A Unified Practical Approach to Stochastic DVS Scheduling

Ruibin Xu, Rami Melhem, Daniel Mossé
Department of Computer Science, University of Pittsburgh
Pittsburgh, Pennsylvania, U.S.A.
{xruibin,melhem,mosse}@cs.pitt.edu

ABSTRACT

This paper deals with energy-aware real-time system scheduling using dynamic voltage scaling (DVS) for energy-constrained embedded systems that execute variable and unpredictable workloads. The goal is to design DVS schemes to minimize the *expected* energy consumption of the *whole* system while meeting the deadlines of the tasks. Researchers have attempted to take advantage of stochastic information about workloads to achieve better energy savings, and accordingly, various stochastic DVS schemes have been proposed. However, the existing stochastic DVS schemes are based on much simplified power models that assume unrestricted continuous frequency, well-defined power/frequency relation, and no speed change overhead. When these schemes are used in practice, they need to be patched in order to comply with realistic power models. Experiments show that some of such DVS schemes perform even worse than certain non-stochastic DVS schemes. Furthermore, even for stochastic schemes that were shown experimentally to outperform non-stochastic schemes, it is not clear how well they perform compared to the optimal solution, which is yet to be found. In this work, we provide a unified practical approach for obtaining optimal (or provably close to optimal) stochastic inter-task, intra-task, and hybrid DVS schemes under realistic power models in which the processor only provides a set of discrete speeds, no assumption is made on power/frequency relation, and speed change overhead is considered. We also evaluate the existing DVS schemes by comparing them with our DVS schemes.

Categories and Subject Descriptors

D.4.1 [**Operating Systems**]: Process Management—*Scheduling*; D.4.7 [**Operating Systems**]: Organization and Design—*Real-time systems and embedded systems*

General Terms

Algorithms

Keywords

Real-time, Dynamic Voltage Scaling, Power management, Stochastic DVS Scheme, Fully Polynomial Time Approximation Scheme

EMSOFT'07, September 30–October 3, 2007, Salzburg, Austria.

1. INTRODUCTION

Power conservation is critically important for energy-constrained embedded systems. Dynamic voltage scaling (DVS), which involves dynamically adjusting the voltage and frequency (speed) of the processor, is a common power management mechanism for any embedded system whose processor accounts for a significant portion of the total power. Through DVS, quadratic energy savings for the processor can be achieved at the expense of just linear performance loss [16]. For real-time embedded systems, the execution of tasks can be slowed down to save processor energy as long as the deadline constraints are not violated. Thus, a key problem is to design DVS schemes that determine execution speeds of tasks in the system.

There are three types of DVS schemes: inter-task DVS schemes, intra-task DVS schemes, and hybrid DVS schemes [8]. Inter-task DVS schemes focus on allotting system time to multiple tasks and schedule speed changes only at each task boundary (i.e., the execution speed of a task is constant for each instance executed), while intra-task DVS schemes focus on how to schedule speed changes within a single task given an allotted amount of time. Hybrid DVS schemes combine inter-task and intra-task DVS, that is, hybrid DVS schemes consider how to allot time to the tasks in the system as well as how to schedule speed changes within each task. We treat intra-task DVS schemes as special cases of hybrid DVS schemes and will only consider designing inter-task DVS and hybrid DVS schemes. Each type of schemes has its advantage and disadvantage. Inter-task DVS schemes are easier to implement than hybrid DVS schemes because the latter require a timer-like interrupt mechanism in order to change speed during the execution of a task. However, hybrid DVS schemes have more flexibility and potentially achieve more energy savings.

In this paper, we focus on *frame-based hard* real-time embedded systems that execute variable and unpredictable workloads. Frame-based real-time systems are special cases of periodic real-time systems. In frame-based real-time systems, all tasks share the same period (also called the *frame*) and deadlines are equal to the end of the period. In each frame, tasks are executed in a predetermined fixed order. Many real-world real-time systems, especially embedded systems, are frame-based due to its simplistic and deterministic nature. On the other hand, periodic real-time systems that use static table driven scheduling approaches [13] such as cyclic executive [2] can be treated as multiple frame-based systems operating in succession because each minor schedule in a cyclic schedule corresponds to a frame. For frame-based systems, the problem of designing a DVS scheme can be reduced to determining the amount of time allotted to a task (and accordingly, deciding the speed(s) to execute the task) before the task is dispatched to execute in the system.

Variable and unpredictable workloads are commonly seen in real-time systems because (1) real-time applications usually exhibit a large variation in actual execution times; (2) the computational requirement of a task cannot be predicted based on recent history (i.e., computational requirements are not correlated in time). The *variability* and *unpredictability* of workloads are mainly caused by different inputs to tasks, and possibly by randomization inside tasks.

The variability of workloads creates the opportunity for dynamic slack reclamation to reduce the execution speed for future tasks. Various DVS scheme using dynamic slack reclamation with different speed reduction aggressiveness have been proposed. For example, a proportional scheme will distribute the slack proportionally among all unexecuted tasks, while a greedy scheme gives all the slack to the next ready-to-run task [11].

More recently, researchers have attempted to take advantage of stochastic information of workloads to deal with unpredictability. Stochastic information of workloads are represented by the probability distribution of the computational requirement of each task. When this information is available (e.g., through profiling [9, 22]), the design goal of DVS schemes becomes *minimizing the expected energy consumption* in the system and those DVS schemes are called *stochastic DVS schemes*.

Optimal stochastic schemes based on the *ideal power model* (or *ideal model* for short) have been proposed in [19, 23]. The ideal model assumes unrestricted continuous frequency, well-defined power /frequency relation, and no speed change overhead. The main reason for using the ideal model is its easy mathematical manipulation. In the real world, however, the processor only provides a limited set of discrete speeds, the power/frequency relation is not well-defined, and speed change overhead may not be ignored. We call this the *realistic power model* (or *realistic model* for short). When the schemes based on the ideal model are used in practice, they need to be patched (e.g. rounding continuous speed to available discrete speed) in order to comply with the realistic model. As a result, the optimal speeds that were derived based on the ideal model are no longer optimal for the realistic model.

Experiments show some anomaly for the patched DVS schemes based on the ideal model. For example, the best of all DVS schemes for the ideal model is the optimal stochastic hybrid DVS scheme called GOPDVS [23]. In Section 6, however, we will see that the patched GOPDVS performs even worse than certain non-stochastic schemes (i.e., schemes that do not use any stochastic information of workloads). This is discouraging since using more information is supposed to lead to better results. Even for the stochastic schemes that were shown experimentally to outperform non-stochastic schemes, it is not clear how well those stochastic schemes perform when compared to the optimal stochastic scheme under the realistic model, which is yet to be found.

In this work, we provide a step function based approach for obtaining the optimal stochastic DVS schemes under the realistic model. To control the computational complexity, we use a function approximation technique to obtain DVS schemes whose resulting expected energy consumption is guaranteed to be within a factor of $1 + \epsilon$ of the optimal solution and whose time complexity is polynomial in $\frac{1}{\epsilon}$ where ϵ is a parameter of the DVS schemes. That is, the system designer has the freedom of controlling the trade-off between the quality of the solution and the time complexity. In fact, our approximation technique falls in the category of fully polynomial time approximation schemes (FPTAS) [5]. Our approach is *unified* in the sense that it can be used to obtain all three types of DVS schemes (i.e., inter-task DVS, intra-task DVS, and hybrid DVS). Furthermore, our approach is practical because it takes into consideration all the practical issues when deriving DVS schemes.

Our contribution is two-fold. (1) we provide optimal (or provably close to optimal) stochastic DVS schemes under the realistic model and thus establish tight upper bounds on the energy savings (note that no upper bound can be estimated when a solution derived under the ideal model is used to approximate the solution for the realistic model because the ideal model is not a generalization of the realistic model); and (2) based on our DVS schemes for the realistic model, we identify good DVS schemes that are based on heuristics.

The remainder of this paper is organized as follows. We first describe the related work in Section 2. The task and system models as well as problem description are presented in Section 3. Section 4 describe the basic idea behind our approach and Section 5 presents the details of our DVS schemes. The evaluation results are reported in Section 6. We end the paper in Section 7 with concluding remarks and future work directions.

2. RELATED WORK

Although much work has been done on exploring DVS in real-time environments since the seminal work by Yao et al. [21], we will focus on the related work that deals with variable and unpredictable workloads, including both stochastic schemes and non-stochastic schemes.

2.1 Inter-task DVS

Inter-task DVS schemes differ in the way slack is allotted to tasks in the system. Mossé et al. [11] introduced the concept of *speculative speed reduction* and proposed three DVS schemes (Greedy, Proportional, and Statistical) with different speed reduction aggressiveness for frame-based real-time systems. The Proportional scheme distributes the slack proportionally among all unexecuted tasks, while the Greedy scheme is much more aggressive and gives all the slack to the next ready-to-run task. The Statistical scheme uses the average number of execution cycles of tasks to predict the future slack. Many existing DVS schemes proposed for real-time systems can be classified as the Proportional or Greedy schemes.

Pillai et al. [12] proposed the cycle-conserving scheme and the look-ahead scheme for periodic real-time systems that execute variable workloads. When used in frame-based systems, the cycle-conserving scheme is equivalent to the Proportional scheme, while the look-ahead scheme is equivalent to the Greedy scheme. Saewong et al. [15] proposed a scheme for fixed-priority real-time systems which, when used in frame-based systems, is equivalent to the Greedy scheme. To be able to navigate the full spectrum of speculative speed reduction, Aydin et al. [1] proposed a DVS scheme in which system designers can set a parameter to control the degree of speed reduction aggressiveness. In fact, the optimal speed reduction aggressiveness depends on the variability of the workloads. Xu et al. [19] presented the optimal stochastic inter-task DVS scheme under the ideal model and proposed patches to make it comply with the realistic model.

2.2 Intra-Task DVS

For a single task and a given deadline, Lorch et al. [9] have shown that if the task's computational requirement is only known probabilistically, then there is no constant optimal speed for the task and the expected energy consumption is minimized by gradually increasing the speed as the task progresses. They obtained the optimal speed schedule under the ideal model and called this approach PACE (*Processor Acceleration to Conserve Energy*). They also

proposed patches to make PACE fit the realistic model. Stochastic schemes similar to PACE have also been proposed in [22, 6]. They differ in the way of patching the speed schedule obtained under the ideal model to fit the realistic model. Xu et al. [20] proposed the PPACE (Practical PACE) scheme that is derived directly under the realistic model and showed its superiority over the previous schemes for a single task. The derivation of PPACE shares some similarity (mainly in approximating the optimal solution) with our approach used in this paper. However, our approach in this paper works for all types of DVS schemes while PPACE only works for intra-task DVS. Moreover, our approach obtains all speed schedules of a task for different values of allotted time while PPACE only obtains the speed schedule of a task given an allotted amount of time.

2.3 Hybrid DVS

To obtain a hybrid DVS scheme, Kim et al. [8] used an inter-task DVS scheme as the basis and plugged in an intra-task DVS scheme. A similar approach was used in [22]. However, such hybrid schemes are inherently suboptimal because they ignore the interaction between inter-task and intra-task DVS. Zhang et al. [23] presented the optimal stochastic hybrid DVS scheme under the ideal model that can be regarded as an extension to PACE. However, they did not provide any solution to patch their scheme to fit the realistic model.

3. MODELS AND PROBLEM DESCRIPTION

3.1 Task and Power Models

We consider a frame-based task model with N periodic tasks in the system. The task set is denoted by $\Gamma = \{T_1, T_2, \ldots, T_N\}$. All tasks share the same period and deadlines are equal to the end of the period. The length of the period (also known as *frame length*) is denoted by D. The execution of the frame is repeated and thus we only need to focus on the first frame which starts at time 0 and ends at time D. All tasks are executed nonpreemptively during each frame in the order of $T_1, T_2, \ldots, T_N$.

Each task T_i ($1 \leq i \leq N$) is characterized by its worst-case number of execution cycles (WCEC) W_i and the probability function of its execution cycles $P_i(x)$, which denotes the probability that task T_i executes for x ($1 \leq x \leq W_i$) cycles. Obviously, $\sum_{x=1}^{W_i} P_i(x) = 1$ and $P_i(W_i) \neq 0$. The corresponding cumulative distribution function is $cdf_i(x) = \sum_{j=1}^{x} P_i(j)$ and $cdf_i(0) = 0$. The average-case number of execution cycles (ACEC) of T_i is $A_i = \sum_{x=1}^{W_i} P_i(x)x$.

In practice, a histogram is used to represent the probability function considering that a task usually takes millions of cycles. In this case, let the number of bins in the histogram that represents $P_i(\cdot)$ be denoted by r_i and denote the bin boundaries by $B_i(x), x = 1, 2, \ldots, r_i$. Thus, $P_i(x)$ ($1 \leq x \leq r_i$) denotes the probability that task T_i executes for $B_i(x)$ cycles.

The tasks are to be executed in a system equipped with a variable voltage processor that has the ability to dynamically adjust its voltage and frequency. The processor only provides M discrete operating frequencies, $f_1 < f_2 < \cdots < f_M$. All frequencies are efficient which means that using a frequency to execute a task always results in lower energy consumption than using higher frequencies [10].

The system power consumption includes the power consumption of processor, memory, and other components. When the system is idle, the system power consumption is p_{idle} and the processor is executing no-op operations at the minimum frequency. The system power consumption when executing task T_i at frequency f_j is $p_i(f_j)$. Because frames are executed repeatedly, we assume that the system is never turned off. Thus, we ignore the idle power in deriving DVS schemes since this portion of power is always consumed in the system. When describing algorithms in Section 5, we use $\hat{p}_i(f_j) = p_i(f_j) - p_{idle}$ to simplify the presentation.

The overhead of changing speeds depends on the current and future speeds of the processor, as follows. When changing the frequency of the processor from f_i to f_j, the time penalty is

$$P_T(f_i, f_j) = \xi_1 |f_i - f_j| \tag{1}$$

and the energy penalty is

$$P_E(f_i, f_j) = \xi_2 |f_i^2 - f_j^2|, \tag{2}$$

where ξ_1 and ξ_2 are constants determined by the voltage switching circuits. Equations (1) and (2) are taken from [3] and are considered to be an accurate modeling of speed change overheads [17]. It is common in the literature to simplify Equations (1) and (2) by considering a constant penalty for the worst-case frequency swing and assuming that $P_T(f_i, f_j) = \xi_1(f_M - f_1)$ and $P_E(f_i, f_j) = \xi_2(f_M^2 - f_1^2)$. Our approach can deal with variable speed overheads modeled by Equations (1) and (2), unlike common practice that considers constant and worst-case penalties for speed changes.

3.2 Problem Description

A DVS scheme consists of N *speed schedule* functions $S_i(\cdot)$ ($i = 1, 2, \ldots, N$). $S_i(t)$ denotes the speed schedule for task T_i, when T_i is ready to execute and there is time t remaining in the frame. A speed schedule for a task dictates what speed(s) to be used for executing this task. For inter-task DVS, a speed schedule is a single speed; for intra-task and hybrid DVS, each speed schedule contains a set of speeds and the corresponding speed scaling points.

Let $e'(\varsigma, x)$ and $t'(\varsigma, x)$ denote the energy consumption and time for executing T_i using speed schedule ς when the actual number of execution cycles of T_i is x. The expected energy consumption for executing $T_i, T_{i+1}, \ldots, T_N$ using time t can be computed as

$$E_i(t) = \sum_{k=1}^{r_i} P_i(k)\big(e_i'(S_i(t), B_i(k)) + E_{i+1}(t - t_i'(S_i(t), B_i(k)))\big)$$

and $E_{N+1}(t) = 0$. Thus, the problem is to find DVS schemes that minimize $E_1(D)$.

We consider three cases of the problem in the paper: (1) SIDVS, which stands for the Simple Inter-task DVS problem that attempts to find inter-task DVS schemes in the absence of speed change overhead; (2) IDVS, which is a generalization of the SIDVS problem that considers speed change overhead; (3) HDVS, which attempts to find hybrid DVS schemes (also considering speed change overhead). Obviously, SIDVS is the simplest case of the problem. The differences between SIDVS under the realistic model and that under the ideal model are discrete speeds vs.continuous speeds, and arbitrary power function vs.well-defined power function. We consider SIDVS because its solution contains all the essential ingredients of our approach and can be easily extended to solve the other two problems.

4. THE BASIC IDEA

In this section, we describe the basic idea behind our approach through the discussion of the main idea of the solution to SIDVS. The purpose of this section is to illustrate all the key elements in our approach without delving into too much mathematical detail. We start by describing the solution to SIDVS, followed by the properties of the solution and how to obtain such a solution.

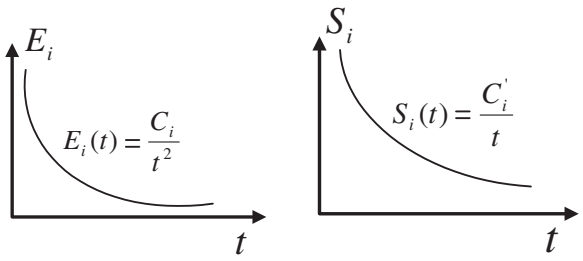
4.1 The Solution to SIDVS

The solution to SIDVS (simple inter-task DVS) is $2N$ functions, two for each task in the system. Specifically, each task T_i corresponds to two functions: $E_i(\cdot)$ and $S_i(\cdot)$. These two functions denote that when task T_i is ready to execute and there is time t remaining in the frame, if we use speed $S_i(t)$ to execute T_i, the expected energy consumption of executing $T_i, T_{i+1}, \ldots, T_N$ will be $E_i(t)$. Computing functions $E_i(\cdot)$ and $S_i(\cdot)$ (i.e., finding all the mappings in the functions) is done offline, which we will discuss in Section 4.3. During the operation of the system, the OS scheduler will consult functions $S_i(\cdot)$ to determine the speed of each task. Specifically, at the beginning of a frame when there is time D available, the OS scheduler will use the speed $S_1(D)$ to execute T_1. After T_1 finishes and it has taken time t', there is time $D - t'$ remaining in the frame and the OS scheduler will use the speed $S_2(D - t')$ to execute T_2. The same process will be applied to the rest of the tasks.

4.2 The Properties of the Solution

Before discussing how to obtain the solution to SIDVS, we examine the properties of functions $E_i(\cdot)$ and $S_i(\cdot)$, which will determine their representation. During the examination, we also consider the solution under the ideal model, which will help us understand the motivation behind our approach.

We first examine functions $E_N(\cdot)$ and $S_N(\cdot)$ because they only involve a single task T_N. For the ideal model (assuming cubic power/frequency relationship), we have [19] $E_N(t) = \frac{C_N}{t^2}$ and $S_N(t) = \frac{C'_N}{t}$ where neither C_N nor C'_N depends on t. Thus, both $E_N(\cdot)$ and $S_N(\cdot)$ (Figure 1) can be represented by just a constant (C_N for $E_N(\cdot)$ and C'_N for $S_N(\cdot)$). This is due to the simplicity of the ideal model.

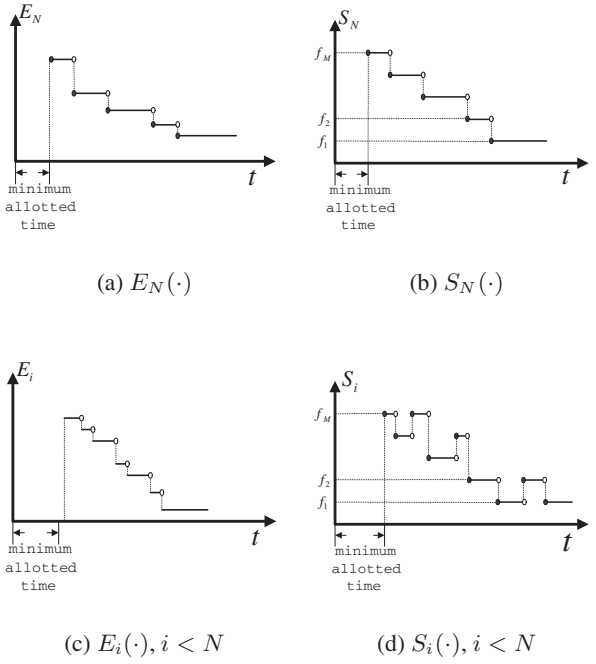


(a) $E_i(\cdot)$ $(1 \le i \le N)$ (b) $S_i(\cdot)$ $(1 \le i \le N)$

Figure 1: Solution for the ideal model

For the realistic model, however, both $E_N(\cdot)$ and $S_N(\cdot)$ are step functions (piece-wise constant functions). Figure 2(b) shows the function of $S_N(\cdot)$ for the realistic model, which can be obtained by rounding its counterpart for the ideal model (Figure 1(b)) to the available discrete speeds. We can see that there are M (M is the number of available discrete speeds) *half-open* line segments in the graph, each corresponding to an available discrete speed. Each line segment can be represented by its left end point because its right end point is the left end point of the line segment to its immediate right or infinity when it is the rightmost line segment of the graph. We call the left end point of a line segment a *turning point*. Thus, $S_N(\cdot)$ can be represented by $2M$ numbers because each turning point can be represented by its two coordinates. Figure 2(a) shows the function of $E_N(\cdot)$, which has also M half-open line segments. Counting from left to right, the k^{th} $(1 \le k \le M)$ line segment of $E_N(\cdot)$ shares the same starting t coordinate and ending t coordinate with the k^{th} line segment of $S_N(\cdot)$. Thus, $E_N(\cdot)$ can be also represented by $2M$ numbers as in $S_N(\cdot)$. Note that $E_N(\cdot)$ does not include the idle energy consumption, as explained in Section 3.1. Computing $E_N(t)$ and $S_N(t)$ can be turned into a table lookup which takes $O(\log M)$ if binary search is used.

Now we examine functions $E_i(\cdot)$ and $S_i(\cdot)$ $(1 \le i < N)$ which involve multiple tasks. For the ideal model, $E_i(\cdot)$ $(1 \le i < N)$ is of the same form as $E_N(\cdot)$, that is, $E_i(t) = \frac{C_i}{t^2}$ where C_i does not depend on t. The same holds for $S_i(\cdot)$ $(1 \le i < N)$. This elegant result, which is proved in [19], is again due to the simplicity of the ideal model. Thus, both $E_i(\cdot)$ and $S_i(\cdot)$ $(1 \le i < N$, see Figure 1) can still be represented by a single constant. This means that the complexity of the representation for the ideal model does not depend on i. However, this is not the case for the realistic model.



(a) $E_N(\cdot)$ (b) $S_N(\cdot)$

(c) $E_i(\cdot), i < N$ (d) $S_i(\cdot), i < N$

Figure 2: Solution for the realistic model

Both $E_i(\cdot)$ and $S_i(\cdot)$ $(1 \le i < N$, see Figure 2(c) and 2(d)) are still step functions. But there are more turning points in $E_i(\cdot)$ $(1 \le i < N)$ than in $E_N(\cdot)$. This is because $E_i(\cdot)$ is the expected energy consumption for multiple tasks and different combination of speeds from these tasks usually results in different energy consumption. In fact, the number of turning points of $E_i(\cdot)$ may suffer from exponential growth as i decreases, which will be clear in Section 5.1. As in the case for $E_N(\cdot)$ and $S_N(\cdot)$, each line segment of $E_i(\cdot)$ can be translated into one in $S_i(\cdot)$. However, the number of possible values of $S_i(\cdot)$ is only M. If two adjacent line segments in $S_i(\cdot)$ share the the same S_i coordinate, they can be combined into one line segment. Thus, the number of turning points of $S_i(\cdot)$ is usually much smaller than that of $E_i(\cdot)$. As for the shape of the function, $E_i(\cdot)$ (Figure 2(c)) is still an non-increasing function, while $S_i(t)$ (Figure 2(d)) may go up and down as t increases.

The latter claim is counter-intuitive, especially for the speed going up when t increases (i.e., if there is more slack, the speed increases to yield lower energy consumption). This is due to the na-

ture of discrete speeds. We describe a scenario where this will happen. Suppose that for some workload it is beneficial to use low speed to execute the tasks following T_i. For a given available time t, increasing the speed for T_i will not give enough room to drop the speed for the following tasks to the next lower discrete speed. However, as the available time t increases, increasing the speed for T_i will eventually be rewarded.

Similar to computing $E_N(t)$ or $S_N(t)$, computing $E_i(t)$ or $S_i(t)$ $(1 \leq i < N)$ is a table lookup, which takes $O(\log K)$ where K is the number of turning points in the function.

4.3 Obtaining the Solution

From Section 4.2, we can see that computing $E_i(\cdot)$ and $S_i(\cdot)$ is equivalent to identifying all the turning points in $E_i(\cdot)$ and $S_i(\cdot)$. From the recursive description of the problem in Section 3.2, it is natural to compute $E_i(\cdot)$ and $S_i(\cdot)$ in reverse order, that is, first compute $E_N(\cdot)$ and $S_N(\cdot)$, then $E_{N-1}(\cdot)$ and $S_{N-1}(\cdot)$, and so on. The computation of $E_i(\cdot)$ and $S_i(\cdot)$ only depends on $E_{i+1}(\cdot)$, as $E_{i+1}(\cdot)$ has already "summarized" functions $E_j(\cdot)$ and $S_j(\cdot)$ where $j = i+2, \ldots, N$. When the computation is done, all $E_i(\cdot)$ $(i = 1, 2, \ldots, N)$ can be discarded because they are not needed for the operation of the system.

As mentioned in Section 4.2, the number of turning points in the functions may suffer exponential growth. Thus, we propose a function approximation technique to limit the number of points. We use an example to illustrate the technique. Consider the two turning points, (e_1, t_1) and (e_2, t_2), inside the circle in Figure 3. Obviously, $e_1 > e_2$ and $t_1 < t_2$. If the difference between e_1 and e_2 is small (formally, if $\frac{e_1 - e_2}{e_2} < \delta$, where δ is a parameter to quantify the difference), we eliminate the point (e_2, t_2). Through this kind of elimination, the number of turning points is reduced and upper bounded by a polynomial in $\frac{1}{\delta}$. However, the resulting function is only an approximation of the original function (i.e., the elimination induces error). This is because when time t where $t_2 \leq t < t_3$ is available, we cannot use the speed schedule corresponding to (e_2, t_2) since it was eliminated. We will have to use the speed schedule corresponding to (e_1, t_1) and thus result in expected energy e_1 which is greater than e_2. Because of the way we eliminate the points, the difference between the resulting function and the original function is guaranteed to be no more than δ times the original function.

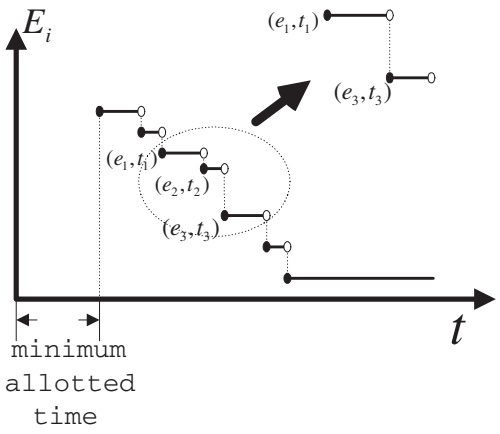


Figure 3: Function approximation

We apply the above function approximation technique to function $E_i(\cdot)$ before computing $E_{i-1}(\cdot)$. Thus, the error accumulates and increases as i decreases. Let the error of $E_1(\cdot)$ be denoted by ϵ. The expected energy consumption of the system, $E_1(D)$, is within a factor of $1 + \epsilon$ of the optimal expected energy consumption. If we let ϵ be a parameter set by system designers, we can derive the value of δ to be used for each function approximation. The technical detail can be found in Section 5.1.3.

5. THE DETAILS

Having described the basic idea behind our approach in the previous section, we provide the technical details in this section. We first give the algorithm to solve the SIDVS problem and present its analysis in Section 5.1. Then we will extend the algorithm to solve other problems, IDVS and HDVS, in Section 5.2 and 5.3, respectively.

5.1 The Algorithm to Solve SIDVS

5.1.1 On Step Functions

From Section 4 we can see that step functions play an important role in our approach. Thus, being able to represent step functions effectively and manipulate step functions efficiently are crucial for the viability of our approach.

We first formally define step function through the following two definitions.

DEFINITION 1. *A point $\mathbb{P}$ is a 2-tuple (e, t), where e and t are nonnegative reals and denote energy and time respectively. We write the energy component as $\mathbb{P}.e$ and the time component as $\mathbb{P}.t$.*

DEFINITION 2. *A step function (piece-wise constant function) $\mathbb{F}(\cdot)$ is defined as a point sequence $\mathbb{S} = [\mathbb{P}_1, \mathbb{P}_2, \ldots, \mathbb{P}_m]$ where $\mathbb{P}_1.t < \mathbb{P}_2.t < \cdots < \mathbb{P}_m.t$. $\mathbb{F}(t)$ is undefined when $t < \mathbb{P}_1.t$, otherwise $\mathbb{F}(t) = \mathbb{P}_i.e$ and $i = \max_{j=1,2,\ldots,m} \{j | t \geq \mathbb{P}_j.t\}$.*

Having formally defined step function, we will use $\mathbb{F}$ to denote a step function $\mathbb{F}(\cdot)$ unless confusion arises. Let $|\mathbb{F}|$ denote the number of points in $\mathbb{F}$. Obviously, computing $\mathbb{F}(t)$ can be done in time $O(\log |\mathbb{F}|)$ by using binary search.

We now look at three operations between a number and a step function.

DEFINITION 3. *The operator $+_e$ is defined between a real x and a step function $\mathbb{F}$ such that $x +_e \mathbb{F} = [(x + \mathbb{P}_1.e, \mathbb{P}_1.t), (x + \mathbb{P}_2.e, \mathbb{P}_2.t), \ldots]$ (i.e., the result is still a step function). Other operators, $\times_e$ and $+_t$ can be defined similarly.*

Obviously, the operators defined in Definition 3 can be performed in time $O(|\mathbb{F}|)$.

Finally, we describe two operations between step functions.

DEFINITION 4. *The sum operator $+_{\mathbb{F}}$ is defined between 2 step functions, $\mathbb{F}_1$ and $\mathbb{F}_2$, such that $\mathbb{F}_1 +_{\mathbb{F}} \mathbb{F}_2 = \mathbb{F}$ and $\mathbb{F}(t) = \mathbb{F}_1(t) + \mathbb{F}_2(t)$. The merge operator $\cup$ is defined between 2 step functions, $\mathbb{F}_1$ and $\mathbb{F}_2$ such that $\mathbb{F}_1 \cup \mathbb{F}_2 = \mathbb{F}$ and $\mathbb{F}(t) = min\,(\mathbb{F}_1(t), \mathbb{F}_2(t))$.*

The resulting step function $\mathbb{F}$ by either the sum or the merge operators over n step functions $\mathbb{F}_i$ $(i = 1, 2, \ldots, n)$ could have as many as $\sum_{i=1}^{n} |\mathbb{F}_i|$ points. The time component of each point in $\mathbb{F}$ comes from one of the $\mathbb{F}_i$'s. Because the points in $\mathbb{F}_i$ are already sorted, the time components of all points in $\mathbb{F}$ can be obtained by a procedure similar to merge sort in time $O((\sum_{i=1}^{n} |\mathbb{F}_i|) \log n)$. To compute the energy component of each point in $\mathbb{F}$, the sum operator takes constant time and the merge operator takes $O(\log n)$ time by using a priority queue. Thus, computing $+_{\mathbb{F}}{}_{i=1}^{n} \mathbb{F}_i$ takes $O((\sum_{i=1}^{n} |\mathbb{F}_i|) \log n)$ time and computing $\cup_{i=1}^{n} \mathbb{F}_i$ takes $O((\sum_{i=1}^{n} |\mathbb{F}_i|) \log^2 n)$ time.

5.1.2 An Optimal Algorithm

Recall form Section 4.3 that we compute functions E_i and S_i in reverse order. For succinct presentation, we do not show the computation of functions S_i because it can be easily performed as a by-product of computing E_i.

To compute function E_i, we first consider M helper functions $\hat{E}_{i,j}$ ($j = 1, 2, \ldots, M$), where M is the number of available discrete speeds. $\hat{E}_{i,j}$ denotes the expected energy function when frequency f_j is used to execute task T_i. A single value of $\hat{E}_{i,j}$ can be computed as

$$\begin{aligned}\hat{E}_{i,j}(t) &= \sum_{k=1}^{r_i} P_i(k)\left(\hat{p}_i(f_j)\frac{B_i(k)}{f_j} + E_{i+1}(t - \frac{B_i(k)}{f_j})\right) \\ &= \hat{p}_i(f_j)\frac{A_i}{f_j} + \sum_{k=1}^{r_i} P_i(k)E_{i+1}(t - \frac{B_i(k)}{f_j})\end{aligned}$$

In the above equations, $\hat{p}_i(f_j)\frac{B_i(k)}{f_j}$ is the energy consumption of executing the first k bins of T_i and $E_{i+1}(t - \frac{B_i(k)}{f_j})$ is the expected energy consumption of executing $T_{i+1}, \ldots, T_N$. Computing the whole function $\hat{E}_{i,j}$ can be expressed using our notations about step functions described in Section 5.1.1 as Line 6 in Figure 4. Function E_i is just the result of merging M $\hat{E}_{i,j}$ functions. During the merging process, the optimal speed corresponding to each point is also determined. The optimal algorithm to solve SIDVS is shown at Lines 1-7 in Figure 4. Line 8 is used for the function approximation technique mentioned in Section 4.3 and will be explained Section 5.1.3.

We now analyze the time complexity and space complexity of computing E_i. The key operation in computing $\hat{E}_{i,j}$ is the sum operation over r_i step functions, each is of size $|E_{i+1}|$. Thus, the time to compute $\hat{E}_{i,j}$ is $O(r_i|E_{i+1}|\log r_i)$ and the number of points in $\hat{E}_{i,j}$ is $O(r_i|E_{i+1}|)$. The key operation in computing E_i is the merge operation over M step functions, each is of size $O(r_i|E_{i+1}|)$. Thus, the time to compute E_i is $O(Mr_i|E_{i+1}|\log^2 M)$ and the number of points in E_i is $O(Mr_i|E_{i+1}|)$. Since the base case is $|E_{N+1}| = 1$, we can obtain the closed forms of the time complexity and space complexity to be $O((Mr_i)^{N-i+1}\log^2 M)$ and $O((Mr_i)^{N-i+1})$, respectively.

PROCEDURE SIDVS(ϵ)
1. $E_{N+1} := \{(0,0)\}$
2. for $i := N$ downto 1 do
3. //compute E_i
4. for $j := 1$ to M do
5. //the case where f_j is used to execute T_i
6. $\hat{E}_{i,j} := \hat{p}_i(f_j)\frac{A_i}{f_j} +_e +_{\mathbb{F}}{}_{k=1}^{r_i} P_i(k) \times_e \left(\frac{B_i(k)}{f_j} +_t E_{i+1}\right)$
7. $E_i := \cup_{j=1}^{M} \hat{E}_{i,j}$
8. $E_i := TRIM(E_i, (1+\epsilon)^{\frac{1}{N}} - 1)$

END

Figure 4: The SIDVS scheme; Line 8 is only for the approximation algorithm

5.1.3 Applying Function Approximation

The time complexity of the optimal algorithm for the SIDVS problem depends greatly on the size of functions E_i. As we can see from the analysis of the optimal algorithm in Section 5.1.2, the size of E_i may grow exponentially as i goes from N to 1. Thus, we need to control the size of function E_i within some polynomial bound. To do that, we trim (i.e., remove some points) function E_i after it is computed at Line 7 in Figure 4. A trimming parameter δ ($0 < \delta < 1$) is used to direct the trimming. After function E_i is trimmed, the energy components of any adjacent points (recall from Definition 2 that the points are stored in the order of increasing t component) differ by at least a factor of δ. The choice of $\delta = (1+\epsilon)^{\frac{1}{N}} - 1$ (ϵ is a parameter of the SIDVS scheme) at Line 8 in Figure 4 will be clear at the end of this section. Figure 5 shows the trimming procedure.

PROCEDURE TRIM($\mathbb{F} = [\mathbb{P}_1, \mathbb{P}_2, \ldots, \mathbb{P}_{|\mathbb{P}|}]$,$\delta$)
1. $\hat{\mathbb{F}} := \{\mathbb{P}_1\}$
2. $l := \mathbb{P}_1$
3. for $i := 2$ to $|\mathbb{F}|$ do
4. if $l.e > (1+\delta)\mathbb{P}_i.e$ then
5. append $\mathbb{P}_i$ onto the end of $\hat{\mathbb{F}}$
6. $l := \mathbb{P}_i$
7. return $\hat{\mathbb{F}}$

END

Figure 5: The TRIM procedure

The function approximation achieved by the trimming procedure is inspired by [5] and is similar to the label elimination technique used in [20]. Thus, we only sketch its analysis for the sake of completeness. Interested readers can refer to [5, 20] for more in-depth details.

Before computing the number of points in E_i after trimming, we prove an important lemma. Let E'_i ($i = 1, 2, \ldots, N$) be the step functions obtained if Line 8 in Figure 4 is omitted. That is, E'_i is the functions returned by the optimal algorithm. By comparing E'_i and E_i, we have the following lemma:

LEMMA 1. *For every point $\mathbb{P}' \in E'_i$ where $1 \le i \le N+1$, there exists a point $\mathbb{P} \in E_i$ such that $\mathbb{P}'.e \le \mathbb{P}.e \le (1+\delta)^{N+1-i}\mathbb{P}'.e$ and $\mathbb{P}'.t \ge \mathbb{P}.t$.*

PROOF. This lemma is equivalent to $E_i(t) \le (1+\delta)^{N+1-i}E'_i(t)$ for any value of t. The proof is by induction on i and the base case for $i = N+1$ obviously holds from Line 1 in Figure 4. In the induction step for E_i, we inspect Line 6 in Figure 4. From the hypothesis, $E_{i+1}(t)$ is within a factor of of $(1+\delta)^{N-i}$ of $E'_{i+1}(t)$. All the operations at Line 6 will preserve this property. After the trimming operation, the factor will be only increased by $(1+\delta)$, which will make $E_i(t)$ with a factor of $(1+\delta)^{N-i+1}$ of $E'_i(t)$. ∎

Using functions E'_i will lead to expected energy consumption of $E'_1(D)$ and using functions E_i will lead to expected energy consumption of $E_1(D)$. From Lemma 1, we have $E_1(D) \le (1+\delta)^N E'_1(D)$. Since we choose δ to be $(1+\epsilon)^{\frac{1}{N}} - 1$, we have $E_1(D) \le (1+\epsilon)E'_1(D)$.

To compute the upper bound of the number of points in E_i, we note that after the trimming procedure, the energy components of any adjacent points differ by at least a factor of δ. Let the leftmost point in E_i be denoted by $\mathbb{P}_l$ (which is upper bounded by the energy consumption when all tasks use the maximum speed) and the rightmost point in E_i be denoted by $\mathbb{P}_r$ (which is lower bounded by the energy consumption when all tasks use the minimum speed). Thus, we have

$$\mathbb{P}_l.e > (1+\delta)^{|E_i|-1}\mathbb{P}_r.e$$

By plugging in $\delta = (1+\epsilon)^{\frac{1}{N}} - 1$ and some algebraic manipulations, we will obtain $|E_i| = O(\frac{N \log \lambda}{\epsilon})$ where $\lambda = \frac{\mathbb{P}_l.e}{\mathbb{P}_r.e}$. Thus, the number of points in E_i is upper bounded by a polynomial in $\frac{1}{\epsilon}$.

5.2 The Algorithm to Solve IDVS

In the IDVS problem, speed change overhead is considered. The solution to SIDVS can be easily extended to solve IDVS. Instead of computing only one expected energy function E_i for each task T_i as in SIDVS, we compute M expected energy functions $E_{i,s}$ ($s = 1, 2, \ldots, M$). $E_{i,s}$ denotes the expected energy consumption of executing tasks $T_i, T_{i+1}, \ldots, T_N$ when the current speed is f_s, that is, when the speed before the execution of T_i starts is f_s. Computing each $E_{i,s}$ in IDVS is similar to computing E_i in SIDVS. The only difference is that computing $E_{i,s}$ takes into consideration the energy penalty and time penalty associated with the speed change. Thus, computing each $E_{i,s}$ in IDVS has the same time and space complexity as computing E_i in SIDVS. Figure 6 shows the details of the IDVS scheme.

PROCEDURE IDVS(ϵ)
1. for $s := 1$ to M do
2. $\quad E_{N+1,s} := \{(0,0)\}$
3. for $i := N$ downto 1 do
4. $\quad$ for $s := 1$ to M do
5. $\quad\quad$ //compute $E_{i,s}$
6. $\quad\quad$ for $j := 1$ to M do
7. $\quad\quad\quad$ //the case where f_j is used to execute T_i
8. $\quad\quad\quad \hat{E}_{i,s,j} := P_E(f_s, f_j) + \hat{p}_i(f_j)\frac{A_i}{f_j} +_e +_{\mathbb{F}}{}_{k=1}^{r_i} P_i(k) \times_e \left(\frac{B_i(k)}{f_j} + P_T(f_s, f_j) +_t E_{i+1,j}\right)$
9. $\quad\quad E_{i,s} := \cup_{j=1}^{M} \hat{E}_{i,s,j}$
10. $\quad\quad E_{i,s} := TRIM(E_{i,s}, (1+\epsilon)^{\frac{1}{N}} - 1)$

END

Figure 6: The IDVS scheme

In the IDVS scheme, $N \times M$ speed schedule functions are computed. During the operation of the system, when task T_i is ready to execute and there is time t remaining in the frame, the OS scheduler will detect the current speed s of the processor and use speed $S_{i,s}(t)$ to execute T_i.

5.3 The Algorithm to Solve HDVS

In the HDVS problem, a task is allowed to change speed during its execution. Because of the speed change overhead, a limited number of speed scaling points at which speed may change are predefined for a task [9]. This is similar to predefining the quantum size for OS due to the context switch overhead. The speed remains constant between any two adjacent speed scaling points of a task. For ease of presentation, we choose the bin boundary of the histogram representing the probability distribution of a task as the speed scaling points for the task. By treating each bin of a task as a subtask, the solution to IDVS can be easily extended to solve the HDVS problem. We add one more dimension to the expected energy functions for each task T_i. That is, we compute function $E_{i,s,b}$ ($s = 1, 2, \ldots, M$ and $b = 1, 2, \ldots, r_i$) that denotes the expected energy consumption of executing bin $b, b+1, \ldots, r_i$ of T_i, and $T_{i+1}, \ldots, T_N$ when the current speed is f_s. There is a catch, however. $E_{i,s,b}$ is not only dependent on $E_{i,\cdot,b+1}$, but also $E_{i+1,\cdot,1}$. This is because task T_i may finish at bin b and the rest of the bins will not be executed. Let X be the number of cycles that T_i executes. We compute $\hat{P}_i(b)$, the probability that bin b of task T_i will be executed given that the previous $b-1$ bins have been executed

$$\begin{aligned}\hat{P}_i(b) &= Prob(X \geq B_i(b) | X \geq B_i(b-1)) \\ &= \frac{Prob(X \geq B_i(b) \wedge X \geq B_i(b-1)}{Prob(X \geq B_i(b-1))} \\ &= \frac{1 - cdf_i(b-1)}{1 - cdf_i(b-2)}\end{aligned}$$

where $cdf_i(0) = cdf_i(-1) = 0$. Similar to the IDVS scheme, the helper function $\hat{E}_{i,s,b,j}$ denotes the expected energy consumption of executing bin $b, b+1, \ldots, r_i$ of T_i, and $T_{i+1}, \ldots, T_N$ when the current speed is f_s and speed f_j is used to execute bin b. Thus, a single value of $\hat{E}_{i,s,b,j}$ can be computed as

$$\begin{aligned}\hat{E}_{i,s,b,j}(t) &= \hat{P}_i(b)(P_E(f_s, f_j) + \hat{p}_(f_j)\frac{w_i(b)}{f_j} + E_{i,j,b+1}(t - \\ &\quad P_T(f_s, f_j) - \frac{w_i(b)}{f_j})) + (1 - \hat{P}_i(b))E_{i+1,s,1}\end{aligned}$$

where $w_i(b) = B_i(b) - B_i(b-1)$. Figure 7 shows the details of the HDVS scheme.

PROCEDURE HDVS(ϵ)
1. for $s := 1$ to M do
2. $\quad E_{N+1,s,1} := \{(0,0)\}$
3. for $i := N$ downto 1 do
4. $\quad$ for $b := r_i$ downto 1 do
5. $\quad\quad$ for $s := 1$ to M do
6. $\quad\quad\quad$ //compute $E_{i,s,b}$
7. $\quad\quad\quad$ for $j := 1$ to M do
8. $\quad\quad\quad\quad$ //the case where f_j is used to run bin b of T_i
9. $\quad\quad\quad\quad \hat{E}_{i,s,b,j} := \hat{P}_i(b) \times_e (P_E(f_s, f_j) + \hat{p}_i(f_j) \frac{w_i(b)}{f_j} +_e (P_T(f_s, f_j) + \frac{w_i(b)}{f_j} +_t E_{i,j,b+1})) + (1 - \hat{P}_i(b)) \times_e E_{i+1,s,1}$
10. $\quad\quad\quad E_{i,s,b} := \cup_{j=1}^{M} \hat{E}_{i,s,b,j}$
11. $\quad\quad\quad E_{i,s,b} := TRIM(E_{i,s,b}, (1+\epsilon)^{\frac{1}{\sum_{k=1}^{N} r_k}} - 1))$

END

Figure 7: The HDVS scheme

In the HDVS scheme, the speed schedule functions for each bin of a task is computed. That is, there are a total of $M \times \sum_{i=1}^{N} r_i$ speed schedule functions. However, we do not need this many speed schedule functions for the operation of the system. This is because for a single task, the execution speed is always non-decreasing [9]. This indicates that the speed schedule of a task for a given amount of time has at most M speeds and M speed scaling points. Thus, we compute new speed schedule functions $\hat{S}_{i,s}$, which denotes the speed schedules for the whole task T_i when the current speed is f_s, from $S_{i,\cdot,b}$ ($b = 1, 2, \ldots, r_i$) and use them during the operation of the system. Note that the number of speed schedules in $\hat{S}_{i,s}$ is $|S_{i,s,1}|$.

6. EVALUATION RESULTS

In this section, we use the IDVS and HDVS schemes as the baselines to experimentally evaluate the existing DVS schemes. The existing DVS schemes can be regarded as heuristic solutions because they do not have any performance guarantee under the realistic model. However, history has shown that for certain hard problems, there exist heuristic solutions that work very well and

even close to the optimal in practice. The purpose of the evaluation is to identify those good heuristic schemes.

6.1 Processor Models

We used two embedded processor models in our simulations.

1. Intel XScale (Table 1) [18]. The idle power of XScale is one half the power consumed at the minimum frequency (i.e., 40 mW) [4]. The time and energy penalties for each speed change are reported as 12μs and 1.2μJ, respectively, in [17]. We assume that these numbers are worst-case speed change overheads, which are equivalent to the overheads incurred when changing from the minimum speed to the maximum speed in our power model described in Section 3. Accordingly, we derived the values of the constants ξ_1 and ξ_2 in Equation (1) - (2) to be used in our experiments.

2. IBM PowerPC 405LP (Table 2) [14]. The idle power is assumed to be half the power consumed at the minimum frequency. The time and energy penalties for each speed change are reported as 1ms and 750μJ, respectively, in [14]. We also translated these numbers into our power model as in the case of XScale.

Table 1: XScale speed settings and power consumptions

Speed (MHz)	150	400	600	800	1000
Voltage (V)	0.75	1.0	1.3	1.6	1.8
Power (mW)	80	170	400	900	1600

Table 2: PowerPC 405LP speed settings and power consumptions

Speed (MHz)	33	100	266	333
Voltage (V)	1.0	1.0	1.8	1.9
Power (mW)	19	72	600	750

The power consumptions listed in Table 1 and 2 are obtained by measuring the processor power when running certain benchmarks. In practice, different applications have different instruction mixes, thus resulting in different dynamic power consumptions. We associate each task with a *power scaling factor* to simulate this reality. For example, if a task's power scaling factor is 0.9, it will consume $40 + (400 - 40) \times 0.9 = 364$ mW when executing at frequency 600MHz for XScale.

6.2 Simulation Setup

A frame-based real-time system is characterized by the number of tasks, the power scaling factor for each task, the WCEC of each task, the probability distribution of the number of execution cycles of each task, and the frame length. We simulated systems consisting of 5 and 10 tasks. We only show the results for the systems with 5 tasks because the results for systems with 10 tasks are similar. The power scaling factor was randomly chosen uniformly from 0.8 to 1.2. The WCEC of each task is 500,000,000 and the minimum number of cycles is 5,000,000. The probability function of each task's actual execution cycles is randomly chosen from the 6 representative distributions shown in Figure 8. The bin width of the histograms denoting the probability functions is 5,000,000 cycles. We experimented with 20 frame lengths chosen evenly from $\frac{5 \times WCEC}{f_M}$ (no slack) to $\frac{5 \times WCEC}{f_1}$. In evaluating a DVS scheme on a simulated system, we performed a *run* in which we generated

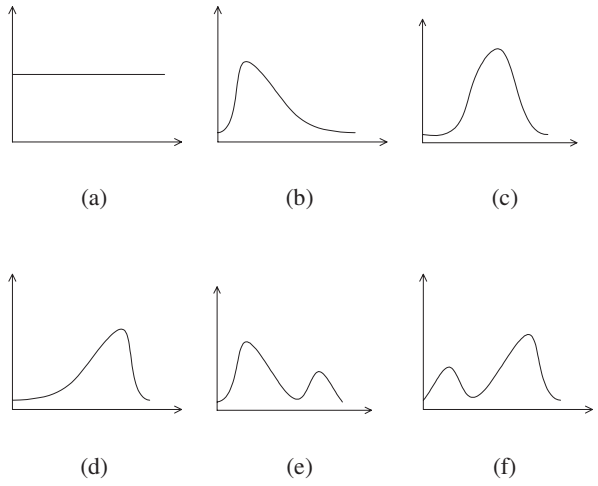


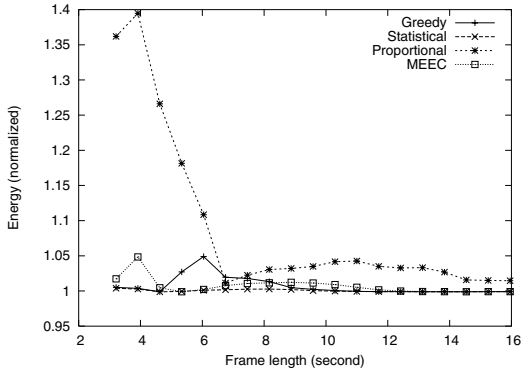
Figure 8: Probability functions: uniform, unimodal1, unimodal2, unimodal3, bimodal1, bimodal2 (from left to right). The Y-axis is probability and the X-axis is number of execution cycles.

100,000 frames and computed the average energy consumption per frame as the energy consumption for that scheme on that system. Under the aforementioned setup, we conducted over 20,000 runs (i.e., 20 billion frames were simulated). For each DVS scheme, we averaged the energy consumption for all systems with the same frame length because we consider slack to be the most influential factor for energy consumption.

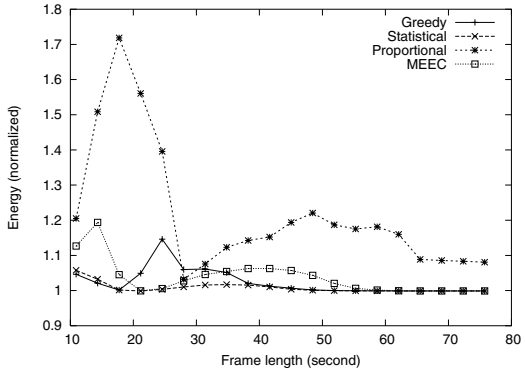
6.3 Evaluation of Inter-task DVS Schemes

We evaluated 4 inter-task DVS schemes: Proportional, Greedy, Statistical, and MEEC. The first 3 schemes [11] are non-stochastic schemes that do not use stochastic information of the workloads. They are all based on the ideal model and need to be patched to fit the realistic model. The MEEC scheme [19] is obtained by patching the optimal stochastic inter-task DVS scheme under the ideal model. The patches for all schemes are similar, including rounding continuous speed up to the lowest feasible discrete speed (i.e., guaranteed to meet deadlines) and subtracting the maximum possible time penalty from the available system time. The energy consumption of all schemes is normalized to that of the IDVS scheme with $\epsilon = 0.05$ (i.e., the energy consumption is guaranteed to be within (1+5%) of the optimal). For all experiments, the number of points in $S_{i,s}$ of the IDVS scheme is at most 97. Recall from Section 4.3 that we only need functions $S_{i,s}$ during the operation of the system. Thus, the space overhead of the IDVS scheme is very small.

Figure 9 shows the evaluation results. We can see that the Statistical scheme is very close to IDVS, which is guaranteed to be within (1+5%) of the optimal and many times is in practice better than the guarantee. Thus, we conclude that Statistical is very close to the optimal even though it is not provably optimal for either the ideal or realistic model. The Greedy scheme also performs relatively well in most cases, comparing with the IDVS scheme. MEEC is outperformed by the Statistical scheme in most cases. This is more evident for the PowerPC 405LP model. Even the Greedy scheme beats MEEC in some cases. This is the so-called anomaly since using more information leads to worse results. The reason is that

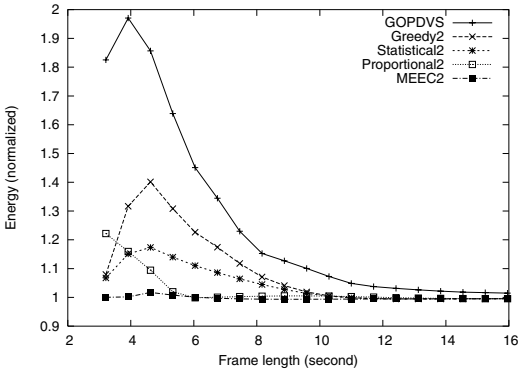


(a) XScale

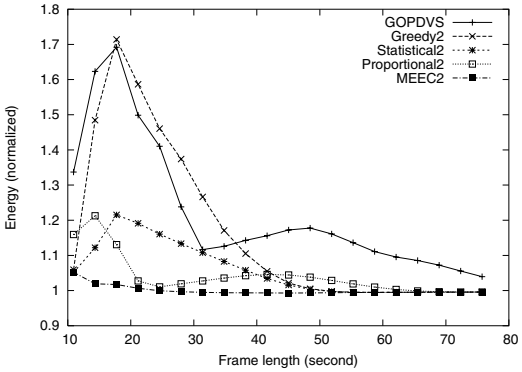


(b) PowerPC 405LP

Figure 9: Evaluation of Inter-task DVS Schemes



(a) XScale



(b) PowerPC 405LP

Figure 10: Evaluation of hybrid DVS Schemes

the rounding-up effect offsets the advantage of using stochastic information.

6.4 Evaluation of Hybrid DVS Schemes

We evaluated 5 hybrid DVS schemes: Proportional2, Greedy2, Statistical2, MEEC2, and GOPDVS. The first 4 schemes are different from their inter-task DVS counterparts only in that up to two speeds are used to execute a task. This is due to the fact that any continuous speed can be emulated by using its two adjacent discrete speeds [7]. In essence, these schemes attempt to emulate inter-task DVS schemes under the ideal model. However, they belong to hybrid DVS schemes technically because the speed may change during the execution of a task (thus they need interrupt support). The GOPDVS scheme [23] is obtained by patching the optimal stochastic hybrid DVS scheme under the ideal model (the patch is a combination of the patch used for inter-task DVS schemes and the patch used by [9] for intra-task DVS schemes). The energy consumption of all schemes is normalized to that of the HDVS scheme with $\epsilon = 0.05$. For all experiments, the number of speed schedules in $\hat{S}_{i,s}$ of the HDVS scheme is at most 1013. Thus, the space overhead of the HDVS scheme is reasonably small.

Figure 10 shows the evaluation results. We note several quantitative differences from the results for inter-task DVS schemes. First, GOPDVS performs poorly, which is a surprising result considering that it is based on the best of all schemes under the ideal model. This is because the excessive rounding of speeds in the GOPDVS scheme makes it drift far away from the optimal solution. Statistical2 performs much worse than its inter-task DVS counterpart. The Greedy2 scheme is the worst of all non-stochastic DVS schemes. The good performance of the MEEC2 scheme is also a surprising result since its main idea is to emulate only the optimal stochastic inter-task DVS scheme under the ideal model.

7. CONCLUSIONS

In this paper, we provide a unified practical approach for obtaining optimal (or provably close to optimal) stochastic inter-task, intra-task, and hybrid DVS schemes under the realistic power model. As a result, we establish tight upper bounds on energy savings for stochastic DVS schemes.

Using our DVS schemes as baselines, we experimentally evaluate previously existing DVS schemes. Based on our experimental results, we conclude that the Statistical scheme [11] is a very good heuristic inter-task DVS scheme and MEEC2 [19] is a very good heuristic hybrid DVS scheme. However, whether this conclusion can be generalized to all systems is still an open question since exhaustive testing is impossible. We will leave it as future work.

8. REFERENCES

[1] H. Aydin, R. Melhem, D. Mossé, and P. Mejia-Alvarez. Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems. In *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, pages 95–105, December 2001.

[2] T. P. Baker and A. Shaw. The cyclic executive model and ada. *The Real-Time Systems Journal*, 1(1):7–26, 1989.

[3] T. Burd and R. Brodersen. Design issues for Dynamic Voltage Scaling. In *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED)*, June 2000.

[4] G. Contreras and M. Martonosi. Power Prediction for Intel XScale Processors Using Performance Monitoring Unit Events. In *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED)*, August 2005.

[5] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, 2001.

[6] F. Gruian. Hard Real-Time Scheduling for Low-Energy Using Stochastic Data and DVS Processors. In *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED)*, August 2001.

[7] T. Ishihara and H. Yasuura. Voltage Scheduling Problem for Dynamically Variable Voltage Processors. In *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED)*, pages 197–202, August 1998.

[8] W. Kim, D. Shin, H. Yun, J. Kim, and S. Min. Performance Comparison of Dynamic Voltage Scaling Algorithms for Hard Real-Time Systems. In *Proceedings of IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2002.

[9] J. Lorch and A. Smith. Improving Dynamic Voltage Scaling Algorithms with PACE. In *Proceedings of ACM SIGMETRICS*, June 2001.

[10] A. Miyoshi, C. Lefurgy, E. V. Hensbergen, R. Rajamony, and R. Rajkumar. Critical Power Slope: Understanding the runtime effects of Frequency Scaling. In *Proceedings of the 16^{th} ACM International Conference on Supercomputing*, June 2002.

[11] D. Mossé, H. Aydin, B. Childers, and R. Melhem. Compiler-Assisted Dynamic Power-aware Scheduling for Real-Time Applications. In *Proceedings of Workshop on Compiler and OS for Low Power (COLP)*, October 2000.

[12] P. Pillai and K. G. Shin. Real-time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)*, pages 89–102, October 2001.

[13] K. Ramamritham and J. Stankovic. Scheduling algorithms and operating systems support for real-time systems. In *Proceedings of the IEEE*, January 1994.

[14] C. Rusu, R. Xu, R. Melhem, and D. Mossé. Energy-Efficient Policies for Request-Driven Soft Real-Time Systems. In *Proceedings of Euromicro Conference on Real-Time Systems (ECRTS)*, Catania, Italy, July 2004.

[15] S. Saewong and R. Rajkumar. Practical Voltage-Scaling for Fixed-Priority RT-Systems. In *Proceedings of IEEE Real-Time Embedded Technology and Applications Symposium (RTAS)*, May 2003.

[16] N.H.E. Weste and K. Eshraghian. *Principles of CMOS VLSI Design*. Addison-Wesley, Reading, MA, 1993.

[17] F. Xie, M. Martonosi, and S. Malik. Compile-Time Dynamic Voltage Scaling Settings: Opportunities and Limits. In *Proceedings of Programming Language Design and Implementation (PLDI)*, June 2003.

[18] Intel XScale Microarchitecture:Benchmarks, 2005. http://web.archive.org/web/20050326232506/http://developer.intel.com/design/intelxscale/benchmarks.htm.

[19] R. Xu, D. Mossé, and R. Melhem. Minimizing Expected Energy in Real-Time Embedded Systems. In *Proceedings of ACM International Conference on Embedded Software (EMSOFT)*, Jersey city, New Jersey, September 2005.

[20] R. Xu, C. Xi, R. Melhem, and D. Mossé. Practical PACE for Embedded Systems. In *Proceedings of ACM International Conference on Embedded Software (EMSOFT)*, Pisa, Italy, September 2004.

[21] F. Yao, A. Demers, and S. Shankar. A Scheduling Model for Reduced CPU Energy. In *Proceedings of IEEE Annual Foundations of Computer Science (FOCS)*, pages 374–382, 1995.

[22] W. Yuan and K. Nahrstedt. Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems. In *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)*, October 2003.

[23] Y. Zhang, Z. Lu, J. Lach, K. Skadron, and M. Stan. Optimal Procrastinating Voltage Scheduling for Hard Real-Time Systems. In *Proceedings of Desing Automation Conference (DAC)*, June 2005.