in the literature can be modeled by concave functions since they have nonincreasing marginal returns. As mentioned above, these areas include image and speech processing, multimedia applications, time-dependent planning, robot control/navigation systems, real-time heuristic search, information gathering, and database query processing.

*Convex* reward functions (a function $f$ is convex if the function $-f$ is concave) represent the case where a considerable benefit is obtained only later during the optional execution. It should be noted that reward functions with 0-1 constraints, where no reward is accrued unless the *entire* optional part is executed, have also received interest in the literature [6,8,13]. Unfortunately, scheduling with 0-1 constraints has been shown to be NP-Complete in [8].

Note that the reward functions must also take into account the worst-case execution time of optional parts: no reward can be accrued by $O_i$ beyond the upper bound $o_i$. Throughout the chapter we will assume that the reward function $R_i(t_i)$ of a task $T_i$ is given by

$$R_i(t_i) = \begin{cases} f_i(t_i) & \text{if } 0 \le t_i \le o_i \\ f_i(o_i) & \text{if } t_i > o_i \end{cases} \tag{36.1}$$

where $t_i$ is the amount of CPU time allocated to the *optional* part $O_i$ (recall that the mandatory part $M_i$ must be executed fully). We will suppose that each $f_i$ above is a nondecreasing, concave and continuously differentiable function over nonnegative real numbers, unless stated otherwise.

We are now ready to define the general RB Scheduling Problem.

### 36.2.1 Reward-Based Scheduling Problem

Consider a RB real-time task set $\mathbf{T} = \{T_1, \ldots, T_n\}$, where each task $T_i$ is decomposed to a mandatory part $M_i$ and an optional part $O_i$. A nondecreasing and concave reward function $R_i(t_i)$ is associated with the execution of each optional part $O_i$. Given a time point $Z$, determine the *optimal schedule* $\mathcal{OPT}$ in the interval $[0, Z]$, such that $\mathcal{OPT}$ is feasible and valid, and each optional part $O_i$ receives service for $t_i \le o_i$ units of time so as to maximize the total system reward $\sum_i R_i(t_i)$.

Determining the optimal schedule for RB tasks clearly involves the computation of optimal optional service times (the $t_i$ values). Noting that the reward accrued by each optional part $O_i$ does not increase beyond the upper bound $o_i$, this computation can be expressed as an optimization problem where the objective is to find $t_i$ values[1] so as to

$$\text{Maximize} \quad \sum_{i=1}^{n} R_i(t_i) \tag{36.2}$$

$$\text{subject to} \quad 0 \leq t_i \leq o_i, \;\; i = 1, \ldots, n \tag{36.3}$$

$$\text{There exists a feasible and valid schedule with } \{m_i\} \text{ and } \{t_i\} \text{ values} \tag{36.4}$$

We reiterate that the "validity" condition in the constraint (36.4) may capture any requirements imposed by the task and system model (such as nonpreemptive scheduling or the existence of precedence constraints).

## 36.3 Related Scheduling Frameworks

A number of research studies that appeared in real-time scheduling literature have common traits with the RB scheduling framework introduced in Section 36.2. Below we briefly review these models and major research studies therein.

### 36.3.1 Imprecise Computation

A large body of work in RB scheduling originated from *Imprecise Computation* model, where the study in [9] can be considered as a starting point. In this work, the mandatory/optional semantic distinction and the imprecise computation model were introduced. The objective of the scheduling problem was defined as guaranteeing the timeliness of mandatory parts, while minimizing the total error. The *error* of a task is the amount of optional work left uncompleted. Later, the concept was generalized to weighted and general error functions. As a task's optional part executes, the *(precision) error* decreases, according to the specified error function. Notice that the error functions in this context are analogous to (dual of) the reward functions of our framework: a concave reward function corresponds to a convex error function, and vice versa.

In [14], optimal preemptive algorithms for Imprecise Computation model were first proposed using network-flow formulation. Faster algorithms were devised later in [8] to optimally schedule $n$ independent tasks with identical linear reward functions (in time $O(n \log n)$) and linear reward functions with different weights (in time $O(n^2)$). Subsequently, Leung, Yu, and Wei [15] gave an $O(n \log n + kn)$-time algorithm for a single processor, where $k$ is the number of distinct weights. Minimizing the total error with 0-1 constraints have been shown to be NP-hard even with identical weights in [8]. Ho, Leung, and Wei [13] proposed a $O(n^2)$ heuristic for this last problem, along with polynomial-time algorithms for maximizing the *number* of optional tasks that are entirely executed under 0-1 constraints. Note that the imprecise computation problem with precedence constraints is not computationally harder than the one in the independent task model, since we can first modify the ready times/deadlines of tasks to "reflect" precedence constraints and then solve that problem [6].

More recently, the *extended imprecise computation model* was introduced by Feng and Liu in [16]. This study describes several heuristics regarding the problem of assigning service times to tasks within chains. All the tasks in a chain share a common, single temporal constraint. Each chain is composed of imprecise computation tasks with linear precedence constraints and the input quality of each task depends upon the output quality of its predecessor. This work assumes linear error functions.

The problem of imprecise computations for *periodic* tasks was first addressed in [10,17]. A more detailed study appeared in [3] where the possible application areas are classified as "error noncumulative" and

[1]When considering the periodic task model, the execution time of each task instance ($t_{ij}$) should be considered as a separate unknown (see Section 36.4).

"error cumulative." In "error cumulative" applications, such as radar tracking, an optional instance must be executed completely at every (predetermined) $k$ invocations. The authors further proved that the case of error cumulative jobs is an NP-Complete problem. On the other hand, in "error noncumulative" applications, errors (or optional parts left unexecuted) have no effect on the future instances of the same task. Well-known examples of this category are image/speech processing, information retrieval, and display tasks. For these jobs, the authors showed how to guarantee a feasible schedule by constantly favoring mandatory parts and proposed a class of heuristics to minimize the total error through *Mandatory-First* approach (see Section 36.4.1). All heuristics, except one, operate independently from the nature of error functions.

On-line scheduling of imprecise computation tasks has been examined in [11]. The algorithms address separately the cases where the workload consists solely of on-line tasks and of a mix of on-line and off-line tasks. Only identical linear error functions are considered. However, Baruah and Hickey later proved [18] that, in general, an on-line algorithm for imprecise computation tasks may perform arbitrarily badly when compared to a clairvoyant scheduler.

Performance metrics other than maximizing the (weighted) total error have been also studied within the imprecise computation model. In [19], two polynomial-time algorithms are presented for unit-weight error functions to obtain a "balanced" distribution of error among imprecise computation tasks. In other words, the aim is to minimize the maximum error. For the problem of minimizing maximum error, Ho, Leung, and Wei [20] gave an $O(n^2)$-time algorithm for a single processor and an $O(n^3 log^2 n)$-time algorithm for multiprocessors.

Leung and Wong investigated the problem of minimizing the number of tardy tasks for a given maximum, tolerable error; and showed it to be NP-hard [21]. In [22], minimizing the average response time, while keeping the total error less than a threshold has been shown to be NP-hard for almost every reasonable model.

### 36.3.2 Increased-Reward-with-Increased-Service Model

The Increased-Reward-with-Increased-Service (IRIS) framework allows tasks to get increasing reward with increasing service, without an upper bound on the execution times of the tasks and without the separation between mandatory and optional parts [4]. A task executes for as long as the scheduler allows. Typically, a nondecreasing *concave* reward function is associated with each task's execution time. In [4,5] the problem of maximizing the total reward in a system of independent aperiodic tasks is explored. An optimal polynomial-time solution with static task sets (identical ready times) is presented, as well as two extensions that include mandatory parts and on-line policies for dynamic task arrivals.

### 36.3.3 Quality-of-Service-Based Resource Allocation Model

A QoS-based resource allocation model has been proposed in [7,12,23,24]. In that study, the problem is to optimally allocate multiple resources to the various applications such that they simultaneously meet their minimum requirements along multiple QoS dimensions and the total system utility is maximized. In one aspect, this can be viewed as a generalization of optimal CPU allocation problem in RB scheduling, to multiple resources and multiple reward metrics (quality dimensions). Further, dependent and independent quality dimensions are separately addressed for the first time in this work.

However, a fundamental assumption of Q-RAM model is that the resource allocation is made in terms of *utilization of resources*, usually expresssed as a fraction (percentage) of the available resource capacity. In the CPU allocation case for the periodic real-time tasks, this translates to identical service time allocation assumption for each instance of a given task. The periodic RB scheduling framework we consider in this chapter assumes that the reward accrued has to be computed separately over *each* instance for the periodic case. In other words, we are not making the assumption that all the instances of a periodic task will have the same CPU allocation time, which can lead to suboptimal results for some settings.

In [7], Rajkumar et al. addressed the case of a single resource and multiple QoS (reward) dimensions and provided a pseudopolynomial time algorithm for continuous concave utility functions. In [12], the problem of a single resource and two or more QoS dimensions (with one dimension being discrete and dependent on another) is explored. The problem is proven to be NP-hard, and an approximation algorithm based on Concave Majorant Optimization is presented. Then, the authors show that the problem of multiple resources and single QoS dimension can be reduced to a mixed integer programming problem; they also report the run-time of example problems when solved by commercial software packages.

Lee et al. further considered discrete QoS dimensions in [23], as opposed to the continuous QoS dimension assumption of their previous research in [7]. They proposed two approximation algorithms for the case of single resource and multiple quality dimensions. The first algorithm yields a solution that is within a known bounded distance from the optimal solution, while the second is capable of returning an allocation whose distance from the optimal solution can be reduced by giving more time to the allocation routine.

Finally, the more general case of multiple resources and QoS dimensions was addressed in [24]. Due to the intractable nature of the problem, a mixed integer programming formulation is developed to yield near-optimal results, followed by an approximation algorithm based on local search techniques. The execution time and quality distance to the optimal solution of these algorithms are experimentally evaluated.

## 36.4 Periodic Reward-Based Scheduling

Many real-time tasks are *periodic* in nature: Assuming that it is ready for execution at $time = 0$, a periodic real-time task $T_i$ with period $P_i$ will generate a new *task instance (job)* every $P_i$ time units. Specifically, the $j^{th}$ instance of $T_i$ (denoted by $T_{ij}$) is invoked at $t = (j-1) \cdot P_i$ and it has to be completed by the time of next invocation at $t = j \cdot P_i$ (in other words, we will assume that the relative deadline of each instance is equal to its period).

A large number of periodic real-time applications can be modeled by the RB scheduling framework. Well-known examples are periodic tasks that receive, process, and transmit video, audio, or compressed images and information retrieval tasks. With the advance of multimedia and networking technologies, this class of applications are likely to become more and more widespread. Moreover, these applications readily admit a mandatory/optional semantic distinction: the mandatory part produces an image, frame, or output of acceptable quality and the optional part improves on it.

The extension of the basic RB task model to the periodic execution settings is relatively simple: a schedule of periodic RB tasks is **feasible** if mandatory parts meet their deadlines at every task invocation. Note that the optional execution times of different instances belonging to a given periodic task $T_i$ can be potentially different.

The common performance metric [2,3] for the feasible periodic RB task schedules is **the average cumulative reward**, given by

$$REW_{CUM} = \frac{P_i}{P} \sum_{i=1}^{n} \sum_{j=1}^{P/P_i} R_i(t_{ij}) \tag{36.5}$$

where $P$ is the *hyperperiod*, that is, the least common multiple of $P_1, P_2, \ldots, P_n$ and $t_{ij}$ is the amount of CPU time assigned to the $j$th instance of optional part of task $T_i$ (i.e., $O_{ij}$). Observe that it is necessary and sufficient to generate a schedule that maximizes the total reward during the hyperperiod $P$, since the schedule will repeat itself every $P$ time units. Also note that, in the above expression, the average reward of $T_i$ is computed over the number of its invocations during the hyperperiod $P$.

Thus, the solution to the periodic RB scheduling problem must

1. ensure that the mandatory part of each periodic task instance meets its deadline, and
2. determine the optional execution times of all the periodic task instances to maximize the performance metric given by Equation (36.5).

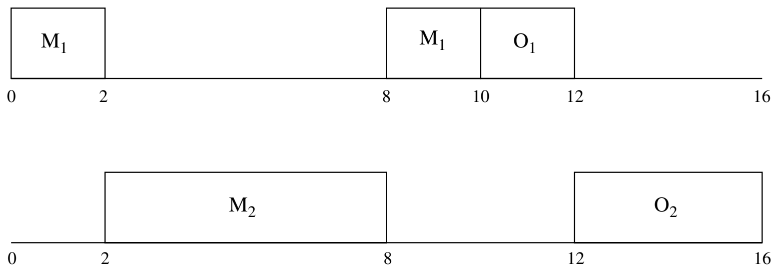

**FIGURE 36.2** The solution produced by a mandatory-first algorithm.

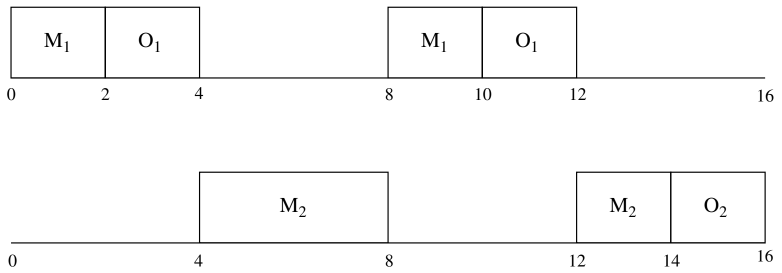

**FIGURE 36.3** An optimal schedule.

### 36.4.1 Mandatory-First Solutions

In one of the earliest attempts attacking the periodic RB scheduling problem, Chung, Liu, and Lin [3] gave priority to guaranteeing the feasibility constraint. To achieve this, they proposed statically assigning higher scheduling priorities to all the mandatory parts with respect to the optional parts. In other words, in this *Mandatory-First* strategy, no optional part is scheduled if there is a ready mandatory part in the system. In their original proposal, the relative scheduling priorities of mandatory parts are determined according to the Rate Monotonic Scheduling policy [25], but it is clear that other policies can be also used. For the priority assignment to the optional parts, the authors proposed a number of heuristics, such as considering the deadlines, periods, or best incremental rewards. As an example, consider an RB task set consisting of two tasks, where $P_1 = 8$, $m_1 = 2$, $o_1 = 2$, $P_2 = 16$, $m_2 = 6$, $o_2 = 10$. Let the reward functions of $T_1$ and $T_2$ be given by $f_1(t_1) = 8 \cdot t_1$ and $f_2(t_2) = 2 \cdot t_2$. Then, an "intelligent" Mandatory-First algorithm, considering that the reward associated by $O_1$ is much higher than that of $O_2$, would produce the schedule shown in Figure 36.2 yielding a total cumulative reward of 12.

Yet, it is easy to see that the optimal solution for this specific problem instance involves delaying the execution of $M_2$ to the favor of "valuable" $O_1$, giving a total cumulative reward of 18 (see Figure 36.3).

We can see that the reward performance of the mandatory-first schemes is inherently limited: In the example above, the best mandatory-first scheme could yield only 2/3 of the optimal reward. In fact, in [2] it is proven that, in the worst-case, the relative performance of the mandatory-first schemes can be arbitrarily bad when compared to the optimal policy (i.e., the reward ratio of the best mandatory-first scheme to the reward of the optimal policy can be arbitrarily close to 0). It is clear that the optimal solution must be based on principles fundamentally different from Mandatory-First heuristic.

### 36.4.2 Optimal Solution

The difficulty involved in the periodic RB scheduling problem stems from the fact that the feasibility and reward maximization objectives must be achieved simultaneously. Moreover, even if we temporarily disregard the feasibility constraint above, the optimization problem itself remains challenging: the number of unknowns to be determined ($t_{ij}$'s in the previous expression (36.5)) can be potentially *exponential* in the number of tasks, $n$. Consequently, any *efficient* solution to the problem hinges in the first place on being able to reduce the number of unknowns without sacrificing optimality.

The theorem below establishes that for the most common (i.e., concave and linear) reward functions, the number of unknowns is only $n$ (the number of tasks):

**Theorem 36.1 (from [2])**

*If all the reward functions are concave, then the periodic RB scheduling problem has an optimal solution in which a given task $T_i$ receives the* same *optional service time at* every *instance, that is, $t_{ij} = t_{ik} = t_i \; \forall j, k$.*

The same study [2] also shows that, once $t_1, \ldots, t_n$ are determined, then *any periodic real-time scheduling policy that can achieve 100% CPU utilization* may be used to obtain a feasible schedule with these assignments. Examples of such policies are Earliest-Deadline-First (EDF) [25] and Least-Laxity-First (LLF) [26].

Interestingly, Theorem 36.1 extends also to the *identical multiprocessor* settings [2]. However, in general, EDF and LLF cannot be used to achieve full utilization on multiprocessors. An example policy that can be adopted for this purpose is proposed by Bertossi and Mancini in [27].

Theorem 36.1 is crucial in reducing the number of unknowns to $n$, but it is based on an existence proof and the problem of computing $t_1, \ldots, t_n$ efficiently (in polynomial-time) is equally important. One can obtain the following nonlinear optimization problem formulation to compute the optimal optional service times $t_1, \ldots, t_n$:

$$\text{Maximize} \quad \sum_{i=1}^{n} f_i(t_i) \tag{36.6}$$

$$\text{subject to} \quad \sum_{i=1}^{n} \frac{P}{P_i} t_i \le P - \sum_{i=1}^{n} \frac{P}{P_i} m_i \tag{36.7}$$

$$0 \le t_i \le o_i, \quad i = 1, \ldots, n \tag{36.8}$$

The constraint set (36.8) reflects that negative service times do not have a physical interpretation and executing an optional part $O_i$ beyond its worst-execution time $o_i$ does not increase the reward. The constraint (36.7) encodes the fact that the sum of optional and mandatory service times cannot exceed the total available time $P$ (the length of the hyperperiod).

The work in [2] presents a polynomial-time algorithm to solve the problem above. Observe that if the available time can accommodote all the optional parts in their entirety, that is, if $\sum_{i=1}^{n} \frac{P}{P_i}(m_i + o_i) \le P$, then setting $t_i = o_i \; \forall i$ clearly maximizes the objective function. Otherwise, because of the nondecreasing nature of reward functions, we should fully use the timeline and consider the constraint (36.7) as an equality constraint. Thus, the algorithm in [2] temporarily ignores the inequality constraints (36.8) and first produces a solution that only satisfies the equality constraint through *Lagrange multipliers* technique. In case that the constraint set (36.8) is violated, then the solution is iteratively improved using the *Kuhn-Tucker optimality conditions* [28] for nonlinear optimization problems. In [2], it is formally proven that this iterative algorithm is guaranteed to converge to the optimal solution in time $O(n^2 \log n)$.

Though the solution above addresses the most common task and reward function models, it is worth investigating the possible extensions to different models. Clearly, Theorem 36.1 is the key, eliminating a potentially exponential number of unknowns (the $t_{ij}$ values) and, arguably, it is very hard to solve the problem without such a property. In fact, [2] also established that the result on the optimality of identical execution times no longer holds if we assume that

1. the relative deadline of a task instance can be smaller than its period, or
2. a static-priority assignment (such as Rate-Monotonic Priority Assignment) is to be used as opposed to EDF/LLF, or
3. the concavity assumption about the reward functions is relaxed.

The solution to the periodic RB scheduling problem with any of the above assumptions is still open as of this writing. Further, it was formally proven in [2] that the concavity assumption is absolutely necessary for computational tractability, *even for tasks sharing a common period and deadline*:

**Theorem 36.2 (from [2])**

*The RB scheduling problem with* convex *reward functions is NP-hard.*

## 36.5 Power-Aware Scheduling

The RB scheduling framework is used when attempting to assign CPU cycles to various applications to maximize the reward (user-perceived utility). Many of these applications do also run on embedded, mobile, and wireless computing devices that rely on *battery power*. Thus, power management has been recently elevated to a major research area in computer science and engineering. Different components of computer systems, such as CPU, memory, disk subsystems, and network interfaces have been subject to various energy-efficiency studies. The reader is referred to [29,30] for comprehensive surveys.

One powerful and increasingly popular technique, known as Variable Voltage Scheduling (*VVS*) or Dynamic Voltage Scaling (*DVS*), is based on reducing CPU energy consumption through the adjustment of the CPU speed by varying both the supply voltage and the clock frequency. The technique is motivated by the observation that the dynamic power dissipation $P_d$ of an on-chip system is given by the following formula:

$$P_d = C_f \cdot V_s^2 \cdot f$$

where $C_f$ is the effective switched capacitance, $V_s$ is the supply voltage, and $f$ is the frequency of the clock. Hence, in principle, it is possible to obtain significant power savings by simultaneously reducing the supply voltage and the clock frequency. On the other hand, clearly, the response time will increase in a linear fashion when we reduce the clock frequency. Thus, trading the speed for energy savings is the main idea in Power-Aware Scheduling. The exact form of power/speed relation depends on the specific technology in use, but as a rule, it is a strictly increasing convex function, specifically a polynomial of at least the second degree. The prospects of reducing the energy consumption through DVS have been materialized in recently-announced processors such as Transmeta's Crusoe processor and Intel Xscale architecture.

Note that in its simplest form, the power/speed relation can be exploited by using the maximum CPU speed when executing tasks and shutting down the CPU whenever the system is (likely to stay) idle for an acceptably long period. However, this *predictive shutdown* technique remains inefficient and suboptimal even for a scheduler with perfect knowledge of idle intervals, due to the convex dependence between the speed and the power consumption: a better approach is to *reduce the CPU speed* while executing tasks.

Early works on DVS focused on task systems with no explicit deadlines. Weiser et al. adopted an approach where time is divided into 10–50 msec intervals, and the CPU clock speed (and the supply voltage) is adjusted using predictions based on processor utilization in recent intervals [31]. Govil, Chan, and Wasserman proposed and evaluated several predictive and nonpredictive approaches for voltage changes [32].

In real-time systems, where tasks have to meet their timing constraints, one cannot arbitrarily reduce the CPU speed. Thus, the real-time power-aware scheduling (RT-PAS) problem[2] can be stated as follows: **Determine the CPU speed to minimize the total energy consumption while still meeting all the deadlines.** Observe that, the solution to the RT-PAS problem depends on the task model under consideration

[2]Alternative names that appeared in the literature for the same framework include *real-time dynamic voltage scaling* (RT-DVS) and *real-time variable voltage scheduling* (RT-VVS).

(aperiodic/periodic/hybrid task sets or preemptive/nonpreemptive scheduling). Further, clearly, the optimal CPU speed in the solution may be *a function of the time and/or identity of the running task*.

RT-PAS research can be traced back to the important work of Yao, Demers, and Shenker [33], where the authors provided a polynomial-time algorithm to compute the optimal speed assignments assuming *aperiodic tasks* and *worst-case execution times*. Other works in this area include heuristics for extended task and system models, such as on-line scheduling of aperiodic and periodic tasks [34], nonpreemptive RT-PAS [35], and RT-PAS with upper bound on the voltage change rates [36].

### 36.5.1 Modeling Real-Time Workload and Energy on a Variable Speed CPU

Traditional real-time scheduling theory mandates the consideration of the worst-case workload, which translates to *worst-case execution times* under a *fixed* CPU speed. On a variable speed CPU, it is clear that the execution time of a task is dependent on both the number of CPU cycles required, and the CPU speed. Consequently, the *worst-case number of CPU cycles* is adopted as the measure of the worst-case workload.

On a variable speed CPU, the processor speed $S$ (expressed in the number of CPU cycles per time unit) can be changed between a lower bound $S_{\min}$ and an upper bound $S_{\max}$. We will assume that the speed can be varied continuously over the range $[S_{\min}, S_{\max}]$ and that the supply voltage is also adjusted in accordance with the speed. We will normalize the speed values with respect to $S_{\max}$; that is, at any time, $0 \leq S_{\min} \leq S \leq S_{\max} = 1$.

We will denote by $C_i$ the number of CPU cycles required by the task $T_i$. If a task executes with constant speed $S$, then its worst-case execution time is given by $C_i/S$. Observe that, since $S_{\max} = 1.0$, the worst-case execution time of the task $T_i$ under the maximum speed is $C_i$ time units.

In current processor arcitectures implemented in CMOS technology, the CPU power consumption of task $T_i$ executing at the speed $S$ is given by a strictly increasing and convex function $g_i(S)$. In general, $g_i(S)$ can be represented by a polynomial of the second or the third degree [33,36]. If the task $T_i$ occupies the processor during the interval $[t_1, t_2]$ and if the processor speed changes according to a function $S(t)$, then the *energy consumed* in this interval is $E(t_1, t_2) = \int_{t_1}^{t_2} g_i(S(t))dt$. Note that the last expression simply becomes $E(t_1, t_2) = g_i(S) \cdot (t_2 - t_1)$ if the speed $S$ is constant during the interval.

### 36.5.2 Correlating Real-Time Power-Aware Scheduling to Reward-Based Scheduling

Before addressing the specific case of *periodic* tasks, in this section, we correlate the *general* RB scheduling problem given by (36.2), (36.3), and (36.4) in Section 36.2 to *general* RT-PAS problem. We underline that the relation we prove is preserved regardless of the specific task model (preemptive/nonpreemptive scheduling, independent/dependent task sets) *or* the number of the processors as long as we make the same assumptions for both RB Scheduling and RT-PAS problems. First, we state the general RT-PAS Problem.

#### 36.5.2.1 Real-Time Power-Aware Scheduling Problem

Consider a CPU with variable voltage/speed $S(S_{\min} \leq S \leq S_{\max})$ facility, and a set $\mathbf{T} = \{T_1, \ldots, T_n\}$ of hard real-time tasks, in which each task $T_i$ is subject to a worst-case workload of $C_i$ expressed in the number of required CPU cycles. The power consumption of the task $T_i$ is given by a strictly increasing convex function $g_i(S)$, which is a polynomial of at least the second degree. Given a time point $Z$, determine *the energy-optimal schedule* ($\mathcal{EOS}$) and *the processor speed* $S(t)$ in the interval $[0, Z]$, such that $\mathcal{EOS}$ is feasible and valid, and the total energy consumption $E(0, Z) = \int_0^Z g_i(S(t))dt$ is minimized.

A major difficulty with the *real-time power-aware scheduling* (RT-PAS) problem lies in the *possibility* of having a *nonconstant* speed in the optimal solution: determining the exact form of $S(t)$ for every point in the interval $[0, Z]$ may be a serious challenge. Fortunately, the *convexity* assumption about $g_i(S)$ makes possible to prove the following helpful property (see [37] for a formal proof).

**Proposition 36.1**

*A task $T_i$ requiring $C_i$ CPU cycles can be executed with a* constant speed $S_i$, *without increasing the energy consumption or the completion time.*

That is, we can assume that the CPU speed will not change while executing a given task (task instance for the periodic task model), without compromising the optimality or feasibility. However, note that the optimal speed *may* be different for different tasks. We can now formulate the general RT-PAS as an optimization problem where the aim is to determine the speeds $S_1, \ldots, S_n$ so as to

$$\text{Minimize} \quad \sum_{i=1}^{n} \frac{C_i}{S_i} \cdot g_i(S_i) \tag{36.9}$$

$$\text{subject to} \quad S_{\min} \leq S_i \leq S_{\max}, \quad i = 1, \ldots, n \tag{36.10}$$

$$\text{There exists a feasible and valid schedule with } \{S_i\} \text{ values} \tag{36.11}$$

The expression $\frac{C_i}{S_i} \cdot g_i(S_i)$ in the objective function (36.9) quantifies the energy consumption of task $T_i$ when run at the speed $S_i$. The constraint set (36.10) encodes the lower and upper bounds on the speed, whereas the constraint (36.11) enforces the feasibility and validity.

**Proposition 36.2**

*Solving an instance of RT-PAS problem is equivalent to solving an instance of RB Scheduling problem with concave reward functions.*

***Proof***

Observe that determining $S_i$ is effectively equivalent to determining *the CPU time allocation of* $T_i$, which will be denoted by $X_i$ ($X_i = C_i/S_i$). In addition, the minimum and maximum speed bounds impose explicit bounds on $X_i$. Specifically, for any task $T_i$, the CPU allocation $X_i$ should lie in the interval $[X_{\min}, X_{\max}]$, where $X_{\min} = \frac{C_i}{S_{\max}}$ and $X_{\max} = \frac{C_i}{S_{\min}}$. Thus, we can rewrite RT-PAS as to determine the CPU time allocations $\{X_i\}$ so as to

$$\text{Minimize} \quad \sum_{i=1}^{n} X_i \cdot g_i\left(\frac{C_i}{X_i}\right) \tag{36.12}$$

$$\text{subject to} \quad \frac{C_i}{S_{\max}} \leq X_i \leq \frac{C_i}{S_{\min}} \quad i = 1, \ldots, n \tag{36.13}$$

$$\text{There exists a feasible and valid schedule with } \{X_i\} \text{ values} \tag{36.14}$$

That is, $T_i$ must receive service for at least $X_{\min}$ units, which can be considered as a *mandatory* execution. Depending on the specific speed assignment $S_i$, $T_i$ can receive an additional service of up to $X_{\max} - X_{\min}$, which can be seen as an *optional* execution. Moreover, the lower the speed $S_i$, the longer the "optional" execution beyond $X_{\min}$ and the lower the energy consumption (see Figure 36.4). In addition, minimizing the energy consumption is equivalent to *maximizing energy savings*. Thus, consider the following variable substitutions:

$$m_i = \frac{C_i}{S_{\max}}$$

$$t_i = X_i - m_i$$

$$o_i = \frac{C_i}{S_{\min}} - \frac{C_i}{S_{\max}}$$

$$R_i(t_i) = -\left(t_i + \frac{C_i}{S_{\max}}\right) g_i\left(\frac{C_i}{t_i + \frac{C_i}{S_{\max}}}\right)$$

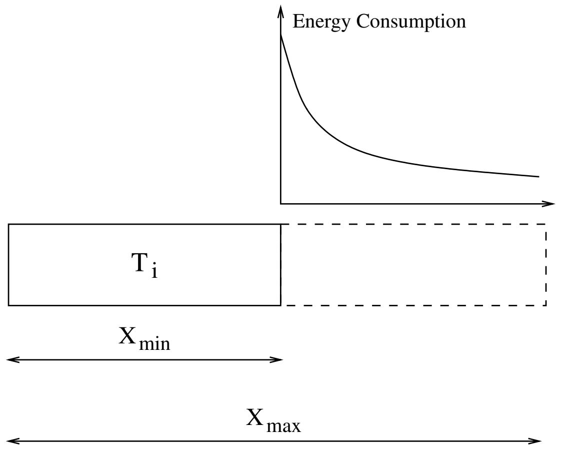

**FIGURE 36.4** Energy consumption as a function of CPU allocation.

Using this transformation, we obtain a new form for the RT-PAS problem:

$$\text{Maximize} \quad \sum_{i=1}^{n} R_i(t_i) \tag{36.15}$$

$$\text{subject to} \quad 0 \le t_i \le o_i \quad i = 1, \dots, n \tag{36.16}$$

$$\text{There exists a feasible and valid schedule with } \{m_i\} \text{ and } \{t_i\} \text{ values} \tag{36.17}$$

The final formulation is an instance of the general RB scheduling problem defined in Section 36.2 by Equations (36.2), (36.3), and (36.4). Further, the reward function $R_i(t_i)$ above is concave, since

$$\left(t_i + \frac{C_i}{S_{\max}}\right) g_i \left(\frac{C_i}{t_i + \frac{C_i}{S_{\max}}}\right)$$

is convex. To see this, we can use the result from [28] stating that if $a$ and $b$ are both convex functions and if $a$ is increasing, then $a(b(x))$ is also convex. Thus, by setting

$$h(t_i) = \frac{C_i}{t_i + \frac{C_i}{S_{\max}}}$$

and observing that the multiplication by $(t_i + \frac{C_i}{S_{\max}})$ does not affect the convexity, we justify the concavity of $R_i(t_i) = -g_i(h(t_i))$.

## 36.5.3 Determining Optimal Speed Assignments for Periodic Real-Time Tasks

In this section, we focus on the most common type of real-time task models, namely, *periodic task sets*. Thus, we consider a set $\mathcal{T} = \{T_1, \dots, T_n\}$ of $n$ periodic real-time tasks. As in Section 36.4, the period of $T_i$ is denoted by $P_i$, which is also equal to the deadline of the current invocation. All tasks are assumed to be independent and ready at $t = 0$. The worst-case number of CPU cycles required by each instance[3] of

[3]Note that we restrict our power-aware scheduling discussion to hard real-time tasks that must be executed in their entirety. That is, we do not address the case where each task can be decomposed to a mandatory and optional part. The reader is referred to a recent study [38] for the power-aware scheduling of RB tasks.

$T_i$ is denoted by $C_i$. We will assume that tasks can be preempted during the execution and the overhead of changing voltage/speed is negligible.

We define $U$ as the total utilization of the task set under maximum speed $S_{\max} = 1$, that is, $U = \sum_{i=1}^{n} \frac{C_i}{P_i}$. Note that the schedulability theorems for periodic real-time tasks [25] imply that $U \leq 1$ is a necessary condition to have at least one feasible schedule.

Thus, *periodic RT-PAS problem* can be stated as: *Given a set of periodic tasks, determine the optimal speed assignments of all the task instances to minimize total energy consumption while still meeting all the deadlines.*

Note that the optimal solution needs to consider the possibility of assigning different speeds to different instances of a given task. Fortunately, using Proposition 36.2 that established the equivalence between RB (but constant-speed) scheduling problem and RT-PAS problem, and Theorem 36.1, we can immediately assert that *in the optimal solution, all the instances of a given task will have the same CPU allocation, thus the same speed*. Moreover, the polynomial-time solution of the periodic RB scheduling problem [2] can be adopted to compute the optimal speed assignments (refer to [39] for details). Finally, thanks to Theorem 36.1, we can state that EDF and LLF policies may be used to obtain a feasible schedule with these speed assignments.

Moreover, if the power consumption functions of all the tasks can be assumed to be identical (a realistic assumption in many systems), that is, if $g_i(S) = g_j(S) = g(S)\, \forall i,\, j$, a stronger result can be proven [40]:

## Proposition 36.3

*If task power consumption functions are identical, then the optimal CPU speed to minimize the total energy consumption while meeting all the deadlines of the periodic hard real-time task set is constant and equal to* $S_{\text{opt}} = \max\{S_{\min}, U\}$, *where* $U = \sum_{i=1}^{n} \frac{C_i}{P_i}$.

In other words, the optimal speed with identical power consumption functions is equal to the utilization of the task set under maximum speed (namely, $U$), subject to the $S_{\min}$ constraint. Observe that this speed choice results in an *effective* utilization of $\sum_{i=1}^{n} \frac{C_i}{S_{\text{opt}} \cdot P_i} = 1.0$ (a fully utilized timeline), assuming that the lower bound on the CPU speed is not violated. Thus, the optimal solution consists in "stretching out" the CPU allocations of all task instances in equal proportions.

Note that the solution presented above is *static* in the sense that it assumes a worst-case workload for each periodic task instance. This is necessary because any solution for hard real-time systems must provision for the worst-case scenarios. However, in practice, many real-time applications complete before presenting their worst-case workload [41]. Thus, at run-time, the static optimal CPU speed can be further reduced when early completions do occur. A class of *dynamic* scheduling techniques to adjust the speed to the *actual* workload were investigated in depth in recent years (see [40,42,43] as representative studies). A crucial consideration in dynamic power-aware scheduling is to avoid deadline misses when reducing the speed and reclaiming unused computation times (*slack*). Thus, the proposed dynamic techniques differ in their ability to exploit slack while preserving the feasibility.

If the actual workload is likely to differ from the worst-case, then additional energy gains can be achieved through a *speculative* strategy. In this approach, the CPU speed is *aggressively* reduced in *anticipation* of early completions. Clearly, such a speculative strategy risks violating the timing constraints if the actual workload of tasks turns out to be higher than predicted. Consequently, the scheduler must guarantee that all the deadlines will be met by adopting a *high* speed in the later parts of the schedule, should there be a need. The idea of speculative speed reduction was first introduced in [44] for tasks sharing a common period. The works in [40,42,45] provide representative research on speculative scheduling policies for general periodic real-time tasks. Finally, the study in [46] presents a detailed performance comparison of state-of-the-art dynamic and speculative dynamic voltage scaling techniques.

### 36.5.4 Practical Considerations

In this section, we look at two issues that must be carefully considered in any practical real-time system that uses Dynamic Voltage Scaling. The first one is the *delay overhead* involved in performing the speed and supply voltage reductions. This overhead may be particularly significant especially for the *dynamic* scheduling schemes that change the CPU speed at run-time upon detecting early completions.

While Namgoong, Yu, and Meg reported the time taken to reach steady state at the new voltage level as under 6 ms/V as of 1996 [47], Burd et al. indicate that this delay is limited by 70 $\mu$s in the new ARM V4 processor based system, including frequency transition [48]. For the Low-Power lpARM processor, clock frequency transitions take approximately 25 $\mu$s [49]. In a study based on StrongARM SA-1100 processor operating under constant supply voltage, this delay is reported to be less than 150 $\mu$s [50]. When modifying the supply voltage and the clock frequency of the StrongARM SA-1100 processor, Pouwelse, Langendoen, and Sips have found that the voltage and speed increase is rapidly handled (40 $\mu$s), but the decrease can take up to 5.5 ms [51]. A full voltage transition can be performed in less than 300 $\mu$s in the Transmeta Crusoe 5400 processor [52]. Thus, the technology trends are toward even more efficient DVS-enabled processors and it can be claimed that the overhead involved can be ignored in many cases, especially when the speed changes occur only at context switch time. In case that this overhead is not negligible, it can be incorporated into the worst-case workload of each task [53].

The second issue is related to the fact that, unlike the assumptions of this section's framework where the CPU speed can be changed continuously in the interval $[S_{\min}, S_{\max}]$, current technologies support only a finite number of speed levels [52], though the number of available speed levels is likely to increase in the future. The solutions obtained for the continuous speed model can be always adopted to these settings by choosing the lowest speed level that is equal to or greater than the value suggested by the algorithms. Our preliminary experimental results indicate that this simple approach results in an energy overhead of 15–17% with respect to continuous speed settings, when the number of available speed levels is only 5. When the number of speed levels exceeds 30, the difference reduces to 3%. More comprehensive recent studies that address this issue and other overhead sources in dynamic voltage scaling can be found in [43,54].

## 36.6 Conclusion

In this chapter, we introduced the RB and the RT-PA Scheduling frameworks, and we established the relationship between these two models. RB scheduling is a general framework which unifies hard and soft real-time scheduling theories by addressing feasibility and maximum utility issues simultaneously. Assuring timely completion of mandatory parts guarantees an acceptable overall performance, while optimal scheduling of optional parts maximizes the total user-perceived utility (reward). RB scheduling promotes graceful degradation and resource utilization where worst-case guarantees cannot be given due to faults, overload conditions and for applications that admit an intratask mandatory/optional division. However, simultaneously achieving feasibility and optimality is not a trivial problem, since the reward functions associated with the optional parts bring another dimension to the traditional scheduling framework.

After introducing various research works and models that can be classified as part of the RT-RB Scheduling, we focused on RB scheduling for periodic real-time tasks. According to a main result due to [2], when the reward functions are linear and/or concave (the most common case), it is always possible to find an optimal schedule where the service time of a given task remains constant from instance to instance. This, in turn, implies that well-known scheduling policies such as EDF and LLF can be used to meet all the timing constraints with these optimal assignments. Moreover, it is possible to compute these optimal service times in polynomial time by solving iteratively a non-linear optimization problem. Finally, we briefly commented on the existing difficulties when one attempts to extend these results to different models.

RT-PAS has recently become a major research area with ever increasing emphasis on energy saving strategies for portable and embedded devices that rely on battery power. Thanks to Dynamic Voltage

Scaling technology, it is possible to obtain considerable energy savings by simultaneously reducing the CPU speed and the supply voltage. However, this is accompanied by a linear increase in response time, which may compromise the correctness of hard real-time systems. Thus, RT-PAS solutions focus on minimizing the total energy consumption (through speed reduction) while still meeting all the deadlines.

We have shown that the RT-PAS problem can be solved within the framework of RB scheduling, through an appropriate transformation. For the most common, periodic real-time task model, it turns out that the optimal speed to meet all the deadlines while minimizing the energy consumption is constant and equal to the utilization (load) of the task set under maximum speed, if the power consumption of CPU is independent of the running task.

## References

[1] G. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications.* Kluwer Academic Publishers, Boston, 1997.

[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): 111–130, February 2001.

[3] J.-Y. Chung, J. W.-S. Liu, and K.-J. Lin. Scheduling periodic jobs that allow imprecise results. *IEEE Transactions on Computers,* 19(9): 1156–1173, September 1990.

[4] J. K. Dey, J. Kurose, D. Towsley, C.M. Krishna, and M. Girkar. Efficient on-line processor scheduling for a class of IRIS (increasing reward with increasing service) real-time tasks. *Proceedings of ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems,* May 1993.

[5] J. K. Dey, J. Kurose, and D. Towsley. On-line scheduling policies for a class of IRIS (increasing reward with increasing service) real-time tasks. *IEEE Transactions on Computers,* 45(7):802–813, July 1996.

[6] J. W.-S. Liu, K.-J. Lin, W.-K. Shih, A. C.-S. Yu, C. Chung, J. Yao, and W. Zhao. Algorithms for scheduling imprecise computations. *IEEE Computer,* 24(5): 58–68, May 1991.

[7] R. Rajkumar, C. Lee, J. P. Lehoczky, and D. P. Siewiorek. A resource allocation model for QoS management. In *Proceedings of 18th IEEE Real-Time Systems Symposium,* December 1997.

[8] W.-K. Shih, J. W.-S. Liu, and J.-Y. Chung. Algorithms for scheduling imprecise computations to minimize total error. *SIAM Journal on Computing,* 20(3), July 1991.

[9] K.-J. Lin, S. Natarajan, and J. W.-S. Liu. Imprecise results: utilizing partial computations in real-time systems. In *Proceedings of 8th IEEE Real-Time Systems Symposium,* December 1987.

[10] J. W.-S. Liu, K.-J. Lin, and S. Natarajan. Scheduling real-time, periodic jobs using imprecise results. In *Proceedings of 8th IEEE Real-Time Systems Symposium,* December 1987.

[11] W.-K. Shih and J. W.-S. Liu. On-line scheduling of imprecise computations to minimize error. *SIAM Journal on Computing,* October 1996.

[12] C. Lee, R. Rajkumar, J. P. Lehoczky, and D. P. Siewiorek. Practical solutions for QoS-based resource allocation problems. In *Proceedings of 19th IEEE Real-Time Systems Symposium,* December 1998.

[13] K.I.J. Ho, J.Y.T. Leung, and W.D. Wei. Scheduling imprecise computations with 0/1 constraint. *Discrete Applied Math.,* 78, 117–132, 1997.

[14] W.-K. Shih, J. W.-S. Liu, J.-Y. Chung, and D.W. Gillies. Scheduling tasks with ready times and deadlines to minimize average error. *ACM Operating Systems Review,* July 1989.

[15] J. Y. T. Leung, V. K. M. Yu, and W. D. Wei. Minimizing the weighted number of tardy task units. *Discrete Applied Math.,* 51, 307–316, 1994.

[16] W. Feng and J.W.-S. Liu. Algorithms for scheduling real-time tasks with input error and end-to-end deadlines. *IEEE Transactions on Software Engineering,* 23(2): 93–106, February 1997.

[17] J.-Y. Chung and J. W.-S. Liu. Algorithms for scheduling periodic jobs to minimize average error. In *Proceedings of 9th IEEE Real-Time Systems Symposium,* December 1988.

[18] S. K. Baruah and M. E. Hickey. Competitive on-line scheduling of imprecise computations. *IEEE Transactions on Computers,* 47(7): 1027–1033, September 1998.

[19] W.-K. Shih and J. W.-S. Liu. Algorithms for scheduling imprecise computations with timing constraints to minimize maximum error. *IEEE Transactions on Computers,* 44(3):466–471, March 1995.

[20] K. I. J. Ho, J. Y. T. Leung, and W. D. Wei. Minimizing maximum weighted error for imprecise computation tasks. *Journal of Algorithms,* 16, 431–452, 1994.

[21] J. Y. T. Leung and C. S. Wong. Minimizing the number of late tasks with error constraint. *Information and Computation,* 106, 83–108, 1993.

[22] J. Y. T. Leung, T. W. Tam, C. S. Wong, and G. H. Young. Minimizing mean flow time with error constraint. *Algorithmica,* 20, 101–118, 1998.

[23] C. Lee, R. Rajkumar, J. P. Lehoczky, and D. P. Siewiorek. On quality of service optimization with discrete QoS options. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium,* June 1998.

[24] C. Lee, J. P. Lehoczky, D. P. Siewiorek, R. Rajkumar, and J. Hansen. A scalable solution to the multi-resource QoS problem. In *Proceedings of 20th IEEE Real-Time Systems Symposium,* December 1999.

[25] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in hard real-time environment. *Journal of ACM* 20(1), 1973.

[26] A. K. Mok. *Fundamental Design Problems of Distributed systems for the Hard Real-Time Environment.* Ph.D. Dissertation, MIT, 1983.

[27] A. Bertossi and L. Mancini. Scheduling algorithms for fault-tolerance in hard real-time systems. *Real Time Systems Journal,* 7, 1994.

[28] D. Luenberger. *Linear and Nonlinear Programming.* Addison-Wesley, Reading Massachusetts, 1984.

[29] R. GrayBill and R. Melhem (editors). *Power Aware Computing,* Series in Computer Science. Kluwer Academic/Plenum Publishers, May 2002.

[30] P. J. M. Havinga and G. J. M. Smith. Design techniques for low-power systems. *Journal of Systems Architecture,* 46(1): 1–21, January 2000.

[31] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *USENIX Symposium on Operating Systems Design and Implementation,* 13–23, 1994.

[32] K. Govil, E. Chan, and H. Wasserman. Comparing algorithms for dynamic speed-setting of a low-power CPU. In *ACM International Conference on Mobile Computing and Networking,* 13–25, 1995.

[33] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. *IEEE Annual Foundations of Computer Science,* 374–382, 1995.

[34] I. Hong, M. Potkonjak, and M. B. Srivastava. On-line scheduling of hard real-time tasks on variable voltage processor. In *Computer-Aided Design, ICCAD'98,* 653–656, 1998.

[35] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. Srivastava. Power optimization of variable voltage core-based systems. In *Design Automation Conference,* 1998.

[36] I. Hong, G. Qu, M. Potkonjak, and M. Srivastava. Synthesis techniques for low-power hard real-time systems on variable voltage processors. In *Proceedings of 19th IEEE Real-Time Systems Symposium, RTSS'98,* Madrid, December 1998.

[37] H. Aydin. *Enhancing Performance and Fault Tolerance in Reward-Based Scheduling.* Ph.D. Dissertation, University of Pittsburgh, August 2001.

[38] C. Rusu, R. Melhem, and D. Mossé. Maximizing rewards for real-time applications with energy constraints. *ACM Transactions on Embedded Computing Systems.* (Unpublished)

[39] H. Aydin, R. Melhem, D. Mossé, and P. M. Alvarez. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *Proceedings of the 13th EuroMicro Conference on Real-Time Systems, ECRTS'01,* June 2001.

[40] H. Aydin, R. Melhem, D. Mossé, and P. M. Alvarez. Dynamic and aggressive power-aware scheduling techniques for real-time systems. In *Proceedings of the 22nd IEEE Real-time Systems Symposium, RTSS'01,* December 2001.

[41] R. Ernst and W. Ye. Embedded program timing analysis based on path clustering and architecture classification. In *Computer-Aided Design, ICCAD'97,* 598–604, 1997.

[42] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low power embedded operating systems. In *Proceedings of the 18th Symposium on Operating System Principles,* October 2001.

[43] S. Saewong and R. Rajkumar. Practical voltage-scaling for fixed-priority real-time systems. In *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS'03,* Washington D.C., May 2003.

[44] D. Mossé, H. Aydin, B. Childers, and R. Melhem. Compiler-assisted dynamic power-aware scheduling for real-time applications. *Workshop on Compilers and Operating Systems for Low-Power, COLP'00,* Philadelphia, PA, October 2000.

[45] F. Gruian. Hard real-time scheduling for low-energy using stochastic data and DVS processors. In *International Symposium on Low Power Electronics and Design 2001,* August 6–7, 2001.

[46] W. Kim, D. Shin, H. S. Yun, J. Kim, and S. L. Min. Performance comparison of dynamic voltage scaling algorithms for hard real-time systems. In *Proceedings of the 8th Real-time Technology and Applications Symposium,* San Jose, 2002.

[47] W. Namgoong, M. Yu, and T. Meg. A high efficiency variable-voltage CMOS dynamic DC–DC switching regulator. *IEEE International Solid-State Circuits Conference,* 380–391, 1996.

[48] T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen. A dynamic voltage scaled microprocessor system. *IEEE Journal of Solid-State Circuits,* 35(11): 1571–1580, November 2000.

[49] T. Pering, T. Burd and R. Brodersen. Voltage scaling in the IpARM microprocessor system. In *Proceedings of the 2000 International Symposium on Low power Electronics and design*.

[50] R. Min, T. Furrer and A. Chandrakasan. Dynamic voltage scaling techniques for distributed microsensor networks. In *Proceedings of the IEEE Computer Society Annual Workshop on VLSI, WVLSI '00,* April 2000.

[51] J. Pouwelse, K. Langendoen, and H. Sips. Dynamic voltage scaling on a low-power microprocessor. In *Proceedings of Mobile Computing Conference, MOBICOM,* 2001.

[52] http://www.transmeta.com

[53] N. AbouGhazaleh, D. Mossé, B. Childers, and R. Melhem. Toward the placement of power management points in real time applications. In *Workshop on Compilers and Operating Systems for Low Power, COLP'01,* Barcelona, Spain, 2001.

[54] B. Mochocki, X. Hu, and G. Quan. A realistic variable voltage scheduling model for real-time applications. In *Proceedings of ICCAD 2002*.

[55] H. Chetto, M. Silly, and T. Bouchentouf. Dynamic scheduling of real-time tasks under precedence constraints. *Journal of Real-Time Systems,* 1990.

[56] V. Gutnik and A. Chandrakasan. An efficient controller for variable supply voltage low power processing. *Symposium on VLSI Circuits,* 158–159, 1996.

[57] C. M. Krishna and Y. H. Lee. Voltage clock scaling adaptive scheduling techniques for low power in hard real-time systems. In *Proceedings of the 6th IEEE Real-Time Technology and Applications Symposium (RTAS'00),* Washington D.C., May 2000.

[58] L. D. Nguyen and A. M. K. Cheng. An imprecise real-time image magnification algorithm. In *International Conference on Multimedia Systems,* 1996.