

Utility Accrual Real-Time Scheduling with Energy Bounds

Abstract

In this paper, we consider timeliness and energy optimization in battery-powered, dynamic embedded real-time systems, which must remain functional during an operation/mission with a bounded energy budget. We consider application activities that are subject to time/utility function time constraints, statistical assurance requirements on timeliness behavior, and an energy budget, which cannot be exceeded at run-time. To account for the inevitable variability in activity arrivals in dynamic systems, we describe arrival behaviors using the unimodal arbitrary arrival model. For such a model, we present a CPU scheduling algorithm, called the Energy-Bounded Utility Accrual Algorithm (or EBUA). EBUA is a polynomial-time algorithm that satisfies energy bounds, and provides statistical assurances on individual activity timeliness behavior. We analytically establish several timeliness properties of EBUA. Further, our simulation experiments confirm the algorithm's effectiveness and superiority.

1 Introduction

With the proliferation of mobile and embedded devices that operate on limited battery power, power and energy management of embedded systems have become critically important. Many such devices have finite energy bounds, embodied by a battery that has a finite lifetime. An important technique used for optimizing the energy consumption of real-time embedded systems is dynamic voltage scaling (DVS). With DVS, an appropriate clock rate and voltage can be determined in response to dynamic application behaviors. This can result in quadratic energy savings at the expense of roughly, linearly increased application activity sojourn times (see [3, 14, 19, 25] and the references therein).

In this paper, we focus on dynamic embedded real-time control systems in domains including

robotics, space, defense, and consumer electronics. A specific example from the robotics/space domain is NASA Jet Propulsion Laboratory’s robotic systems (e.g., Mars Rover), which are envisioned for long-lived, scientific exploration missions on the Mars planet [8]. Such systems are fundamentally time-critical, as they must sense external objects in a timely manner and produce timely control responses (e.g., to avoid obstacles in the physical world). Further, they are energy-critical, as they must operate on battery, with finite energy budgets and limited battery recharging time. Prolonging their battery life often requires bounding and minimizing the *system’s* energy consumption, and not just the CPU’s energy consumption.

Moreover, such systems operate in environments with dynamically uncertain properties. These uncertainties include transient and sustained overloads on the CPU (due to context dependent execution times) and arbitrary arrival patterns for application activities. Nevertheless, such systems desire assurances on activity timeliness behaviors and energy consumption behaviors. Consequently, their non-deterministic operating situations must be characterized with stochastic or extensional (rule-based) models.

The most distinguishing property of such systems, however, is that they are subject to time constraints that are “soft” (besides those that are hard) in the sense that completing an activity at any time will result in some (positive or negative) utility to the system, and that utility depends on the activity’s completion time. These soft time constraints are subject to scheduling optimality criteria such as completing all time-constrained activities as close as possible to their optimal completion times—so as to yield maximal collective utility.

Jensen’s time/utility functions [12] (or TUFs) allow the semantics of soft time constraints to be precisely specified. A TUF, which is a generalization of the deadline constraint, specifies the utility to the system resulting from the completion of an activity as a function of that activity’s completion time. A TUF’s utility values are derived from application-level quality of service metrics. Figure 1 shows example time constraints from real applications specified using TUFs. Figures 1(a)–1(c) show some time constraints of two applications in the defense domain [6,18]. The classical deadline is a binary-valued, downward “step” shaped TUF; 1(d) shows examples.

When activity time constraints are expressed with TUFs, the scheduling optimality criteria are typically based on accrued activity utility—e.g., maximizing sum of the activities’ attained utilities or assuring satisfaction of lower bounds on activities’ maximal utilities. Such criteria are called *utility accrual* (or UA) criteria, and sequencing (scheduling, dispatching) algorithms that consider UA criteria are called UA sequencing algorithms.

UA criteria directly facilitate adaptive behaviors during resource overloads, when (optimally

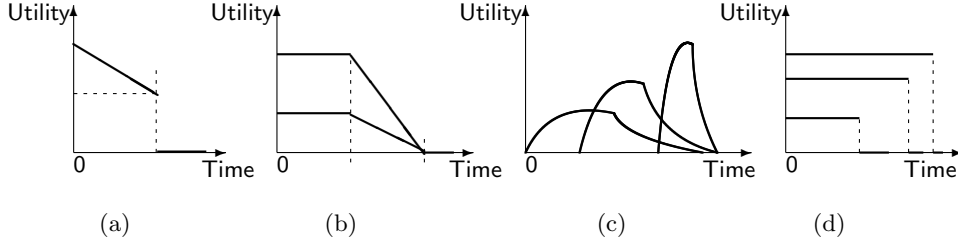


Figure 1: Example TUF Time Constraints. (a): AWACS *association* TUF [6]; (b-c): Air defense *plot correlation*, *track maintenance*, & *missile control* TUFs [18]; (d): step TUFs.

or sub-optimally) completing more important activities, irrespective of activity urgency, is often desirable. For example, UA algorithms that maximize summed utility under downward step TUFs (or deadlines) meet all activity deadlines during under-loads [7, 16]. When overloads occur, they favor activities that are more important (since more utility can be attained from them), irrespective of their urgency. Thus, UA algorithms’ timeliness behaviors subsume the optimal timeliness behavior of deadline scheduling [9].

1.1 Contribution, Paper Outline

In this paper, we focus on dynamic real-time embedded systems, which are subject to a finite energy budget for the entire duration of their operational/mission life. Example motivations for such a constraint include non-availability of battery recharging time and/or energy source. Consequently, they must operate without violating their energy budgets. Thus, the energy budget is a *hard* constraint.

If the system’s energy budget does not (transiently or permanently) allow the execution of all jobs—e.g., due to (transient or permanent) overloads, some jobs should be deferred or rejected in a controlled fashion so as to enable maximal utility to be accrued by the mission, without adversely affecting the system’s functionality during the mission. Further, assurances on activity timeliness behavior must be provided, whenever possible. Furthermore, the system’s energy consumption must be minimized and timeliness performance must be maximized, as much as possible, at all times.

We solve this exact problem in this paper. We consider repeatedly occurring real-time application activities whose time constraints are specified using TUFs. To better account for uncertainties in activity behaviors, we stochastically describe activity execution demands, and describe activity arrivals using the unimodal arbitrary arrival model (or UAM) [22]. We consider Martin’s system-level energy consumption model [17], where each system component’s

energy consumption is individually modeled and aggregated to account for system’s energy consumption. For such an activity model, our objective is to: (1) provide statistical assurances on timeliness behavior including probabilistically-satisfied lower bounds on individual activity utility; and (2) maximize system-level energy efficiency; while ensuring that the system’s energy consumption never exceeds the specified energy budget.

This problem is \mathcal{NP} -hard [2]. We present a polynomial-time, heuristic algorithm for the problem called *Energy-Bounded Utility Accrual Algorithm* (or EBUA). We prove that EBUA never violates the energy budget. Further, we establish that EBUA’s timeliness behavior subsumes the optimal timeliness behavior of EDF [9] as a special case, and identify the conditions under which EBUA provides assurances on individual activity timeliness behavior. Finally, our simulation studies confirm the effectiveness and superior performance of EBUA.

Most of the past efforts on energy-bounded real-time scheduling focus on the deadline constraint, step TUFs, or deadline-based timeliness optimality criteria (e.g., meeting all or some percentage of deadlines). For example, [20] considers maximizing the total reward in a system under a fixed energy budget, with periodic and frame-based tasks (where reward is equivalent to our utility notion). Static scheduling of periodic tasks with continuous resource/utility functions and a fixed budget is explored in [13]. In [2], static and dynamic solutions for energy-constrained periodic task systems with or without DVS capacity are investigated. All these works are restricted to either deadlines or step TUFs (see Figure 1(d)).

It is interesting to note that energy-efficient real-time scheduling efforts which do not consider energy budgets are restricted to deadline-based optimality criteria [3, 10, 19]. This restriction was overcome in energy-efficient UA real-time scheduling works [21–24]. However, none of these works consider scheduling under fixed energy budgets.

Thus, the paper’s central contribution is the EBUA algorithm. We are not aware of any other efforts that solve the *energy-bounded, TUF/UA scheduling problem* solved by EBUA.

The rest of the paper is organized as follows: Section 2 describes our models and assumptions. Section 3 states the scheduling objective, and presents EBUA. In Section 4, we establish EBUA’s timeliness properties. Section 5 discusses the simulation studies. We conclude the paper in Section 6.

2 Models and Assumptions

2.1 System and Task Model

We consider a preemptive real-time system which consists of a set of independent tasks, denoted as $\mathbf{T} = \{T_1, T_2, \dots, T_n\}$. The target variable voltage processor can be operated at m frequencies $\{f_1, \dots, f_m | f_1 < \dots < f_m\}$. Each task T_i has a number of instances (*jobs*). With the UAM model, we associate a tuple $\langle a_i, P_i \rangle$ with a task T_i , meaning the maximal number of its instance arrivals during any sliding time window of P_i is a_i . Instances may arrive simultaneously. Note that the periodic model is a special case of UAM model with $\langle \bar{1}, P_i \rangle$, 1 being both the upper and lower bound.

We refer to the j^{th} job (or invocation) of task T_i as $J_{i,j}$. The basic scheduling entity that we consider is the job abstraction. Thus, we use J to denote a job without being task specific, as seen by the scheduler at any scheduling event.

2.2 Timeliness Model

A task's time constraint is specified using a TUF. We use $U_i(\cdot)$ to denote task T_i 's TUF, which has the same shape as the TUF of its each job i.e., $U_{i,j}(\cdot)$. Without being task specific, U_{J_k} means the TUF of a job J_k ; completion of J_k at a time t will yield a utility $U_{J_k}(t)$. In this paper, we restrict our focus to *non-increasing*, unimodal TUFs i.e., those TUFs for which utility never increases as time advances. Figures 1(a), 1(b), and 1(d) show examples.

Each TUF $U_{i,j}$, $i \in \{1, \dots, n\}$ has an initial time $I_{i,j}$ and a termination time $X_{i,j}$. Initial and termination times are the earliest and the latest times for which the TUF is defined, respectively. We assume that $I_{i,j}$ is equal to the arrival time of $J_{i,j}$, and $X_{i,j} - I_{i,j}$ is equal to the sliding time window P_i of the task T_i . If a job's termination time is reached and its execution has not been completed, an exception is raised. Normally, this exception will cause the job's abortion and execution of exception handlers.

2.3 Statistical Timeliness Requirement

Similar to that in [24], the *statistical performance requirement* of a task T_i is denoted as $\{\nu_i, \rho_i\}$, which implies that task T_i should accrue at least ν_i percentage of its maximum possible utility with a probability of at least ρ_i . For step TUFs, ν can only be 0 or 1.

During some situations, it is possible that such statistical assurances cannot be provided.

When that happens, the objective is to maximize the total utility per system-level energy consumption.

2.4 Activity Cycle Demands

Both UA scheduling and DVS depend on the prediction of task cycle demands. Similar to [24] and [22], we estimate the statistical properties (e.g., mean and variance) of the demand rather than the worst-case demand.

Let Y_i be the random variable representing T_i 's cycle demand i.e., the number of processor cycles required by T_i . We assume that the mean and variance of Y_i , denoted as $E(Y_i)$ and $Var(Y_i)$ respectively, are finite and determined through either online or off-line profiling. Under a frequency f (given in cycles per second), the expected execution time of a task T_i is given by $e_i = \frac{E(Y_i)}{f}$.

2.5 Power and Energy Consumption Model

We consider Martin's system-level energy consumption model to derive the energy consumption per cycle (detailed model descriptions can be found in [17, 21, 24]). In this model, when operating at a frequency f , a component's dynamic power consumption is denoted as P_d . P_d of CPU is given by $S_3 \times f^3$, where S_3 is constant.

Besides the CPU, *other* system components also consume energy. P_d of those that must operate at a fixed voltage (e.g., main memory) is given by $S_1 \times f$, while P_d of those that consume constant power with respect to the frequency (e.g., display devices) can be represented as an constant S_0 . In practice, the quadratic term $S_2 \times f^2$ is also included to account for the appearance of variations in DC-DC regulator efficiency across the range of output power, CMOS leakage currents, and other second order effects [17].

Summing the power consumption of all system components together, the system-level energy consumption of a task is obtained as: $E_i = e_i \times (S_3 \times f^3 + S_2 \times f^2 + S_1 \times f + S_0)$. Therefore, the expected energy consumption per cycle is given by:

$$E(f) = S_3 \times f^2 + S_2 \times f + S_1 + \frac{S_0}{f} \quad (1)$$

We consider a single processor system that relies on battery power. Thus, the system has a limited energy budget bound E_{bnd} . Further, we assume that the system must remain operational in the mission time interval $[0, MT]$. Therefore, the system is subject to a *hard* energy constraint,

and the total system-level energy consumption should not exceed E_{bnd} for MT time units. Recharging the battery is not possible or feasible during the mission.

For a system with DVS capability and m possible frequencies, we assume that, during any time interval $[t_1, t_2]$, total cycles executed with CPU speed f_i are denoted by $cyc_i, i \in \{1, \dots, m\}$. Thus, total system-level energy consumption during the time interval $[t_1, t_2]$ is given by $E(t_1, t_2) = \sum_{i=1}^m E(f_i) \times cyc_i$, where $E(f_i)$ is derived from Equation 1.

3 The EBUA Algorithm

3.1 Definition and Scheduling Objective

For a real-time system with statistical performance requirements that must remain operational during the mission time $[0, MT]$, there is minimum amount of energy needed to meet all the timeliness requirements when the system is not overloaded, while sustaining the system's operation until the end of the mission. We refer to this required threshold energy as E_{rqd} hereafter.

Definition 1 *If $E_{bnd} < E_{rqd}$, a real-time system is **energy-bounded**; energy-efficient UA scheduling in such a system is called energy-bounded UA scheduling.*

If $E_{bnd} \geq E_{rqd}$, we should try to satisfy the performance requirements. For an energy-bounded system with $E_{bnd} < E_{rqd}$, trying to execute all the jobs may result in a situation where the system runs out of energy in the middle of the mission. Thus, our aim then is to provide maximum energy efficiency while sustaining the system operation until the end of the mission.

With the problem definition, we consider a two-fold scheduling criterion: (1) assure that each task T_i accrues the specified percentage ν_i of its maximum possible utility with at least probability ρ_i ; and (2) maximize the system-level "energy efficiency," under the condition of assuring the feasibility of the system with limited energy budget i.e., the consumed energy during an operation/mission never exceeds the energy bound E_{bnd} . When it is not possible to satisfy $\{\nu_i, \rho_i\}$ for each task, our objective goes to (2).

Equation 1 indicates that there is an optimal value (not necessarily the lowest one) for clock frequency that minimizes system-level energy consumption. This adds to the difficulty to decide whether a system is energy-bounded. Some simple cases can be derived. For example, the optimal CPU speed for periodic tasks that always execute their worst-case cycles is constant and equal to the worst-case aggregate CPU demand (see [3,24]). Thus, with this periodic task

model, if E_{bnd} is smaller than the energy needed to execute all tasks with the speed equal to the aggregate CPU demand, which is just E_{rqd} , then the system is energy-bounded. Otherwise, it is not.

Intuitively, for a system that is not energy-bounded, during overloads the scheduling objective becomes utility maximization under energy constraints, since DVS tends to select the highest frequency, making the system consume constant energy. During under-loads, the algorithm delivers the performance assurances. Thus, the scheduling objective becomes the dual criterion problem of utility maximization, i.e., minimizing energy while achieving the given utility within the given time constraint. Such intuitions are slightly changed for energy-bounded systems. When $E_{bnd} < E_{rqd}$, our objective becomes utility maximization under fixed, limited energy consumption bound.

3.2 Task Critical Time and Demand

For non-increasing TUFs, satisfying a designated ν_i requires that task T_i 's sojourn time is upper bounded by a ‘‘critical time’’ (D_i). D_i is calculated from $\nu_i = \frac{U_i(D_i)}{U_i^{max}}$. If there are more than one points on the time axis that correspond to $\nu_i \times U_i^{max}$, we choose the latest point. Note that $P_i = D_i$ for a downward step TUF whose utility drops to a zero value at time P_i .

Knowing the mean and variance of task T_i 's demand Y_i , by a one-tailed version of the Chebyshev's inequality, we have $Pr[Y_i < c_i] \geq \frac{(c_i - E(Y_i))^2}{Var(Y_i) + (c_i - E(Y_i))^2}$, when $c_i \geq E(Y_i)$. To satisfy the requirement ρ_i , we let ρ_i equal the right half of the inequality, and obtain the minimal required $c_i = E(Y_i) + \sqrt{\frac{\rho_i \times Var(Y_i)}{1 - \rho_i}}$.

Thus, the scheduler allocates c_i cycles to each job $J_{i,j}$, so that the probability that job $J_{i,j}$ requires no more than the allocated c_i cycles is at least ρ_i i.e., $Pr[Y_i < c_i] \geq \rho_i$.

3.3 Energy-Bounded UA Scheduling

For an energy-bounded system, the jobs to be executed should be carefully selected in order to make best use of available energy. Such selection process is guided by the performance metric Utility and Energy Ratio (UER), which is defined to integrate timeliness and energy consumption. A job's UER measures the utility that can be accrued per unit energy consumption by executing the job. The UER of T_i under frequency f at time t is calculated as $\frac{U_i(t + c_i/f)}{c_i \times E(f)}$, where $E(f)$ is derived using Equation 1. Equation 1 indicates that there is an optimal value for clock frequency to maximize T_i 's UER.

For the UAM model, we denote C_i as the total cycles of a_i jobs in the time window P_i , i.e., $C_i = a_i c_i$. With C_i , the aggregate CPU demand of the task set \mathbf{T} is defined as $CPU_{dmd} = \sum_{i=1}^n C_i / P_i$, and the system load is defined as $Load = \frac{1}{f_m} \sum_{i=1}^n \frac{C_i}{D_i}$. Note that we do not denote CPU_{dmd} as $\sum_{i=1}^n C_i / D_i$, and the reason is elaborated in Section 3.5.

The scheduling events of EBUA include the arrival and completion of a job, and the expiration of a time constraint such as the arrival of the termination time of a TUF. To describe EBUA, we define the following variables and auxiliary functions:

- $E_{rem} \leq E_{bnd}$; it is the system's remaining energy for execution.
- For task T_i , during a time window P_i , D_i^a and c_i^r are its earliest job's absolute critical time and remaining cycles, respectively. We define the task-level flag SEL_i to represent whether task T_i 's instances are selected in the job selection phase. $SEL_i = skipped$ indicates that the task is skipped, and $SEL_i = selected$ indicates that it is, or can be selected for execution.
- $\mathcal{J}_r = \{J_1, J_2, \dots, J_{n'}\}$ is the current unscheduled job set; σ is the ordered output schedule. $J_k.D$ is job J_k 's critical time; $J_k.X$ is its termination time, and $J_k.c$ is its remaining cycles.
- $T(J_k)$ returns the corresponding task of job J_k . $headOf(\sigma)$ returns the first job in σ ; $sortByUER(\sigma)$ sorts σ by each job's UER, in a non-increasing order. $selectFreq(x)$ returns the lowest frequency $f_i \in \{f_1 < \dots < f_m\}$ such that $x \leq f_i$.
- $offlineComputing()$ runs at $t = 0$. It computes c_i and D_i as described in Section 3.2, and determines its optimal frequency $f_{T_i}^o \in \{f_1, \dots, f_m\}$, which maximizes T_i 's UER. Each task initially runs at the speed $f_{T_i}^{ini} = \max(f_{T_i}^o, selectFreq(CPU_{dmd}))$.
- $insert(T, \sigma, I)$ inserts T in the ordered list σ at the position indicated by index I ; if there are already entries in σ at the index I , T is inserted after them.
- $feasible(\sigma)$ returns a boolean value denoting schedule σ 's feasibility. For σ to be feasible, the predicted completion time of each job in σ , calculated at the highest frequency f_m , must not exceed its termination time. $eBounded(\sigma)$ checks whether the predicted energy consumption of σ is less than the allowed bound of energy consumption before σ 's predicted completion time.
- $updateSel(\mathbf{T})$ updates the flag SEL_i for the task set \mathbf{T} .

A high level description of EBUA is shown in Algorithm 1. When EBUA is invoked at time t_{cur} , it first updates each task's remaining cycle (line 5–8). The algorithm then checks the feasibility of the jobs. If a job is infeasible, then it can be safely aborted (line 10). Otherwise, its UER is calculated (line 13).

Algorithm 1: EBUA: High Level Description

```

1: input      :  $\mathbf{T} = \{T_1, \dots, T_n\}$ ,  $\mathcal{J}_r = \{J_1, \dots, J_{n'}\}$ ,  $E_{rem}$ 
2: output    : selected job  $J_{exe}$  and frequency  $f_{exe}$ 
3: offlineComputing ( $\mathbf{T}$ );
4: Initialization:  $t := t_{cur}$ ,  $\sigma := \emptyset$ , update  $E_{rem}$ ;
5: switch triggering event do
6:   case task_release( $T_i$ )            $c_i^r = c_i$ ;
7:   case task_completion( $T_i$ )        $c_i^r = 0$ ;
8:   otherwise                          Update  $c_i^r$ ;
9: for  $\forall J_k \in \mathcal{J}_r$  do
10:  if feasible( $J_k$ )=false then
11:     $c_{T(J_k)}^r := 0$ ;
12:    abort( $J_k$ );
13:  else
14:     $J_k.UER := U_{J_k}(t + \frac{J_k.c}{f_m}) / (E(f_m) \times J_k.c)$ ;
15:  $\sigma_{tmp} := \text{sortByUER}(\mathcal{J}_r)$ ;
16: for  $\forall J_k \in \sigma_{tmp}$  from head to tail do
17:  if  $J_k.UER > 0$  then
18:    copy  $\sigma$  into  $\sigma_{tent}$ :  $\sigma_{tent} := \sigma$ ;
19:    insert( $J_k$ ,  $\sigma_{tent}$ ,  $J_k.D$ );
20:    if feasible( $\sigma_{tent}$ ) and eBounded( $\sigma_{tent}$ ) then
21:       $\sigma := \sigma_{tent}$ ;
22:  else break;
23: updateSel( $\mathbf{T}$ );
24:  $J_{exe} := \text{headOf}(\sigma)$ ;
25:  $f_{exe} := \text{decideFreq}(\mathbf{T}, J_{exe}, t)$ ;
26: return  $J_{exe}$  and  $f_{exe}$ ;

```

The jobs are then sorted by their UERs (line 15). In each step of the **for** loop from line 16 to 22, the job with the largest UER is inserted into σ , if it can produce a positive UER, and keep the schedule after insertion feasible. Also we check to assure that the energy consumption from its execution will not exceed the energy bound. Thus, σ is a feasible and energy-bounded schedule sorted by the jobs' critical times, in a non-decreasing order.

At line 23, EBUA updates the flag SEL_i for each task. SEL_i is set to be *skipped* if and only if T_i has instances in the ready job queue \mathcal{J}_r , but none of them are selected in σ . Note that even when T_i has no jobs in \mathcal{J}_r , SEL_i is set to be *selected*.

Finally, from line 24 to 26, with algorithm `decideFreq()`, EBUA analyzes the demands of the task set and applies DVS to decide the execution frequency f_{exe} for the selected job J_{exe} at the head of σ . DVS can reduce the energy consumption of the selected jobs whenever possible, which enables us to further increase excess energy that can be later used for job selection.

3.4 Monitoring Energy Consumption Online

Since our energy bound E_{bnd} is associated with the mission time $[0, MT]$, we need to dynamically monitor the system-level energy consumption and adjust the selected jobs to execute. The online monitoring is conducted with the function `eBounded(σ_{tent})` in line 20 of Algorithm 1.

`offlineComputing(T)` sets the initial speed $f_{T_i}^{ini}$ for each task. We assume the last job in σ_{tent} has the predicted completion time D_{tent} . Thus, for σ_{tent} where each job is executed at its initial speed, its expected energy consumption $E(t_{cur}, D_{tent})$ can be calculated by the mechanism described in the last paragraph of Section 2.5. When $E_{bnd} - E_{rem} + E(t_{cur}, D_{tent}) \leq \frac{D_{tent}}{MT} \times E_{bnd}$, `eBounded(σ_{tent})` returns *true*; otherwise, it returns *false*.

3.5 DVS with Energy Bound

We have CPU-time reclamation through DVS. 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 sojourn times of already selected jobs).

In [22], Wu et al. consider the “processor demand approach” [4] to analyze the feasibility of tasks with stochastic parameters. They prove that for a task T_i with a UAM arrival pattern $\langle a_i, P_i \rangle$ and critical time D_i , all its jobs can meet their D_i , if T_i is executed at a frequency no lower than $\frac{C_i}{D_i}$, where $C_i = a_i c_i$. They also propose a DVS mechanism for tasks with UAM arrivals based on LaEDF [19]. We extend and adapt their algorithm to energy-bounded real-time systems.

For a task T_i , when $P_i > D_i$, T_i 's CPU demand can be denoted as $\frac{C_i}{D_i}$ [22]. But the task set's aggregate CPU demand will be overestimated if we denote it as $\sum_{i=1}^n \frac{C_i}{D_i}$. This is because, when $P_i = D_i$, our calculation will safely assume that beyond D_i , the next invocation of task T_i starts immediately and consumes $\frac{C_i}{P_i}$ processing resources. But when $P_i > D_i$, substituting the critical time D_i for P_i will underestimate the future processing capacity that is available.

We elaborate such estimation through the processor demand approach [4, 5]. The necessary and sufficient condition for satisfying a task set's critical times is $fL \geq \sum_{i=1}^n \left\lfloor \frac{L+P_i-D_i}{P_i} \right\rfloor C_i$, $\forall L > 0$, where f is the processor frequency allocated to the task set, and $\left\lfloor \frac{L+P_i-D_i}{P_i} \right\rfloor C_i$ is the cycle demand of task T_i during the time interval $[0, L]$. Since $\left\lfloor \frac{L+P_i-D_i}{P_i} \right\rfloor \leq \left(\frac{L}{P_i} + \frac{P_i-D_i}{P_i} \right)$, it is sufficient to have:

$$f \geq \sum_{i=1}^n \frac{C_i}{L} \left(\frac{L}{P_i} + \frac{P_i - D_i}{P_i} \right) = \sum_{i=1}^n \frac{C_i}{P_i} + \sum_{i=1}^n \left(\frac{P_i - D_i}{L} \times \frac{C_i}{P_i} \right), \forall L > 0. \quad (2)$$

We study the part $\sum_{i=1}^n \left(\frac{P_i - D_i}{L} \times \frac{C_i}{P_i} \right)$ in Equation 2. In addition to $\sum_{i=1}^n \frac{C_i}{P_i}$, this part represents the CPU demand resulting from the fact that $D_i < P_i$. Note that for a task T_i , if $L < D_i$, $\left\lfloor \frac{L+P_i-D_i}{P_i} \right\rfloor C_i = 0$. This is because, in the interval $[0, L]$, no job of T_i has a critical time earlier than D_i , and thus there are no cycle demand from task T_i . If we assume from T_1 to T_n ,

$D_1 < D_2 < \dots < D_n$, then only when $L \geq D_n$, each task can contribute to $\sum_{i=1}^n \left(\frac{P_i - D_i}{L} \times \frac{C_i}{P_i} \right)$. Further, it is easy to see that $\left(\frac{P_i - D_i}{L} \times \frac{C_i}{P_i} \right)$ monotonically decreases with the increase of L . Therefore, in Equation 2, $\sum_{i=1}^n \left(\frac{P_i - D_i}{L} \times \frac{C_i}{P_i} \right)$ has very limited contribution to the required frequency f , and the majority of CPU demand is consumed by $\sum_{i=1}^n \frac{C_i}{P_i}$. So we denote the aggregate CPU demand of \mathbf{T} as $CPU_{dmd} = \sum_{i=1}^n \frac{C_i}{P_i}$.

We validated the efficiency of the definition of CPU_{dmd} through experimental comparison. We observed in our experiments that estimating CPU_{dmd} as $\sum_{i=1}^n \frac{C_i}{D_i}$ is safe but very conservative, resulting in much less energy savings than the more aggressive estimation. Thus, we adopt the measurement $CPU_{dmd} = \sum_{i=1}^n \frac{C_i}{P_i}$, and propose the aggressive energy-conserving DVS approach, `decideFreq()`, in Algorithm 2.

Algorithm 2: decideFreq()

```

1: input:  $\mathbf{T}, J_{exe}, t_{cur}$ ; output:  $f_{exe}$  ;
2:  $CPU_{dmd} := 0$ ;
3: for  $T_i \in \mathbf{T}$  do
4:   if  $SEL_i = selected$  then
5:      $CPU_{dmd} := CPU_{dmd} + C_i/P_i$ ;
6:  $s := 0$ ;
7: for  $i = 1$  to  $n$ ,  $T_i \in \{T_1, \dots, T_n \mid D_1^a \geq \dots \geq D_n^a\}$  do
8:   if  $SEL_i = skipped$  then
9:     continue;
10:   /* reverse EDF order of tasks */
11:    $CPU_{dmd} := CPU_{dmd} - C_i/D_i$ ;
12:    $x := \max(0, C_i^r - (f_m - CPU_{dmd}) \times (D_i^a - D_n^a))$ ;
13:    $CPU_{dmd} := \begin{cases} f_m, & \text{if } D_i^a - D_n^a = 0 \\ CPU_{dmd} + \frac{C_i^r - x}{D_i^a - D_n^a}, & \text{otherwise} \end{cases}$  ;
14:    $s := s + x$ ;
15:  $f := \min(f_m, s/(D_n^a - t_{cur}))$ ;
16:  $f_{exe} := \text{selectFreq}(f)$ ;
17:  $f_{exe} := \max(f_{exe}, f_{T(J_{exe})}^o)$ ;

```

In line 3–5 of Algorithm 2, if the task-level flag SEL_i is *skipped*, the skipped task can be considered with zero actual execution cycles, enabling us to further reduce the speed. EBUA keeps track of the remaining computation cycles C_i^r . For the current time window P_i with a_i' instances, C_i^r is calculated as $C_i^r = \min((a_i' - 1)c_i + c_i^r, (a_i - 1)c_i + c_i^r)$. Note that the actual number of jobs a_i' can be larger than the maximum job arrivals a_i , because there may be unfinished jobs from the previous time window P_i . But we only need to consider at most a_i instances of them.

In line 6–14, EBUA considers the interval until the next task critical time and attempts to “push” as much work as possible beyond the critical time. Similar to LaEDF [19], the algorithm considers the tasks in the latest-critical-time-first order in line 7. But since there may be more than one job of T_i in P_i , D_i^a is set to be the earliest invocation’s absolute critical time.

LaEDF [19] updates each task’s D_i^a immediately when a task instance completes. In our DVS approach, we delay such an update until the next task instance is released, which results in additional energy savings. s reflects the minimum number of cycles that must be executed by D_n^a in order for the selected tasks to meet their critical times (line 13).

Thus, `decideFreq()` capitalizes on early task completion by deferring work for future tasks in favor of scaling the current task’s frequency. Also, during overloads, the required frequency may be higher than f_m , and `selectFreq()` will fail to return a value. In line 14, we solve this by setting the upper limit of the required frequency to be the highest frequency f_m . Finally, f_{exe} is compared with $f_{T(J_{exe})}^o$. The higher frequency is selected to provide the timeliness assurance—we cannot decrease f_{exe} , but may increase it to maximize the system-level energy efficiency.

4 Timeliness Properties

We first establish EBUA’s assurance on energy-bounded systems.

Theorem 1 *EBUA assures that the consumed energy during an interval $[0, MT]$ never exceeds the energy bound E_{bnd} .*

Proof EBUA’s dynamic monitoring of energy consumption described in Section 3.4 assures that, at any scheduling event during $[0, MT]$, the energy consumption of selected jobs never exceeds the allowed portion of E_{bnd} . Thus, the theorem holds. \square

The periodic model is a special case of the UAM model. If $E_{bnd} \geq E_{rqd}$, then under the conditions of (1) a non-energy-bounded system; (2) a set of periodic tasks with $\langle \bar{1}, P_i \rangle$ and step TUFs; and (3) absence of CPU overloads (under Liu and Layland’s condition [15]), we can establish EBUA’s timeliness properties.

Theorem 2 *Under conditions (1), (2), and (3), a schedule produced by EDF [11] is also produced by EBUA, yielding equal total utilities. This is a critical time-ordered schedule.*

Proof We prove this by examining Algorithm 1. For periodic tasks with step TUFs during non-overload situations, σ from line 20 of Algorithm 1 is critical time-ordered. The step TUF’s critical time that we consider is analogous to a deadline in [11]. As proved in [11, 15], a deadline-ordered schedule is optimal (with respect to meeting all deadlines) when there are no overloads. Thus, σ yields the same total utility as that by EDF. \square

Some important corollaries about EBUA’s timeliness behavior during under-loads can be deduced from EDF’s optimality [11].

Corollary 3 *Under conditions (1), (2), and (3), EBUA always meets all task critical times of T_i with a probability of at least ρ_i .*

Corollary 4 *Under conditions (1), (2), and (3), EBUA minimizes the maximum lateness.*

Theorem 5 *Under conditions (1), (2), and (3), EBUA meets the statistical timeliness performance requirements.*

We also derive Theorem 5’s counterpart for non-step and non-increasing TUFs in Theorem 6, where critical times are less than termination times. The proof for it can be found in [4].

Theorem 6 *In a non-energy-bounded system, for a set of independent periodic tasks, where each task is subject to a non-increasing TUF, the task set is schedulable and can meet all statistical timeliness requirements under the condition of Baruah, Rosier, and Howell [4].*

5 Experimental Results

5.1 Experimental Settings

To evaluate the energy efficiency of EBUA, we perform simulation experiments to compare EBUA with other energy-bounded algorithms, i.e., OFC [2] and REW-Pack [20]. OFC statically (off-line) selects tasks with the heuristic of Larger Reward Density (LRD), based only on the worst-case workload information. REW-Pack works for frame-based or periodic tasks.

We simulate the AMD k6 processor with PowerNow! mechanism [1]. The CPU can operate at seven different frequencies, {360, 550, 640, 730, 820, 910, 1000 MHz}.

We select task sets with 10 to 50 tasks in three applications for our study. Their parameters are summarized in Table 1. Within each range, the time window P is uniformly distributed. The synthesized task sets simulate the varied mix of short and long time windows. For each task cycle demand Y_i , we keep $Var(Y_i) \approx E(Y_i)$, and generate normally-distributed demands. Finally, according to the calculation of c_i in Section 3.2, the cycle demands $E(Y_i)$ s are scaled by a constant k , and $Var(Y_i)$ s are scaled by k^2 ; k is chosen such that the system load reaches a desired value. The U^{max} of the TUFs in A_1 , A_2 , and A_3 are uniformly generated in the range [50, 70], [300, 400], and [1, 10], respectively.

The energy consumption per cycle at a particular frequency is calculated using Equation 1. In practice, the S_3 , S_2 , S_1 , and S_0 terms depend on the power management state of the system and its subsystems [17, 21, 24]. We test three energy settings similar to those in [24], as shown

Table 1: Task Settings

Applications	# tasks	UAM $\langle a, P \rangle$	U^{max}
A_1	4	$\langle 5, 22-28 \rangle$	[50, 70]
A_2	18	$\langle 8, 50-70 \rangle$	[300, 400]
A_3	8	$\langle 3, 2.4-9.6 \rangle$	[1, 10]

Table 2: Energy Settings

Energy Model	S_3	S_2	S_1	S_0
E_1	1.0	0	0	0
E_2	0.75	0	0	$0.25f_m^3$
E_3	0.5	0	0	$0.5f_m^3$

in Table 2. Note that E_1 is the same as the conventional energy model, which only considers the CPU's energy consumption.

5.2 Performance with Step TUFs

We first evaluate the performance with step TUFs, so that EBUA can be compared with the other strategies. We set $\{\nu_i = 1, \rho_i = 0.96\}$, and apply different schemes on independent periodic task sets under different energy settings. We vary $Eratio$, which is defined as the ratio of E_{bnd} to E_{rqd} , from 0.1 to 1.0, and show the accrued utility at $Load = 0.7$ and $Load = 1.5$, respectively. Note that REW-Pack requires the hyper-period of periodic tasks, which can be very large due to our synthesized task sets, so we approximate it.

When $Eratio < 1$, the system is energy-bounded; when $Load > 1$, the system is CPU overloaded. Thus, by changing $Eratio$ and $Load$, we can generate interesting scenarios to study the tradeoffs between energy and utility.

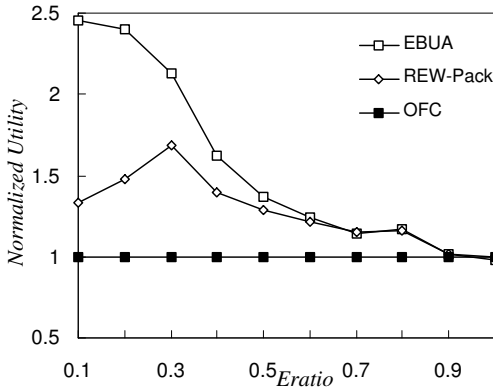
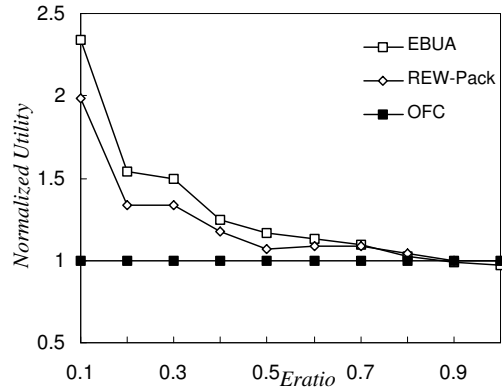
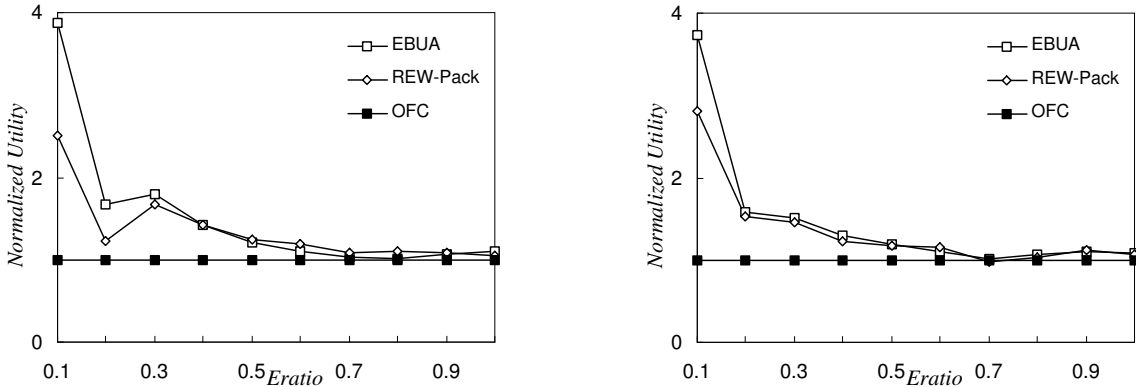
(a) $Load = 0.7, E_1$ (b) $Load = 0.7, E_3$ **Figure 2:** Normalized Utility vs. $Eratio$ during under-loads under E_1 and E_3

Figure 2 shows the utilities normalized to that of OFC under energy setting E_1 and E_3 . It shows the performance of different strategies in the scenario of CPU under-loads ($Load = 0.7$) in an energy-bounded system. From Figure 2(a) and Figure 2(b), we observe that when E_{bnd} is low, the system has a strict energy bound and it is crucial to utilize excess energy due to early completions. Hence, under such conditions the difference in performance is significant. As

E_{bnd} increases, the system becomes less energy-bounded. The schemes converge when $Eratio = 100\%$, because the system has enough energy to meet all the critical times and accrue its maximum utility, due to EDF's optimality [11] in such cases.

Figure 3(a) and Figure 3(b) show the scenarios of CPU overloads ($Load = 1.5$) in an energy-bounded system, under energy setting E_1 and E_3 , respectively. Plots in Figure 3 bear the same trends as those in Figure 2. But when $Eratio = 100\%$, the plots in Figure 3 do not converge. This is because, although the system is not energy-bounded, during CPU overloads different schemes still show different abilities in utility accrual.



(a) $Load = 1.5, E_1$

(b) $Load = 1.5, E_3$

Figure 3: Normalized Utility vs. $Eratio$ during overloads under E_1 and E_3

In Figure 2 and Figure 3, E_{bnd} is recalculated as a percentage of E_{rqd} , which is also a function of the system load. In our next set of experiments, E_{bnd} is set to a fixed value, namely the energy required to meet all the critical times when $Load = 0.7$. We represent this value as $E_{rqd}(Load = 0.7)$. Figure 4 shows the normalized utilities of the three schemes with $E_{bnd} = E_{rqd}(Load = 0.7)$, when $Load$ varies from 0.2 to 1.8 under energy setting E_1 .

Figure 4 shows more complicated combinations of system energy requirement and CPU loads. When $Load \leq 0.7$, the system has enough energy to meet all critical times (i.e., $E_{bnd} \geq E_{rqd}$, and system is under-loaded). Therefore, all schemes yield the same utility. As $Load$ increases beyond 0.7 but below 1.0, the system becomes effectively more energy-bounded, but is still under-loaded. When $Load$ exceeds 1.0, the system is both energy-bounded and overloaded. The performance gap with the increase of $Load$ shown in Figure 4 demonstrates that EBUA accrues higher utility with fixed energy bound.

5.3 Performance with Non-Increasing TUFs and UAM

We then consider non-step and non-increasing TUFs, and UAM tasks with EBUA and EUA* [22]. each task is allocated a linear TUF, and its slope is calculated as $-\frac{U^{max}}{P}$, P being the time window. We set $\{\nu_i = 0.3, \rho_i = 0.9\}$ to each task, and use the energy model E_1 in the experiments of this section.

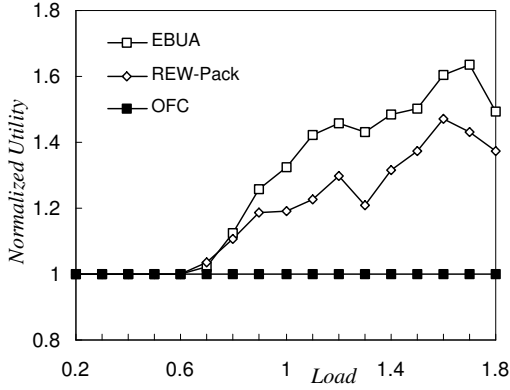


Figure 4: Utility vs. *Load* with Fixed E_{bnd}

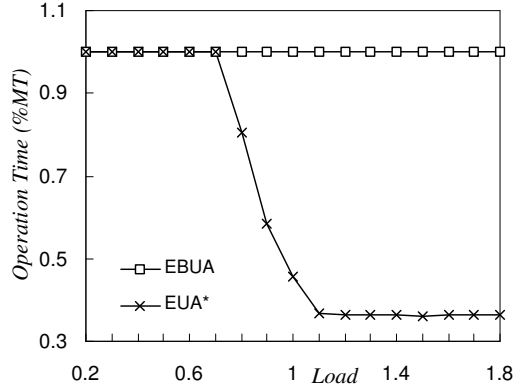


Figure 5: Operation Time vs. *Load*

In our experiments, we study the operation time of EBUA and EUA*, within which the system remains functional. We still set $E_{bnd} = E_{rqd}$ ($Load = 0.7$), and vary *Load* from 0.2 to 1.8 to study how EBUA handles the energy-bound. Figure 5 shows operation times normalized to the results of EBUA.

We observe that, the operation time of EBUA is always the whole mission time MT , since EBUA dynamically monitors energy consumption and keeps system functional during $[0, MT]$. When $Load \leq 0.7$, EUA* can remain functional during $[0, MT]$, because the system is neither energy-bounded nor overloaded. But when $Load \geq 0.7$, the operation time of EUA* decreases to far below MT until $Load = 1.0$, as it cannot deal with energy-bounded systems. Note that when $Load \geq 1.0$, the operation time of EUA* becomes constant. This is because during system overloads, DVS always picks the highest frequency f_m , and the system-level energy consumption, hence E_{rqd} , also becomes constant. This also means that $E_{ratio} = \frac{E_{bnd}}{E_{rqd}}$ is constant, when $Load \geq 1.0$.

5.4 Comparison of Dynamic Energy Drain Rates

For a battery, its discharge rate decides its life. Usually we hope this discharge rate can be as constant as possible, which is represented by small variances on the discharge curve.

In this section, we approximately measure and compare the dynamic energy drain rates of different mechanisms. According to the relationship between energy, voltage, and current, a constant energy drain rate implies a constant battery discharge rate in terms of discharge current, assuming that the battery operates under a constant voltage.

For the same task sets as in Section 5.2, we uniformly take 20 sampling time points during the interval $[0, MT]$, and measure the dynamic energy drain rate at each point, as the ratio of consumed energy to system operation time so far. In our experiments, we set $Load = 0.7$ and $E_{bnd} = 0.5 \times E_{rqd} (Load = 0.7)$. The measurements are normalized to the average energy drain rate, E_{bnd}/MT , and the normalized results under energy settings E_1 and E_2 are shown in Figure 6. Note that these two figures only represent our qualitative study.

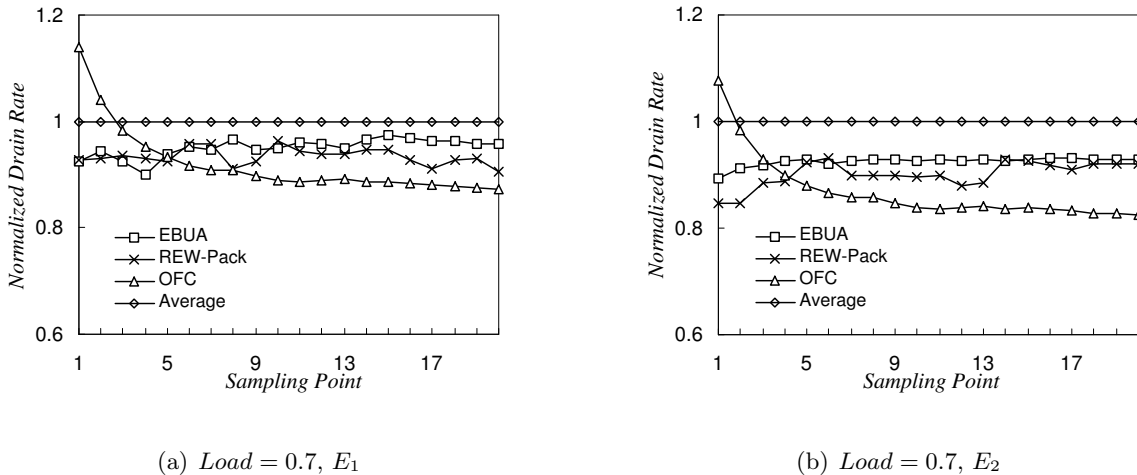


Figure 6: Normalized Energy Drain Rates with $E_{bnd} = 0.5 \times E_{rqd} (Load = 0.7)$ under E_1 and E_2

We observe from the plots in both Figure 6(a) and Figure 6(b) that, all strategies have dynamic energy drain rates less than the average rate. This is because the energy consumptions of all methods during the interval $[0, MT]$ are no larger than E_{bnd} . From the figures, EBUA has a constant energy drain rate. The initial drain rate of OFC is larger than the average rate, because before the mission starts, OFC has off-line selected tasks to execute, and at the beginning those tasks may generate a high energy drain rate. Also note that the drain rate of EBUA is higher than that of OFC and REW-Pack. This implies that the amount of energy consumed by EBUA is closer to the energy budget E_{bnd} . Thus, EBUA is more effective for energy-bounded systems.

6 Conclusions

Many battery-powered, dynamic embedded systems have energy bounds, embodied by a battery that has a finite lifetime. The work presented in this paper is aimed at addressing the feasibility and timeliness performance optimization problem for such systems, which must remain functional during an operation/mission with a bounded energy budget. We present an energy-bounded, UA real-time scheduling algorithm called EBUA that considers activities subject to TUF time constraints and UAM arrival model, statistical timeliness performance requirements, and system-level energy consumption concerns.

EBUA considers utility maximization under energy bounds, and dynamically skips less important jobs for execution to achieve timeliness objectives, while assuring that the system remains functional until the end of its mission. The algorithm dynamically scales voltage and frequency to reduce system-level energy consumption, and to obtain additional energy savings for selecting new jobs in a dynamic fashion. We analytically establish several timeliness properties of EBUA. Our simulation experiments confirm EBUA's assurances on energy consumption and timeliness behaviors, and improvement in system-level energy efficiency.

Several aspects of the work are directions for further research. Examples include considering more general task arrival models (than UAM), and reward functions for tasks, where tasks accrue reward as a function of their execution cycles.

References

- [1] Advanced Micro Devices Corporation. Mobile AMD-K6-2+ Processor Data Sheet. Publication #23446, June 2000.
- [2] T. A. AlEnawy and H. Aydin. On Energy-Constrained Real-Time Scheduling. In *Euromicro Conference on Real-Time Systems*, pages 165–174, June 2004.
- [3] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 95–105, December 2001.
- [4] S. K. Baruah, L. E. Rosier, and R. R. Howell. Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor. *Real-Time Systems*, 2(4):301–324, November 1990.
- [5] G. C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, 2nd edition, 2005.
- [6] R. Clark, E. D. Jensen, A. Kanevsky, J. Maurer, P. Wallace, T. Wheeler, Y. Zhang, D. Wells, T. Lawrence, and P. Hurley. An Adaptive, Distributed Airborne Tracking System. In *Proceedings of The IEEE Workshop on Parallel and Distributed Systems*, volume 1586 of *LNCS*, pages 353–362. Springer-Verlag, April 1999.
- [7] R. K. Clark. *Scheduling Dependent Real-Time Activities*. PhD thesis, Carnegie Mellon University, 1990. CMU-CS-90-155, <http://www.real-time.org> (last accessed: January 22, 2005).

- [8] R. K. Clark, E. D. Jensen, and N. F. Rouquette. Software Organization to Facilitate Dynamic Processor Scheduling. In *Proceedings of IEEE Parallel and Distributed Processing Symposium*, April 2004.
- [9] M. Dertouzos. Control Robotics: the Procedural Control of Physical Processes. *Information Processing*, 74, 1974.
- [10] R. Graybill and R. Melhem. *Power Aware Computing*. Kluwer Academic/Plenum Publishers, 2002.
- [11] W. Horn. Some Simple Scheduling Algorithms. *Naval Research Logistics Quarterly*, 21:177–185, 1974.
- [12] E. D. Jensen, C. D. Locke, and H. Tokuda. A Time-Driven Scheduling Model for Real-Time Systems. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 112–122, December 1985.
- [13] D.-I. Kang, S. P. Cargo, and J. Suh. A Fast Resource Synthesis Technique for Energy-Efficient Real-Time Systems. In *Proceedings of IEEE Real-Time Systems Symposium*, December 2002.
- [14] W. Kim, J. Kim, and S. L. Min. A Dynamic Voltage Scaling Algorithm for Dynamic-Priority Hard Real-Time Systems Using Slack Time Analysis. In *Proceedings of IEEE/ACM Design, Design, Automation and Test in Europe (DATE)*, March 2002.
- [15] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, 20(1):46–61, 1973.
- [16] C. D. Locke. *Best-Effort Decision Making for Real-Time Scheduling*. PhD thesis, Carnegie Mellon University, 1986. CMU-CS-86-134, <http://www.real-time.org> (last accessed: January 22, 2005).
- [17] T. Martin. *Balancing Batteries, Power and Performance: System Issues in CPU Speed-Setting for Mobile Computing*. PhD thesis, Carnegie Mellon University, August 1999.
- [18] D. P. Maynard, S. E. Shipman, R. K. Clark, J. D. Northcutt, R. B. Kegley, B. A. Zimmerman, and P. J. Keleher. An Example Real-Time Command, Control, and Battle Management Application for Alpha. Technical report, Department of Computer Science, Carnegie Mellon University, December 1988. Archons Project Technical Report 88121.
- [19] P. Pillai and K. G. Shin. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *Proceedings of the ACM symposium on Operating Systems Principles*, pages 89–102, 2001.
- [20] C. Rusu, R. Melhem, and D. Mosse. Maximizing the System Value while Satisfying Time and Energy Constraints. *IBM Journal of Research and Development*, 47(5/6):689–702, September/November 2003.
- [21] J. Wang, B. Ravindran, and T. Martin. A Power Aware Best-Effort Real-Time Task Scheduling Algorithm. In *Proceedings of The IEEE Workshop on Software Technologies for Future Embedded Systems, IEEE International Symposium on Object-oriented Real-time Distributed Computing*, pages 21–28, May 2003.
- [22] H. Wu, B. Ravindran, and E. D. Jensen. Energy-Efficient, Utility Accrual Real-Time Scheduling Under the Unimodal Arbitrary Arrival Model. In *Proceedings of IEEE/ACM Design, Design, Automation and Test in Europe (DATE)*, pages 474–479, March 2005.
- [23] H. Wu, B. Ravindran, E. D. Jensen, and P. Li. CPU Scheduling for Statistically-Assured Real-Time Performance and Improved Energy Efficiency. In *Proceedings of 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES/ISSS)*, pages 110–115, September 2004.
- [24] H. Wu, B. Ravindran, E. D. Jensen, and P. Li. Energy-Efficient, Utility Accrual Scheduling under Resource Constraints for Mobile Embedded Systems. In *Proceedings of Fourth ACM International Conference on Embedded Software (EMSOFT)*, pages 64–73, September 2004.
- [25] W. Yuan and K. Nahrstedt. Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 149–163. ACM Press, 2003.