Energy-Aware Modeling and Scheduling of Real-Time Tasks for Dynamic Voltage Scaling

Xiliang Zhong and Cheng-Zhong Xu
Department of Electrical and Computer Engineering
Wayne State University, Detroit, Michigan 48202
{xlzhong, czxu}@wayne.edu

***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 scheduled task set. This paper presents an analytical model of general tasks for DVS assuming job timing information is known only after task releases. The voltage scaling process is modeled as a transfer function-based filter system. The filtering model facilitates the design of two efficient scaling algorithms. The first is a time-invariant scaling policy with a constant time complexity based on a voltage scaling function independent of input jobs over time. Several existing approaches are special cases of the policy with respect to energy savings. A time-variant policy is derived for more energy savings with a time complexity of $O(N)$, where N is the number of distinct deadlines. It can be not only applied to scheduling based on worst case execution times, but also to on-line slack distribution when jobs complete earlier. We further establish two relationships between computation capacity and deadline misses. The relationships make it possible to the provisioning of statistical real-time guarantees.**

I. 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. Dynamic voltage scaling (DVS) is an effective approach to power reduction by scaling the processor voltage and frequency when the system is not fully loaded. Chip makers like Intel, AMD, and Transmeta have commercial processors with this DVS feature.

It is known that the energy consumed by a CMOS microprocessor is dominated by its dynamic power consumption, which is proportional to the square of its operating voltage ($E \propto V^2$) [4]. Reducing the voltage also drops the maximum operating frequency approximately proportionally ($V \propto f$). Thus energy consumption can be approximated as being proportional to the frequency squared ($E \propto f^2$), which means substantial energy can be saved by operating at a lower frequency and setting the voltage accordingly. On the other hand, a reduction of the operating frequency leads to an increase of service time. A challenge of applying DVS algorithms to delay-sensitive applications is to achieve maximum energy savings while still meeting all temporal requirements of the system [24]. 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 the application time [27]. This makes it possible for DVS algorithms to reduce the processor frequency and still ensure no tasks miss their deadlines.

There have been lots of 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 job timing information including release time, execution time (at the maximum available processor speed), and deadline is known in advance [2], [10], [22], [23], [30], [41]. Many practical real-time applications involve both types of periodic and aperiodic tasks. Algorithms in support of aperiodic tasks can be found in [12], [17], [21], [25], [32], [35] 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 on-line scheduling algorithms without assumed timing knowledge before job arrivals.

In addition to the power delay trade-off, there are two other design challenges for on-line aperiodic tasks scheduling. The first is to provide an efficient voltage scaling algorithm with a low time complexity. This is important as the algorithm needs to be invoked at every context switch. An efficient scaling algorithm also lends itself to be easily integrated into a real-time operating system with low overhead. The second 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 on-line strategies in support of real-time aperiodic tasks do not deal with all the three issues. For example, the slacked EDF in [31] is targeted at energy minimization at the cost of occasional deadline misses. The algorithm in [28] applies a synthetic utility bound to conserve energy in a fixed-priority system with a constant time complexity. DVSST in [24] focuses on a processor that only provides frequency scalings. The energy savings can be improved if we consider a dynamic priority system with a processor capable of dynamic voltage adjustment. The optimal periodic and sporadic task scheduling in [11] provides maximum energy saving in a pseudo-polynomial time complexity. Readers are referred to Section VII for a concise review of the algorithms. All the algorithms are insightful to voltage scaling for real-time aperiodic tasks by assuming accurate execution time

estimation available after a job is released. They do not consider the effect of workload variation. In this paper, we tackle these design issues for real-time aperiodic tasks using a model-based approach. Job releases (or arrivals) are modeled as a random process with a general distribution. The voltage scaling is modeled as a transfer function-based filter system, which characterizes output load as a convolution of aggregated input requests. The filtering model facilitates the design of two efficient voltage scaling algorithms.

The first is a time-invariant scaling policy based on a voltage scaling function, independent of input jobs over time. We prove that AVR for aperiodic tasks [35], DVSST for sporadic tasks [24], static voltage scaling for periodic tasks [2], [23], are special cases of the scaling with respect to energy consumption. The second is a more energy-efficient time-variant scaling algorithm. The algorithm shows a water-filling structure of information theory [5] with a time complexity of $O(N)$, where N is the number of distinctive deadlines and smaller than the number of tasks. The scaling algorithm can not only be applied to WCET-based scheduling, but also to on-line slack distribution when jobs do not consume all their WCETs.

To provide deadline guarantees for real-time tasks, a schedulability test is often used. The schedulability test in [24] for a set of sporadic tasks considers the worst case scenario when all tasks are released at their maximum rate. This is conservative because a sporadic task usually has a much lower average release rate than its maximum. The maximum processor speed may be over-provisioned. In light of this, we establish relationships between capacity and deadline miss rate and provide statistical deadline guarantees to reduce the computation capacity.

The rest of the paper is organized as follows. In Section II, we define an analytic model for DVS. In Section III, we present a time-invariant voltage scaling. In Section IV, we propose a time-variant voltage scaling and show it is also effective for on-line slack distribution. Section V shows the effectiveness of the proposed scaling policies in energy savings by simulation. Statistical deadline guarantees due to capacity configuration are discussed in Section VI. Section VII reviews related work. Section VIII concludes the article.

II. A Filtering Model of Voltage Scaling

A. 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 scheduling time index. The input is viewed as a set of jobs released in a certain sequence, rather than a set of periodic/aperiodic tasks. Job arrivals are modeled as a general fluid type process $\{n(1), n(2), n(3), ..., n(t), ...\}$, where $n(t)$ is the number of jobs arrived between time $t-1$ and t. Request size of a job at time t, $w_i(t)$, is the required CPU cycles of the ith job that arrives during the last time slot. Taking the size into consideration, the input process can be written as:

$$\left\{\{w_i(t)\}_{i=1,2,..n(t)}\right\}_{t=0,1,...}.$$

The aggregated input process $\tilde{w}(t)$ at time t is the sum of $n(t)$ input requests. Let t_b denote the release (begin) time of a job, and t_d as the relative deadline. We assume job parameters are known after the job is released.

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 [12], [17]. The approaches actually enable a processor to run at any speed. We therefore do not explore the effect of 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 100μs, and the actual time period when the processor is unavailable in a transition is as short as 10μs [8]. The overhead is small compared to the execution times of real-time tasks. If it is not negligible, we assume the overhead is incorporated into the worst-case workload of each task. The energy function is first assumed as the widely used frequency square f^2 in Section III and later extended to a general increasing convex function, denoted as $\mathcal{P}(f)$ in Section IV.

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 $f(t)$. The energy consumed by a schedule S in any time interval $[t_0, t_1]$ is $E(S) = \int_{t_0}^{t_1} \mathcal{P}(f(t))dt$ [35]. 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 $f(t)$ at one time unit. Assume no speed change is made during time $(t, t+1)$, i.e., $f(t)$ is a constant. We rewrite the total energy in a discrete form as

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

The voltage scaling problem is to assign appropriate voltage levels and set corresponding processor speeds so that the energy consumption $E(S)$ is minimized while providing real-time guarantees.

B. The Filtering Model

We next show that voltage scaling activities can be modeled as a transfer function-based filter system. The CPU resource allocation is characterized by a time-varying function. Formally, it is a function of system time t, job release time t_b, job size $w_i(t)$, and the current load $l(t)$, denoted as $d(t, t_b, w_i(t), l(t))$. It represents the amount of CPU cycles allocated at each time slot for job i. In fact, the allocation function should also depend on job deadline t_d. However, we consider a job that completes before its deadline to effectively complete at its deadline to achieve maximum energy savings.

The deadline t_d then becomes a model parameter instead of a variable. We first consider the case when all tasks have the same deadlines and later extend the model to tasks with different deadlines.

With a specific scheduling policy, the adaptation of scheduling to job size and system load does not change over time. The allocation operation with an input $w_i(t)$ can be decomposed into three steps:

1) Determine the total amount of CPU cycles to be allocated to request $w_i(t)$, defined as $g(w_i(t))$. Generally, the allocated number of cycles is smaller or equal to the requested amount, i.e, $g(w_i(t)) \leq w_i(t)$;
2) Determine the number of cycles to be allocated at current time slot $[t, t+1)$. Denote $h(t, t_b)$ as the percent of cycles allocated at the current slot. The allocation at the slot to $w_i(t)$ is then $h(t, t_b) \cdot g(w_i(t))$;
3) Adjust the allocation to the current system load by a time-varying scaling factor $s(l(t))$. This leads to an adapted allocation as $(h(t, t_b) \cdot g(w_i(t))) \cdot s(l(t)$.

Hence, the allocation function $d(t, t_b, w_i(t), l(t))$ can be represented as a three-step process:

$$d(t, t_b, w_i(t), l(t)) = h(t, t_b)g(w_i(t))s(l(t)). \tag{2}$$

The system load at time t is equal to the sum of all CPU cycles allocated to the jobs that release from time $t - t_d$ to t. Thus the load function can be expressed as

$$\begin{aligned} l(t) &= \sum_{t_b=t-t_d}^{t} \sum_{i=1}^{n(t_b)} d(t, t_b, w_i(t), l(t)) \\ &= \sum_{t_b=t-t_d}^{t} \sum_{i=1}^{n(t_b)} h(t, t_b)g(w_i(t))s(l(t)). \end{aligned} \tag{3}$$

The defined load function is a general form of voltage scaling. In this paper, we focus on causal algorithms, which means no CPU cycles will be reserved before job releases, i.e., for all $t < t_b$, $h(t, t_b) = 0$. The scheduler tries to allocate CPU resource needed by all jobs, $g(w_i(t)) = w_i(t)$. We assume the scheduling is non-adaptive to system load which implies $s(l(t)) = 1$. The allocation can then be simplified as

$$\begin{aligned} l(t) &= \sum_{t_b=t-t_d}^{t} \sum_{i=1}^{n(t_b)} h(t, t_b) \cdot w_i(t) \\ &= \sum_{t_b=t-t_d}^{t} h(t, t_b) \cdot \tilde{w}(t_b), \end{aligned} \tag{4}$$

where

$$\tilde{w}(t_b) = \sum_{i=1}^{n(t_b)} w_i(t). \tag{5}$$

Given a job process, the properties of the compounded process $\tilde{w}(t_b)$ are derivable. In the case when both $n(t)$ and $w_i(t)$ are identical independent distributed (i.i.d.) random variables, the process $\tilde{w}(t_b)$ is a simple random process following a distribution of *random sum*. 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 [33]. Jobs in each task are similar in the sense that they share the same statistical behavior and the same timing requirement. Their interarrival times are constant for periodic tasks and i.i.d. random variables with a certain distribution for aperiodic tasks. Similarly, the execution times of jobs in each task are also i.i.d. random variables. The statistical behavior of the system does not change with time; that is, the system is stationary [19]. With known job interarrival and execution time distributions of a task set, it is easy to get marginal distribution and autocorrelation of the compound input process.

Meanwhile, the amount of CPU cycles consumed by a job must be equal to request size $w_i(t)$. In other words, the scheduler will always allocate enough amount of cycles to a job by its specified deadline t_d. Similar to the load function, we have

$$\begin{aligned} w_i(t) &= \sum_{t=t_b}^{t_b+t_d} d(t, t_b, w_i(t), l(t)) \\ &= \sum_{t=t_b}^{t_b+t_d} h(t, t_b)g(w_i(t))s(l(t)) = \sum_{t=t_b}^{t_b+t_d} h(t, t_b)w_i(t). \end{aligned}$$

That is

$$\sum_{t=t_b}^{t_b+t_d} h(t, t_b) = 1. \tag{6}$$

The load function (4) and the scheduling (scaling) function (6) together determine the voltage scaling process.

III. Time-Invariant Voltage Scaling

If the scheduling process is time-invariant, in the sense that the scaling function $h(t)$ does not adapt in response to input jobs over time, we have $h(t, t_b) = h(t - t_b)$. We can further simplify the load function (4) as

$$l(t) = \sum_{t_b=t-t_d}^{t} h(t - t_b) \cdot w(t_b) = h(t) * w(t), \tag{7}$$

where "*" is a convolution operator. Similarly, the scaling function (6) becomes

$$\sum_{t=t_b}^{t_b+t_d} h(t - t_b) = 1 \ or \ \sum_{t=0}^{t_d} h(t) = 1. \tag{8}$$

Equations (7) and (8) represent the scheduling process as a perfect format 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 t_d number of input requests.

Define $\mathbf{h}(t) = [h(t), h(t+1), ..., h(t+t_d-1)]'$ and $\mathbf{w}(t) = [\tilde{w}(t), \tilde{w}(t-1), ..., \tilde{w}(t-t_d+1)]$ as vector forms of the scaling functions and aggregated job sizes. Define

$\Omega(t) = E[\mathbf{w}(t)\mathbf{w}'(t)]$ as the covariance matrix of input process $\tilde{w}(t)$ in the order of t_d. Denote $\mathbf{u}$ as a row vector with t_d components equal to 1. We present the optimal time-invariant voltage scaling policy in minimizing energy consumption in Theorem 3.1. Its proof can be seen in [39].

Theorem 3.1: The optimal time-invariant voltage scaling has a unique solution $\mathbf{h}^*$ that minimizes energy consumption. The solution is

$$\mathbf{h}^* = \frac{\Omega^{-1}\mathbf{u}'}{\mathbf{u}\Omega^{-1}\mathbf{u}'}. \tag{9}$$

The solution shows that the scaling function can be determined by the covariance matrix of input process in the order of t_d. The matrix is an indication of job size correlation structure. The time-invariant scaling has a constant time complexity. A special case is when the compounded job 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 resource to a request at each time unit before its deadline.

The major assumptions that lead to a uniform allocation are independent input process and a square energy function. We are trying to relax the energy function assumption. The optimal scaling is not necessarily uniform in this case. We leave studies on the time-invariant scaling along the line in our future work. The uniform solution for input process can also be obtained by the on-line AVR for aperiodic tasks in [35]. In essence, it is a special case of the time-invariant scaling when the input process is independent over time. In [39], we also prove that an existing algorithm (DVSST in [24]) for sporadic task model and the static voltage scaling [2], [23] for periodic tasks are special cases of the time-invariant voltage scaling with respect to energy savings.

IV. 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 savings can be further reduced if we adapt the function in response to input over time. In this section, we describe an adaptive scaling policy and prove its optimality in minimizing energy consumption under the constraint that the scheduling is performed on-line with job request size available upon job release.

Let t_d denote the supreme of the deadlines of all the jobs. We make t_d separate running queues each containing jobs with the same absolute deadline. Scheduling is conducted in all the t_d queues in the order of increasing deadlines. Newly admitted jobs can be dispatched into respective queues according to their deadlines. Using absolute or relative deadlines generates the same queue backlog when dispatching requests arrived at the same time t. After jobs are served at current scheduling slot, jobs with a deadline of j will have a deadline of $j-1$. We then rotate the queues so that queue 1 becomes empty and queues 2, 3, ..., t_d become queues 1, 2, ..., $t_d - 1$. A job entering queue j at time t with a deadline of j will enter queue $j-1$ at time $t+1$, queue 1 at time $t+j-1$, and out of the system 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 loads of all queues, $\sum_{j=1}^{t_d} l_j(t)$.

A. The Optimal Time-Variant Voltage Scaling

Consider any time interval $[0, t_d]$. We start from time zero and omit the variable t for brevity. Define q_j as the backlog of queue j at time t. 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 S that

$$\text{Minimize } E(S) = \sum_{i=0}^{t_d-1} \mathcal{P}(\sum_{j=i+1}^{t_d} l_j(i)) \tag{10}$$

$$\text{Subject to } l_j(i) > 0,\ 0 \le i \le t_d - 1, 1 \le j \le t_d$$

$$\sum_{i=0}^{j-1} l_j(i) = q_j,\ 1 \le j \le t_d. \tag{11}$$

The optimal solution can be achieved by the following process for queue j, $1 \le j \le t_d$,

$$l_j(i) = (s_j - \sum_{k=i+1}^{j-1} l_k(i))^+,\ 0 \le i \le (j-1), \tag{12}$$

where the notion $(x)^+ = max(x, 0)$ and s_j is determined by evaluating the backlog constraint in (11).

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

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

It is the average of the energy consumption summed over all queues in the next t_d time slots. This shows the minimization of energy consumption is to minimize L in (10).

The constraint in (11) means 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

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

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

in [39] that there exists λ'_j satisfying

$$\sum_{k=i+1}^{j-1} l_k(i) + l_j(i) + \sum_{k=j+1}^{t_d} l_k(i) - \lambda'_j = 0$$

$$l_j(i) = \lambda'_j - \sum_{k=j+1}^{t_d} l_k(i) - \sum_{k=i+1}^{j-1} l_k(i). \quad (13)$$

The conditions in (13) give the solution for queue j at time i, depending on values of all other queues. This dependency makes it hard to solve. Define the variable $s_j = \lambda'_j - \sum_{k=j+1}^{t_d} l_k(i)$ for queue j. The conditions (13) become

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

Consider the fact that the load $l_j(i)$ is nonnegative. This leads to the condition in (12) with an unknown value s_j. The value can be determined by the constraint in (11) given queue backlog q_j. Therefore, we get the values of s_j and $l_j(i)$ that satisfy the condition in (13). Note that the conditions only give necessary conditions for the existence of a minimum. Since the optimization function is convex, the conditions are also sufficient. □

The variable s_j can be interpreted as the maximum system load, or processor speed similarly, that will be allocated for queue j during the next j time slots. The reserved cycle allocation at the ith time slot is $\sum_{k=i+1}^{j-1} l_k(i)$. The solution is simply a substraction between the cap s_j and the aggregated load. If the resulted load is negative, no more cycles will be allocated at the time slot. The key part of the solution is to determine the bound s_j given a queue backlog q_j. Once we get the bound, any scheduling policy that can fully utilize the processor can be used to obtain a feasible schedule, e.g., the schedule in (12) or EDF.

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. Algorithm 1 lists a pseudo-code of the process. By taking queue 5 as an example, Figure 1 illustrates the basic idea graphically. The vertical white areas indicate the accumulated reserved allocation at different time. 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 the queue backlog has been distributed, we get the bound s_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 [5]. Many problems in information theory can be solved by constructing solutions in a form of water-filling structure. Representative examples include multiuser power control in Digital Subscriber Lines [36], maximization of the sum of communication rate in single user scenarios [9], in multi-user environments subject to individual power constraints [37] or a sum power constraint [15], and minimization of data transmission energy in a Gaussian channel [16], in a fading channel adapting to varying channel states [7], in a fading channel adapting to both channel states and queue backlog [34]. To the best of our knowledge, this study is the first to relate delay-sensitive energy-efficient CPU scheduling to a water-filling process.

Algorithm 1 Pseudo code of the water-filling process for speed/voltage setting.

```
1: function SELECT_SPEED( )
2:     for j ← 2, t_d; tm ← 1, j − 1 do            ▷ Queues rotation
3:         load(j − 1, tm − 1) ← load(j, tm)
4:         if j = t_d then
5:             load(j, tm) ← 0
6:         end if
7:     end for
8:     for tm ← 0, t_d − 1 do                      ▷ Accumulated allocation
9:         sum_load[tm] = Σ_{j=tm+1}^{t_d} load[j, tm]
10:    end for
11:    for all backlog q_j with newly arrived requests, starting from
       queues with earliest deadline do
12:        s_j ← GET_LEVEL(sum_load, j, q_j)
13:        for i ← 0, j − 1 do
14:            load[j, i] ← load[j, i] + max(s_j − sum_load[i], 0)
15:        end for
16:    end for
17:    system_load(t) ← Σ_{j=1}^{t_d} load[j, 0]
18:    return speed ← system_load/system_max_load
19: end function

20: function GET_LEVEL(sum_load, j, q_j)
21:    level ← min(sum_load[0..j − 1])
22:    total ← level ∗ j;
23:    while total < q_j do
24:        level ← level + (q_j − total)/j;
25:        total ← Σ_{i=0}^{j−1} max((sum_load[i] − level), 0);
26:    end while
27:    return level;
28: end function
```

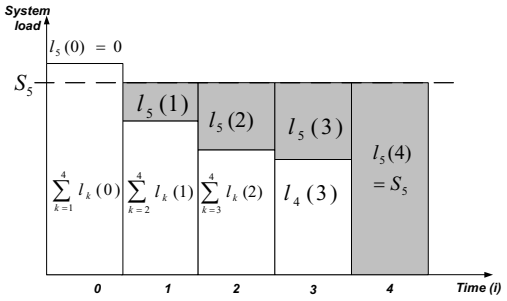

Fig. 1. Water-filling illustration of the scaling for queue 5.

The solution to the problem gives optimal load for all queues for the next t_d time slots. The sum of load from all queues at the current time, $\sum_{j=1}^{t_d} l_j(0)$, forms the optimal solution for current allocation. However, for all other time, the solutions are not necessarily optimal if there are new jobs released after the current time. It is because the new jobs may change the queue backlog and the optimization process needs to be performed again to get the optimal load. Since different jobs may have different delay constraints, successive water-filling must be performed sequentially for queues from low delay to high. The process treats all other queues yet to

be performed as empty queues. This guarantees no conflict between different queues. The time complexity of the process is determined by the number of distinctive deadlines of all ready jobs, denoted as N. In the worst case scenario, we may have t_d different deadlines. In all cases, N is smaller the number of tasks because tasks with the same deadline are aggregated to the same queue.

B. An Illustrative Example

Let us consider a set of three sporadic tasks $T_1(1,4)$, $T_2(2,4)$, $T_3(1,4)$, where the pair of parameters denotes the WCET at full speed and relative deadline. We assume there is a minimal inter-separation time 4. T_1 releases jobs at times 0 and 5; T_2 at times 1 and 7; T_3 at times 3 and 9. Let job $J_{i,j}$ denote the jth instance of task i. We compare the time-variant voltage scaling (referred as TimeVar) with DVSST [24] for jobs released during the first 10 time slots. Figures 2 and 3 illustrate the schedules under EDF without DVS and with DVSST.

Figure 4 shows the speed setting and schedule by TimeVar. At time $t = 0$, job $J_{1,1}$ is released and it is evenly distributed in $[0, 4)$ with a constant speed 0.25, as shown in Figure 4(a). At $t = 1$, job $J_{2,1}$ arrives and it is distributed starting from the latest allowable time-slot, $[4, 5)$. After the speed surpasses 0.25, the remaining job is equally distributed to the slots 1 to 5. A constant speed 0.69 was set during the time range $[1, 5)$, as in Figure 4(b). At $t = 3$, job $J_{3,1}$ is released. Similarly, we first distribute the workload to the farthest two slots, $[5, 7)$. The resulted speed 0.5 is smaller than 0.69 so we do not need to distribute the job to other slots. We get two speeds in $[3, 7)$ as in Figure 4(c): the first is 0.69 in $[3, 5)$; the other is 0.5 in $[5, 7)$. Although we do not get a constant speed during the time range, it is the best solution we can get at time 3 in terms of energy savings without known job release times. The complete schedule is shown in Figure 4(d) up to time 13. The policy adjusts frequency and voltage on each job release/finish and in each step a solution is obtained based on jobs not yet finished. With an energy model as speed square, TimeVar can save 35% of energy compared with EDF without DVS; 8% more energy savings compared with DVSST.

C. On-line Slack Management

Real-time applications usually have non-deterministic 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 [2]. The basic idea is later extended to different environments. Recently Lee and Shin [18] proposed an on-line slack management algorithm, called OLDVS, 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 first update remaining execution time of all ready jobs with lower priority. Then the solution process in (11) is used to compute the next speed setting. EDF is used to schedule jobs under the speed due to its optimality even for aperiodic tasks. As no task periodicity is assumed, the approach applies for general tasks. The solution can be illustrated using a simple example with three jobs released at time 0: $J_1(2,2)$, $J_2(2,4)$, $J_3(1,5)$, where the pair of parameters denotes WCET and relative deadline. Suppose the actual execution time of the first job is 1 and job 2, 3 take their WCETs. Figure 5(a) is a schedule without on-line slack distribution. During time $[1, 2)$, the processor is idle. Figure 5(b) shows the slack distribution using priority based slack stealing. In this simple example, both OLDVS and a slighted adapted DRA can give the schedule. Only job 2 claims the on-line slack in the schedule. However, job 2 and 3 are both ready for execution at time 1 and job 3 should also be considered in the slack distribution. TimeVar computes a new speed considering all ready jobs after job 1 finishes. Figure 5(c) is a schedule using TimeVar 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. Intuitively, the speed can be determined by redistributing the over-provisioned speed of job 3 evenly to the time interval $[1, 5)$, a way identical to the water-filling process of the time-variant scaling. TimeVar remains efficient even if job 2 or 3 finishes earlier. If there is only one job ready, TimeVar would give the same schedule as the priority based slack distribution.

Aydin *et al.* [2] argued that the benefit of DRA should come from assigning the entire slack to the *first* low-priority task and that this greedy assignment improved over the cycle-conserving technique of Pillai and Shin [23], which reduces the speed of *all* ready jobs in equal proportions. This approach was also used for a general task model by Lee and Shin [18]. TimeVar acts more greedily by considering all ready jobs and leads to more energy saving in comparison with assigning the slack to the first task. We will verify this by simulation. It is possible that a more aggressive speculation-based algorithms (e.g. [2]) could be adapted for slack distribution for WCET-based TimeVar. It is beyond the scope of this work.

In TimeVar, deadline misses occur only when the required computation speed is higher than the maximum processor speed. Intuitively, the resulted 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 and leave its proof in [39].

Theorem 4.2: Using TimeVar for on-line slack distribution preserves the feasibility of the WCET-based schedule.

V. 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

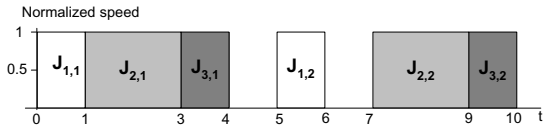

Fig. 2. Schedule for a sporadic task set using EDF w/o DVS.

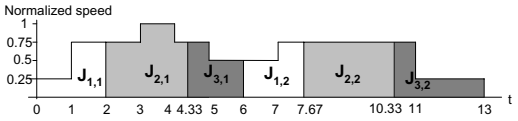

Fig. 3. Schedule for a sporadic task set using EDF with DVSST.

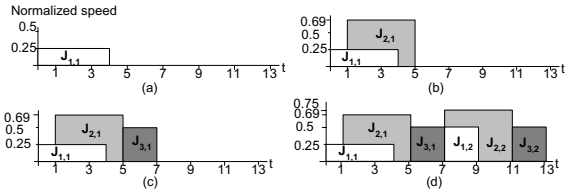

Fig. 4. Schedule for a sporadic task set using TimeVar

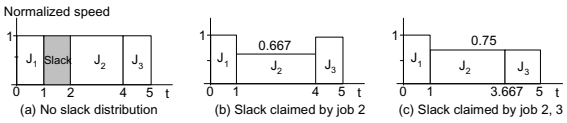

Fig. 5. Example solutions for slack distribution.

speed. We ignored the overhead to switch to power-down mode and assumed no energy was consumed in the power-down mode for simplicity. As the proposed approaches only uses processor slowdown, the relative performance of the approaches will be better if we consider the power-down overhead in comparison with EDF combined with power-down.

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.* [35]. It serves as a theoretical lower bound on energy savings.
- DVSST: An on-line processor scaling algorithm for a sporadic task model due to Qadi *et al.* [24].
- TimeInvar: The time-invariant voltage scaling.
- TimeVar: The time-variant voltage scaling.

To investigate the impact of variation in job execution times, three on-line 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 [23].
- TimeVar + OLDVS: The time-variant voltage scaling and the reclaiming algorithm of Lee and Shin [18].
- TimeVar + TimeVar: A unified solution for both the worst case schedule and slack distribution based on TimeVar.

We point out that the cycle conserving technique [23] is readily applicable to DVSST for on-line slack distribution. This is because DVSST scales processor speed using a frequency scaling factor, similar to the utilization approach. The only modification to the DVSST algorithm is to reduce the frequency factor by the amount of r_i/p_i at the time a job finishes earlier, where r_i and p_i are the unused WCET of the job and task deadline.

One unexpected result of the experiment is that although TimeInvar can take autocorrelation structure of input process into account, the structure has little effect on energy savings. We do not present the results of TimeInvar for energy savings to improve the readability of the figures.

A. Input with Deterministic Execution Times

We first evaluate the algorithms with a real-world application: the Robotic Highway Safety Marker (RSM) used in [24]. 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 theses 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 [24] in the simulation. The processor had four available frequency levels: 18.532MHz, 9.266MHz, 4.633MHz, and 2.3165MHz. More details of the task timing parameters can be found in [24].

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. Figure 6 shows the normalized energy consumption with respect to that of No-DVS. It shows 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 TimeVar leads to more than 40% energy savings compared with No-DVS, about 10% better than DVSST. One remarkable result is the consumed energy of TimeVar is close to the offline solution. The performance gap is around 5%. Recall that Offline requires a priori knowledge about both timing and size information of job releases and its time complexity is $O(n^2)$. In contrast, TimeVar is on-line and be easily integrated.

Since different voltage scaling algorithms may benefit from the task timing parameters differently, it is necessary to investigate different input patterns and processor utilization

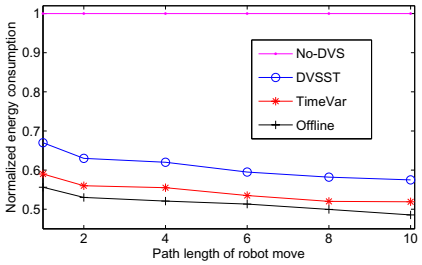

Fig. 6. Energy consumption with the Robotic Highway Safety Marker application.

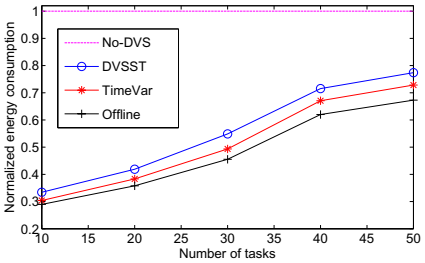

Fig. 7. Energy consumption of synthetic workload with different number of tasks.

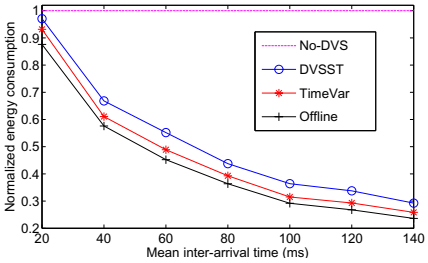

Fig. 8. Energy consumption of synthetic workload with different interarrival times.

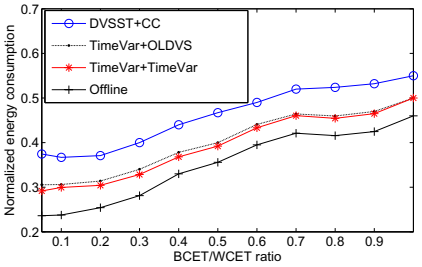

Fig. 9. Impact of variability in actual execution time.

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 number of tasks. Jobs interarrival time was assumed to follow an exponential distribution with mean 50 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, 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 Figure 7 and 8 show that all the DVS policies lead to significant energy savings with different task patterns. Although with slightly different performance gaps, TimeVar consistently outperforms DVSST and is close to Offline. When the processor has more idle intervals, characterized by small number of tasks and large inter-arrivals, all the DVS algorithms offer more energy savings. However, the energy savings gaps became 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 the DVS algorithms, which in turn leads to the minimum energy saving.

B. Impact of Variability in Actual Workload

Actual execution times of real-time applications are often non-deterministic and smaller than their WCETs. We investigate the impact of variability in the actual workload with on-line 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 [30]. Under this setting, 99.7% 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 inter-arrival time was set to 60 ms. Simulated energy consumption of the techniques is presented in Figure 9. When BCET/WCET=1, no slack distribution is possible. The performance is the same as the one in Figure 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 TimeVar+TimeVar provide 18% and 21% more energy savings compared with their worst case execution, respectively. The relative performance of all schemes also increases with more variable workload. Offline has definite advantage over other techniques as it is based on known actual execution times. TimeVar provides consistently more energy savings than the other two on-line approaches. However, only marginal improvement is achieved over TimeVar+OLDVS. Recall that the use of TimeVar for on-line reclamation only outperforms OLDVS 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. We also experimented with different number of tasks and different mean interarrival times. Although the actual values are different, there is no major performance difference with respect to energy consumption.

VI. Statistical Real-Time Guarantee

To guarantee that all tasks will meet their deadlines, conditions must be placed on an allowed task set. The provisioning of statistical real-time guarantee was motivated by the schedulability test for sporadic tasks [24]. It was proved that

for a sporadic task set with $d_i = p_i$, 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 rate,

$$\sum_{i=1}^{n} \frac{e_i}{p_i} \leq 1. \tag{14}$$

Recall that w_i denotes the required number of CPU cycles. Let f_{max} denote the maximum CPU frequency. Then e_i equals w_i/f_{max}. The condition can be expressed in an alternative form as $\sum_{i=1}^{n} w_i/p_i \leq f_{max}$. It is conservative by considering the worst case scenario: all tasks are released at maximum rate with the minimum interarrival time p_i. 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, $prob(l(t) > f_{max})$. 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 resource, similar to the task transformation technique in [33]. Although the overload probability is different from deadline miss rate in concept, it provides a safe bound for deadline misses.

A. Bounds for Deadline Misses

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. Readers are referred to [39] for the proof of the theorem.

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

$$v \leq \frac{\sigma_l^2}{\sigma_l^2 + (f_{max} - \mu_l)^2}. \tag{15}$$

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.

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 $p(\tilde{w}(t))$. Once we get the output distribution, the overload probability v can be calculated by considering its tail distribution $v = prob(l(t) > f_{max}) = \int_{f_{max}}^{\infty} p(l(t))dl(t)$. 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 $p(\cdot)$. For a time-independent input process, the output distribution is a scaled convolution in a form as

$$p(l(t)) = t_d p^{t_d^*}(t_d \cdot \tilde{w}(t)), \tag{16}$$

where t_d^* means t_d-fold convolution of $p(\cdot)$ with itself. Although the convolution is computationally expensive, performance analysis can be done off-line by the use of statistical information of job requests. This does not interfere with the on-line voltage scaling decisions. For a more desirable on-line analysis, we can simplify the computation by using characteristic function. Denote $\Phi(\omega)$ as the characteristic function of the input PDF. The characteristic function of the output can be simply calculated by multiplication instead of expensive convolution, as $t_d \Phi^{t_d}(t_d \cdot \omega)$. The output PDF can be readily obtained by taking the inverse transform of the function.

When the characteristic function of input distribution is not available or in the case of TimeVar, we can use a histogram-based technique. Histogram of output load can be obtained by a profiling or on-line monitoring in a time window. The output values are a series of numbers ranging from the minimum l_{min} to the maximum l_{max} number of cycles per unit time. 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 ith 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 $v = prob(l(t) > l_{min} + (i+1) \cdot l_m) = 1 - P(l_{min} + (i+1) \cdot l_m)$. The boundary of the ith bin $l_{min} + (i+1) \cdot l_m$ can be used as a tight capacity bound corresponding to the overload setting. Histogram-based technique has been shown to be effective in estimating distribution of task cycle demands in DVS [27], [38]. In contrast, we use it to estimate the system load distribution and bound deadline misses.

B. Experimental Results

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

1) Capacity Configuration: Consider the sporadic task set used in the previous experiments with 20 tasks and the minimal inter-separation 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 (14), 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 1230K cycles. The minimum capacity 123MHz was chosen as the worse case configuration.

To relax the capacity bound, we first estimate the output load distribution for both TimeInvar and TimeVar. Then a tight capacity bound is computed. The histogram-based technique is

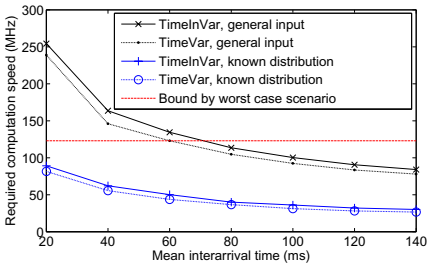

Fig. 10. Comparison of capacity configuration.

used for simplicity. Figure 10 shows the capacity bounds with varied mean interarrival times given an overload probability 1%. The bound computed with 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% when the mean interarrival reaches 100 ms. A 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 TimeVar scheduling consistently outperforms TimeInvar from 6% to 15%. This implies TimeVar is more effective in reducing load variance.

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 handing 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 resource. 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 TimeInvar is 113.6 MHz for a general input; 40 MHz when underlying distribution is known. Both capacities are computed with a target overload probabilities 1%. The bounds are tightened than the worst case estimate, as can be observed from Figure 10. Similarly, the capacity is 105.7 MHz for a general input and 37 MHz with known distribution using TimeVar. We experimented with the general bounds and found no system overload or deadline misses for both TimeInvar and TimeVar. This shows the conservativeness of the estimate because of the target 1% overload probability.

Simulation results of scheduling with a tightened capacity bound are shown in Table I. 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% of the jobs are rejected in TimeInvar and 0.61% in TimeVar. The same amount of jobs misses 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 job only at time the system has enough resource, it does not increase the overload number. In both cases, the deadline miss rates are effectively bounded by the 1% overload probability in simulation setting.

VII. Related Work

There are extensive research in real-time systems on DVS. A major focus is on 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 [2], [10], [22], [23], [30], [41]. On-line slack distribution techniques are then applied to reclaim slacks when tasks finish earlier. For example, Pillai and Shin [23] 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. Cycle-conserving (CC) and Look-Ahead algorithms were proposed to claim unused CPU time. Aydin *et al.* used a similar voltage scaling approach for periodic tasks based on worst case executions [2]. They applied a priority-based slack distribution utilizing the unused execution times of high priority tasks. The basic idea is later applied to periodic tasks using rate monotonic scheduling in case of task synchronization [13], to a hard real-time task graph with precedence constraints for a multi-processor environment [26], [40], and to slack distribution between processor slowdown and shutdown [14]. In this paper, we study a preemptive general task model in a single-processor system considering processor slowdown.

Another research focus on DVS is for algorithms in support of real-time aperiodic tasks [12], [17], [21], [25], [32], [35]. For example, Yao *et al.* provided an optimal preemptive off-line real-time scaling algorithm for a set of independent aperiodic tasks with arbitrary arrival times and deadlines on a variable speed processor [35]. Its time complexity is $O(n^2)$ (or $O(nlog^2n)$ for a sophisticated implementation), where n is the number of jobs. Manzak and Chankrabari presented an algorithm that could be used for both periodic and aperiodic tasks with prior knowledge of job timing information [21]. Quan and Hu extended the algorithm in [35] by a near-optimal *fixed-priority* scheduling algorithm with the same assumption that all timing parameters are known off-line [25]. The effect of a limited number of available processor speeds was considered in [12], [17], [32]. For example, Kwon and Kim proposed an optimal procedure to transform the scheduling resulted from [35] with continuous range of voltage levels into that with limited number of levels [17]. The procedure was based on the results of [12], which showed that an optimal voltage

TABLE I

STATISTICS OF TIMEINVAR/TIMEVAR SCHEDULING WITH A TIGHTENED CAPACITY. ($Interarrival \sim exp(80ms)$, $t_d = 10ms$)

Scheduling	TimeInvar+Rejection	TimeInvar+Besteffort	TimeVar+Rejection	TimeVar+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%

allocation is to use the two voltages that are the immediate neighbors to the ideal voltage.

The above algorithms for aperiodic tasks are off-line because they assume known job timing information before job releases. Aperiodic tasks usually have arbitrary release times, which calls for on-line DVS algorithms. An on-line scheduling heuristic approach, called Average Rate heuristic (AVR), was proposed also by Yao *et al.* for aperiodic tasks [35]. 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 [31]. They assumed a stationary processor utilization over the scheduling slots which is not always valid. Both algorithms may result in more deadline misses by scaling the processor for energy reduction. Since their focus was on energy saving, the impact on deadline misses was not analyzed and no feasibility condition was used to guarantee schedulability.

On-line algorithms in [11], [28], [24] can provide hard real-time support to aperiodic tasks. Hong *et al.* considered a mixed real-time workload of both aperiodic (on-line) and periodic (off-line) tasks [11]. They used an on-line acceptance test for both tasks to provide hard real-time guarantee (OPASTS). But its time complexity is pseudo-polynomial with the number of requests in a hyperperiod. This limits its usage in a real-time environment. Algorithms with reduced complexity in the order of $O(n)$ were also considered for aperiodic (STS) and mixed periodic/aperiodic tasks (HPASTS). Although deadlines of periodic tasks can be guaranteed, their algorithms may reject more aperiodic tasks by scaling the processor without knowledge of future aperiodic task release. The impact of the algorithms on rejection was not addressed. The analytical model in this paper enables the statistical guarantee to both deadline miss and rejection rates.

Sharma *et al.* investigated the use of DVS in web servers [28]. They applied a synthetic utilization bound for a set of fixed-priority aperiodic tasks and tended to fix the utilization as close as possible to the bound by use of admission control. In contrast, we consider a dynamic priority system. Qadi *et al.* considered a special type of hard aperiodic task referred as a canonical sporadic task model [24]. They proposed a CPU scaling algorithm that could guarantee all tasks meet their deadlines and proved its optimality when only CPU frequency could be scaled. In this work, we proposed an scaling policy that could lead to more energy savings when processor voltage can be changed dynamically with frequency setting.

A general task model was recently addressed by Lee and Shin [18]. They focused on slack management when jobs finish earlier. In contrast, we proposed a unified solution for both WCET based scheduling and on-line slack distribution.

A related mixed periodic and aperiodic task model was considered for energy saving by Aydin and Yang [3], Shin and Kim [29]. Their objective was to improve responsiveness of aperiodic tasks while retaining the deadlines of periodic tasks. The approach in this paper can be used to provide hard real-time to all admitted jobs.

We note that there exist work on stochastic analysis of periodic real-time systems, but with no energy considerations, see [1], [6], [33] for examples. There also exists stochastic analysis for DVS dealing with the uncertainty of execution time demand [10], [20], [38]. It assumes or measures the demand distribution of applications and adjusts CPU speed based on the distribution. Gruian [10], Lorch and Smith [20] developed similar approaches for stochastic DVS. In the approaches, a job starts slowly and then accelerates as it progresses. Yuan [38] extended the approach for soft real-time support to multimedia applications by on-line demand distribution monitoring. In this paper, we take a different approach by considering the effect of energy-efficient scheduling to real-time guarantees.

VIII. CONCLUSION

In this paper, we have presented techniques for power-efficient real-time computing through dynamic voltage scaling. While most existing DVS algorithms can only be used for periodic tasks, the proposed solution does not assume task periodicity. The parameters of each job, including release time, deadline, and execution cycles, are known only after job releases. The job releases are modeled as a random process with a general distribution. An analytical model is proposed to characterize the relationship of input process, scheduling, and system load. The system load is proved to be a linear combination of the aggregated input requests and a CPU resource allocation function. The voltage scaling algorithm design is turned into a filter design problem in a linear system. An optimal time-invariant voltage scaling policy with a constant time complexity is derived. A more energy-efficient time-variant scaling policy is also derived. It has a time complexity determined by the number of distinctive task deadlines. The solution process shows a time water-filling structure, which makes it easy to integrate into existing schedulers. We show it is effective in energy savings for WCET based schedule as well as on-line slack distribution.

We have further provided statistical performance guarantees to reduce computation requirement. Two capacity bounds have been derived depending on different amount of required statistical information. One is a loose upper bound for a general input in which only input mean and variance are known. The other is tight with a known marginal distribution of the input process. Both can effectively bound the deadline miss rate.

REFERENCES

[1] A. Atlas and A. Bestavros. Statistical rate monotonic scheduling. In *IEEE Real-Time Systems Symposium*, pages 123–132, 1998.

[2] H. Aydin, R. G. Melhem, D. Mossé, and P. Mejía-Alvarez. Power-aware scheduling for periodic real-time tasks. *IEEE Trans. Computers*, 53(5):584–600, 2004.

[3] H. Aydin and Q. Yang. Energy-responsiveness tradeoffs for real-time systems with mixed workload. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symp.*, pages 74–83, 2004.

[4] A. Chandrakasan, S. Sheng, and R. Brodersen. Low-power CMOS digital design. *IEEE Journal of Solid-State Circuits*, 27(4):473–484, 1992.

[5] T. Cover and J. Thomas. *Elements of Information Theory (pages 250-253)*. John Wiley & Sons, Inc, 1991.

[6] J. L. Díaz, D. F. García, K. Kim, C.-G. Lee, L. L. Bello, J. M. López, S. L. Min, and O. Mirabella. Stochastic analysis of periodic real-time systems. In *IEEE Real-Time Systems Symposium*, pages 289–300, 2002.

[7] A. Fu, E. Modiano, and J. N. Tsitsiklis. Optimal energy allocation for delay-constrained data transmission over a time-varying channel. In *Proceedings of the IEEE Infocom*, 2003.

[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 Journal*, 7(2):31–36, May 2003.

[9] A. Goldsmith and P. Varaiyai. Capacity of fading channel with channel side information. *IEEE Trans. Inform. Theory*, 43(6):1986–1992, 1995.

[10] F. Gruian. Hard real-time scheduling for low-energy using stochastic data and dvs processors. In *Proc. of the International Symp. on Low-Power Electronics and Design*, pages 46–51, 2001.

[11] I. Hong, M. Potkonjak, and M. B. Srivastava. On-line scheduling of hard real-time tasks on variable voltage voltage processor. In *Proc. of the Int'l Conference on Computer-Aided Design*, November 1998.

[12] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *Proc. of the International Symp. on Low-Power Electronics and Design*, August 1998.

[13] R. Jejurikar and R. K. Gupta. Dual mode algorithm for energy aware fixed priority scheduling with task synchronization. In *In Workshop on Compilers and Operating Systems for Low Power*, September 2003.

[14] R. Jejurikar and R. K. Gupta. Dynamic slack reclamation with procrastination scheduling in real-time embedded systems. In *DAC*, pages 111–116, 2005.

[15] N. Jindal, W. Rhee, S. Vishwanath, S. A. Jafar, and A. Goldsmith. Sum power iterative water-filling for multi-antenna gaussian broadcast channels. *IEEE Trans. Inform. Theory*, 51(4):1570–1580, 2005.

[16] M. A. Khojastepour and A. Sabharwal. Delay-constrained scheduling: Power efficiency, filter design, and bounds. In *Proc. of the 23rd IEEE Infocom*, March 7-11 2004.

[17] W.-C. Kwon and T. Kim. Optimal voltage allocation techniques for dynamically variable voltage processors. *ACM Trans. on Embedded Computing Sys.*, 4(1):211–230, 2005.

[18] C.-H. Lee and K. G. Shin. On-line dynamic voltage scaling for hard real-time systems using the edf algorithm. In *Proc. of the 25th IEEE International Real-Time Systems Symp.*, December 2004.

[19] J. W. Liu. *Real-Time Systems*. Prentice-Hall, 2000.

[20] J. R. Lorch and A. J. Smith. Improving dynamic voltage scaling algorithms with PACE. In *Proc. of the ACM SIGMETRICS 2001 Conference*, pages 50–61, 2001.

[21] A. Manzak and C. Chakrabarti. Variable voltage task scheduling for minimizing energy or minimizing power. In *Proc. of the IEEE Int. Conf. on Acoustic, Speech, and Signal Processing*, June 2000.

[22] P. Mejía-Alvarez, E. Levner, and D. Mossé. Adaptive scheduling server for power-aware real-time tasks. *ACM Trans. Embedded Comput. Syst.*, 3(2), 2004.

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

[24] A. Qadi, S. Goddard, and S. Farritor. A dynamic voltage scaling algorithm for sporadic tasks. In *Proc. of the 24th IEEE International Real-Time Systems Symp.*, December 2003.

[25] G. Quan and X. S. Hu. Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors. In *Proc. of the Design Automation Conference*, June 2001.

[26] D. Roychowdhury, I. Koren, C. M. Krishna, and Y.-H. Lee. A voltage scheduling heuristic for real-time task graphs. In *International Conf. on Dependable Systems and Networks*, pages 741–750, 2003.

[27] C. Rusu, R. Xu, R. G. Melhem, and D. Mossé. Energy-efficient policies for request-driven soft real-time systems. In *Euromicro Conference on Real-Time Systems*, pages 175–183, 2004.

[28] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu. Power-aware qos management in web servers. In *Proc. of the 24th IEEE International Real-Time Systems Symp.*, December 2003.

[29] D. Shin and J. Kim. Dynamic voltage scaling of periodic and aperiodic tasks in priority-driven systems. In *Proc. of the Asia Pacific Design Automation Conference*, 2004.

[30] Y. Shin and K. Choi. Power conscious fixed priority scheduling for hard real-time systems. In *Proc. of the Design Auto. Conf.*, 1999.

[31] A. Sinha and A. P. Chandrakasan. Energy efficient real-time scheduling. In *Proc. of the Int'l Conf. on Computer-Aided Design*, November 2001.

[32] V. Swaminathan and K. Chakrabarty. Network flow techniques for dynamic voltage scaling in hard real-time systems. *IEEE Trans. on Computer-aided Design of Integrated Circuits and Systems*, 23(10), 2004.

[33] 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 tasks with varying computation times. In *IEEE Real Time Technology and Applications Symp.*, pages 164–173, 1995.

[34] E. Uysal-Biyikoglu and A. E. Gamal. On adaptive transmission for energy efficiency in wireless data networks. *IEEE Trans. Inform. Theory*, 50(12):3081–3094, 2004.

[35] F. Yao, A. Demers, and A. Shenker. A scheduling model for reduced cpu energy. In *IEEE Foundations of Computer Science*, pages 374–382, 1995.

[36] W. Yu, G. Ginis, and J. Cioffi. Distributed multiuser power control for digital subscriber lines. *IEEE Journal on Selected Areas in Communications*, 20(5):1105–1115, 2002.

[37] W. Yu, W. Rhee, S. Boyd, and J. M. Cioffi. Iterative water-filling for gaussian vector multiple-access channels. *IEEE Trans. Inform. Theory*, 50(1):145–152, 2004.

[38] W. Yuan and K. Nahrstedt. Energy-efficient soft real-time cpu scheduling for mobile multimedia systems. In *Proc. of the 19th ACM Symp. on Operating Systems Principles*, pages 149–163, 2003.

[39] X. Zhong and C.-Z. Xu. Energy aware modeling and scheduling of real-time tasks for dynamic voltage scaling. Technical report, Cluster and Internet Computing Lab, Wayne State University, May 2005. http://www.ece.eng.wayne.edu/˜czxu/paper/dvs-model-scheduling.pdf.

[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 Distrib. Syst.*, 14(7):686–700, 2003.

[41] Y. Zhu and F. Mueller. Feedback edf scheduling exploiting dynamic voltage scaling. In *IEEE Real-Time and Embedded Technology and Applications Symp.*, pages 84–93, 2004.