

Energy-Aware Modeling and Scheduling for Dynamic Voltage Scaling with Statistical Real-Time Guarantee

Xiliang Zhong, *Student Member, IEEE*, and Cheng-Zhong Xu, *Senior Member, IEEE*

Abstract—Dynamic voltage scaling (DVS) is a promising technique for battery-powered systems to conserve energy consumption. Most existing DVS algorithms assume information about task periodicity or a priori knowledge about the task set to be scheduled. This paper presents an analytical model of general tasks for DVS assuming job timing information is known only after a task release. It models the voltage scaling process as a transfer function-based filtering system, which facilitates the design of two efficient scaling algorithms. The first is a time-invariant scaling policy and it is proved to be a generalization of several popular DVS algorithms for periodic, sporadic, and aperiodic tasks. A more energy efficient policy is a time-variant scaling algorithm for aperiodic tasks. It is optimal in the sense that it is online without assumed information about future task releases. The algorithm turns out to be a water-filling process with a linear time complexity. It can be applied to scheduling based on worst-case execution times as well as online slack distribution when jobs complete earlier. We further establish two relationships between computation capacity and deadline misses to provide a statistical real-time guarantee with reduced capacity.

Index Terms—Real-time systems, power-aware scheduling, dynamic power management, dynamic voltage scaling.



1 INTRODUCTION

LOW energy consumption is a key design requirement in real-time systems. This is especially important for battery-powered systems, such as cellular phones and portable medical devices, because low energy consumption extends their limited battery life. An effective approach to power reduction is based on dynamic voltage scaling (DVS), which exploits the super linear dependence of the power consumption of a CMOS processor on its operating voltage. However, a reduction of the operating frequency leads to an increase of service time. A challenge in applying DVS algorithms to delay-sensitive applications is to achieve maximum energy savings while still meeting all temporal requirements of the system. It is known that most real-time systems are designed according to their peak performance requirements. Their capacities often exceed the average throughput demand and their busy time usually accounts for only a small fraction of application running time. This makes it possible for DVS algorithms to reduce the processor frequency and still ensure no tasks miss their deadlines.

There are significant studies on DVS-based real-time scheduling for energy savings in the past decade. Most of the algorithms are targeted at periodic tasks, assuming all task timing information, including release time, execution time (at the reference processor speed), and deadline, is known in advance [1], [9], [20], [22], [29], [41]. Many practical real-time applications involve both types of

periodic and aperiodic tasks. Algorithms in support of aperiodic tasks can be found in [11], [15], [19], [24], [31], [34] with explicit deadlines under the assumption that task timing information is known offline. In contrast to periodic tasks that have regular job releases, the release times of an aperiodic task may be arbitrary with irregular intervals. The irregularity calls for online scheduling algorithms without assumed timing knowledge before job arrivals.

In addition to energy-delay trade-off, there are two other design requirements for online aperiodic task scheduling. One is to provide an algorithm with low time complexity since the speed assignment is made online. The other is to consider the impact of workload variation because the actual execution time of a task can be far less than its worst-case execution time (WCET). Existing online strategies in support of real-time aperiodic tasks [30], [10], [28], [23], [16] do not solve all of the issues completely. For example, uncontrolled occasional deadline misses are possible by using the slacked EDF [30]. The optimal periodic and sporadic task scheduling due to [10] has a high time complexity. DVSST [23] is a low complexity energy-efficient algorithm. However, its energy saving is suboptimal.

1.1 Motivational Examples

Consider a simple real-time system with three aperiodic tasks, $T_1(1, 4)$, $T_2(2, 4)$, $T_3(1, 4)$, where the pair of parameters denotes WCET at full speed and relative deadline. Consider jobs released during the first 10 time units. We assume T_1 releases jobs at times 0 and 5, T_2 at times 1 and 7, and T_3 at times 3 and 9.

Let $J_{i,j}$ denote the j th instance of task i . Fig. 1a shows the scheduling under EDF with the online algorithm DVSST due to [23]. The speed setting is energy-efficient and can achieve 38 percent energy savings compared to a schedule

• The authors are with the Department of Electrical and Computer Engineering, Wayne State University, 5050 Anthony Wayne Drive, Detroit, MI 48202. E-mail: {xlzhong, czxu}@wayne.edu.

Manuscript received 5 Oct. 2005; revised 7 May 2006; accepted 23 Aug. 2006 published online 22 Jan. 2007.

For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0345-1005. Digital Object Identifier no. XXX

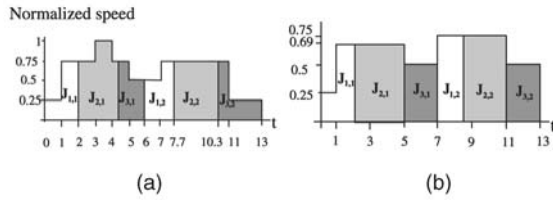


Fig. 1. Schedule solutions for a sporadic task set. (a) Schedule with DVSSST. (b) An alternative schedule.

without DVS under a quadratic energy model. However, the solution is suboptimal in comparison with an alternative schedule in Fig. 1b, which has 8 percent more energy savings. In fact, the schedule in Fig. 1b is optimal in the sense it is online without knowledge of future job releases. We will propose an efficient algorithm in deriving the optimal speed setting in Section 4.

Both schedules in Fig. 1 save energy by slowing down job executions. Due to the irregular releases of aperiodic tasks, it is possible to have deadline misses for online scheduling algorithms without assumptions about future job releases. Consider a simple example with two jobs, both with relative deadline 2. One is released at time 0 with a WCET 1, the other at time 1 with a WCET 2. Under a schedule without speed scaling, both jobs can finish before their deadlines under the maximum speed, as shown in Fig. 2a. However, an online energy-aware schedule will slow down the speed of the first job. As a result, the second job cannot finish before its deadline. If a job cannot finish before its deadline, admission control can be integrated to reject the job to ensure no deadline misses. Alternatively, it can be admitted and served in a best-effort mode. Studies using schedulability test can guarantee no deadline miss for a set of sporadic tasks (a special type of aperiodic task with a minimal interrelease time) under the worst-case scenario assuming all tasks release jobs at their maximum rates [23]. However, as the interarrival times of a sporadic task can vary widely, the worst-case scenario may seldom occur and using the schedulability test is conservative by requiring a high computation capacity or, equivalently, admitting few tasks. Instead, we propose to allow a controllable deadline miss so that the computation capacity can be significantly reduced. The basic idea is to obtain output load distribution under the condition that all jobs finish before their deadlines. We then consider the tail distribution to bound the deadline misses under a given capacity.

1.2 Paper Organization and Contributions

In this paper, we tackle the design challenges for real-time tasks using a model-based approach. Job releases (or arrivals) are represented as a random process with a general distribution. The voltage scaling process is modeled as a transfer function-based filtering system which characterizes output load as a linear combination of cycle demands of uncompleted jobs. The major contributions are:

- A generalization of DVS policies. Existing DVS algorithms were designed assuming knowledge of task periodicity and any generalization is hardly possible. Our scheduling model is based on jobs

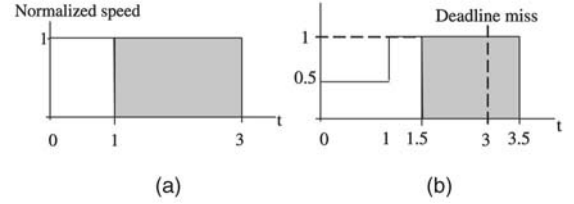


Fig. 2. Deadline miss with an energy-aware schedule. (a) Schedule without DVS. (b) An energy-aware schedule.

instead of tasks. The model facilitates the derivation of a time-invariant policy, which is proved to be a generalization of several existing algorithms with respect to energy consumption. Examples include AVR for aperiodic tasks [34], DVSSST for sporadic tasks [23], and static voltage scaling for periodic tasks [1], [22].

- An online efficient energy-efficient policy for aperiodic tasks. The policy is proved to be optimal among all online scheduling policies without assumptions about future job releases. The algorithm is a water-filling process of information theory with a linear time complexity in the number of tasks. Considering the fact that most jobs seldom execute up to their WCETs, we show the algorithm is readily applicable to online slack distribution and outperforms a recent slack management scheme for general tasks [16].
- Statistical real-time guarantee. We derive the inherent relationships between computation capacity and deadline miss rate and provide statistical deadline guarantees to reduce the computation capacity. Experimental results show that, with a small deadline miss rate of 1 percent, the maximum computation speed can be cut in half compared with the worst-case scenario.

The rest of the paper is organized as follows: Section 2 defines a model for DVS, assuming continuous voltage levels and ignorable time and energy overheads in voltage switches. Section 3 provides a generalization of existing policies. In Section 4, we propose a time-variant scheduling policy for aperiodic tasks and show its applicability in online slack distribution. Section 5 shows the effectiveness of the proposed scaling policies in energy savings by simulation. Statistical deadline guarantees due to capacity configuration are discussed in Section 6. Section 7 reviews related work. Section 8 concludes the paper.

2 SYSTEM MODEL

We consider a dynamic priority single-processor system with a set of independent, preemptive tasks. The voltage scaling process is based on a discrete time model, with t as the scheduling time index. The input is viewed as a set of jobs released in a certain sequence. The set of jobs under consideration is denoted as \mathcal{J} , which can be infinite in general. Let $n(t)$ be a subset of jobs released during time $[t-1, t)$. The request size of a job $i \in \mathcal{J}$, w_i , is the required number of CPU cycles. The relative deadline of job i is

denoted as d_i . We assume the parameters of a job are known only after its release.

The processor under consideration is assumed to support a continuous range of voltage and speed. Considering the fact that most commercially available processors only provide a limited number of voltage levels, researchers have proposed algorithms to map continuous voltage levels to discrete ones [11], [15], [2]. The approaches actually enable a processor to run at any speed. We therefore do not explore the effect of a limited number of processor speeds. We also assume the time and energy overhead during frequency scaling is negligible. This is valid in modern processors because the speed transition time is usually under 100 μs and the actual time period when the processor is unavailable in a transition is as short as 10 μs [8]. The overhead is small compared to the execution times of real-time tasks. Transition energy is the energy consumed during the transition time. We assume the energy is also ignorable. The energy function is first assumed as a quadratic function of CPU frequency f^2 in Section 3 and later extended to a monotonically increasing convex function, denoted as $\mathcal{P}(f)$ in Section 4.

Since the processor's clock speed is determined by the voltage setting, it is convenient to think of energy consumption as a function of the processor's time-dependent frequency. Let $l(t)$ denote the number of CPU cycles allocated (or load) during time $[t, t + 1)$. It is the same as the processor frequency per unit time. No speed change is made during time $(t, t + 1)$. The total energy consumed in any time interval $[t_0, t_1]$ under a schedule is

$$E = \sum_{t=t_0}^{t_1} \mathcal{P}(l(t)). \quad (1)$$

The voltage scaling problem is to assign appropriate voltage levels and set corresponding processor speeds so that the energy consumption is minimized and tasks finish before their deadlines.

3 TIME-INVARIANT VOLTAGE SCALING

We show that voltage scaling activities can be modeled as a transfer function-based filtering system. We first consider the case when jobs have the same deadline in Sections 3.1 and 3.2. We will extend the model to support different deadlines in Section 3.3.

3.1 The Filtering Model

We define the symbols used in describing the jobs and the model in Table 1. Consider an aggregated input at time t_0 , $\tilde{w}(t_0)$. The CPU resource allocation at time t , $t \geq t_0$, according to the arrivals at t_0 is

$$l(t, t_0) = \tilde{w}(t_0) \cdot h(t, t_0), \quad (2)$$

where $h(t, t_0)$ is the portion of cycles allocated at time t to job set $n(t_0)$. As we intend to find the system load in order to meet all job deadlines, the load at time t is equal to the sum of all CPU cycles allocated to the jobs released during the last d_m time slots, i.e., from $t - d_m + 1$ to t . The load can be expressed as

TABLE 1
Nomenclature

\mathcal{J}	A set of jobs under consideration
t, t_0	Scheduling time slot index
$n(t)$	$n(t) \subseteq \mathcal{J}$, a subset of jobs released during $[t - 1, t)$
w_i, r_i, d_i	Cycle demand, release time, and relative deadline of job i
d_m	Supreme of the deadlines of jobs in \mathcal{J}
$\tilde{w}(t)$	$\sum_{i \in n(t)} w_i$, aggregated cycle demand at time t
$h(t, t_0)$	Portion of cycles allocated at time t for job set $n(t_0)$
$l(t, t_0)$	Allocated number of cycles at time t for job set $n(t_0)$
$l(t)$	Total number of allocated cycles at time t

$$l(t) = \sum_{t_0=t-d_m+1}^t l(t, t_0) = \sum_{t_0=t-d_m+1}^t \tilde{w}(t_0) \cdot h(t, t_0). \quad (3)$$

Meanwhile, the number of CPU cycles consumed by a job must be equal to its request size. In other words, the scheduler will always allocate enough cycles to a job by its specified deadline. Similarly to the load function, we have

$$\tilde{w}(t_0) = \sum_{t=t_0}^{t_0+d_m-1} l(t, t_0) = \sum_{t=t_0}^{t_0+d_m-1} \tilde{w}(t_0) \cdot h(t, t_0), \quad (4)$$

that is,

$$\sum_{t=t_0}^{t_0+d_m-1} h(t, t_0) = 1. \quad (5)$$

The load function (3) and the scheduling (scaling) function (5) together determine the voltage scaling process.

As we consider only jobs in the model, our method can be applied for both periodic and aperiodic tasks. Given either a set of periodic or aperiodic tasks, it is possible to obtain the task statistics by acquiring (e.g., by sampling technique) detailed timing information about the tasks or by simulation on the target system [32]. Jobs in each task are similar in the sense that they share the same statistical behavior and the same timing requirement. Their inter-arrival times are constant for a periodic task and identically distributed random variables with a certain distribution for an aperiodic task. Similarly, the execution times of jobs in a task are also identical random variables. The statistical behavior of the system does not change with time, that is, the system is stationary [17]. With known job interarrival and execution time distributions of a task set, we can get marginal distribution and autocorrelation of the compound input process $\tilde{w}(t)$ by a measurement-based approach. For example, the distribution can be measured by using a histogram technique [36]; the autocorrelation can be measured by summing products of input [21]. A special case is, when the number of arrivals at each time and the execution times of jobs are time independent random variables with certain probability distributions, the process $\tilde{w}(t)$ is a simple random process following a distribution of *random sum* [7].

A scheduling process is time-invariant in the sense that the scaling function does not depend on the release time of a job. For a time-invariant scaling policy, we have $h(t, t_0) = h(t - t_0)$. We can further simplify the load function (3) as

$$\begin{aligned}
l(t) &= \sum_{t_0=-\infty}^{\infty} h(t-t_0) \cdot \tilde{w}(t_0) = \sum_{t_0=t-d_m+1}^t h(t-t_0) \cdot \tilde{w}(t_0) \\
&= h(t) * \tilde{w}(t),
\end{aligned} \tag{6}$$

where “*” is the convolution operator. The second equality is due to the fact that $h(t) = 0$ for $t < 0$ and $t \geq d_m$. Similarly, the scaling function (5) can be written as

$$\sum_{t=t_0}^{t_0+d_m-1} h(t-t_0) = 1 \text{ or } \sum_{t=0}^{d_m-1} h(t) = 1. \tag{7}$$

Equations (6) and (7) represent the scheduling process as a perfect form of linear filtering system with a transfer function $h(t)$. This not only facilitates the application of vast filter design techniques in literature for optimal scheduling algorithms design, but also simplifies the analysis of the system load as it is only a linear combination of the last d_m input requests.

3.2 The Optimal Time-Invariant Voltage Scaling

Define $\mathbf{h} = [h(t), h(t+1), \dots, h(t+d_m-1)]^T$ and $\mathbf{w}(t) = [\tilde{w}(t), \tilde{w}(t-1), \dots, \tilde{w}(t-d_m+1)]$ as vector forms of the scaling functions and aggregated job sizes. Define $\Omega(t) = \mathbb{E}[\mathbf{w}(t)\mathbf{w}'(t)]$ as the covariance matrix of input process $\tilde{w}(t)$, in the order of d_m . We present the optimal time-invariant voltage scaling policy in energy minimization in Theorem 3.1.

Theorem 3.1. *The optimal time-invariant voltage scaling for a processor with a quadratic energy-speed relationship has a unique solution that minimizes energy consumption.*

Proof. Write (6) in vector form as $l(t) = \mathbf{h}'\mathbf{w}(t)$. Consider the minimization of expected energy consumption $\mathbb{E}[l^2(t)]$. It is equivalent to minimizing the total energy consumption in (1). We have:

$$\mathbb{E}[l^2(t)] = \mathbb{E}[\mathbf{h}'\mathbf{w}(t)\mathbf{h}'\mathbf{w}(t)] = \mathbb{E}[\mathbf{h}'\mathbf{w}(t)\mathbf{w}'(t)\mathbf{h}] = \mathbf{h}'\Omega\mathbf{h}, \tag{8}$$

subject to the constraint in (7).

The covariance matrix Ω is a symmetric positive definite matrix. There exists an orthogonal matrix \mathbf{U} such that $\mathbf{U}^{-1}\Omega\mathbf{U} = \Lambda$ and $\Lambda = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_{d_m})$, where λ_i are the eigenvalues of the matrix Ω and $\lambda_i > 0$.

Let $y = \mathbf{h}'\Omega\mathbf{h}$. It follows that $y = \mathbf{h}'\mathbf{U}\Lambda\mathbf{U}^{-1}\mathbf{h}$. Define $\mathbf{g} = \mathbf{U}^{-1}\mathbf{h}$. It follows that $y = \mathbf{g}'\Lambda\mathbf{g}$. This is a standard form. It is minimized when $\lambda_i g_i^2 = \lambda_j g_j^2$ for any $1 \leq i, j \leq d_m$. Since $\mathbf{h}'\mathbf{1} = 1$ and $\mathbf{U}\mathbf{g} = \mathbf{h}$, there exists a unique solution of \mathbf{h} for the optimization problem. \square

The solution shows that the scaling function can be determined by the covariance matrix of input process in the order of d_m . The matrix is an indication of job size correlation structure. A special case is when the input process $\tilde{w}(t)$ is time independent. The covariance matrix becomes diagonal and the solution becomes uniform. Formally, we have:

Corollary 3.1. *If the compound input process $\tilde{w}(t)$ is independent over time, the optimal time-invariant voltage scaling is to allocate equal amount of cycles to a job at each time unit before its deadline.*

The uniform solution for input process can also be obtained by the online AVR for aperiodic tasks in [34]. In essence, it is a special case of the time-invariant scaling when the input process is time independent.

3.3 Extension for Tasks with Different Deadlines

The proposed model can be readily extended to support different deadlines. We achieve this by making a set of running queues up to d_m containing jobs with the same deadlines. Scheduling is conducted in all of the queues independently. At time t , the system load is a sum of load from all queues, $l(t) = \sum_{j=1}^{d_m} l_j(t)$. The analytical model can be applied to each queue with the same deadline. We only need to show the optimality is maintained after the sum of load. The expected energy consumption is as follows:

$$\begin{aligned}
\mathbb{E}[l^2] &= \mathbb{E}\left[\left(\sum_{j=1}^{d_m} l_j\right)^2\right] = \mathbb{E}\left[\sum_{j=1}^{d_m} l_j^2\right] + 2\mathbb{E}\left[\sum_{i=1}^{d_m} \sum_{j=i+1}^{d_m} l_i l_j\right] \\
&= \sum_{j=1}^{d_m} \mathbb{E}[l_j^2] + 2 \sum_{i=1}^{d_m} \sum_{j=i+1}^{d_m} \mathbb{E}[l_i l_j],
\end{aligned} \tag{9}$$

where we omit the variable t for brevity. Assume the input processes to different queues are statistically independent. Each item $\mathbb{E}[l_i l_j]$ can be simplified as $\mathbb{E}[l_i]E[l_j]$. Note that we provide no service degradation for all jobs. The expected load in any queue must equal the expected computation requirement of input arrivals. For a system with stationary input, the expected computation requirement is a constant. The total energy consumption is then determined by the sum of energy consumed from different queues, represented by the first item in the last equation of (9). Due to task independence, the total energy consumed is minimized if and only if energy consumed in each queue is minimized.

3.4 Energy Savings

In this part, we will prove that an existing algorithm (DVSST in [23]) for sporadic tasks and the static voltage scaling [1], [22] for periodic tasks are special cases of the time-invariant voltage scaling with respect to energy savings. Consider a sporadic task set $T = \{T_1, T_2, \dots, T_n\}$. Task j has two associated parameters, the minimum interarrival time between two consecutive jobs of the task and WCET, denoted as p_j , e_j , respectively. It was assumed that the relative deadline of task j equals p_j and CPU speed is scaled by a frequency-scaling factor, α , the normalized speed with respect to the maximum processor speed [23].

Theorem 3.2. *In a DVS enabled processor, the DVSST algorithm for sporadic tasks and the uniform allocation lead to the same amount of energy consumption.*

Proof. In DVSST, the speed will only be scaled under two cases. One is, at the time a job is released, the factor is increased by e_j/p_j for task j if the deadline of the last job from the task has passed; the other is, when a job finishes, the factor is scaled down by e_j/p_j . No change is made in all other cases. Use α_n to denote the n th change to α at time t . We summarize how α_n changes as

$$\alpha_n = \begin{cases} \alpha_{n-1} + \frac{e_j}{p_j}, & \text{a new instance of } T_j \text{ is released} \\ \alpha_{n-1} - \frac{e_j}{p_j}, & \text{deadline of last instance of } T_j \text{ passes} \\ \text{no change otherwise (including idle).} \end{cases}$$

To apply the analytical results to the sporadic task model, we consider a time-independent process from a sporadic task set. Applying the result in Corollary 3.1, cycle allocation for job j is uniform before its deadline. The allocated cycles for the job in each time slot are w_j/p_j . If there is a job release at time t , the load is increased by the amount w_j/p_j in the next p_j time slots. After time $t + p_j$, the job finishes and the load is decreased by the same amount. If multiple job releases exist at the same time, the procedure repeats. Similarly to DVSSST, the load at t for the n th change is

$$l_n(t) = \begin{cases} l_{n-1}(t) + \frac{w_j}{p_j}, & \text{a new instance of } T_j \text{ is released} \\ l_{n-1}(t) - \frac{w_j}{p_j}, & \text{deadline of last instance of } T_j \text{ passes} \\ \text{no change otherwise (including idle).} \end{cases}$$

Let τ_s be the set of all scaling-factor change intervals for a set of sporadic tasks during time $[0, \tau]$, i.e., $\tau_s = \{\tau_0, \tau_1, \dots, \tau_n\}$. Let $\alpha_s = \{\alpha_0, \alpha_1, \dots, \alpha_n\}$ be the set of all scaling factors corresponding to the intervals in τ_s . Denote f_j as the operating frequency in interval τ_j and $f(t)$ as the frequency at time t . According to (1), the total energy consumed in interval τ using DVSSST can be expressed as

$$\begin{aligned} E_{DVSSST} &= \sum_{t=0}^{\tau} l^2(t) = \sum_{t=0}^{\tau} (f(t) \cdot 1)^2 = \sum_{j=0}^n f_j^2 \tau_j \\ &= \sum_{i=0}^n \alpha_i^2 f_{max}^2 \tau_i = \sum_{j=0}^n \frac{e_j^2 f_{max}^2 \tau_j}{p_j^2}. \end{aligned} \quad (10)$$

Note that $l(t) = f(t) \cdot 1$ holds because the load is the number of cycles in a time unit. Similarly, energy consumed by the uniform allocation is

$$\begin{aligned} E_{uni} &= \sum_{t=0}^{\tau} l^2(t) = \sum_{j=0}^n l_j^2(t) \cdot \tau_j = \sum_{j=0}^n \left(\frac{w_j}{p_j}\right)^2 \cdot \tau_j \\ &= \sum_{j=0}^n \frac{e_j^2 f_{max}^2 \tau_j}{p_j^2}. \end{aligned} \quad (11)$$

We use $l_j(t)$ to denote the load at time slot t during the j th interval, which is a constant during the interval as there is no speed change. The last equation holds because the requested number of cycles, w_j , equals the worst-case execution time, e_j , multiplied by the maximum processor frequency, f_{max} . Equations (10) and (11) are equal. This proves the two voltage scaling approaches lead to the same energy consumption. \square

According to [23], DVSSST will give the same results as the static scaling approach [1], [22] for periodic tasks if all sporadic tasks are released at their maximum rates. We hence have the following corollary:

Corollary 3.2. *The static voltage scaling for periodic tasks and the uniform allocation lead to the same amount of energy consumption.*

4 TIME-VARIANT VOLTAGE SCALING

The scaling policy discussed in the preceding section is optimal in the sense that the scaling function does not change over time. The energy consumption can be further reduced if we adapt the function in response to unfinished jobs and new job releases.

Similarly to the time-invariant case, we make separate running queues, each containing jobs with the same deadlines. Scheduling is conducted in the queues in the order of increasing deadlines. A difference is the existence of interdependence between different queues when the scaling function varies with time. We tackle this issue by letting queue j contain only jobs with the same residue deadline j . Newly released jobs are dispatched into respective queues according to their deadlines. After jobs with a deadline of j are served at current scheduling slot, their new deadline becomes $j - 1$. Jobs entering queue j at time t with a deadline of j will enter queue $j - 1$ at time $t + 1$ and complete at $t + j$. Denote the load at time t in queue j as $l_j(t)$. The system load $l(t)$ at t is a sum of the load of all queues, $\sum_{j=1}^{d_m} l_j(t)$.

4.1 The Optimal Time-Variant Voltage Scaling

Consider a time interval $[0, d_m]$. We start from time zero and omit the variable t for brevity. Define q_j as the backlog of queue j , including jobs released at the current time. We formulate and solve the optimization problem in the following theorem:

Theorem 4.1. *The optimal delay maximized scheduling can be achieved by finding a schedule that*

$$\text{Minimize } E = \sum_{i=0}^{d_m-1} \mathcal{P} \left(\sum_{j=i+1}^{d_m} l_j(i) \right) \quad (12)$$

Subject to $l_j(i) > 0$, $0 \leq i \leq d_m - 1$, $1 \leq j \leq d_m$

$$\sum_{i=0}^{j-1} l_j(i) = q_j, \quad 1 \leq j \leq d_m. \quad (13)$$

The optimal solution can be achieved by the following process for queue j , $1 \leq j \leq d_m$:

$$l_j(i) = \left(l_j - \sum_{k=i+1}^{j-1} l_k(i) \right)^+, \quad 0 \leq i \leq (j-1), \quad (14)$$

where the notion $(x)^+ = \max(x, 0)$ and l_j is determined by evaluating the backlog constraints (13).

Proof. According to (1), the energy consumed is $\sum_{t=t_0}^{t_1} \mathcal{P}(l(t))$. The load at the i th time slot is the sum over all queues with larger residual delay than i , i.e., $\sum_{j=i+1}^{d_m} l_j(i)$. Based on all of the current and past jobs information, the optimization criterion becomes

$$E = \sum_{i=0}^{d_m-1} \mathcal{P}(l(i)) = \sum_{i=0}^{d_m-1} \left(\mathcal{P} \left(\sum_{j=i+1}^{d_m} l_j(i) \right) \right).$$

It is the average of the energy consumption summed over all queues in the next d_m time slots. This shows the minimization of energy consumption is to minimize (12).

The constraints (13) mean the backlog in queue j must be scheduled in the next j time slots. Including the constraint, the minimization becomes a standard constrained optimization problem. It can be solved by using Lagrange Multipliers. For all i and j , we form the Lagrangian

$$L(l_j(i), \lambda_j) = \sum_{i=0}^{d_m-1} \mathcal{P} \left(\sum_{j=i+1}^{d_m} l_j(i) \right) - \lambda_j \left(\sum_{i=0}^{j-1} l_j(i) - q_j \right)$$

and differentiate L with respect to $l_j(i)$, assuming it is differentiable. Applying the Kuhn-Tucker conditions, we have

$$\frac{\partial \mathcal{P}(\sum_{j=i+1}^{d_m} l_j(i))}{\partial l_j(i)} - \lambda_j = 0.$$

Solve the condition for $\sum_{j=i+1}^{d_m} l_j(i)$. We get

$$\mathcal{P} \left(\sum_{j=i+1}^{d_m} l_j(i) \right) = \lambda_j l_j(i) + C,$$

where C is a constant. As the power function is assumed to be monotonically increasing, the inverse function of $\mathcal{P}(\cdot)$ exists, denoted as $\mathcal{P}^{-1}(\cdot)$. The conditions then become

$$\begin{aligned} \sum_{j=i+1}^{d_m} l_j(i) &= \mathcal{P}^{-1}(\lambda_j l_j(i) + C) \\ \sum_{k=i+1}^{j-1} l_k(i) + l_j(i) + \sum_{k=j+1}^{d_m} l_k(i) - \mathcal{P}^{-1}(\lambda_j l_j(i) + C) &= 0. \end{aligned}$$

For brevity, we let $\lambda'_j = \mathcal{P}^{-1}(\lambda_j l_j(i) + C)$. The conditions can then be written as

$$\begin{aligned} \sum_{k=i+1}^{j-1} l_k(i) + l_j(i) + \sum_{k=j+1}^{d_m} l_k(i) - \lambda'_j &= 0 \\ l_j(i) &= \lambda'_j - \sum_{k=j+1}^{d_m} l_k(i) - \sum_{k=i+1}^{j-1} l_k(i). \end{aligned} \quad (15)$$

The conditions in (15) give the solution for queue j at time i , depending on the values of all other queues. This dependency makes it hard to solve. Let $l_j = \lambda'_j - \sum_{k=j+1}^{d_m} l_k(i)$ for queue j . The conditions become

$$l_j(i) = l_j - \sum_{k=i+1}^{j-1} l_k(i).$$

Consider the fact that the load $l_j(i)$ is nonnegative. This leads to the conditions (14) with an unknown value l_j . The value can be determined by the constraints (13) given queue backlog q_j . Therefore, we get the values of l_j and $l_j(i)$ that satisfy the conditions (15). Note that the conditions are only necessary for the existence of a global minimum. Since the optimization function is convex, the conditions are also sufficient. \square

The key part of the solution is to determine the bound l_j given a queue backlog q_j . Once we get the bound, any feasible schedule can be used for task executions. The

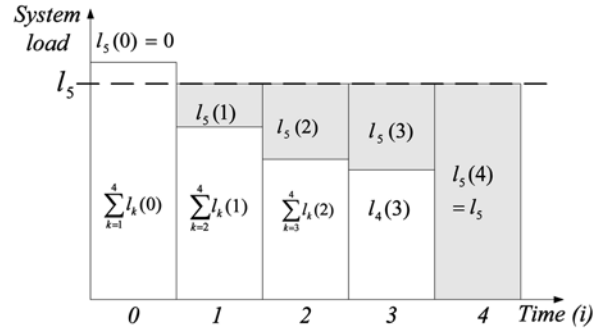


Fig. 3. Water-filling illustration of the speed assignment for queue 5.

variable l_j can be interpreted as the maximum system load, or processor speed similarly, which is set for queue j during the next j time slots based on current queue backlog. The reserved cycle allocation at the i th time slot is $\sum_{k=i+1}^{j-1} l_k(i)$. The solution is simply a subtraction between the cap l_j and the aggregated load. If the resulting load is negative, no more cycles will be allocated at the time slot. As a result of the solution, the load is a nonincreasing function of the time slot. Formally, we have the following property:

Corollary 4.1. *The speed setting according to Theorem 4.1 for any queue j , $l_j(i)$, is a nondecreasing function of i for $i = 0, 1, \dots, j-1$.*

To distribute the queue backlog q_j between different queues, the solution pushes the starting time of each job as far as possible under the deadline constraint. By taking queue 5 as an example, Fig. 3 illustrates the basic idea graphically. The vertical white areas indicate the accumulated reserved allocation at different times. The queue backlog is first distributed to the *latest* time slot with the lowest accumulated load, denoted by the shaded slot 4. As the input increases further, parts of the requests will be put into busier slots. When all of the queue backlog has been distributed, we get the bound l_j . The request distribution is similar to a *water-filling* process in which power is distributed among a set of parallel Gaussian channels for maximum information capacity [3]. Many problems in information theory can be solved by constructing solutions in a form of water-filling structure. Recent examples include maximization of the sum of the communication rate in multiuser environments subject to individual power constraints [35], minimization of data transmission energy in a Gaussian channel [14], and in a fading channel adapting to both channel states and queue backlog [33].

4.2 Speed Assignment Algorithm

In this subsection, we develop an iterative algorithm to find the optimal speed setting. In [38], we presented a schedule due to (14). An alternative policy under EDF is listed in Algorithm 1 to find the speed of a group of jobs released at time 0. In the general case, when jobs are released at different times, the algorithm can be applicable by considering all ready jobs, in a similar way to [4], [10]. We assume the jobs are arranged in a nondecreasing order by their deadlines.

As speed change is allowed only at job release or completion, we group queues and jobs with the same speed into one state to improve time efficiency. The algorithm

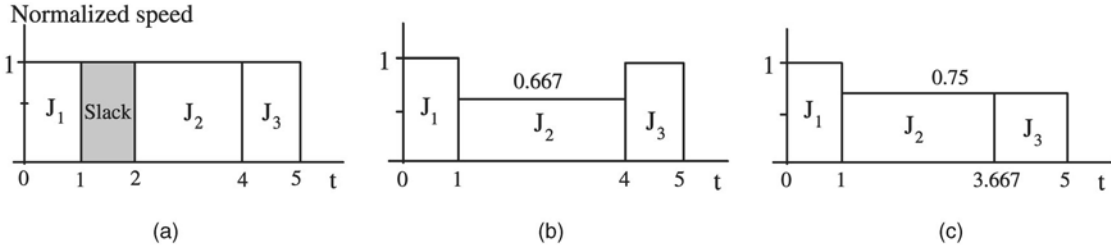


Fig. 4. Example solutions for slack distribution. (a) No slack distribution. (b) Slack claimed by job 2. (c) Slack claimed by jobs 2 and 3.

maintains a set of states. Each state contains indices of jobs in a subset S having the same service speed l_S with a cycle demand w_S summed over all jobs in S . Initially, we have a list L containing the first job from a set of jobs. In each iteration, job i is added with a speed either smaller than or identical to the last speed in L . In the former case, we continue to the next job; in the latter, we keep checking the next speed in L and merge jobs with the same speed to one state until all cycles of job i are distributed. After n iterations, the speeds of all jobs are stored in list L . The maximum number of states in L is n , which occurs when the **else** part of the while loop is executed up to its maximum number, $n - 1$. On the other extreme, there is only one state in L , meaning the maximum number of state merging is taking, $n - 1$. Any other case is a combination of $n - 1$ executions of the **if** and **else** parts. Therefore, the algorithm has a linear time in determining speed settings of n jobs. The average running time for each job is constant.

Algorithm 1 Getting optimal speed setting online for a set of jobs ready at time 0 under EDF.

```

1:  $S_1 = \{1\}$ ,  $w_{S_1} = w_1$ ,  $l_{S_1} = w_1/d_1$ 
2: Initialize  $L$  with state  $(S_1, w_{S_1}, l_{S_1})$ , denoted by  $S_1$ 
3: for  $i = 2$  to  $n$  do  $\triangleright$  add job  $i$  at the  $i$ th iteration
4:    $S_i = \{i\}$ ,  $w_{S_i} = 0$ ,  $l_{S_i} = 0$ ,  $\tau = d_i - d_{i-1}$ 
5:   append  $S_i$  to the end of  $L$ 
6:   while  $w_i > 0$  do
7:     let the last state in  $L$  as  $S'$ 
8:     let the second to the last state in  $L$  as  $S''$ ;
9:      $S'' = (0, 0, \infty)$  if there is only one state in  $L$ 
10:    if  $w_i > (l_{S''} - l_{S'})\tau$  then  $\triangleright$  job  $i$  has a larger
11:    speed than  $l_{S''}$ ; need to check next speed in  $L$ 
12:       $w_i = w_i - (l_{S''} - l_{S'})\tau$ 
13:      merge  $S'$  and  $S''$  into one state  $\bar{S}$ ,
14:       $w_{\bar{S}} = l_{S''}\tau + w_{S''}$ ,  $l_{\bar{S}} = l_{S''}$ 
15:       $\tau = \tau + w_{S''}/l_{S''}$ 
16:    else  $\triangleright$  job  $i$  has a smaller speed than  $l_{S''}$ ; done with
17:    job  $i$ 
18:       $w_{S'} = w_{S'} + w_i$ ,  $l_{S'} = l_{S'} + w_i/\tau$ ,  $w_i = 0$ 
19:    end if
20:  end while
21: end for
22: for each state  $S_i \in L$  and each  $j \in S_i$  do  $\triangleright$  size of
23:    $S_1 \cup S_2 \dots \cup S_{|L|}$  is  $n$ 
24:   return normalized speed  $l_{S_i}$  with respect to
25:   maximum load for job  $j$ 
26: end for

```

When we use the algorithm online for dynamic job releases, we need to replace job execution time and deadline

with its residue time and deadline. The algorithm is invoked whenever there is a new job and makes a speed assignment based on uncompleted jobs and the newly released job. Its worst-case running time for one job is still $O(n)$. This occurs when a newly released job has the earliest deadline, in which case, all jobs finished later may have to be rescheduled. The best cases occur either when all ready jobs are newly released or when we only need one speed for all uncompleted jobs and the newly released job has the largest deadline. In both cases, the time needed for one job is constant. The average running of the algorithm is between constant and $O(n)$.

4.3 Online Slack Management

Real-time applications usually have nondeterministic execution times or complete earlier than their WCETs. Aydin et al. applied a priority-based slack distribution based on the remaining execution times of high priority tasks for periodic tasks, named DRA [1]. Recently, Lee and Shin proposed an online slack management algorithm, called OLDVVS [16], for a general task model. They used a similar priority-based approach to determine whether a slack is reclaimed by other tasks.

The time-variant voltage scaling can also be used for slack distribution combined with the priority-based slack determination. Each time a request finishes earlier, we apply the solution process in (13) to compute the next speed setting using residual execution times for unfinished jobs. EDF is used to schedule jobs under the speed due to its optimality even for aperiodic tasks. The solution can be illustrated using a simple example with three jobs with $J_i(r_i, w_i, d_i)$ as $J_1(0, 2, 2)$, $J_2(0, 2, 4)$, $J_3(0, 1, 5)$. Suppose the actual execution time of the first job is 1 and jobs 2, 3 take their WCETs. Fig. 4a is a schedule without online slack distribution. During time $[1, 2)$, the processor is idle. Fig. 4b shows the slack distribution using priority-based slack stealing. In this simple example, both OLDVVS and a slighted adapted DRA can give the same schedule. Only job 2 claims the online slack in the schedule. However, jobs 2, 3 are both ready for execution at time 1 and job 3 should also be considered in the slack distribution. TV-DVS computes a new speed considering all ready jobs after job 1 finishes. Fig. 4c is a schedule using TV-DVS for speed setting and EDF for scheduling. We get a constant speed of 0.75 during the time $[1, 5)$, which is optimal due to the convexity of the energy function. TV-DVS remains efficient even if job 2 or 3 finishes earlier. If there is only one job ready, TV-DVS would give the same schedule as the priority-based slack distribution.

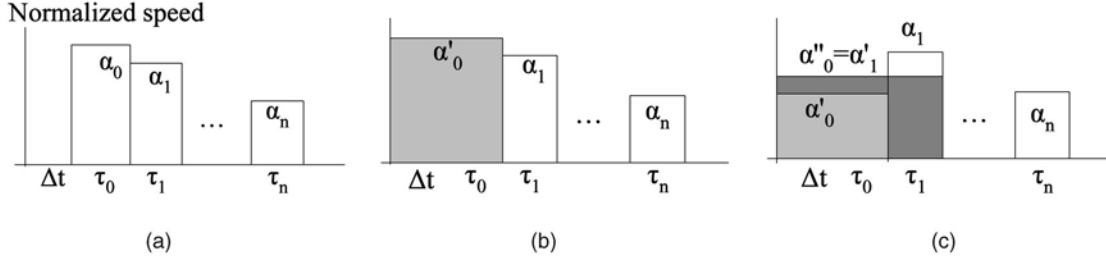


Fig. 5. Slack distribution to all ready jobs. (a) Slack Δt to be reclaimed by n jobs. (b) Case 1, $\alpha'_0 \geq \alpha_1$. (c) Case 2, $\alpha'_0 < \alpha_1$.

In TV-DVS, deadline misses occur only when the required computation speed is higher than the maximum processor speed. Intuitively, the resulting processor speed will not be increased by starting a job earlier than its worst-case planning. If the worst-case planning guarantees no deadline miss, the guarantee also holds after reducing the speed. Formally, we present this result in Theorem 4.2.

Theorem 4.2. *Using TV-DVS for online slack distribution preserves the feasibility of the WCET-based schedule.*

Proof. Suppose at time t , a job finishes earlier with a slack time Δt . If the slack is not claimed by other jobs, the WCET-based schedule and feasibility do not change. We assume the slack will be claimed by n ready jobs. Let τ_s be the set of all scaling-factor (or normalized speed) change intervals in the worst-case schedule of the n jobs, $\tau_s = \{\tau_0, \tau_1, \dots, \tau_n\}$. Let $\alpha_s = \{\alpha_0, \alpha_1, \dots, \alpha_n\}$ be the set of all scaling-factors corresponding to the intervals in τ_s , as shown in Fig. 5a. According to Corollary 4.1, $\alpha_0 \geq \alpha_1 \geq \dots \geq \alpha_i \dots \geq \alpha_n$. Denote the updated factor after slack distribution for interval i as α'_i . In TV-DVS, speed setting is determined by distributing a request between its start and end times. If there is a deadline miss at time interval i , the required CPU speed must be larger than the maximum available speed, i.e., $\alpha'_i > 1$. As TV-DVS guarantees $\alpha'_0 \geq \alpha'_1 \geq \dots \geq \alpha'_i \dots \geq \alpha'_n$, we have $\alpha'_0 > 1$ when there is a deadline miss. In other words, $\alpha'_0 \leq 1$ with a feasible WCET-based schedule.

We consider the distribution of the slack time Δt . The scaling-factor after the first job claims the slack time is computed as $\alpha'_0 = \alpha_0 \frac{\tau_0}{\tau_0 + \Delta t}$. In the case $\alpha'_0 \geq \alpha_1$, as in Fig. 5b, no further speed change is necessary and α'_0 is the speed setting after slack distribution. As $\alpha'_0 < \alpha_0 \leq 1$, this ensures no deadline miss.

In the second case, $\alpha'_0 < \alpha_1$, the speed setting of the first job will be increased. We perform the water-filling process considering the interval τ_1 and the updated factor is $\alpha''_0 = \alpha'_1 = \alpha'_0 \frac{\Delta t + \tau_0}{\Delta t + \tau_0 + \tau_1} = \alpha_0 \frac{\tau_0}{\Delta t + \tau_0 + \tau_1}$ in Fig. 5c. As $\alpha'_0 < \alpha_1 \leq \alpha_0 \leq 1$, we have $\alpha''_0 < 1$. Again, it ensures no deadline miss.

Finally, if the updated α''_0 is smaller than the next speed α_2 , the above two-case analysis must be repeated until we either get a speed higher than the speed in the next interval (finish with case 1) or we reach the end of the job list (finish with case 2). In each analysis step, the speed setting is guaranteed to be smaller than α_0 ($\alpha_0 \leq 1$). Therefore, no deadline miss is possible. \square

5 EXPERIMENTAL EVALUATION

We evaluated the performance of the proposed schemes in a simulation environment with a processor capable of voltage and frequency scalings. The main objective is to demonstrate the effectiveness of the proposed scalings in energy savings. We assumed the energy consumption was a square of processor speed. We ignored the overhead to switch to power-down mode and assumed no energy was consumed in the power-down mode for simplicity.

We first assumed each job took its worst-case time to execute or had deterministic execution time. We implemented the following schedulers for performance evaluation in this case:

- No-DVS, which schedules jobs using the maximum available speed and shuts down the processor whenever there is no ready job.
- Offline: The offline optimal algorithm of Yao et al. [34]. It serves as a theoretical lower bound on energy savings. We implemented the algorithm with time complexity $O(n^2)$.
- DVSST: An online processor scaling algorithm for a sporadic task model with constant time complexity due to Qadi et al. [23].
- TI-DVS: The time-invariant voltage scaling, which has a constant time complexity.
- TV-DVS: The time-variant voltage scaling with a linear time complexity.

To investigate the impact of variation on job execution times, three online slack distribution policies were integrated:

- DVSST + CC (Cycle-conserving EDF): Worst-case schedule using DVSST combined with the reclaiming algorithm due to Pillai and Shin [22]. Both have constant complexity.
- TV-DVS + OLDVS: The time-variant voltage scaling and the reclaiming algorithm of Lee and Shin [16] with a linear and constant complexity, respectively.
- TV-DVS + TV-DVS: A unified solution for both schedule based on WCET and slack distribution based on TV-DVS with a linear complexity.

We point out that the cycle conserving technique proposed by Pillai and Shin is readily applicable to DVSST for online slack distribution. This is because DVSST scales processor speed using a frequency scaling factor, similarly to the utilization approach. The only modification to the DVSST algorithm is to reduce the frequency factor by the amount of r_j/p_j at the time when a job finishes earlier,

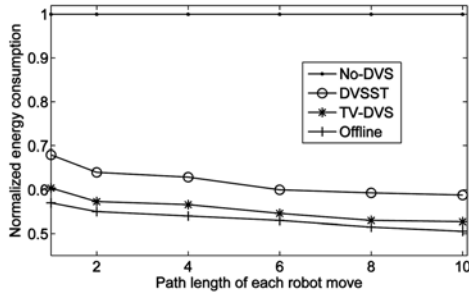


Fig. 6. Energy consumption with the Robotic Highway Safety Marker application.

where r_j and p_j are the unused WCET of the job and deadline for task j . If there is no ready job, the processor is shut down to conserve energy.

Recall that TI-DVS can take the autocorrelation structure of the input process into account. However, experimental results show that the structure has little effect on energy savings. The scheduling function was computed using Theorem 3.1. We compared the energy consumption with that of a uniform allocation by Corollary 3.1. The performance gap is within 1 percent for all of the test cases. This suggests that, for input processes with different autocorrelation degrees, a uniform allocation is generally enough as far as energy consumption is concerned. As pointed out before, the uniform allocation is equivalent to DVSST in energy savings, which means the performance gap between DVSST and TI-DVS is within 1 percent. We do not present the results of TI-DVS for energy savings to improve the readability of the figures.

5.1 Input with Deterministic Execution Times

We first evaluate the algorithms with a real-world application: the Robotic Highway Safety Marker (RSM) used in [23]. The RSM application was designed to control robot movements. It can be roughly divided into two stages: the path length calculation of robot movement and a serial of tasks for each move. The task set can be modeled as a sporadic task set because each task has a minimum separation period. The execution times of these tasks are very deterministic. We assumed the tasks were run in a processor in which voltage was adjusted with the frequency and a decrease in speed led to a quadratic energy saving. We incorporated the Rabbit 2000 processor speed specifications [23] in the simulation.

We experimented with a scenario in which the robot was constantly moving without idle periods. Each experiment simulates 500 robot moves with different path lengths. The results of No-DVS and Offline are upper and lower bounds of energy consumption. Fig. 6 shows the normalized energy consumption with respect to that of No-DVS. It reveals that an increased path length leads to decreased energy consumption. Since it takes fixed time to determine the path length, a longer path will amortize the overhead and result in a lower processor utilization. More energy savings are possible with low average load because more idle intervals exist in the schedule without DVS. With all the lengths, the proposed TV-DVS leads to more than 40 percent energy savings compared with No-DVS, about 10 percent

better than DVSST. One remarkable result is that the consumed energy of TV-DVS is close to the offline solution. The performance gap is around 5 percent. Recall that Offline requires a priori knowledge about both timing and size information of job releases. In contrast, TV-DVS is online and can be easily integrated.

We compare the running times of two online algorithms, TV-DVS and DVSST with linear and constant running times, respectively. Because we do not have the actual running time of the algorithms on the simulated processor, we estimate their running time by first running the algorithms on a Pentium M laptop at 700 MHz clock rate and then project their running times onto the simulated processor. For tasks generated from RSM in Fig. 6, the total execution time of 500 robot moves adds up to 77.6 s. The running times of DVSST and TV-DVS are estimated as 0.44 s and 0.57 s, respectively. Although TV-DVS is more complex than DVSST, which has a constant running time, its overhead is about 0.7 percent of the overall task execution time. The energy overhead for each speed computation is determined by a cubic function of speed multiplied by algorithm running time. The reported energy in Section 5.1 is a combination of algorithm energy overhead and tasks energy expenditure. As TV-DVS does not introduce significant running time overhead, the impact of algorithm energy overhead on total energy expenditure is hardly noticeable.

Since different voltage scaling algorithms may benefit from the task timing parameters differently, it is necessary to investigate different input patterns and processor utilization for a fair comparison. We next evaluate the schedulers by using a randomly generated input. Because of the irregular job releases of aperiodic tasks, the processor utilization is not easy to control. We characterize the utilization by interarrival times of jobs and number of tasks. Two groups of input were used. The first was generated with different numbers of tasks. Jobs interarrival time was assumed to follow an exponential distribution with mean 100 ms. To enable the use of DVSST, we assumed there was a minimal 10 ms interarrival time between any two consecutive jobs of a task. The required execution cycles of each job were generated following a normal distribution $n(100, 10)K$. In the second group, we fixed the number of tasks as 20 and varied the interarrival time. In both tests, the deadline of each task was set to 10 ms. We assumed a processor model with continuous frequency levels ranging from 10 MHz to 200 MHz. The results in Figs. 7 and 8 show that all of the DVS policies lead to significant energy savings with different task patterns. Although with slightly different performance gaps, TV-DVS consistently outperforms DVSST and is close to Offline. When the processor has more idle intervals, characterized by a small number of tasks and large interarrivals, all of the DVS algorithms offer more energy savings. However, the energy savings gaps are reduced. This is because, with low processor utilization, few jobs are running in the system. More jobs can be run at a constant speed with all of the DVS algorithms, which leads to the minimum energy consumption.

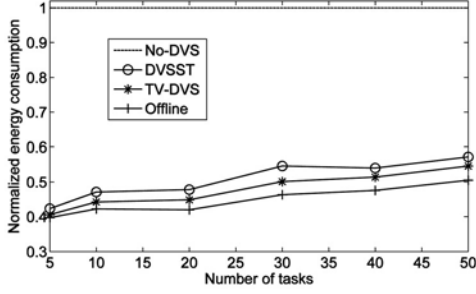


Fig. 7. Energy consumption of synthetic workload with different number of tasks.

5.2 Impact of Variability in Actual Workload

The actual execution times of real-time applications are often nondeterministic and smaller than their WCETs. We investigate the impact of variability in the actual workload with online slack distribution. Workload variation is measured by a ratio of best-case execution time, BCET, to WCET. In each run, the actual execution time of a job was drawn from a normal distribution. The mean and the standard deviation were set to $(BCET + WCET)/2$ and $(WCET - BCET)/6$, respectively, as suggested in [29]. Under this setting, 99.7 percent of the execution times fall in the range $[BCET, WCET]$ approximately. The input process was generated following a sporadic task model in the same way as the last experiment with 30 tasks. The mean interarrival time was set to 60 ms. Simulated energy consumption of the techniques is presented in Fig. 9. When $BCET/WCET = 1$, no slack distribution is possible. The performance is the same as the one in Fig. 8 with mean interarrival 60 ms. As the ratio decreases, i.e., the actual execution times are more variable, the performance of all schemes improves by reclaiming unused CPU slack. For example, DVSST + CC and TV-DVS provide 18 percent and 21 percent more energy savings compared with their worst-case execution, respectively. The relative performance of all schemes also improves with a more variable workload. Offline has definite advantages over other techniques as it has exact knowledge of the actual execution times of all jobs. TV-DVS provides consistently more energy savings than the other two online approaches. However, only marginal improvement is achieved over TV-DVS + OLDVDS. Recall that the use

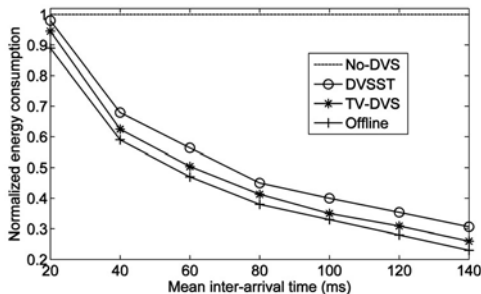


Fig. 8. Energy consumption of synthetic workload with different interarrival times.

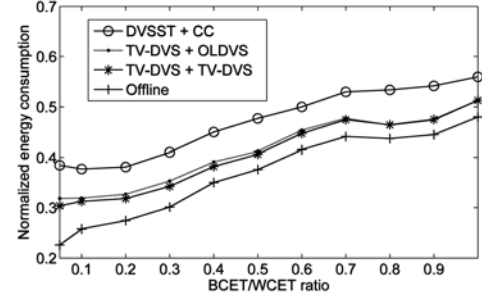


Fig. 9. Impact of variability in actual execution time.

of TV-DVS for online reclamation only outperforms OLDVDS in case of multiple ready jobs at the time of slack distribution. The marginal performance gap implies this situation does not occur frequently under the experimental setting.

6 STATISTICAL REAL-TIME GUARANTEE

To guarantee that all tasks meet their deadlines, conditions must be placed on the task set. The provisioning of statistical real-time guarantee was motivated by the schedulability test for sporadic tasks [23]. It was proven that, for a sporadic task set with deadline equal to minimum task interrelease time p_j , DVSST will succeed in scheduling the tasks if and only if the summed utilization is smaller than one when all sporadic tasks are released at their maximum rates,

$$\sum_{j=1}^n \frac{e_j}{p_j} \leq 1. \quad (16)$$

It is conservative by considering the worst-case scenario. In fact, the average interarrival time can be much larger than the minimum interval for a sporadic task. As a result, the average processor utilization would be very low under the worst-case configuration. To relax the condition, we allow the total utilization larger than one. This can result in system overload or deadline misses. We characterize the deadline misses by an overload probability v , defined as the probability that the required CPU speed surpasses the maximum frequency f_{max} or capacity, i.e., $prob(l(t) > f_{max})$. Admission control needs to be enforced in the presence of overload. We can either reject the job causing overload or admit it and schedule the unfinished part when the processor has enough resources, similarly to the task transformation technique in [32]. Although the overload probability is different from the deadline miss rate, it provides a safe bound for deadline misses.

6.1 Bounds for Deadline Misses

6.1.1 A General Bound

We first provide an upper bound of the overload probability for a general input with the minimum amount of input statistics required, the load mean μ_l and standard deviation σ_l in Theorem 6.1.

Theorem 6.1. *An upper bound of the overload probability for a general input is*

$$v \leq \frac{\sigma_l^2}{\sigma_l^2 + (f_{max} - \mu_l)^2}. \quad (17)$$

Proof. From Chebyshev's inequality, it is known that

$$F\{I\} \leq a^{-1} \mathbb{E}[u(y)],$$

where y is a random variable, I is an interval, $F\{I\}$ is a distribution function, and $u(y) > a > 0$ for all y in I .

Substitute y with system load $l(t)$ and define $u(l(t)) = (l(t) + x)^2$ with $x > 0$. It can be verified that $u(l(t)) > (f_{max} + x)^2 > 0$ for $l(t) > f_{max} > 0$. Therefore,

$$\text{prob}(l(t) > f_{max}) \leq \frac{1}{(f_{max} + x)^2} \mathbb{E}[(l(t) + x)^2].$$

Since

$$\mathbb{E}[(l(t) + x)^2] = \sigma_l^2 + \mu_l^2 + 2x\mu_l + x^2,$$

we have

$$\text{prob}(l(t) > f_{max}) \leq \frac{1}{(f_{max} + x)^2} (x^2 + 2x\mu_l + \sigma_l^2 + \mu_l^2).$$

The right side of the inequality takes the minimum value at $x = -\mu_l + \sigma_l^2/f_{max}$ and the minimum value is the same as (17). This proves the theorem. \square

The theorem reveals that the overload probability is determined by the system load mean and variance. In case the input arrivals are highly variable, the system load variance can be a dominating factor in determining the capacity. The theorem is useful in the sense that it provides a unified solution that applies to all distributions with finite first and second order moments. From the theorem, it is easy to get an estimate of the computation capacity f_{max} and overload probability given statistics of input process.

Corollary 6.1. *If the compounded input process $\tilde{w}(t)$ is time-independent, the chance of overload using TI-DVS can be characterized by*

$$v \leq \frac{\sigma_{\tilde{w}}^2}{\sigma_{\tilde{w}}^2 + (f_{max} - \mu_{\tilde{w}})^2 d_m}, \quad (18)$$

where $\mu_{\tilde{w}}$ and $\sigma_{\tilde{w}}^2$ are the mean and variance of the marginal distribution of the input process.

Proof. According to Corollary 3.1, the function $h(t)$ should be set to a constant value $1/d_m$ for an independent process. Thus,

$$\sigma_l^2 = \sigma_{\tilde{w}}^2 \sum_{t=-\infty}^{t=\infty} h^2(t) = \frac{\sigma_{\tilde{w}}^2}{d_m}.$$

The load mean must equal the input mean as we will allocate enough resources to all requests, $\mu_l = \mu_{\tilde{w}}$. Substituting the two variables into (17) completes the proof. \square

6.1.2 A Tight Bound with Known Distribution

The bound by Theorem 6.1 is loose as it is for a general input. It can be tightened if the underlying distribution is known. In the time-invariant voltage scaling, the load is

modeled as a linear combination of input and a voltage scaling function $h(t)$. The output distribution $p(l(t))$ can be readily obtained by convolutions of the input distribution $p(\tilde{w}(t))$ [38]. Once we get the output distribution, the overload probability v can be calculated by considering its tail distribution $v = \text{prob}(l(t) > f_{max}) = \int_{f_{max}}^{\infty} p(l(t)) dl(t)$. Similarly, we can calculate the capacity requirement f_{max} with a given overload probability v as $f_{max} = P^{-1}(1 - v)$, where $P(\cdot)$ is the CDF of load distribution $p(\cdot)$.

As the convolution is computationally expensive, we can use a histogram-based technique in estimating the output distribution to simplify the analysis. A histogram of output load can be obtained by a profiling or online monitoring in a time window. The output values are a series of numbers ranging from the minimum l_{min} to the maximum number of cycles per unit time l_{max} . We put the numbers into m equal size bins. Each bin contains $(l_{max} - l_{min})/m$ values, denoted as l_m . Let n_i be the number of values in the i th bin. The number $\frac{n_i}{n}$ denotes the percent of values that is in the range $[l_{min} + i \cdot l_m, l_{min} + (i + 1) \cdot l_m]$. The cumulative distribution function of the output load can be estimated as $P(l_{min} + (i + 1) \cdot l_m) = \sum_{j=0}^i \frac{n_j}{n}$. With an overload probability setting as v , we can find the index of the bin by

$$\begin{aligned} v &= \text{prob}(l(t) > l_{min} + (i + 1) \cdot l_m) \\ &= 1 - P(l_{min} + (i + 1) \cdot l_m). \end{aligned}$$

The boundary of the i th bin $l_{min} + (i + 1) \cdot l_m$ can be used as a tight capacity bound corresponding to the overload setting. The histogram-based technique has been shown to be effective in estimating the distribution of task cycle demands in DVS [27], [36]. In contrast, we use it to estimate the system load distribution and bound deadline misses.

6.2 Experimental Results

We next demonstrate the effectiveness of the analytical results in providing a controllable deadline guarantee.

6.2.1 Capacity Configuration

Consider the sporadic task set used in the previous experiments with 20 tasks and the minimal interseparation time 10 ms. Task deadline was set to the minimal interarrival. The worst-case execution cycles were uniformly drawn from [10, 100] K. According to the schedulability test based on the worst-case scenario (16), the capacity for the sporadic task set can be computed as $\sum_{i=1}^{20} w_i/10ms \leq f_{max}$. For the task set we used in the following experiment, the sum of all task sizes is 1,230 K cycles. The minimum capacity 123 MHz was chosen as the worst-case configuration.

To relax the capacity bound, we first estimate the output load distribution for both TI-DVS and TV-DVS. Then, a tight capacity bound is computed. The histogram-based technique is used for simplicity. Fig. 10 shows the capacity bounds with varied mean interarrival times given an overload probability 1 percent. The bound computed with a known distribution cuts the worst-case bound in half for most interarrival times. The improvement increases with a larger interarrival because the worst-case scenario becomes less likely to take place. The bound is reduced to as far as 25 percent when the mean interarrival reaches 100 ms. A

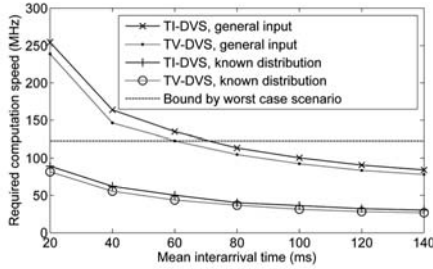


Fig. 10. Comparison of required CPU computation speed.

loose upper bound for general input from Theorem 6.1 is also included for comparison. The bound is loose as it assumes less input statistical information and works for a general input distribution. In addition, it does not consider the 10 ms minimal interarrival in the experimental setting. However, it can still give tightened capacity bounds for more than half of the test cases. Another observation is that the TV-DVS scheduling consistently outperforms TI-DVS from 6 percent to 15 percent. This implies TV-DVS is more effective in reducing load variance.

6.2.2 Deadline Misses

The capacity bounds can be tightened compared with the bounds based on the worst-case scenario. However, this is achieved by allowing deadline misses. We investigated the amount of deadline misses by a given tightened capacity configuration. As the system may become unstable in the presence of overload, we implemented two types of overload handling strategies. One is to reject the job resulting in overload so that hard real-time support is provided to all admitted jobs. The other is to provide soft real-time support in which the overload part of the job is put in a FIFO queue and scheduled in a best-effort manner whenever the system has enough resources. The processor is run at its maximum speed until the waiting queue is empty. After that, we continue to scale the processor using the scaling policies.

We generated the input sporadic task with mean interarrival time 80 ms. The corresponding capacity using TI-DVS is 113.6 MHz for a general input; 40 MHz when the underlying distribution is known. Both capacities are computed with a target overload probabilities 1 percent. The bounds are tightened more than the worst-case estimate, as can be observed from Fig. 10. Similarly, the capacity is 105.7 MHz for a general input and 37 MHz with known distribution using TV-DVS. We experimented with the general bounds and found no system overload or deadline misses for both TI-DVS and TV-DVS. This shows

the conservativeness of the estimate because of the target 1 percent overload probability.

The simulation results of scheduling with a tightened capacity bound are shown in Table 2. By buffering jobs resulting in overload and scheduling the overloaded part later, we have slightly heavier load using the best-effort overload handling. Similarly, the mean completion time is also higher as more jobs are admitted. Because of occasional overload, 0.63 percent of the jobs are rejected in TI-DVS and 0.61 percent in TV-DVS. The same amount of jobs miss their deadlines with the best-effort strategy. Both numbers equal the actual overload because both rejection and job delay occur only at the time of system overload. Since the best-effort approach runs overloaded jobs only at the time when the system has enough resources, it does not increase the overload number. In both cases, the deadline miss rates are effectively bounded by the 1 percent overload probability in simulation setting.

7 RELATED WORK

There is extensive research in real-time systems on DVS. A major focus is on the periodic task model under RM or EDF scheduling. A general approach is to allocate CPU resource to applications based on their WCETs with known job release times and deadlines [1], [9], [20], [22], [29], [41]. Online slack distribution techniques are then applied to reclaim slacks when tasks finish earlier. For example, Mejía-Alvarez et al. assigned speed to a group of periodic tasks by mapping the energy minimization to an NP-hard Multiple-Choice Knapsack Problem and proposed approximated solutions [20]. Pillai and Shin proposed a Static Voltage Scaling algorithm to scale the processor voltage by a factor equal to the minimum utilization required for a group of periodic tasks to remain schedulable [22]. Cycle-conserving (CC) and Look-Ahead algorithms were proposed to claim unused CPU time. Zhu and Mueller improved the slack reclamation for dynamic workload by a feedback controller within the EDF scheduling algorithm [41]. Aydin et al. used a similar voltage scaling approach for periodic tasks based on WCET [1]. They applied a priority-based slack distribution utilizing the unused execution times of high priority tasks. The basic idea was later applied to periodic tasks with nonpreemptible sections [37], to a hard real-time task graph with precedence constraints for a multiprocessor environment [26], [40], and to slack distribution between processor slowdown and shutdown [12]. In this paper, we study a general task model in a single-processor system considering processor slowdown.

TABLE 2
Statistics of TI-DVS/TV-DVS with a Tightened Capacity (Interarrival $\sim \exp(80ms)$)

Scheduling	TI-DVS+Rejection	TI-DVS+Besteffort	TV-DVS+Rejection	TV-DVS+Besteffort
Load mean	21.6	21.7	21.6	21.74
Load variance	166.7	168.9	141.3	142.8
Completion time mean	10.1	10.09	10.4	10.07
Completion time var.	8.7	8.4	8.4	8.8
Overload/deadline misses	0.63%	0.63%	0.61%	0.61%

Another research focus on DVS is for algorithms in support of real-time aperiodic tasks [11], [15], [19], [24], [31], [34]. For example, Yao et al. provided an optimal preemptive offline real-time scaling algorithm for a set of independent aperiodic tasks with arbitrary arrival times and deadlines on a variable speed processor [34]. Its time complexity is $O(n^2)$ (or $O(n \log^2 n)$ for a sophisticated implementation), where n is the number of tasks. Manzak and Chankrabari presented an algorithm that could be used for both periodic and aperiodic tasks with prior knowledge of job timing information [19]. Quan and Hu extended the algorithm in [34] by a near-optimal *fixed-priority* scheduling algorithm with the same assumption that all timing parameters are known offline [24]. The effect of a limited number of available processor speeds was considered in [11], [15], [31], [2]. For example, Swaminathan and Chakrabarty modeled energy minimization using DVS as a network flow graph for both periodic and aperiodic tasks and solved it optimally in moderate running time [31]. Kwon and Kim proposed an optimal procedure to transform the scheduling resulting from [34] with a continuous range of voltage levels into that with a limited number of levels [15]. The procedure was based on the results of [11], which showed that an optimal voltage allocation is to use the two voltages that are the immediate neighbors to the ideal voltage.

The above algorithms for aperiodic tasks are offline because they assume known job timing information before job releases. Aperiodic tasks usually have arbitrary release times, which calls for online DVS algorithms. An online scheduling heuristic approach, called Average Rate heuristic (AVR), was proposed also by Yao et al. for aperiodic tasks [34]. Each task is associated with its average-rate requirement, defined by dividing its required number of cycles by its relative deadline. Sinha and Chandrakasan proposed an algorithm for aperiodic task scheduling [30]. They assumed a stationary processor utilization over the scheduling slots, which is not always valid. Both algorithms may result in deadline misses by scaling the processor for energy reduction. With a focus on energy saving, the impact on deadline misses was not analyzed and no feasibility condition was used to guarantee schedulability.

Online algorithms in [10], [28], [23], [16] can provide hard real-time support to aperiodic tasks. Hong et al. considered a mixed real-time workload of both aperiodic (online) and periodic (offline) tasks [10]. They used an online acceptance test for both tasks to provide hard real-time guarantee (OPASTS). Its time complexity is pseudo-polynomial with the number of jobs in a hyperperiod. Algorithms with reduced complexity were also considered for a set of aperiodic tasks (STS) with $O(n^2)$ for scheduling of n ready tasks ($O(n)$ time averaged for each task). In this paper, we proposed a more efficient algorithm (TV-DVS), which only needs $O(n)$ for speed assignment of n tasks. In addition, STS may reject more aperiodic jobs by scaling the processor without knowledge of future job releases. The impact of the algorithms on rejection was not addressed. The proposed relation in this paper between computation capacity and an overload probability can be used to provide a bound for rejection ratio. Finally, they did not consider the

fact that a job can finish earlier than its WCET. In contrast, we propose a unified efficient solution to both WCET-based scaling and online slack management.

There exist efficient online algorithms for aperiodic tasks without deadline misses [28], [23], [16]. Sharma et al. investigated the use of DVS (RTDVS) in Web servers [28]. They applied a synthetic utilization bound for a set of *fixed-priority* aperiodic tasks to guarantee no deadline misses. However, their utilization bound is only a sufficient condition, which implies that a larger utilization does not necessarily lead to CPU overload. As a result, they may conservatively deny more requests than necessary. We consider a dynamic priority system to fully exploit the energy saving opportunities.

Qadi et al. considered a special type of hard real-time task, referred to as a canonical sporadic task model with minimum job interrelease time [23]. They proposed a CPU scaling algorithm based on task WCET that could guarantee all tasks meet their deadlines and proved its optimality when only CPU frequency could be scaled. In this work, we proposed a scaling policy that could lead to more energy savings when processor voltage can be changed dynamically with frequency setting. In [23], deadline guarantee is provided by a schedulability test requiring the summed utilization of all tasks released at their maximum rates smaller than one. But, the test is conservative as the interarrival time of a sporadic task can vary widely and the worst-case utilization may seldom take place. In this case, it is more efficient to let the worst-case utilization exceed one in which the schedulability condition fails to provide any guarantee. By considering the frequency with which the worst-case scenario happens, we allow a controllable deadline miss by investigating the relationship between computation capacity and deadline miss rate. The computation capacity can be greatly reduced with a small QoS constraint.

A general task model was recently studied by Lee and Shin [16]. They proposed an algorithm, referred to as OLDVS, focusing on slack management when jobs finish earlier. In contrast, we proposed a unified solution for both WCET-based scheduling and online slack distribution. The solution for online slack distribution outperforms OLDVS by allowing slack distribution to multiple ready jobs.

We note that there is work on stochastic analysis of periodic real-time systems, but with no energy considerations; see [5], [32] for examples. There also exists stochastic analysis for DVS dealing with the uncertainty of execution time demand [9], [18], [36]. They assume or measure the demand distribution of applications and adjust CPU speed based on the distribution. In our approach, instead of statistical analysis of input time demand, we analyze the tail distribution of system load.

A categorization of the most closely related investigations for online aperiodic tasks is summarized in Table 3 in terms of

1. whether the schemes provide a real-time guarantee,
2. whether they generate a schedule based on WCET,
3. whether they support online slack management when tasks finish earlier,

TABLE 3
Comparing the Most-Related Studies for Online Aperiodic Tasks with the Proposed TV-DVS

Algorithms	Real-time guarantee?	Schedule based on WCET?	Slack management?	Energy optimal?	Complexity
AVR [34]	No	Yes	No	No	$O(1)$
STS [10]	Hard	Yes	No	Yes	$O(n^2)$
RTDVS [28]	Hard	Yes	No	No	$O(1)$
DVSST [23]	Hard	Yes	No	No	$O(1)$
OLDVS [16]	Hard	No	Yes	No	$O(1)$
TV-DVS	Hard / Statistical	Yes	Yes	Yes	$O(n)$

4. whether they optimize energy consumption among all online algorithms without knowledge of future task releases, and
5. time complexity of the algorithm.

8 CONCLUSION

In this paper, we have presented a filtering model to characterize the relationship of input process, scheduling, and system load. The model facilitates the derivation of two DVS algorithms, time-invariant and time-variant. The time-invariant algorithm is proved to be a generalization of several existing approaches for different task models with respect to energy savings. The time-variant algorithm is essentially a water-filling process. It is more efficient and easily integrated into existing schedulers. It is effective in energy savings for WCET-based schedules as well as online slack distribution. We further provide statistical performance guarantees to bound the deadline miss rate with reduced computation capacity. Two bounds are derived, depending on different amounts of required statistical information. One is a loose upper bound for a general distribution with limited first and second moments. The other is tight based on load distribution.

We have limited our focus to minimizing the dynamic power consumption of a processor. It is expected that the leakage will significantly affect the overall energy consumption in a CMOS circuit [6]. The impact of leakage power has been studied using both continuous speed levels [25] and discrete ones [13]. A recent generalization of existing approaches with a practical processor model was presented in [2]. It has been shown that the existence of leakage power causes a critical speed under which it is no longer energy efficient for the overall energy consumption [13]. The proposed solutions in this paper are still optimal if the derived speeds are not lower than the critical speed. Otherwise, the speeds should be rounded to the speed for energy-efficiency, which destroys the optimality of the solution. However, the existence of critical speed introduces another constraint in getting the optimal solution. An exact analysis of its impact is beyond the scope of this paper. We investigated this issue and the impact of I/O standby power in [39].

ACKNOWLEDGMENTS

This research was supported in part by US National Science Foundation grants ACI-0203592, CCF-0611750, and DMS-0624849, and NASA grant 03-OBPR-01-0049. Part of the results were published in [38]. The authors thank the reviewers for their valuable suggestions.

REFERENCES

- [1] H. Aydin, R.G. Melhem, D. Mossé, and P. Mejía-Alvarez, "Power-Aware Scheduling for Periodic Real-Time Tasks," *IEEE Trans. Computers*, vol. 53, no. 5, pp. 584-600, May 2004.
- [2] E. Bini, G. Buttazzo, and G. Lipari, "Speed Modulation in Energy-Aware Real-Time Systems," *Proc. Euromicro Conf. Real-Time Systems*, pp. 3-10, 2005.
- [3] T. Cover and J. Thomas, *Elements of Information Theory*, pp. 250-253. John Wiley & Sons, 1991.
- [4] J.K. Dey, J.F. Kurose, and D.F. Towsley, "On-Line Scheduling Policies for a Class of Iris (Increasing Reward with Increasing Service) Real-Time Tasks," *IEEE Trans. Computers*, vol. 45, no. 7, pp. 802-813, July 1996.
- [5] J.L. Díaz, D.F. García, K. Kim, C.-G. Lee, L.L. Bello, J.M. López, L. Min, and O. Mirabella, "Stochastic Analysis of Periodic Real-Time Systems," *Proc. IEEE Real-Time Systems Symp.*, pp. 289-300, 2002.
- [6] D. Duarte, N. Vijaykrishnan, M.J. Irwin, H.S. Kim, and G. McFarland, "Impact of Scaling on the Effectiveness of Dynamic Power Reduction Schemes," *Proc. Int'l Conf. Computer Design*, pp. 382-387, 2002.
- [7] W. Feller, *An Introduction to Probability Theory and Its Applications (Volume II)*. John Wiley & Sons, 1971.
- [8] S. Gochman, R. Ronen, I. Anati, A. Berkovits, T. Kurts, A. Naveh, A. Saeed, Z. Sperber, and R.C. Valentine, "The Intel Pentium M Processor: Microarchitecture and Performance," *Intel Technology J.*, vol. 7, no. 2, pp. 31-36, 2003.
- [9] F. Gruian, "Hard Real-Time Scheduling for Low-Energy Using Stochastic Data and DVS Processors," *Proc. Int'l Symp. Low-Power Electronics and Design*, pp. 46-51, 2001.
- [10] I. Hong, M. Potkonjak, and M.B. Srivastava, "On-Line Scheduling of Hard Real-Time Tasks on Variable Voltage Processor," *Proc. Int'l Conf. Computer-Aided Design*, pp. 653-656, 1998.
- [11] T. Ishihara and H. Yasuura, "Voltage Scheduling Problem for Dynamically Variable Voltage Processors," *Proc. Int'l Symp. Low-Power Electronics and Design*, pp. 197-202, 1998.
- [12] R. Jejurikar and R.K. Gupta, "Dynamic Slack Reclamation with Procrastination Scheduling in Real-Time Embedded Systems," *Proc. Design Automation Conf.*, pp. 111-116, 2005.
- [13] R. Jejurikar, C. Pereira, and R.K. Gupta, "Leakage Aware Dynamic Voltage Scaling for Real-Time Embedded Systems," *Proc. Design Automation Conf.*, pp. 275-280, 2004.
- [14] M.A. Khojastepour and A. Sabharwal, "Delay-Constrained Scheduling: Power Efficiency, Filter Design, and Bounds," *Proc. IEEE Infocom*, pp. 1938-1949, 2004.
- [15] W.-C. Kwon and T. Kim, "Optimal Voltage Allocation Techniques for Dynamically Variable Voltage Processors," *ACM Trans. Embedded Computing Systems*, vol. 4, no. 1, pp. 211-230, 2005.
- [16] C.-H. Lee and K.G. Shin, "On-Line Dynamic Voltage Scaling for Hard Real-Time Systems Using the EDF Algorithm," *Proc. IEEE Int'l Real-Time Systems Symp.*, pp. 319-327, 2004.
- [17] J.W. Liu, *Real-Time Systems*. Prentice Hall, 2000.
- [18] J.R. Lorch and A.J. Smith, "Improving Dynamic Voltage Scaling Algorithms With PACE," *Proc. ACM SIGMETRICS Conf.*, pp. 50-61, 2001.
- [19] A. Manzak and C. Chakrabarti, "Variable Voltage Task Scheduling Algorithms for Minimizing Energy/Power," *IEEE Trans. Very Large Scale Integration Systems*, vol. 11, no. 2, pp. 270-276, 2003.
- [20] P. Mejía-Alvarez, E. Levner, and D. Mossé, "Adaptive Scheduling Server for Power-Aware Real-Time Tasks," *ACM Trans. Embedded Computing Systems*, vol. 3, no. 2, pp. 284-306, 2004.
- [21] P. Peebles Jr., *Probability, Random Variables, and Random Signal Principles*, chapter 6. McGraw-Hill, 2001.

- [22] P. Pillai and K.G. Shin, "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems," *Proc. 18th Symp. Operating Systems Principles*, pp. 89-102, 2001.
- [23] A. Qadi, S. Goddard, and S. Farritor, "A Dynamic Voltage Scaling Algorithm for Sporadic Tasks," *Proc. IEEE Real-Time Systems Symp.*, pp. 52-62, 2003.
- [24] G. Quan and X.S. Hu, "Energy Efficient Fixed-Priority Scheduling for Real-Time Systems on Variable Voltage Processors," *Proc. Design Automation Conf.*, pp. 828-833, 2001.
- [25] G. Quan, L. Niu, S. Hu, and B. Mochocki, "Fixed Priority Scheduling for Reducing Overall Energy on Variable Voltage Processors," *Proc. IEEE Real-Time Systems Symp.*, pp. 309-318, 2004.
- [26] D. Roychowdhury, I. Koren, C.M. Krishna, and Y.-H. Lee, "A Voltage Scheduling Heuristic for Real-Time Task Graphs," *Proc. Int'l Conf. Dependable Systems and Networks*, pp. 741-750, 2003.
- [27] C. Rusu, R. Xu, R.G. Melhem, and D. Mossé, "Energy-Efficient Policies for Request-Driven Soft Real-Time Systems," *Proc. Euromicro Conf. Real-Time Systems*, pp. 175-183, 2004.
- [28] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu, "Power-Aware QoS Management in Web Servers," *Proc. IEEE Real-Time Systems Symp.*, pp. 63-72, 2003.
- [29] Y. Shin and K. Choi, "Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems," *Proc. Design Automation Conf.*, pp. 134-139, 1999.
- [30] A. Sinha and A.P. Chandrakasan, "Energy Efficient Real-Time Scheduling," *Proc. Int'l Conf. Computer-Aided Design*, pp. 458-470, 2001.
- [31] V. Swaminathan and K. Chakrabarty, "Network Flow Techniques for Dynamic Voltage Scaling in Hard Real-Time Systems," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 10, pp. 1385-1398, 2004.
- [32] T.-S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J.W.-S. Liu, "Probabilistic Performance Guarantee for Real-Time Scheduling with Varying Computation Times," *Proc. IEEE Real-Time Technology and Applications Symp.*, pp. 164-173, 1995.
- [33] E. Uysal-Biyikoglu and A.E. Gamal, "On Adaptive Transmission for Energy Efficiency in Wireless Data Networks," *IEEE Trans. Information Theory*, vol. 50, no. 12, pp. 3081-3094, 2004.
- [34] F. Yao, A. Demers, and A. Shenker, "A Scheduling Model for Reduced CPU Energy," *Proc. IEEE Symp. Foundations of Computer Science*, pp. 374-382, 1995.
- [35] W. Yu, W. Rhee, S. Boyd, and J.M. Cioffi, "Iterative Water-Filling for Gaussian Vector Multiple-Access Channels," *IEEE Trans. Information Theory*, vol. 50, no. 1, pp. 145-152, 2004.
- [36] W. Yuan and K. Nahrstedt, "Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems," *Proc. 19th ACM Symp. Operating System Principles*, pp. 149-163, 2003.
- [37] F. Zhang and S.T. Chanson, "Blocking-Aware Processor Voltage Scheduling for Real-Time Tasks," *ACM Trans. Embedded Computing Systems*, vol. 3, no. 2, pp. 307-335, 2004.
- [38] X. Zhong and C.-Z. Xu, "Energy Aware Modeling and Scheduling of Real-Time Tasks for Dynamic Voltage Scaling," *Proc. IEEE Real-Time Systems Symp.*, pp. 366-375, 2005.
- [39] X. Zhong and C.-Z. Xu, "System-Wide Energy Minimization for Real-Time Tasks: Lower Bound and Approximation," *Proc. Int'l Conf. Computer-Aided Design*, 2006.
- [40] D. Zhu, R.G. Melhem, and B.R. Childers, "Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multiprocessor Real-Time Systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 14, no. 7, pp. 686-700, July 2003.
- [41] Y. Zhu and F. Mueller, "Feedback EDF Scheduling Exploiting Dynamic Voltage Scaling," *Proc. IEEE Real-Time and Embedded Technology and Applications Symp.*, pp. 203-212, 2004.



Xiliang Zhong received the BS degree in radio engineering from Southeast University, China, in 1997 and the MS degree in electrical engineering from Beijing University of Posts and Telecommunications, China, in 2000. He worked as a software engineer at the Zhongxing Telecom Corporation from 2000 to 2002. He is currently a PhD student in the Department of Electrical and Computer Engineering, Wayne State University, Detroit, Michigan. His current research interests include power management in embedded systems, mobile computing, and resource management in distributed systems. He is a student member of the IEEE.



Cheng-Zhong Xu received the BS and MS degrees in computer science from Nanjing University in 1986 and 1989, respectively, and the PhD degree in computer science from the University of Hong Kong in 1993. He is an associate professor in the Department of Electrical and Computer Engineering at Wayne State University, Detroit, Michigan. His research interests are in distributed and parallel systems, particularly in scalable and secure Internet services, adaptive and highly reliable networked computer systems, and resource management in cluster and grid computing. He has published more than 100 peer-reviewed articles in journals and conference proceedings in these areas. He is the author of the book *Scalable and Secure Internet Services and Architecture* (Chapman & Hall/CRC Press, 2005) and a coauthor of the book *Load Balancing in Parallel Computers: Theory and Practice* (Kluwer Academic/Springer-Verlag, 1997). He serves on the editorial boards of the *Journal of Parallel and Distributed Computing*, *Journal of Parallel, Emergent, and Distributed Systems*, *Journal of High Performance Computing and Networking*, and *Journal of Computers and Applications*. He was the founding program cochair of the International Workshop on Security in Systems and Networks (SSN), the general cochair of the IFIP 2006 International Conference on Embedded and Ubiquitous Computing (EUC '06), and a member of the program committees of numerous conferences. His research was supported in part by the US National Science Foundation and NASA. He is a recipient of the Faculty Research Award of Wayne State University in 2000, the President's Award for Excellence in Teaching in 2002, and the Career Development Chair Award in 2003. He is a senior member of the IEEE.

► For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.