

Energy-Efficient, Utility Accrual Real-Time Scheduling Under the Unimodal Arbitrary Arrival Model *

Haisang Wu* Binoy Ravindran*

*ECE Dept., Virginia Tech
Blacksburg, VA 24061, USA
{hswu02,binoy}@vt.edu

E. Douglas Jensen†

†The MITRE Corporation
Bedford, MA 01730, USA
jensen@mitre.org

Abstract

We present an energy-efficient real-time scheduling algorithm called *EUA**, for the unimodal arbitrary arrival model (or *UAM*). *UAM* embodies a “stronger” adversary than most arrival models. The algorithm considers application activities that are subject to time/utility function time constraints, *UAM*, and the multi-criteria scheduling objective of probabilistically satisfying utility lower bounds, and maximizing system-level energy efficiency. Since the scheduling problem is intractable, *EUA** allocates CPU cycles, scales clock frequency, and heuristically computes schedules using statistical estimates of cycle demands, in polynomial-time. We establish that *EUA** achieves optimal timeliness during under-loads, and identify the conditions under which timeliness assurances hold. Our simulation experiments illustrate *EUA**’s superiority.

1. Introduction

Mobile and embedded devices are becoming increasingly popular at the workplace and at home. As batteries are the primary energy source of such devices, increasing the energy-efficiency of computing and communications is of critical importance. In many embedded systems, the CPU consumes a substantial fraction of the total energy, making it a prime target for energy saving in past efforts. Besides the CPU, other components also consume energy.

The characteristics of the power/performance tradeoffs of CMOS circuits dictate that the power consumption changes linearly with frequency and quadratically with voltage, yielding potential energy savings for reduced speed/voltage. Dynamic Voltage Scaling (DVS) is the technique for exploiting this tradeoff—an appropriate clock rate and voltage is determined in response to dynamic application behaviors (see [2, 8, 13] and the references therein).

In this paper, we focus on dynamic, adaptive, embedded real-time control systems that occur in many domains including robotics, space, defense, consumer electronics, and financial markets. Such systems are fundamentally time-critical and energy-critical, as they must produce timely control responses, while running on batteries. Further, they 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, whenever possible. Consequently, the 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 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 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. The optimality of the soft time constraints is generally as mission- and safety-critical as that of the hard ones.

Jensen’s time/utility functions [7] (or TUFs) allow the semantics of soft time constraints to be precisely specified. A TUF, which generalizes the deadline constraint, specifies the utility to the system resulting from the completion of an activity as a function of its 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 time constraints of two applications in the defense domain [4, 12]. The classical deadline is a binary-valued, downward “step” shaped TUF; 1(d) shows an example.

When activity time constraints are expressed with TUFs, the timeliness 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

* This work was supported by the US Office of Naval Research under Grant N00014-00-1-0549 and The MITRE Corporation under Grant 52917.

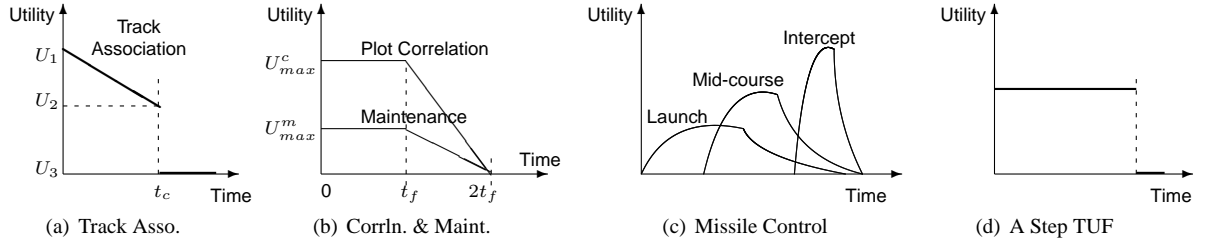


Figure 1. Example TUF Time Constraints from AWACS [4] (a) and Coastal Air Defense [12] (b-c), and a Step TUF (d).

called *Utility Accrual* (or UA) criteria, and sequencing (scheduling, dispatching) algorithms that consider UA criteria are called UA sequencing algorithms.

UA criteria facilitate adaptivity during overloads, when completing activities that are more important than those which are more urgent is often desirable. During overloads, UA algorithms that maximize summed utility typically favor activities that are more important (since more utility can be attained from them) than those which are more urgent.

Most past efforts on energy-efficient, real-time scheduling consider activity arrival models that are either periodic, or frame-based (where all periods are equal), or sporadic. These include past works that consider deadline-based timeliness optimality criteria (e.g., meeting all or some percentage of deadlines) [2, 8, 13, 18], and those that consider UA criteria (e.g., maximizing summed utility) [14–17]. As far as we know, the only exception is [15], which allows aperiodic arrivals. However, [15] provides no timeliness assurances. Thus, prior efforts are concentrated on two extremes: (1) those that provide timeliness assurances, but under highly restrictive periodic, frame-based, or sporadic arrivals; or (2) those that allow aperiodic arrivals, but provide no timeliness assurances. Both these extremes are inappropriate for the applications/domains of interest to us.

In this paper, we bridge these extremes by considering the *unimodal arbitrary arrival model* (or UAM) [5]. UAM embodies a “stronger” adversary than many traditional arrival models, and subsumes traditional models as special cases. We consider activities subject to TUF time constraints, arriving according to UAM. To better account for uncertainties in execution behaviors, we stochastically describe activity execution demands. We adopt Martin’s system-level energy consumption model [11] that accounts for each system component’s energy consumption and aggregates them to obtain the system’s energy consumption.

For such a model, our objective is to: (1) probabilistically satisfy lower bounds on individual activity’s maximal utility; and (2) maximize system-level energy efficiency.

This problem is \mathcal{NP} -hard. We present a polynomial-time, heuristic algorithm for the problem called *Energy-efficient Utility Accrual Algorithm** (or EUA*). We establish that EUA*’s timeliness behavior subsumes the optimal timeliness behavior of deadline scheduling as a special case. Further, we identify the conditions under which EUA* provides timeliness assurances. Our simulation studies confirm

the effectiveness and superior performance of EUA*.

Thus, the paper’s contribution is the EUA* algorithm. We are not aware of any other efforts that consider the problem of *TUF/UA scheduling under UAM*, solved by EUA*.

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

2. Models and Objective

2.1. System and Task Model

We consider a preemptive 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 $\{f_1, \dots, f_m \mid f_1 < \dots < f_m\}$. 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 and Statistical Requirement

A job’s time constraint is specified using a TUF. Jobs of a task have the same TUF. We use $U_i(\cdot)$ to denote task T_i ’s TUF, and use $U_{i,j}(\cdot)$ to denote the TUF of T_i ’s j^{th} job. 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.

Similar to that in [17], the *statistical timeliness 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 take the value 0 or 1.

During some situations, it is possible that such statistical assurances cannot be provided; then the objective is to maximize the total utility per unit energy consumption.

2.3. Activity Cycle Demands

Both UA scheduling and DVS depend on the prediction of task cycle demands. Similar to [16] and [17], 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 $e_i = \frac{E(Y_i)}{f}$.

2.4. 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 [11, 15, 17]). 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 a 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 [11].

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

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

2.5. Scheduling Objective

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's "energy efficiency." When it is not possible to satisfy $\{\nu_i, \rho_i\}$ for each task, our objective is to maximize the system-level energy efficiency.

Intuitively, when the system is overloaded, DVS tends to select the highest frequency f_m for the processor execution. Therefore, during overloads, with the constant energy consumption at f_m , the scheduling objective becomes utility maximization under energy constraints. During underloads, the algorithm delivers the performance assurances. Thus, the objective becomes the dual criterion problem of utility maximization, that is, minimizing energy while achieving the given utility within the given time constraint.

3. The EUA* Algorithm

3.1. Determining Task Critical Time and Demand

For non-increasing TUFs, to satisfy the designated ν_i on accrued utility, we should bound task T_i 's sojourn time to less than its "critical time" (D_i). With $E(Y_i)$ and $Var(Y_i)$, by the Chebyshev's inequality, we can derive the minimal required cycles c_i to allocate to each job of T_i , so that $Pr[Y_i < c_i] \geq \rho_i$. These are addressed in our prior work [16, 17], where D_i is calculated from $\nu_i = \frac{U_i(D_i)}{U_i^{max}}$, and $c_i = E(Y_i) + \sqrt{\frac{\rho_i \times Var(Y_i)}{1 - \rho_i}}$.

3.2. Utility Accrual Scheduling

We define a performance metric, *Utility and Energy Ratio* (UER) 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 (not necessarily the lowest one) for clock frequency that maximizes T_i 's UER.

The scheduling events of EUA* include the arrival and completion of a job, and the expiration of a time constraint such as the arrival of a TUF's termination time. We define the following variables and auxiliary functions for EUA*:

- During a time window P_i , D_i^a is task T_i 's earliest invocation's absolute critical time; c_i^r is its earliest job's remaining computation cycles.
- $\mathcal{J}_r = \{J_1, J_2, \dots, J_{n'}\}$ is the current unscheduled job set; σ is the ordered schedule. $J_k \in \mathcal{J}_r$ is a job.
- $J_k.D$ is job J_k 's critical time; $J_k.X$ is its termination time; $J_k.c$ 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 $x \leq f_i$.
- $offlineComputing()$ is computed at $t = 0$. For task T_i , it computes c_i and D_i as described in Section 3.1,

and determines its optimal frequency $f_{T_i}^o \in \{f_1, \dots, f_m\}$, which maximizes the task's UER.

- `insert(T, σ, 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(σ)` 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.

```

1: input   :  $\mathbf{T} = \{T_1, \dots, T_n\}, \mathcal{J}_r = \{J_1, \dots, J_{n'}\}$ 
2: output  : selected job  $J_{exe}$  and frequency  $f_{exe}$ 
3: offlineComputing( $\mathbf{T}$ );
4: Initialization:  $t := t_{cur}, \sigma := \emptyset$ ;
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 abort( $J_k$ );
11:  else  $J_k.UER := U_{J_k}(t + \frac{J_k.c}{f_m}) / (E(f_m) \times J_k.c)$ ;
12:  $\sigma_{tmp} := \text{sortByUER}(\mathcal{J}_r)$ ;
13: for  $\forall J_k \in \sigma_{tmp}$  from head to tail do
14:  if  $J_k.UER > 0$  then
15:   copy  $\sigma$  into  $\sigma_{tent}$ ;  $\sigma_{tent} := \sigma$ ;
16:   insert( $J_k, \sigma_{tent}, J_k.D$ );
17:   if feasible( $\sigma_{tent}$ ) then  $\sigma := \sigma_{tent}$ ;
18:  else break;
19:  $J_{exe} := \text{headOf}(\sigma)$ ;
20:  $f_{exe} := \text{decideFreq}(\mathbf{T}, J_{exe}, t)$ ;
21: return  $J_{exe}$  and  $f_{exe}$ ;

```

Algorithm 1: EUA*: High Level Description

A high level description of EUA* is shown in Algorithm 1. When EUA* is invoked at time t_{cur} , the algorithm first updates each task's remaining cycles (the `switch` starting from line 5). 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 11).

The jobs are then sorted by their UERs (line 12). In each step of the `for` loop from line 13 to 18, the job with the largest UER is inserted into σ , if it can produce a positive UER and keep the schedule after insertion feasible. Thus, σ is a feasible schedule sorted by the jobs' critical times, in a non-decreasing order.

Finally, EUA* analyzes the demands of the task set and applies DVS to decide the frequency f_{exe} with algorithm `decideFreq()`. The selected job J_{exe} at the head of σ is executed with the frequency f_{exe} (line 19–21).

3.3. DVS with the UAM model

We consider the “processor demand approach” [3] to analyze the feasibility of tasks with stochastic parameters.

Theorem 1 *For a task T_i with a UAM 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 is the total cycles of a_i jobs in the time window P_i , i.e., $C_i = a_i c_i$.*

Proof The necessary and sufficient condition for satisfying job critical times is $fL \geq C_i(0, L), \forall L > 0$, where f is the processor frequency allocated to T_i , and

$C_i(0, L)$ is the cycle demand on the time interval $[0, L]$, i.e., $C_i(0, L) = \left(\left\lfloor \frac{L - D_i}{P_i} \right\rfloor + 1 \right) C_i$. Thus, we need $f \geq \frac{1}{L} \left(\left\lfloor \frac{L - D_i}{P_i} \right\rfloor + 1 \right) C_i \geq \frac{C_i}{P_i} \left(1 + \frac{P_i - D_i}{L} \right), \forall L > 0$. Since $P_i \geq D_i$, f monotonically decreases with L . Furthermore, notice that if $L \leq D_i$, $C_i(0, L) = 0$ because no job has a critical time earlier than D_i . Thus, it is sufficient to consider the case of $L = D_i$, where $f \geq C_i/D_i$. \square

```

1: input:  $\mathbf{T}, J_{exe}, t_{cur}$ ; output:  $f_{exe}$ ;
2:  $Util := C_1/D_1 + \dots + C_n/D_n$ ;
3:  $s := 0$ ;
4: for  $i = 1$  to  $n, T_i \in \{T_1, \dots, T_n, D_1^a \geq \dots \geq D_n^a\}$  do
   /* reverse EDF order of tasks */
5:    $Util := Util - C_i/D_i$ ;
6:    $x := \max(0, C_i^r - (f_m - Util) \times (D_i^a - D_n^a))$ ;
    $x := \begin{cases} 1, & \text{if } D_i^a - D_n^a = 0 \\ Util + \frac{C_i^r - x}{D_i^a - D_n^a}, & \text{otherwise} \end{cases}$ ;
7:    $s := s + x$ ;
8:    $f := \min(f_m, s / (D_n^a - t_{cur}))$ ;
9:  $f_{exe} := \text{selectFreq}(f)$ ;
10:  $f_{exe} := \max(f_{exe}, f_{T(J_{exe})}^o)$ ;

```

Algorithm 2: `decideFreq()`

Algorithm 2 shows `decideFreq()`, the stochastic DVS technique of EUA*. Based on Theorem 1, we use C_i for utilization analysis. For the current time window P_i with a_i' instances, EUA* keeps track of the remaining computation cycles C_i^r , which 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. But we only need to consider at most a_i instances of them. In line 2–9, EUA* considers the interval until the next task critical time and tries to “push” as much work as possible beyond the critical time. Similar to LaEDF [13], the algorithm considers the tasks in the latest-critical-time-first order in line 4. If task T_i has more than one jobs in P_i , D_i^a is set to be its earliest invocation's absolute critical time.

x counts the minimum number of cycles that a task must execute before the closest critical time, D_n^a , in order for it to complete by its own critical time (line 6), assuming worst-case aggregate CPU demand $Util$ by tasks with earlier critical times. In line 7, $Util$ is adjusted to reflect the actual demand of the task for the time after D_n^a , with the consideration of the case that jobs of different tasks have the same termination times, which can occur, especially during overloads. s is simply the sum of the x values calculated for all of the tasks, and therefore reflects the minimum number of cycles that must be executed by D_n^a in order for all tasks to meet their critical times (line 8).

Thus, `decideFreq()` capitalizes on early task completion by deferring work for future tasks in favor of scaling the current task. Also, during overloads, the required frequency may be higher than f_m , and `selectFreq()` would fail to return a value. In line 9, 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 $f_{T(J_{exe})}^o$.

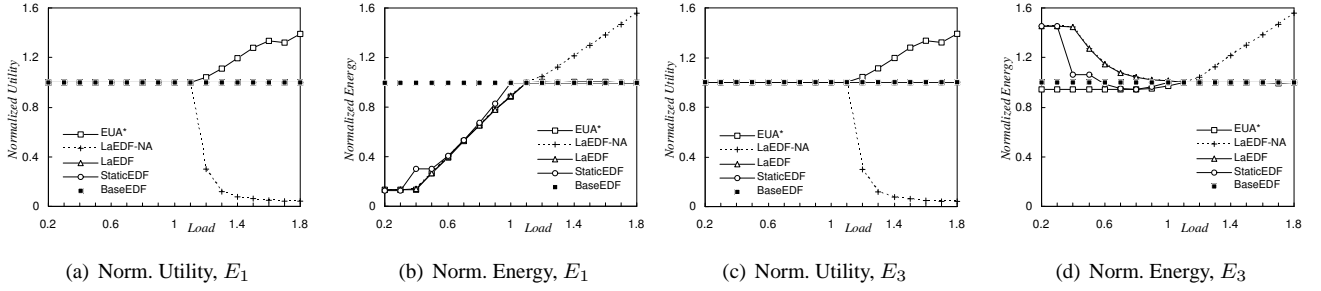


Figure 2. Normalized Energy and Utility vs. Load under E_1 and E_3

The higher frequency is selected—to provide performance assurances, we cannot decrease f_{exe} , but may increase it to maximize the system-level energy efficiency.

4. EUA*’s Timeliness Properties

The periodic model is a special case of UAM. Thus, under the conditions: (1) a set of periodic tasks with $\langle \bar{1}, P_i \rangle$ subject to downward step TUFs; and (2) absence of CPU overloads (i.e., the conditions in [9]), we establish:

Theorem 2 Under conditions (1) and (2), a schedule produced by EDF [6] is also produced by EUA*, yielding equal total utilities. This is a critical time ordered schedule.

Corollary 3 Under conditions (1) and (2), EUA* always meets all task critical times.

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

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

The proofs can be found in [17]. In Theorem 6, we also derive the above theorems’ counterparts for non-step and non-increasing TUFs, with which critical times are less than termination times. The proof for it can be found in [3].

Theorem 6 For a set of independent periodic tasks, where each task has a single computational thread with a non-increasing TUF, the task set is schedulable and can meet all statistical performance requirements under the condition of Baruah, Rosier, and Howell [3].

5. Experimental Results

We simulate EUA* on the AMD k6 processor with PowerNow! mechanism [1], which operates at seven frequencies, $\{360, 550, 640, 730, 820, 910, 1000 \text{ MHz}\}$. For comparison, we consider StaticEDF, LaEDF [13], and LaEDF-NA. While StaticEDF and LaEDF abort infeasible tasks during overloads, LaEDF-NA is LaEDF without abortion. We normalize the results to BaseEDF, which is EDF that always uses the highest frequency.

We select task sets with 10 to 50 tasks in three applications, whose 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 3.1, $E(Y_i)$ 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 ($Load = \frac{1}{f_m} \sum_{i=1}^n \frac{C_i}{D_i}$) 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.

Table 1: Task Settings

App.	# tasks	UAM $\langle a, P \rangle$
A_1	4	(5, 22–28)
A_2	18	(8, 50–70)
A_3	8	(3, 2.4–9.6)

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	$0.5f_m^3$

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 [11, 15]. We test three energy settings similar to those in [17], as shown in Table 2. Note that E_1 is the same as the conventional energy model, which only considers the CPU’s energy consumption.

5.1. Performance with Step TUFs

We first focus on step TUFs, for which EUA* can be compared with the other strategies. We set $\{\nu_i = 1, \rho_i = 0.96\}$, and apply different schemes on periodic task sets under different energy settings. The other strategies, e.g., LaEDF, are based on the worst case workload; here we use cycles allocated by EUA* as their inputs.

Figure 2 shows the normalized utility and energy under energy setting E_1 and E_3 , as Load varies from 0.2 to 1.8. From Figure 2(b) and 2(d), we observe that EUA* saves more energy than others during under-loads. LaEDF-NA’s energy consumption increases linearly with Load, because it does not abort jobs and executes all jobs that arrive. During overloads, the results of all schemes except LaEDF-NA converge to 1, because they select the highest frequency.

As Figure 2(a) and 2(c) show, during under-loaded situations, all schemes accrue the same (optimal) utility because of EDF’s optimality in such cases [6]. But during

overloads, LaEDF-NA suffers domino effects [10] and accrues almost no utility. On the other hand, EUA^* schedules jobs with higher UERs, and thus accrues remarkably higher utility than others. Results under E_2 are similar.

Thus, the performance gap demonstrates that EUA^* accrues higher utility during overloads, and handles the dual criterion problem during under-loads—saving energy while achieving the given utility within the given time constraint.

5.2. Performance with Non-Increasing TUFs

We now consider non-step and non-increasing TUFs with EUA^* , and study the impact of UAM model on the energy consumption. We allocate a linear TUF to each task, and its slope is calculated as $-\frac{U_i^{max}}{P}$, where P is 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.

We change the parameter a_i in the UAM model $\langle a_i, P_i \rangle$ for each task from 1 to 3, and run EUA^* on the task set. Figure 3 shows the energy consumption of EUA^* under different system loads. The energy consumption is normalized to the results of EUA^* without DVS, which always selects f_m .

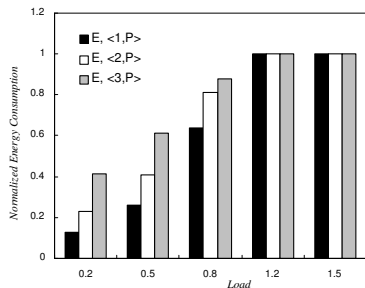


Figure 3. Energy Consumption of Different UAM Settings

We observe that, during overloads, the energy consumption does not change with a_i , because EUA^* tends to select the highest frequency. However, during under-loads, as a_i increases, EUA^* 's energy consumption increases, even under the same *Load*. For example, when *Load* = 0.5, the normalized energy consumption of $\langle 1, P \rangle$ is 0.26; that of $\langle 2, P \rangle$ is 0.41, and with $\langle 3, P \rangle$, this number increases to 0.61. This is because DVS is dependent on the prediction of future workload and slack time estimation. When a_i increases, more complicated job arrivals negatively affect the accurate estimation of slack times.

6. Conclusions, Future Work

We present an energy-efficient, UA scheduling algorithm called EUA^* , which allows activities to arrive according to the UAM model. UAM embodies a “stronger” adversary than most arrival models. The algorithm considers TUF time constraints and the scheduling objective of probabilistically satisfying utility lower bounds,

and minimizing system-level energy consumption (during under-loads). During overloads, EUA^* considers the dual criterion problem, utility maximization under energy constraints. EUA^* allocates cycles, scales CPU frequency, and computes schedules using statistical estimates of cycle demands. We establish the conditions under which EUA^* 's timeliness assurances hold. Our simulation experiments confirm EUA^* 's timeliness behavior and improvement on system-level energy efficiency.

Future work includes scheduling under finite energy budgets, and considering activity models where activities accrue utility as a function of their progress.

References

- [1] Advanced Micro Devices Corporation. Mobile AMD-K6-2+ Processor Data Sheet. Publication #23446, June 2000.
- [2] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems. In *IEEE RTSS*, pages 95–105, December 2001.
- [3] 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, Nov. 1990.
- [4] R. Clark, E. D. Jensen, and et al. An Adaptive, Distributed Airborne Tracking System. In *IEEE WPDRTS*, volume 1586 of *LNCSS*, pages 353–362. Springer-Verlag, April 1999.
- [5] J.-F. Hermant and G. L. Lann. A Protocol and Correctness Proofs for Real-Time High-Performance Broadcast Networks. In *The 18th ICDCS*, pages 360–369, 1998.
- [6] W. Horn. Some Simple Scheduling Algorithms. *Naval Research Logistics Quarterly*, 21:177–185, 1974.
- [7] E. D. Jensen, C. D. Locke, and H. Tokuda. A Time-Driven Scheduling Model for Real-Time Systems. In *IEEE RTSS*, pages 112–122, Dec. 1985.
- [8] W. Kim, J. Kim, and S. L. Min. Dynamic Voltage Scaling Algorithm for Fixed-Priority Real-Time Systems Using Work-Demand Analysis. In *ACM/IEEE ISLPED*, August 2003.
- [9] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *JACM*, 20(1):46–61, 1973.
- [10] C. D. Locke. *Best-Effort Decision Making for Real-Time Scheduling*. PhD thesis, Carnegie Mellon University, 1986. CMU-CS-86-134.
- [11] T. Martin. *Balancing Batteries, Power and Performance: System Issues in CPU Speed-Setting for Mobile Computing*. PhD thesis, Carnegie Mellon University, August 1999.
- [12] D. P. Maynard, S. E. Shipman, et al. An Example Real-Time Command, Control, and Battle Management Application for Alpha. Technical Report Archons Project TR-88121, CMU CS Dept., Dec. 1988.
- [13] P. Pillai and K. G. Shin. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *ACM SOSP*, pages 89–102, 2001.
- [14] C. Rusu, R. Melhem, and D. Mosse. Multi-Version Scheduling in Rechargeable Energy-aware Real-time Systems. In *Euromicro Conference on Real-Time Systems*, pages 95–104, July 2003.
- [15] J. Wang, B. Ravindran, and T. Martin. A Power Aware Best-Effort Real-Time Task Scheduling Algorithm. In *IEEE WSTFES/ISORC Workshop*, pages 21–28, May 2003.
- [16] H. Wu, B. Ravindran, E. D. Jensen, and P. Li. CPU Scheduling for Statistically-Assured Real-Time Performance and Improved Energy Efficiency. In *IEEE/ACM CODES+ISSS*, pages 110–115, Sept. 2004.
- [17] H. Wu, B. Ravindran, E. D. Jensen, and P. Li. Energy-Efficient, Utility Accrual Scheduling under Resource Constraints for Mobile Embedded Systems. In *ACM EMSOFT'04*, pages 64–73, Sept. 2004.
- [18] W. Yuan and K. Nahrstedt. Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems. In *ACM SOSP*, pages 149–163, 2003.