

CPU Scheduling for Statistically-Assured Real-Time Performance and Improved Energy Efficiency

Haisang Wu*, Binoy Ravindran*, E. Douglas Jensen†, and Peng Li*

*ECE Dept., Virginia Tech
Blacksburg, VA 24061, USA
{hswu02,binoy,peili2}@vt.edu

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

ABSTRACT

We present a CPU scheduling algorithm, called *Energy-efficient Utility Accrual Algorithm* (or EUA), for battery-powered, embedded real-time systems. We consider an embedded software application model where repeatedly occurring application activities are subject to deadline constraints specified using step time/utility functions. For battery-powered embedded systems, system-level energy consumption is also a primary concern. We consider CPU scheduling that (1) provides assurances on individual and collective application timeliness behaviors and (2) maximizes system-level timeliness and energy efficiency. Since the scheduling problem is intractable, EUA heuristically computes CPU schedules with a polynomial-time cost. Several properties of EUA are analytically established, including timeliness optimality during under-load situations and statistical assurances on timeliness behavior. Further, our simulation results confirm EUA's superior performance.

Categories and Subject Descriptors: D.4.7 [Operating Systems]: Organization and Design—*real-time systems and embedded systems*; D.4.1 [Operating Systems]: Process Management—*scheduling*; J.7 [Computers in Other Systems]: Real-time; C.3 [Special-Purpose and Application-Based Systems]: Real-time and embedded systems;

General Terms: Algorithms, Experimentation, Performance

Keywords: Real-time systems, time/utility functions, energy-efficient scheduling, utility accrual scheduling

1. INTRODUCTION

Energy consumption has become an important consideration in the design of embedded systems, especially in those that are mobile and battery-powered. 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.

Dynamic voltage scaling (DVS) is a common mechanism employed in the past to trade CPU's energy consumption for performance (see [2, 12, 14] and the references therein). DVS takes into account an important characteristic of CMOS-based processors: the maximum clock frequency scales almost linearly with the power supply voltage, and the energy dissipated per cycle scales quadratically to the supplied voltage [3]. A lower frequency hence enables a lower voltage and yields a quadratic energy reduction.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CODES+ISSS'04, September 8–10, 2004, Stockholm, Sweden.
Copyright 2004 ACM 1-58113-937-3/04/0009 ...\$5.00.

Saving energy without substantially affecting application performance is crucial for battery-driven, embedded real-time systems, because real-time applications must satisfy time constraints (e.g., deadlines) on activity sojourn times [2]. Real-time systems are particularly well-suited to profit from DVS, since in real-time applications, the peak computing rate needed is usually much higher than the average throughput that must be sustained. Thus, through DVS, it is generally possible to obtain quadratic energy savings at the expense of roughly linearly increased sojourn time [6].

In this paper, we consider battery-powered, embedded real-time systems that operate in environments with dynamically uncertain properties. These uncertainties include transient and sustained overloads on the CPU and other resources due to context dependent execution times. Such situations are non-deterministic, and must be characterized with stochastic or extensional (rule-based) models.

During resource overloads, meeting all deadlines of all application activities is impossible as the demand exceeds the supply. The urgency of an application activity is typically orthogonal to the relative importance of the activity—e.g., the most urgent activity can be the least important, and vice versa. Hence when overloads occur, completing activities that are more important than those which are more urgent is often desirable. Thus, a clear distinction has to be made between the urgency and the importance of activities.

Deadlines by themselves cannot express both urgency and importance. Thus, we consider the abstraction of time/utility functions (or TUFs) [8] that express the utility of completing an application activity as an application- or situation-specific function of when that activity completes. We specify the deadline constraint of an activity as a binary-valued, downward “step” shaped TUF.

Figure 1 shows example step TUFs. Note that a TUF decouples importance and urgency—i.e., urgency is measured as a deadline on the X-axis, and importance is denoted by utility on the Y-axis.

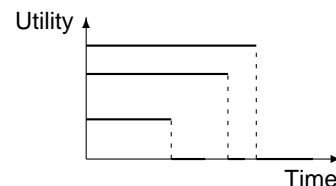


Figure 1: Example Step TUFs (that specify deadlines)

When time constraints are expressed with TUFs, the scheduling optimality criteria are based on maximizing accrued utility from those activities—e.g., maximizing the sum of the activities' attained utilities. Such criteria are called *Utility Accrual* (or UA) criteria, and sequencing (scheduling, dispatching) algorithms that consider UA criteria are called UA sequencing algorithms.

A UA algorithm that maximizes the sum of activities' attained utilities will seek to meet all activity deadlines when sufficient CPU time is available for doing so. Further, when overloads occur, such an algorithm will naturally tend to favor activities that are more

important (from whom greater utility can be accrued) than those which are more urgent.

In this paper, we consider the problem of CPU scheduling to maximize activities' attained utilities, and reduce system-level energy consumption for prolonging battery life. We consider repeatedly occurring application activities that are subject to deadlines expressed using downward step TUFs. To better account for uncertainties in activity behaviors, we consider a stochastic model, where activity execution demand is stochastically expressed. We consider a system-level energy consumption model [11], where each system component's energy consumption is individually modeled and aggregated to obtain system-level energy consumption.

For such an activity model, our objective is to: (1) provide statistical assurances on individual activity timeliness behavior including application-defined, probabilistic, lower bounds on individual activity utility; (2) provide assurances on system-level timeliness behavior including lower bound on the sum of activities' attained utilities; (3) maximize the sum of activities' attained utilities; and (4) maximize system-level energy efficiency.

This problem has not been studied in the past and is \mathcal{NP} -hard. We present a polynomial-time, heuristic algorithm for the problem called *Energy-efficient Utility Accrual Algorithm* (or EUA). We analytically establish EUA's several timeliness properties. These include: (1) timeliness optimality during under-loads and (2) assurances on timeliness behavior including lower bounds on individual activity utility and system-wide, collective utility. Further, our simulation studies confirm that EUA provides statistical assurances on timeliness behavior and improves system-level energy-efficiency.

Most of the past efforts on energy-efficient, real-time scheduling consider deadline-based optimality—i.e., meeting all or some percentage of activity deadlines [2, 6, 12, 13]. The only energy-efficient, UA scheduling effort that we are aware of is [13]. However, [13] does not provide any assurances on timeliness behavior—this is the central contribution of our work. To the best of our knowledge, EUA is the first energy-efficient, real-time scheduling algorithm that provides assurances on individual and collective timeliness behavior.

The paper is organized as follows: Section 2 describes our models and state 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 Activity Model

We consider the application to consist of a set of tasks, denoted as $\mathbf{T} = \{T_1, T_2, \dots, T_n\}$. Each task T_i has a number of instances, and these instances may be released either periodically or sporadically with a known minimal inter-arrival time. The period or minimal inter-arrival time of a task T_i is denoted as P_i . All tasks are assumed to be independent—i.e., they do not share resources or have any precedence relationships.

An instance of a task is called a *job*. 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. Jobs can be preempted at arbitrary times.

2.2 Timeliness Model

A job's time constraint is specified using a step TUF. (It is possible to have non step TUF time constraints [4, 5, 8, 10]; here, we focus on step TUFs.) Jobs of a task have the same TUF. We use $U_i(\cdot)$ to denote task T_i 's TUF. The TUF of task T_i 's j^{th} job is denoted as $U_{i,j}(\cdot)$, which has the same shape as $U_i(\cdot)$. Without being task specific, we use U_{J_k} to denote the TUF of a job J_k ; thus completion of the job J_k at a time t will yield a utility $U_{J_k}(t)$.

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 the arrival time of job $J_{i,j}$, and $X_{i,j} - I_{i,j}$ is the period or minimal inter-arrival time P_i of the task T_i . If $J_{i,j}$'s $X_{i,j}$ is reached and execution of the corresponding job has not been completed, an exception is raised. Normally, this exception will cause $J_{i,j}$'s abortion and execution of exception handlers.

2.3 Statistical Timeliness Requirement

Each task needs to accrue some percentage of its maximum possible utility. The *statistical performance requirement* of a task T_i is denoted as ρ_i , which implies that task T_i should accrue at least ρ_i percentage of its maximum possible utility. For task T_i with a step TUF, ρ_i also denotes the probability that it should meet its job termination times. For example, if $\rho_i = 0.93$, then the task T_i needs to accrue at least 93% of its maximum possible utility; i.e., it needs to meet no less than 93% of job termination times.

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 are dependent on the prediction of task cycle demands. We estimate the statistical properties (e.g., distribution, mean, variance) of the demand rather than the worst-case demand for three reasons: (1) many embedded real-time applications exhibit a large variation in their *actual* workload [4]. Thus, the statistical estimation of the demand is much more stable and hence more predictable than that of the actual workload; (2) worst-case workload is usually a very conservative prediction of the actual workload [2], resulting in resource over-supply, and exacerbates the power consumption problem; and (3) allocating cycles based on the statistical estimation of tasks' demands can provide statistical performance assurances, which is sufficient for the applications that are of interest to us.

Let Y_i be the random variable of a task T_i 's cycle demand. We assume that the mean and variance of task cycle demands are finite and determined through either online or off-line profiling. We denote the expected workload, i.e., the expected number of processor cycles required by a task T_i as $E(Y_i)$, and the variance on the workload as $Var(Y_i)$. Note that, under a constant speed i.e., 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 Energy Consumption Model

We consider Martin's system-level energy consumption model [11, 13] to derive the energy consumption per cycle. In this model, the CPU's dynamic power consumption, denoted P_d , when operating at a frequency f is given by $P_d = C_{ef} \times V_{dd}^2 \times f$, where C_{ef} is the effective switch capacitance and V_{dd} is the supply voltage. The clock frequency is almost linearly related to the supply voltage, since $f = k \times \frac{(V_{dd} - V_t)^2}{V_{dd}}$, where k is constant and V_t is the threshold voltage [13]. By approximation, $f = a \times V_{dd}$, where a is constant. Thus, $P_d = \frac{C_{ef}}{a^2} \times f^3 = S_3 \times f^3$, where S_3 is constant. Here, both the voltage and the frequency can be scaled.

Besides the CPU, there are also *other* system components that consume energy. Some system components must operate at a fixed voltage and thus their power can only scale with frequency. Examples include main memory. In this case, $C_{ef} \times V_{dd}^2$ can be represented as another constant, say, S_1 , and P_d becomes $P_d = S_1 \times f$. Other components in the system consume constant power with respect to the frequency—e.g., display devices. Thus, their power consumption can be represented as S_0 , where S_0 is constant.

For completeness in fitting the measured power of a system to the

cubic equation, another term is included to represent the quadratic term—i.e., $P_d = S_2 \times V_{dd}^2$. Since f is almost linearly related to V_{dd} , P_d is represented as $P_d = S_2 \times f^2$. While this term does not represent the dynamic power consumption of CMOS, because it implies that V_{dd} is being lowered without also lowering f , in practice, this term may appear because 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 system components together, an equation for system-level power consumption is obtained as: $P = S_3 \times f^3 + S_2 \times f^2 + S_1 \times f + S_0$. The corresponding energy consumption of a task T_i is given by: $E_i = P \times e_i$, where e_i denotes T_i 's expected execution time. Therefore, the expected energy consumption per cycle is given by:

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

2.6 Scheduling Objective

We define a system-level performance metric, called *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 *system-level* UER is defined as the ratio of the total accrued utility to the total system-level energy consumption—i.e., $UER = \frac{\sum_{i=1}^n U_i}{\sum_{i=1}^n E_i}$.

We define a two-fold scheduling criterion: (1) assure that each task T_i accrues the specified percentage ρ_i of its maximum possible utility; and (2) maximize the system-level UER. We also desire to obtain a lower bound on the total activities' attained utilities. Note that by maximizing UER, we maximize the total activities' attained utilities as well as system-level energy efficiency. As discussed previously, when it is not possible to satisfy ρ_i for each task, our objective is to maximize the system-level UER.

This problem is \mathcal{NP} -hard because it subsumes the \mathcal{NP} -hard problem of scheduling dependent tasks with step TUFs [5].

3. THE EUA ALGORITHM

3.1 Statistical Estimation of Demand

EUA first needs to decide the number of cycles that must be allocated to each task. To provide statistical timeliness assurances while maximizing energy efficiency, EUA allocates cycles based on the statistical requirements and demand of each task. Knowing the mean and variance of task T_i 's demand Y_i , by a one-tailed version of the Chebyshev's inequality, when $y \geq E(Y_i)$, we have:

$$Pr[Y_i < y] \geq \frac{(y - E(Y_i))^2}{Var(Y_i) + (y - E(Y_i))^2} \quad (2)$$

Equation 2 is the direct result of the cumulative distribution function of the task T_i 's cycle demands. Now, let ρ_i be the statistical performance requirement of T_i i.e., each job $J_{i,j}$ of task T_i must accrue ρ_i percentage of utility. To satisfy this requirement, we let $\rho_i = \frac{(C_i - E(Y_i))^2}{Var(Y_i) + (C_i - E(Y_i))^2}$ 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., $Pr[Y_i < C_i] \geq \rho_i$.

3.2 UA Scheduling with DVS

The parameter C_i determines *how long* (in number of cycles) to execute each task. EUA then needs to determine *when* and *how fast* (i.e., CPU speed scaling) to execute each task.

We suppose that there are n tasks and each task is allocated C_i cycles within its P_i . Assuming that each task presents its worst-case workload to the processor at every instance, the static, optimal CPU speed to minimize the total energy while meeting all the P_i under the traditional energy consumption model is given by the aggregate CPU demand of the concurrent tasks [2]:

$$\sum_{i=1}^n \frac{C_i}{P_i} \quad (3)$$

million cycles per second (MHz). This speed can be assigned to execute all tasks until the task set changes.

However, the cycle demands of tasks often vary greatly. In particular, a task may, and often does, complete a job before using up its allocated cycles. Consequently, dynamically monitoring or anticipating such early computations, and adjusting the CPU speed can be a powerful approach to obtain energy savings.

We consider the system's energy consumption as given by Equation 1. The equation implies that there is an optimal value (not necessarily the lowest) for clock frequency that minimizes E_i for a task T_i . We assume that the CPU can be operated at m frequencies $\{f_1, \dots, f_m \mid f_1 < \dots < f_m\}$.

EUA first decides the optimal frequency for each task T_i that maximizes the task's local UER. At each scheduling event, for the n' jobs $\mathcal{J}_r = \{J_1, J_2, \dots, J_{n'}\}$ currently in the ready queue, the algorithm sorts them based on their UERs in a non-increasing order. EUA then inserts the jobs into a tentative schedule in the order of their termination times (earliest termination time first).

We define the *system load* (Load) as:

$$Load = \frac{1}{f_m} \sum_{i=1}^n \frac{C_i}{P_i} \quad (4)$$

If the system is overloaded, it is possible that the queue \mathcal{J}_r , whose *queue load* is defined as $\frac{1}{f_m} \sum_{k=1}^{n'} (C_{J_k} / (J_k.X - t_{cur}))$, is also overloaded. Note that $J_k.X$ refers to the termination time of J_k . Thus, upon inserting a job, EUA performs feasibility check, and ensures the feasibility of the tentative schedule by dropping some jobs; that is, the predicted completion time of each job left in the tentative schedule never exceeds its termination time.

To calculate a CPU frequency for the currently selected job i.e., the one at the head of the tentative schedule, we adopt a stochastic DVS technique similar to the Look-Ahead EDF (LaEDF) technique discussed in [12]. The calculated value is compared with the job's local optimal frequency, and the higher one is selected as the CPU frequency. This process is elaborated later.

Intuitively, during overloads it is very possible for the DVS technique to select the highest frequency f_m for the execution of the processor, since the aggregate CPU demand defined in Equation 3 is higher than f_m . Therefore, during overloads, with the constant energy consumption at frequency f_m , to maximize the collective utility per unit energy as our objective, we need to maximize the collective utility. This is exactly why we sort the jobs based on their UERs and perform the feasibility check. Such heuristics are explained in detail in the next section.

3.3 Algorithm Description

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 the termination time of a TUF. To describe EUA, we define the following variables and auxiliary functions:

- \mathbf{T} is the task set. X_i is task T_i 's current invocation's termination time; C_i^r denotes its current job's remaining computation cycles.
- \mathcal{J}_r is the current unscheduled job set; σ is the ordered schedule. $J_k \in \mathcal{J}_r$ is a job. $J_k.X$ is job J_k 's termination time; $J_k.C$ is its remaining cycle.

- $T(J_k)$ returns the corresponding task of job J_k . Thus, if $T_i = T(J_k)$, then $J_k.C = C_i^r$, and $J_k.X = X_i$.
- $\text{headOf}(\sigma)$ returns the first job in σ ; $\text{sortByUER}(\sigma)$ sorts σ by each job's UER. $\text{selectFreq}(x)$ returns the lowest frequency $f_i \in \{f_1, \dots, f_m \mid f_1 < \dots < f_m\}$: $x \leq f_i$.
- $\text{offlineComputing}()$ is computed at $t = 0$. For task T_i , it computes C_i as $C_i = E(Y_i) + \sqrt{\frac{\rho_i \times \text{Var}(Y_i)}{1 - \rho_i}}$, and determines its optimal frequency $f_{T_i}^o \in \{f_1, \dots, f_m\}$, which maximizes the task's UER. T_i 's UER is defined as $U_i(t + \frac{C_i}{f}) / (C_i \times E(f))$, where $E(f)$ is derived using Equation 1.
- $\text{Insert}(T, \sigma, I)$ 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.
- $\text{feasible}(\sigma)$ 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.

Algorithm 1: EUA: High Level Description

```

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:  $\text{offlineComputing}(\mathbf{T})$ ;
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  $\text{feasible}(J_k) = \text{false}$  then
11:    abort( $J_k$ );
12:  else  $J_k.UER := U_{J_k}(t + \frac{J_k.C}{f_m}) / (E(f_m) \times J_k.C)$ ;
13:  $\sigma_{tmp} := \text{sortByUER}(\mathcal{J}_r)$ ;
14: for  $\forall J_k \in \sigma_{tmp}$  from head to tail do
15:  if  $J_k.UER > 0$  then
16:    copy  $\sigma$  into  $\sigma_{tent}$ :  $\sigma_{tent} := \sigma$ ;
17:    Insert( $J_k, \sigma_{tent}, J_k.X$ );
18:    if  $\text{feasible}(\sigma_{tent})$  then
19:       $\sigma := \sigma_{tent}$ ;
20:  else break;
21:  $J_{exe} := \text{headOf}(\sigma)$ ;
22:  $f_{exe} := \text{decideFreq}(\mathbf{T}, J_{exe}, t)$ ;
23: return  $J_{exe}$  and  $f_{exe}$ ;

```

A description of EUA at a high level of abstraction is shown in Algorithm 1. When EUA is invoked at time t_{cur} , the algorithm first updates each task's remaining cycle (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 11). Otherwise, its UER is calculated (line 12).

The jobs are then sorted by their UERs (line 13). In each step of the `for` loop from line 14 to 20, 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' termination times.

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 21–23).

Algorithm 2 shows `decideFreq()`, the stochastic DVS technique adopted by EUA, which is similar to LaEDF [12].

In Algorithm 2, EUA keeps track of the remaining computation cycles C_i^r , as updated from line 5 to line 8 of Algorithm 1. From line 2 to line 10, the algorithm considers the interval until the next task termination time and tries to “push” as much work as possible beyond the termination time. The algorithm considers the tasks in the latest-termination-time-first order in line 4.

x is the minimum number of cycles that the task must execute before the closest termination time, X_n , in order for it to complete

Algorithm 2: DecideFreq()

```

1: input:  $\mathbf{T}, J_{exe}, t_{cur}$ ; output:  $f_{exe}$ ;
2:  $Util := C_1/P_1 + \dots + C_n/P_n$ ;
3:  $s := 0$ ;
4: for  $i = 1$  to  $n, T_i \in \{T_1, \dots, T_n \mid X_1 \geq \dots \geq X_n\}$  do
5:   /* reverse EDF order of tasks */;
6:    $Util := Util - C_i/P_i$ ;
7:    $x := \max(0, C_i^r - (f_m - Util) \times (X_i - X_n))$ ;
8:    $Util := \begin{cases} 1, & \text{if } X_i - X_n = 0 \\ Util + \frac{C_i^r - x}{X_i - X_n}, & \text{otherwise} \end{cases}$ ;
9:    $s := s + x$ ;
10:  $f := \min(f_m, s / (X_n - t_{cur}))$ ;
11:  $f_{exe} := \text{selectFreq}(f)$ ;
12:  $f_{exe} := \max(f_{exe}, f_{T(J_{exe})}^o)$ ;

```

by its own termination time (line 7), assuming worst-case aggregate CPU demand $Util$ by tasks with earlier termination times. The aggregate demand $Util$ is adjusted to reflect the actual demand of the task for the time after X_n (line 8). 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 X_n in order for all tasks to meet their termination times (line 9). In line 10, the operating CPU frequency is set just fast enough to execute s cycles over this interval.

Thus, `decideFreq()` capitalizes on early task completion by deferring work for future tasks in favor of scaling the current task. Further, in line 8 we consider the case that jobs of different tasks have the same termination times, which can occur, especially during overloads. Also, during overloads, the required frequency may be higher than f_m , and `selectFreq()` would fail to return a value. In line 10, we solve this by setting the upper limit of the required frequency to be the highest frequency f_m .

Finally, the result of `selectFreq()` is compared with the optimal frequency of $T(J_{exe})$ decided in `offlineComputing()` (line 12). The higher frequency is selected to preserve the statistical performance assurance and maximize system-level UER.

4. TIMELINESS PROPERTIES

We establish EUA's timeliness properties under conditions: (1) a set of independent periodic tasks; and (2) absence of CPU overloads—i.e., sufficient cycles exist for meeting all task termination times.

THEOREM 1. *Under conditions (1) and (2), a schedule produced by EDF [7] is also produced by EUA, yielding equal total utilities. This is a termination time-ordered schedule.*

Proof We prove this by examining Algorithms 1. For periodic tasks with step TUFs during non-overload situations, σ from line 19 of Algorithm 1 is termination time ordered. The TUF termination time that we consider is analogous to a deadline in [7]. As proved in [7, 9], 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 EDF. \square

Some important corollaries about EUA's timeliness behavior during under-loads can be deduced from EDF's optimality [7].

COROLLARY 2. *Under conditions (1) and (2), EUA always meets all task termination times.*

COROLLARY 3. *Under conditions (1) and (2), EUA minimizes the maximum lateness.*

From Corollary 2, we can derive the properties of EUA on performance assurances. With known height for each task's TUF, we can also derive the system-level utility assurance.

THEOREM 4. *Under conditions (1) and (2), EUA meets the statistical performance requirements.*

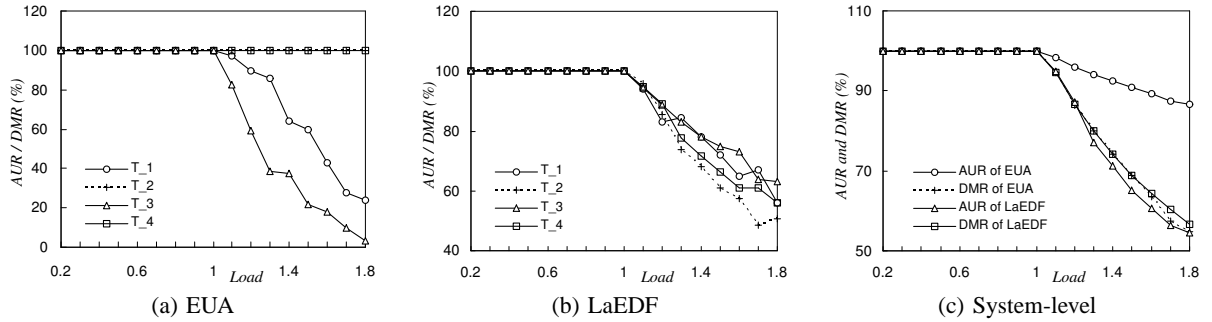


Figure 2: AUR and DMR vs. *Load* of G under E_1

Proof According to Equation 2, at least ρ_i demanded processor cycles of task T_i are less than the allocated cycles. From Corollary 2, under conditions (1) and (2), EUA can meet all task termination times; i.e., all the allocated cycles can be completed before their termination times. Thus, for task T_i , EUA can meet at least ρ_i termination times and accrue at least ρ_i utility. \square

THEOREM 5. *Under conditions (1) and (2), if a task T_i 's TUF has a height h_i , then the system-level utility ratio, defined as the utility accrued by EUA with respect to the system's maximum possible utility, is at least $\frac{\sum_{i=1}^n \rho_i h_i}{\sum_{i=1}^n h_i}$.*

Proof We denote the number of jobs released by task T_i as m_i . The system-level accrued utility to the system's maximum possible utility is $\frac{\rho_1 h_1 m_1 + \dots + \rho_n h_n m_n}{h_1 m_1 + \dots + h_n m_n}$. Thus, when m_i ($i = 1, \dots, n$) approaches $+\infty$, this formula becomes $\frac{\sum_{i=1}^n \rho_i h_i}{\sum_{i=1}^n h_i}$. \square

5. EXPERIMENTAL RESULTS

5.1 Experimental Settings

We performed extensive simulations to experimentally evaluate EUA's performance. The task settings in our simulation study are summarized in Table 1. Table 1 shows task period/minimum inter-arrival times (or P/I.A.) and maximum utility (or U_{max}). For each demand Y_i , we keep $Var(Y_i) \approx E(Y_i)$, and generate normally-distributed cycle demands. We change the tasks' cycle demands to change the system load (*Load* of Equation 4).

Table 1: Task Settings

Task	Jobs	P/I.A.	U_{max}
T_1	154	25	10
T_2	138	28	60
T_3	160	24	10
T_4	148	26	60

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	$0.25 f_m^3$
E_3	0.5	0	0	$0.5 f_m^3$

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, 13].

We use experimental settings similar to that in [11], but de-normalize the terms. For comparison, the experiments are carried out under three energy model settings, 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.

We consider a CPU with seven different frequencies including {360, 550, 640, 730, 820, 910, 1000 MHz}. These frequencies reflect the setting that is available on a platform incorporating an AMD k6 processor with AMD's PowerNow! mechanism [1].

For comparison, we consider past energy-efficient, real-time scheduling algorithms including BaseEDF, LaEDF, StaticEDF, and LaEDF-NA. BaseEDF is EDF without DVS support and uses the highest

frequency. LaEDF is the Look-ahead RT-DVS for EDF [12]. StaticEDF uses the constant speed given by Equation 3 and a "ceiling" up to the lowest suitable frequency in $\{f_1, f_2, \dots, f_m\}$. StaticEDF switches to the lowest frequency whenever there is no ready task. Combining the static schemes in [2] and [12], StaticEDF is the static optimal solution to the DVS problem for the periodic task model with step TUFs under the available frequency set. The previous three schemes abort infeasible tasks during overloads. Thus, LaEDF-NA is LaEDF with no abortion.

LaEDF, LaEDF-NA, and StaticEDF perform DVS on periodic tasks with known worst-case workload, which is unavailable in our application model. Thus, we use the minimum inter-arrival time and cycles allocated by EUA as their inputs.

5.2 Performance Assurances

Our first set of experiments evaluates EUA's statistical performance assurances. Because no strategies except EUA consider the system-level energy consumption, we only use the energy model E_1 in the simulation experiments of this section. We consider the task set G with the performance requirement of $(\rho_1, \rho_2, \rho_3, \rho_4) = (0.93, 0.92, 0.95, 0.90)$.

Figure 2(a) and 2(b) show the accrued utility ratio (AUR) and termination-time meet ratio (DMR) of each task under increasing *Load*, by EUA and LaEDF, respectively. AUR is the ratio of accrued aggregate utility to the maximum possible utility, and DMR is the ratio of the jobs meeting their termination times to the total job releases of a task. For a task with a step TUF, its AUR and DMR are identical. But the system-level AUR and DMR can be different due to the mix of different utility of tasks.

As Figure 2(a) shows, with EUA during under-loads, all tasks accrue 100% AUR and DMR, except task T_1 , whose AUR and DMR is 99.57% at *Load* = 0.5. Thus, EUA delivers the statistical assurance of being able to accrue the required percentage of task-level maximum utility for all tasks. This also validates Theorem 4.

Comparing the results during overloads in Figure 2(a) and 2(b), we observe that EUA still achieves near 100% AUR/DMR of task T_2 and T_4 , but achieves less AUR/DMR of T_1 and T_3 . On the other hand, LaEDF decreases the AUR/DMR of T_2 and T_4 more than the other two. This is because, T_2 and T_4 have TUFs with higher "heights" and thus higher utility; so EUA accrues more system-wide utility by completing these tasks before their termination times. Schemes based on EDF cannot make such scheduling decisions— T_2 and T_4 are not favored by LaEDF, since they have longer termination times than T_1 and T_3 .

Figure 2(c) shows the system-level AUR and DMR of EUA and LaEDF under increasing *Load*. According to Theorem 5, with large number of job releases, the system-level AUR should be at least $(10 \times 0.93 + 60 \times 0.92 + 10 \times 0.95 + 60 \times 0.90) / 140 = 91.4\%$. We observe that the AUR and DMR of EUA during under-loads are above 98%. This validates Theorem 5. Further, EUA accrues much more system-wide utility during overloads than LaEDF—although

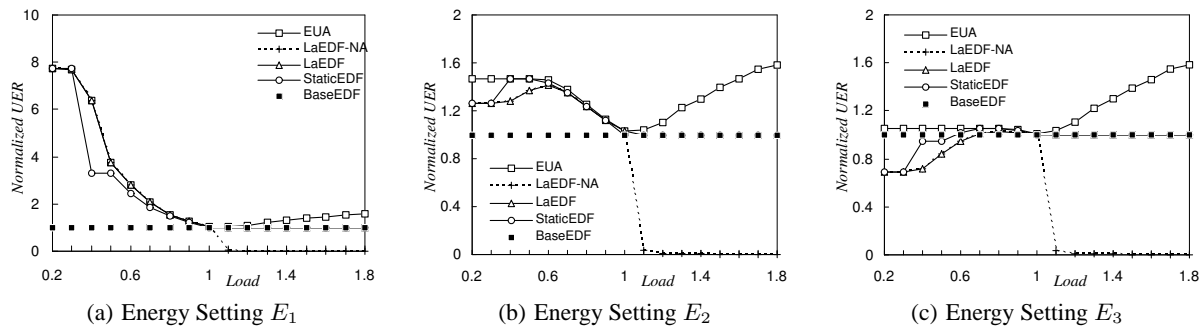


Figure 3: Normalized UER vs. Load under Various Energy Model Settings

its DMR is lower than that of LaEDF, EUA can obtain much higher AUR than LaEDF in such cases.

5.3 Impact of Energy Models

To examine the energy model's effects, we apply different schemes on the task set under different energy settings. The task set has the same statistical performance requirement as that in Section 5.2.

Figure 3 shows the UER for all the DVS schemes normalized to the BaseEDF under energy model settings E_1 , E_2 , and E_3 , as Load varies from 0.2 to 1.8. We observe that under all three energy settings, EUA performs the best among all strategies under all loads, especially during overloads. We also observe that LaEDF-NA yields almost zero UER during overloads.

As Figure 3 shows, during overloads, the normalized UERs produced by LaEDF, StaticEDF, and BaseEDF converge to 1. This is because, all three algorithms select the highest frequency by DVS calculation during overloads, and bear no difference in scheduling. As the S_0 term increases, EUA adjusts the selected frequency to accrue more UER. This effect is more pronounced under E_3 , when LaEDF, LaEDF-NA, and StaticEDF perform worse than BaseEDF, while EUA still outperforms BaseEDF during all loads.

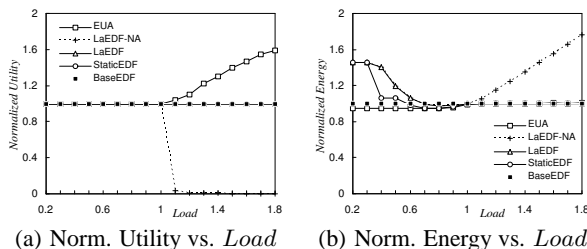


Figure 4: Normalized Energy and Utility under E_3

We speculate that, the UER gap between EUA and the other schemes is because EUA saves more energy during under-loads, and accrues higher utility during overloads. Our speculation is verified in Figure 4, which shows the accrued utility and energy consumption normalized to BaseEDF, under energy setting E_3 .

We observe in Figure 4(a) that, during under-loaded situations, all schemes accrue the same (optimal) utility because of EDF's optimality [7] in such cases. But during overloads, LaEDF-NA suffers domino effects and accrues almost no utility [10]. On the other hand, EUA schedules jobs with higher UERs, and thus accrues remarkably higher utility than the others.

From Figure 4(b), we observe that EUA saves more energy than the other schemes during under-loads. Further, this portion of the curves is nearly symmetric to the corresponding portion of Figure 3(c). LaEDF-NA's energy consumption increases linearly with Load, because it performs no abortion and executes all jobs.

6. CONCLUSIONS, FUTURE WORK

This paper presents an energy-efficient, UA real-time scheduling algorithm called EUA that considers activities subject to deadlines expressed using step TUFs and system-level energy consumption concerns. EUA's objective is to probabilistically satisfy application-specified lower bounds on activity utility, obtain a lower bound on collective utility, and to maximize collective utility and energy efficiency. EUA statistically allocates cycles to individual activities and executes their allocated cycles at different speeds with DVS.

We establish EUA's timeliness optimality during under-loads and assurances on individual and collective timeliness behavior. Our simulation experiments confirm EUA's timeliness assurances and improvement in system-level energy efficiency.

Several aspects of the work are directions for further research. Examples include considering application-specified, lower bounds on collective utility as the algorithm's input (instead of deriving that bound), allowing aperiodic tasks, considering non-step TUFs, and accounting for scheduler overhead in computing schedules.

7. 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] A. Chandrakasan, S. Sheng, and R. W. Brodersen. Low-power CMOS digital design. In *IEEE Journal of Solid-State Circuits*, volume 27, pages 473–484, April 1992.
- [4] R. Clark, E. D. Jensen, and et al. An adaptive, distributed airborne tracking system. In *IEEE WPDRTS*, volume 1586 of *LNCS*, pages 353–362. Springer-Verlag, April 1999.
- [5] R. K. Clark. *Scheduling Dependent Real-Time Activities*. PhD thesis, CMU, 1990. CMU-CS-90-155.
- [6] R. Graybill and R. Melhem. *Power Aware Computing*. Kluwer Academic/Plenum Publishers, 2002.
- [7] W. Horn. Some simple scheduling algorithms. *Naval Research Logistics Quarterly*, 21:177–185, 1974.
- [8] E. D. Jensen, C. D. Locke, and H. Tokuda. A time-driven scheduling model for real-time systems. In *IEEE RTSS*, pages 112–122, December 1985.
- [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] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *ACM SOSP*, pages 89–102, 2001.
- [13] 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.
- [14] W. Yuan and K. Nahrstedt. Energy-efficient soft real-time cpu scheduling for mobile multimedia systems. In *ACM SOSP*, pages 149–163. ACM Press, 2003.