# System-Wide Energy Minimization for Real-Time Tasks: Lower Bound and Approximation

XILIANG ZHONG and CHENG-ZHONG XU
Wayne State University

We present a dynamic voltage scaling (DVS) technique that minimizes system-wide energy consumption for both periodic and sporadic tasks. It is known that a system consists of processors and a number of other components. Energy-aware processors can be run in different speed levels; components like memory and I/O subsystems and network interface cards can be in a standby state when they are active, but idle. Processor energy optimization solutions are not necessarily efficient from the perspective of systems. Current system-wide energy optimization studies are often limited to periodic tasks with heuristics in getting approximated solutions. In this paper, we develop an exact dynamic programming algorithm for periodic tasks on processors with practical discrete speed levels. The algorithm determines the lower bound of energy expenditure in pseudopolynomial time. An approximation algorithm is proposed to provide performance guarantee with a given bound in polynomial running time. Because of their time efficiency, both the optimization and approximation algorithms can be adapted for online scheduling of sporadic tasks with irregular task releases. We prove that system-wide energy optimization for sporadic tasks is NP-hard in the strong sense. We develop (pseudo-) polynomial-time solutions by exploiting its inherent properties.

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

Additional Key Words and Phrases: Real-Time systems, power-aware scheduling, dynamic power management, dynamic voltage scaling

**ACM Reference Format:**

Zhong, X., and Xu, C.-Z. 2008. System-wide energy minimization for real-time tasks: lower bound and approximation. ACM Trans. Embedd. Comput. Syst. 7, 3, Article 28 (April 2008), 24 pages. DOI = 10.1145/1347375.1347381 http://doi.acm.org/10.1145/1347375.1347381

Authors' address: Xiliang Zhong and Cheng-Zhong Xu, Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI 48202; email: {xlzhong,czxu}@wayne.edu.

## 1. INTRODUCTION

Power management is important for battery-powered embedded systems. Dynamic voltage scaling (DVS) is one of the most effective techniques in power-aware design because the energy consumption of a CMOS circuit has a superlinear dependency on the supply voltage, whereas an approximately linear relationship between voltage and delay [Pering et al. 2000].

Conventional DVS algorithms consider only processor energy, with a focus on optimizing dynamic power consumption. More recently, as leakage power is growing rapidly with each technical generation, researchers study the problem of reducing both the static and dynamic power of a processor [Martin et al. 2002; Jejurikar et al. 2004] . In addition to processors, most applications use other system components during execution, such as memory, flash drives, and wireless interface. Most of the components support multiple power states, such as active, standby (active but idle), and shutdown. If the components need to be active during application execution, processor speed slowdown may lead to an increase of the standby time of other components. The power of those components in a standby state can be significant. For example the standby power of memory and wireless interface can be as large as 0.4 and 1 W in comparison with 2 W CPU power [Jejurikar and Gupta 2004]. Since the overall system energy consumption could be adversely affected by processor slowdown, it is necessary to consider resource standby power to compute energy-efficient processor speeds.

There were recent studies that take into account memory and communication interface with [Cho and Chang 2004; Li and Chou 2005] and without [Zhang and Chanson 2005; Swaminathan and Chakrabarty 2005] the presence of processor slowdown. Studies for system-wide energy optimization, based on a DVS processor, have shown that there is a critical speed for any application, below which it is no longer energy efficient [Jejurikar and Gupta 2004; Choi et al. 2004; Zhu et al. 2004]. The best strategy for system-wide energy minimization is to run a task at a speed no lower than its critical speed and then put the system in low-power state when there are no ready jobs. In an environment with multiple tasks having different deadlines and resource requirements, system-wide energy optimization is to assign the appropriate speed for each task so that all tasks can finish before their deadlines and the total energy consumption is optimized. Existing studies for system-wide energy optimization with a DVS processor focus on periodic tasks with complete timing information known in advance. Even for periodic tasks, it is still hard to get an accurate analysis on a processor with discrete voltage levels. Researchers have tried to solve simplified versions of the problem with relaxed timing requirements [Choi et al. 2004] or on a processor with continuous speed levels [Zhuo and Chakrabarti 2005]. There are also heuristic algorithms to get approximated solutions [Jejurikar and Gupta 2004].

The first contribution of this paper is system-wide energy optimization solutions for periodic tasks on processors with a limited number of speed scales. We prove that the minimization problem is an instance of the NP-hard multiple-choice 0-1 knapsack problem (MCKP) with noninteger coefficients. Existing

solutions to MCKP often assume integer coefficients and this assumption is invalid in the energy optimization. We develop a dynamic programming algorithm to solve the energy optimization problem in pseudopolynomial time. As its worst-case running time may grow rapidly with large number of tasks, we propose a fully polynomial time approximation scheme (FPTAS) to bound the worst-case performance by a predefined value. Experimental results show that the optimal solution leads to an energy consumption bound much lower than that of the heuristic in Jejurikar and Gupta [2004]. The performance degradation of the approximation solution can be bounded by the predefined value with a much smaller average error.

Many typical real-time systems have both periodic and sporadic (aperiodic) tasks. A sporadic task model is effective for applications with unpredictable arrival times and external events, such as operator's commands. The irregular releases of sporadic tasks call for online decisions, assuming no a prior timing information until their releases. Recent studies of online sporadic tasks energy saving, based on DVS, were limited to reduction of dynamic energy consumption of processors [Qadi et al. 2003; Lee and Shin 2004; Zhong and Xu 2007].

The second contribution of this paper is system-wide energy optimization solutions for online scheduling of sporadic tasks on processors with discrete speed scales. The solution can also take static processor energy consumption into consideration. The speed assignment is optimal in the sense that the decision is made online without assumptions about future task releases. We show the problem is essentially a multidimensional multiple-choice 0-1 knapsack problem (MMKP). In general, there is not a pseudopolynomial solution nor an FPTAS that provides bounded performance degradation in moderate running time for an MMKP. In this paper, we present both a pseudopolynomial algorithm and an FPTAS by exploiting inherent properties of online sporadic task scheduling, in which the feasibility of a task does not depend on tasks finished later. Experimental results show that energy consumption of the optimal solution can improve upon a recently proposed competitive algorithm [Zhong and Xu 2007] by up to 57%. The approximation scheme strikes a better trade-off between energy-efficiency and time complexity.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 provides power and task models of the system. We present the optimal and approximated solutions for periodic tasks and sporadic tasks in Section 4 and Section 5, respectively. We evaluate the performance of the algorithms in Section 6. Section 7 concludes the article.

## 2. RELATED WORK

Extensive studies on energy savings have been conducted for periodic tasks under DVS. An emphasis was on dynamic energy consumption of a processor. Recently, the impact of nonnegligible leakage power on DVS was explored. For examples, in Jejurikar et al. [2004], the authors showed the existence of processor critical speed, below which it is not energy-efficient for processor slowdown. Leakage-aware task scheduling with fixed priorities was studied in Quan et al. [2004]. Task scheduling over multiple leakage-aware processors was investigated in Chen et al. [2006]. In this paper, we consider system-wide energy

expenditure including both static processor power and standby power of various system components.

There are several initial studies on system-wide energy optimization. In Choi et al. [2004], the authors proposed an interval based frequency setting policy that would minimize system power consumption of a program subject to performance loss in terms of increased execution time. Their approach can be best applied to applications with soft deadlines. Processors with continuous speed levels were studied in Zhuo and Chakrabarti [2005] with a static speed-setting policy and extensions to online slack distribution and preemption control. In Cheng and Goddard [2005], the authors studied system-wide energy savings of periodic tasks with nonpreemptive shared resources. They considered the case when devices have nonzero transition delays and did not put devices in sleep in order to guarantee all jobs meet their deadlines.

A closely related work is presented in Jejurikar and Gupta [2004]. The authors considered periodic tasks on a processor with discrete speed levels and proposed a heuristic algorithm to the system-wide energy optimization problem. The algorithm starts by setting all tasks to their critical speeds and adjusts speed of a task that can lead to the minimum energy increase per unit time. The adjustment continues until a feasible schedule is found. Although the algorithm is more energyefficient than traditional DVS, it remains unclear how good the algorithm is and whether there exists a solution producing more energy savings. In this paper, we answered these questions by proposing an optimal solution and an approximation scheme with bounded performance degradation.

Furthermore, we adapt the optimal and approximated algorithms to online scheduling of sporadic tasks. A general online DVS approach for sporadic tasks is to set the processor to the lowest speed at which there is no deadline miss. For example, an adaptive algorithm based on linear filter theories was proposed for sporadic tasks on processors with continuous speed levels [Zhong and Xu 2007]. This time-variant algorithm is able to find the lowest feasible speed without future task-release knowledge. In this paper, we consider system-wide energy minimization for sporadic tasks on processors with a more practical discrete speed scale.

The connection between DVS for periodic tasks energy optimization and an MCKP was first established in Mejía-Alvarez et al. [2004]. The authors modeled the processor energy minimization problem as an MCKP, transformed it to a standard knapsack problem, and adapted a heuristic algorithm to solve the problem. In Chen et al. [2005], the authors formulated energy minimization as a subset sum problem, which is known to be a special case of the knapsack problem when power characteristics of all tasks are the same. They then proposed an approximation solution with performance guarantee. An MCKP formulation for energy minimization was presented in Rakhmatov and Vrudhula [2003] with a focus on the impact of battery recovery and discharging rate. Heuristic algorithms were developed for approximated solutions.

All the related formulation only considers dynamic energy reduction of a processor. System-wide energy optimization involves other system devices, in addition to a processor. Their standby states make the lowest feasible speed not necessarily energy efficient. There exist studies on power management for

devices without using processor slowdown. For example, in Swaminathan and Chakrabarty [2005], the authors considered dynamic power management via I/O device scheduling. They showed that the energy minimization can reduce to a *sequencing-within intervals* problem and solved it with both optimal and heuristic algorithms. In this paper, we consider an environment with a DVS processor and multiple devices. We prove that the energy optimization for periodic tasks is an MCKP and develop both optimal and approximated solutions. The approximation algorithm differs from existing heuristics in that it provides performance guarantee for any predefined bounds. In addition, existing studies for system energy optimization are limited to periodic tasks. We make one step forward by connecting online DVS for sporadic tasks to an MMCK problem.

We note that there exist studies that model intratask DVS, which allows many scaling points during the execution of a task, as special cases of subset sum [Xu et al. 2004] and knapsack problems [Xie et al. 2005; Yuan and Nahrstedt 2006]. In contrast, we consider intertask DVS, because it is easy to implement and needs fewer number of voltage/speed switches.

## 3. PRELIMINARIES

### 3.1 Task Model

We study two types of hard real-time tasks, periodic and sporadic (aperiodic) tasks. We assume all tasks to be independent and preemptive, scheduled by the earliest-deadline first (EDF) policy. Consider $n$ periodic tasks in a uniprocessor system. Task $i$ is characterized by a tuple $\{C_i, T_i, D_i\}$, where $T_i$ denotes task period and $C_i$ is task execution time under the reference frequency. The relative deadline $D_i$ is assumed to be equal to task period $T_i$. To guarantee the task set is schedulable, we assume the cumulative utilization does not exceed one, i.e., $\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$.

Another type of hard real-time task under consideration is sporadic tasks. Each sporadic task, denoted by a tuple $\{A_i, C_i, D_i\}$, is characterized by its release time $A_i$, execution time $C_i$, and deadline $D_i$. The arrival time of a sporadic task can be arbitrary, with irregular intervals. We, therefore, do not assume knowledge of future task releases at the time of speed assignment and parameters of a sporadic task are known only after its release.

### 3.2 Processor Energy Model

We consider a uniprocessor computing system. The processor can scale its supply voltage and clock speed with $m$ discrete levels within its operational ranges. We let $S$ denote the normalized clock speed with respect to the reference frequency. Note that whenever the speed of a processor is scaled, its supply voltage is scaled according to a roughly linear relation. For example, according to the power model in Martin et al. [2002], the relation is expressed as

$$f = \frac{(V_{dd} - V_{th})^{\alpha}}{L_d K_6}, \tag{1}$$

where $V_{dd}$ is the supply voltage, $V_{th}$ is the threshold voltage, and $L_d$, $K_6$, and $\alpha$ are technology constants for a given processor. Denote the speed for task $i$ as

$S_i$. The execution time of task $i$ under slowdown speed $S_i$ is $C_i/S_i$. The scaled system utilization is $\sum_{i=1}^{n} \frac{C_i}{T_i S_i}$.

Power consumption of the processor at a frequency $f$ is denoted as $P_{cpu}(f)$. It includes both a dynamic and a static (leakage) power consumption, which can be represented as [Martin et al. 2002]

$$P_{cpu}(f) = C_{eff} V_{dd}^2 f + I_{subn} V_{dd} + P_{bs}, \tag{2}$$

where $C_{eff}$ is the effective switched capacitance and can be application specific, $I_{subn}$ and $P_{bs}$ are used to represent effects of the subthreshold leakage and the reverse bias junction current respectively. In this work, we do not assume any specific form of the power function. By combining (1) and (2), recent studies conclude the existence of *critical speed* that minimizes processor energy [Jejurikar et al. 2004]. Below the speed, static processor energy dominates and it is no longer energy efficient.

### 3.3 System Energy Model

In addition to processors, the system under consideration contains other resources. Power consumption of a resource includes both active and standby power. It is assumed that active and standby power of resources is independent of processor speed. If a resource is not used by an active task, it is put into a low-power state. Task $i$ requires a subset $R_i$ of the resources for execution. Standby energy of resource $j$ for task $i$, denoted by $E_{ij}^{std}$, is equal to the standby power multiplied by task execution time, i.e., $E_{ij}^{std} = P_{ij}^{std} \cdot C_i/S_i$. If the task uses the resource only during a portion of its execution time, we can replace the task execution time with the actual standby time. Similarly, the active energy is the active power multiplied by access time. We assume the access time remains independent of $S_i$. That is, the active energy consumption $E_i^{act}$ does not change with $S_i$. The system energy for task $i$ under speed $S_i$ becomes

$$E_i(S_i) = P_{cpu}(S_i) \cdot \frac{C_i}{S_i} + \sum_{j \in R_i} P_{ij}^{std} \cdot \frac{C_i}{S_i} + E_i^{act}. \tag{3}$$

Similar system-level energy models have been defined in Zhuo and Chakrabarti [2005] Jejurikar and Gupta [2004],

In a processor with a limited number of speed levels, the critical speed of task $i$ is the one that minimizes $E_i(S_i)$. Figure 1 show the energy consumption in execution of one cycle for the Intel's XScale processor [Intel ]. The processor can be run at five different speed levels, 150, 400, 600, 800, and 1000 MHz, with power consumption of 80, 170, 400, 900, and 1600 mW, respectively. We assume there are three other components in the system with standby power of 0.2, 0.4, and 1.0 W, respectively. These are typical values for memory, flash card, and wireless interface [Jejurikar and Gupta 2004]. We draw energy consumption per cycle for applications that use processor only, memory (0.2 W), memory and flash drive (0.6 W), memory and wireless interface (1.2 W). The critical speeds of the applications increase with standby power. For example, the critical speed of the application with standby power of 0.2 W has a normalized critical speed of 0.4; it increases to 0.8 when standby power becomes 1.2 W.

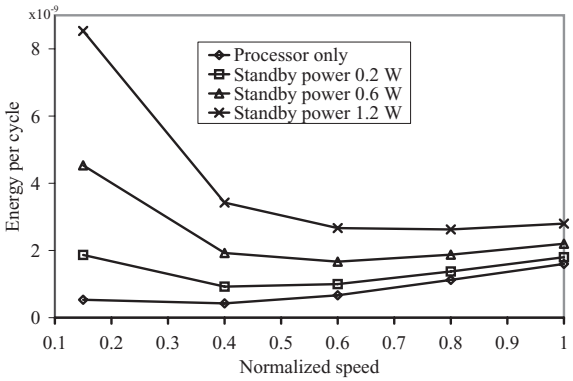


Fig. 1. Energy per cycle versus normalized speed for Intel XScale-based systems with a different standby power.

We denote the system level critical speed of task $i$ as $S_i^{crit}$. Because of the effect of resource standby power, $S_i^{crit}$ is no smaller than the processor critical speed. The minimum speed of task $i$, denoted by $S_i^{min}$, is the larger value of the speed under minimum processor speed, denoted by $S^{min}$, and the critical speed, $S_i^{crit}$. That is, $S_i^{min} = \max\{S^{min}, S_i^{crit}\}$. The set of speeds for task $i$, represented by $\Omega_i$, is a subset of available speeds from $S_i^{min}$ to 1. Its cardinality $m_i$ can be much smaller than the possible number of processor speeds $m$ if the task has a large critical speed.

## 4. PERIODIC TASKS

### 4.1 Problem Formulation

For a set of periodic tasks, we define hyperperiod $T$ to be the least common multiple of $T_1$, ..., $T_n$. As the schedule repeats every $T$ time units, our objective is to minimize the overall energy consumption during each hyperperiod. According to studies in Aydinet al. [2001], for any task $i$, it is optimal for all its task instances to run at the same processor speed. We formulate the optimal voltage scheduling problem as

$$\text{minimize} \sum_{i=1}^{n} \frac{T}{T_i} E_i(S_i) \tag{4}$$

$$\text{subject to} \sum_{i=1}^{n} \frac{C_i}{T_i S_i} \leq 1 \tag{5}$$

$$S_i^{min} \leq S_i \leq 1,\ 1 \leq i \leq n. \tag{6}$$

The constraint (5) ensures the task set under EDF scheduling is feasible. We show, in Theorem 4.1, that the optimization problem is NP-hard.

THEOREM 4.1. *The problem defined by Equations (4–6) is an MCKP with 0-1 variables, which is NP-hard.*

PROOF. Let $x_{ij}$ denote a 0-1 decision variable. If task $i$ is assigned to speed $j \in \Omega_i$, $x_{ij} = 1$; otherwise, $x_{ij} = 0$. Equations (4–6) can be rewritten as:

$$\begin{aligned} \text{minimize } & \sum_{i=1}^{n} \sum_{j \in \Omega_i} \frac{T}{T_i} E_i(S_{ij}) x_{ij} \\ \text{subject to } & \sum_{i=1}^{n} \sum_{j \in \Omega_i} \frac{C_i}{T_i S_{ij}} x_{ij} \leq 1 \\ & \sum_{j \in \Omega_i} x_{ij} = 1, 1 \leq i \leq n. \end{aligned}$$

With a unitary capacity denoted by $c$, we let $-\frac{T}{T_i} E_i(S_{ij})$ be the profit of item $j$ from class $i$, denoted as $p_{ij}$, $\frac{C_i}{T_i S_{ij}}$ be the corresponding weight $w_{ij}$. The problem formulation is then identical to

$$\begin{aligned} \text{maximize } & \sum_{i=1}^{n} \sum_{j \in \Omega_i} p_{ij} x_{ij} \\ \text{subject to } & \sum_{i=1}^{n} \sum_{j \in \Omega_i} w_{ij} x_{ij} \leq c \\ & \sum_{j \in \Omega_i} x_{ij} = 1, 1 \leq i \leq n, \end{aligned}$$

which is an instance of the well-known NP-hard MCKP. □

### 4.2 Optimal Solution

Studies showed that MCKP can be solved optimally in pseudopolynomial time using dynamic programming [Dudzinski and Walukiewicz 1987]. Its optimal solution could be achieved if all its coefficients are integers. With all values being integers, the algorithm forms a dynamic table with indices from 1 to the maximum value and all values in between can be enumerated. To apply this solution to the energy optimization problem with noninteger coefficients, a straightforward approach is to scale and round the energy or utilization values to integers. This is only an approximation and it remains unclear to what extend we should scale the values. In the following, we develop a dynamic programming algorithm for the energy optimization problem with multiple choices of CPU speeds. It is along the line of recent advance of single choice knapsack problem [Kellerer et al. 2004].

4.2.1 *Dynamic Programming with Lists.* The basic idea of the proposed approach is to solve a subproblem with fewer number of tasks and then extend the solution iteratively until the complete problem with $n$ tasks is solved. Consider a subproblem of Equations (4–6) consisting of the first $i$ tasks. We use a pair of two variables $(\bar{u}_{ik}, \bar{e}_{ik})$ to denote the aggregate state of the $i$ tasks, where $\bar{u}_{ik}$ is the accumulated utilization and $\bar{e}_{ik}$ is the corresponding energy sum. Let $l_i$ denote the number of states after task $i$ is enumerated. For $i = 1, \ldots, n$, we

form a list of all the utilization-energy values, arranged in a nondecreasing order of energy,

$$L_i = \langle(\bar{u}_{i1}, \bar{e}_{i1}), (\bar{u}_{i2}, \bar{e}_{i2}), \ldots, (\bar{u}_{il_i}, \bar{e}_{il_i})\rangle.$$

Initially, we have list $L_0$ with zero energy and utilization values. List $L_i$ is obtained in three steps. We first get a number of lists $L'_{ij}$ by adding utilizations and energy values of task $i$ under speed $j \in \Omega_i$ to each state in list $L_{i-1}$. We denote the component-wise addition as

$$L'_{ij} = L_{i-1} \oplus (u_{ij}, e_{ij}).$$

Second, we merge the lists $L'_{ij}$, $j \in \Omega_i$, into one list in a nondecreasing order of energy sums. We then prune the list according to the following conditions.

1. Feasible condition. A schedule is feasible if no task misses its deadline. A state does not lead to a feasible schedule if the sum of the accumulative utilization of the first $i$ tasks and the minimum utilization of the remaining tasks exceeds one, i.e.,

$$\bar{u}_{i*} + \sum_{k=i+1}^{n} \frac{C_k}{T_k} > 1.$$

   It means the resultant solution is not feasible even if the remaining tasks run at the maximum speed. We remove these infeasible states.
2. Energy pruning. After the first condition, we get a partial solution to the first $i$ tasks that can generate a feasible complete solution by simply setting the remaining tasks to the maximum speed. Consider the state with the smallest energy in the solution, $(\bar{u}_{i1}, \bar{e}_{i1})$. With all the remaining tasks running at the maximum speed, their energy consumption $\bar{e}_{i1} + \sum_{k=i+1}^{n} E_k(1)$ becomes an upper bound, because the solution is feasible and the optimal solution should not consume more energy. If any other states in the partial solution require more energy, the states would not appear in the optimal solution and can be removed. The minimum energy required for a state can be obtained by running the remaining tasks at their minimum speeds. This leads to the following pruning condition

$$\bar{e}_{i*} + \sum_{k=i+1}^{n} E_k(S_k^{min}) > \bar{e}_{i1} + \sum_{k=i+1}^{n} E_k(1). \tag{7}$$

3. Dominance pruning. We define a total order on list $L_i$. If two states $(\bar{u}_{ij}, \bar{e}_{ij})$, $(\bar{u}_{ik}, \bar{e}_{ik})$ satisfy the conditions that $\bar{u}_{ij} > \bar{u}_{ik}$ and $\bar{e}_{ij} \geq \bar{e}_{ik}$, or $\bar{u}_{ij} \geq \bar{u}_{ik}$ and $\bar{e}_{ij} > \bar{e}_{ik}$, then the former state is said to be dominated by the latter. The former state will not enter the optimal solution and can be removed from the list. Intuitively, if a state uses more energy while leading to a larger utilization rate than others, it can always be replaced in each iteration.

**Algorithm 1.** Pruning of a list by removing dominated states.

```
1: procedure DOMINANCE-PRUNE(L')
2:     add a state (0, ∞) to the end of list L'
3:     L'' = ∅
4:     repeat
5:         delete the smallest state (ū', ē') from L'
6:         if (ē' < ∞) then
7:             let ū'' = ∞ if L'' == ∅          ▷ (ū'', ē'') is the smallest state in L''
8:             if (ū' < ū'') then
9:                 add (ū', ē') to the end of list L''
10:            end if
11:        end if
12:    until ē' = ∞
13:    return L''
14: end procedure
```

We list the pruning procedure dominance-prune() in Algorithm 1. The procedure finds the list of undominated states in $L'$ and returns it as $L''$. The smallest state in the list $L'$ (or $L''$) is $(\bar{u}'_{*1}, \bar{e}'_{*1})$ (or $(\bar{u}''_{*1}, \bar{e}''_{*1})$). For brevity, we use $(\bar{u}', \bar{e}')$ (or $(\bar{u}'', \bar{e}'')$) to represent the smallest state in the respective lists in Algorithm 1. As we consider the states in a nondecreasing order of energy sums, we have $\bar{e}' \geq \bar{e}''$. There are two cases:

1. $\bar{u}' < \bar{u}''$ and $\bar{e}' \geq \bar{e}''$. Neither state is dominated by the other and we add state $(\bar{u}', \bar{e}')$ to the end of list $L''$.
2. $\bar{u}' \geq \bar{u}''$ and $\bar{e}' \geq \bar{e}''$. State $(\bar{u}', \bar{e}')$ is dominated by $(\bar{u}'', \bar{e}'')$ or $(\bar{u}', \bar{e}') = (\bar{u}'', \bar{e}'')$. We do not add the former state to list $L''$.

Algorithm 2 lists pseudocode for the complete dynamic programming solution; it takes as input a set of periodic tasks and returns the minimum energy. In the first line, we determine the order of task iteration. For the purpose of getting the final solution, tasks can be enumerated in any order, because an

**Algorithm 2.** Energy minimization for periodic tasks using dynamic programming.

```
1: sort the n tasks in a non-increasing order of energy E_i(1), 1 ≤ i ≤ n
2: L_0 = ⟨(0, 0)⟩
3: for i = 1 to n do
4:     for all speeds j ∈ Ω_i do
5:         L'_ij = L_{i-1} ⊕ (u_ij, e_ij)
6:     end for
7:     merge L'_ij into a list L'_i in a non-decreasing order of energy
8:     delete all states in L'_i with ū_{i*} + Σ_{k=i+1}^{n} C_k/T_k > 1
9:     delete all states in L'_i with ē_{i*} + Σ_{k=i+1}^{n} E_k(S_k^min) > ē_{i1} + Σ_{k=i+1}^{n} E_k(1)
10:    L_i = Dominance-Prune(L'_i)
11: end for
12: return the smallest state in L_n
```

arbitrary order does not affect the optimal value and speed assignment. The task order, however, can be an important factor affecting the number of states. We choose to enumerate tasks in a nonincreasing order of task energy under the maximum speed, $E_i(1)$, for $1 \leq i \leq n$. This is motivated by the energy-pruning condition (7). In each iteration, we only need to keep energy values in the interval $[\bar{e}_{i1}, \bar{e}_{i1} + \sum_{k=i+1}^{n} E_k(1)]$. By enumerating the first $i$ tasks with large energy values, the remaining tasks from $i+1$ to $n$ consume the least amount of energy and, hence, result in an interval with a small length of $\sum_{k=i+1}^{n} E_k(1)$.

4.2.2 *Complexity.* Consider the running time of the $i$th iteration that adds task $i$ into the list. Denote $l_i$ as the number of states in list $L_i$, and $l'_i$ as the number of states in list $L'_i$, respectively. Lines 4–6 in Algorithm 2 are linear in the number of states in $L'_i$, which is equal to the multiplication of $l_{i-1}$ and the number of speeds for task $i$. Since $L'_{ij}$ are sorted lists, the running time of the merging of the lists into one sorted list will multiply the sum of the lengths of the lists, $l'_i$, by the time to choose the lowest energy value [Knuth 1997]. As the minimum energy value can be determined in no more than $m_i$ comparisons, line 7 can be completed in $O(m_i l'_i)$. Lines 8 and 9 can be completed in $O(l'_i)$. The running time of dominance-prune() of the $i$th task is also linear in $l'_i$, because all operations within the repeat loop can be finished in a constant time with the states in a sorted order. As a result, the running time of adding task $i$ is in the order of $m_i l'_i$. Assume the number of states in $L'_i$ is bounded by $K$, i.e., $\max_{1 \leq i \leq n} l'_i \leq K$. An upper bound of the running time adding $n$ tasks is $O(\sum_{i=1}^{n} m_i K)$. Similarly, the necessary memory requirement of the algorithm is bounded by the number of states in a list. As we only need to keep the latest list, the space requirement is $O(K)$.

Let $\gamma$, $E_{min}$, and $E_{max}$ denote the computation precision of a real number, lower, and upper bounds of the energy consumption of the given $n$ tasks, respectively. The lower and upper bounds will be taken when all tasks run at their minimum and maximum speeds, respectively. The maximum number of states could be bounded by

$$\frac{E_{max} - E_{min}}{\gamma},$$

which can be unpleasantly large. Algorithm 2 may significantly reduce the computation time by considering quantity sparseness and the pruning conditions. To demonstrate the effectiveness of the state pruning operations in reducing the algorithm complexity, we run Algorithm 2 with a set of randomly generated tasks. Figure 2 shows the number of states during each iteration for ten periodic tasks. Cases with different utilizations under the maximum processor speed are presented. We first run Algorithm 2 without pruning. The number of states reported in each iteration increases rapidly. However, the pruning technique reduces the state space dramatically and maintains no more than 200 states for all utilizations. The total number of states is reduced to 0.1% of its original value. The state space generally expands in the enumeration of first few tasks as few nodes can be pruned by the feasibility and energy conditions. As more tasks are enumerated, more nonoptimal nodes can be eliminated. The

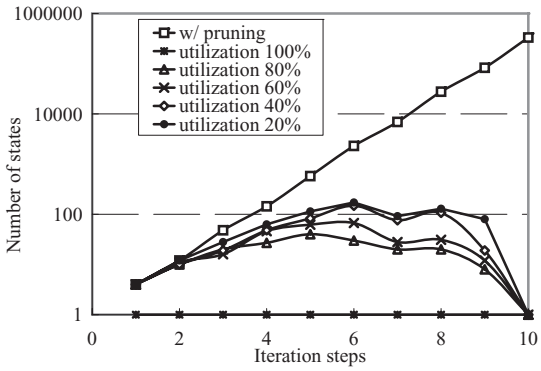


Fig. 2. Effectiveness of state pruning with ten periodic tasks.

state space expands with utilization. This is because with a higher utilization rate, the feasibility condition becomes more effective in identifying infeasible states. When the system becomes fully utilized, the feasibility condition ensures there is only one state in all iterations.

4.2.3 *Optimal Speed Settings.* Algorithm 2 gives the minimum energy value, but does not explicitly return the corresponding optimal speed settings. A straightforward approach in getting the speeds is to maintain an array of speed settings for each state and update the array whenever a task is added. In this case, an array of size $i$ is needed for each state in $L_i$. Storing and updating the arrays for each subproblem increase both the memory requirement and the running time by a factor of $i$ in the $i$th iteration. We can improve the approach by noting that the speed settings after the $i$th iteration will differ from those after the $(i-1)$th iteration only by the speed for task $i$. It is only necessary to store the latest speed settings in each iteration. We can achieve this by extending the state to a tuple of three values $(\bar{u}_{ik}, \bar{e}_{ik}, j)$, with $j$ as the speed index of task $i$. The optimal speed settings can be constructed by backtracking the $n$ lists. Starting from task $n$, we find the speed with accumulative utilization closest to one, update utilization under the assigned speed, and repeat the procedure for the remaining tasks. A simple upper bound of the running time of backtracking is $O(nK)$, which does not change the order of the overall time complexity. However, we need to keep lists of all iterations, which would lead to a space complexity of $O(nK)$. In short, the time and space complexities in getting the optimal speed settings are $O(\sum_{i=1}^{n} m_i K)$ and $O(nK)$.

### 4.3 A Fully Polynomial Time Approximation Scheme

The state space defined in Algorithm 2 can grow rapidly with the number of tasks; it might be computationally expensive to get the optimal solution with a large number of tasks and speed levels. Practically, it is often not necessary to find the optimal solution with limited time and resources. An approximation algorithm providing a nearly optimal performance, especially with a worst-case performance guarantee, is more desirable if it can be finished in reasonable time. The quality of approximation is measured by a relative performance ratio.

An algorithm is said to be an $\epsilon$-approximation scheme if, for a given value of $\epsilon \in (0, 1)$, we have $\frac{\widetilde{E}-E^*}{E^*} \leq \epsilon$, where $\widetilde{E}$ and $E^*$ are the energy consumption of the approximated and the optimal solutions. A desirable approximation scheme should have a running time that increases with polynomial time in both the number of tasks and the performance ratio. This leads to a classification of schemes called fully polynomial time approximation scheme (FPTAS) [Kellerer et al. 2004]. We will propose an FPTAS for the energy optimization problem.

The approximation scheme works by reducing the number of states in each iteration. The basic idea is to divide the energy values into a number of groups each of length $r$. We will show how to determine $r$ according to $\epsilon$ later. Each energy value belongs to one of the groups. For a group size $r$, we scale energy for task $i$ under speed $j$ and then round it up to the next integer, $\lceil \frac{e_{ij}}{r} \rceil$, which will be used to represent the energy values in each group. As a result of the rounding up, the number of states can be reduced greatly so are the running time and space required to solve the scaled problem.

Let $S_i^r$ be the optimal speed setting of task $i$ for the scaled problem, which is not necessarily the same as the optimal setting in the original problem $S_i^*$. Denote the approximated energy consumption of the scaled problem with group length $r$ as $\widetilde{E}^r$. We have

$$\begin{aligned}\widetilde{E}^r &= \sum_{i=1}^{n} E_i(S_i^r) \leq \sum_{i=1}^{n} r\lceil \frac{E_i(S_i^r)}{r} \rceil \leq \sum_{i=1}^{n} r\lceil \frac{E_i(S_i^*)}{r} \rceil \\ &\leq \sum_{i=1}^{n} r(\frac{E_i(S_i^*)}{r} + 1) = \sum_{i=1}^{n} (E_i(S_i^*) + r) = E^* + nr. \end{aligned} \tag{8}$$

The second inequality is because the minimum energy value of the scaled problem is no larger than the energy under the speed settings from the original problem.

According to Equation (8), the absolute error of the approximated solution is, at most, $n \cdot r$. To bound the relative error below a given ratio $\epsilon$, we need to have

$$\frac{\widetilde{E}^r - E^*}{E^*} \leq \frac{nr}{E^*} \leq \epsilon.$$

It follows that

$$r \leq \frac{\epsilon E^*}{n}. \tag{9}$$

As we mentioned, a simple lower bound of $E^*$ is to run each task under its minimum speed, which is $E_{min}$. The right-hand side of Equation (9) can be bounded by

$$\frac{\epsilon E^*}{n} \geq \frac{\epsilon E_{min}}{n}.$$

Hence, setting $r = \epsilon E_{min}/n$ satisfies condition (9) and bounds the relative performance error under $\epsilon$. We list an outline of the approximation in Algorithm 3. For completeness, we include modifications for backtracking the speed settings.

**Algorithm 3.** A fully polynomial time approximation scheme for periodic tasks.

1: sort the $n$ tasks in a non-increasing order of energy $E_i(1)$, $1 \leq i \leq n$
2: $L_0 = \langle (0, 0, 0) \rangle$
3: calculate $E_{min}$ and set $r = \frac{\epsilon E_{min}}{n}$
4: **for** $i = 1$ to $n$ **do**
5: **for all** speeds $j \in \Omega_i$ **do**
6: $L'_{ij} = L_{i-1} \oplus (u_{ij}, \lceil \frac{e_{ij}}{r} \rceil, j)$
7: **end for**
8: merge $L'_{ij}$ into a list $L'_i$ in a non-decreasing order of energy
9: delete all states in $L'_i$ with $\bar{u}_{i*} + \sum_{k=i+1}^{n} C_k/T_k > 1$
10: delete all states in $L'_i$ with $\bar{e}_{i*} + \sum_{k=i+1}^{n} E_k(S_k^{min}) > \bar{e}_{i1} + \sum_{k=i+1}^{n} E_k(1)$
11: $L_i$ = Dominance-Prune($L'_i$)
12: **end for**
13: backtrack the lists from $L_n$ to $L_1$ to get the speed settings $S^r = [S_1^r, ..., S_n^r]$
14: return speed settings $S^r$ and the smallest state in $L_n$

For a given performance bound $\epsilon$, we can determine an upper bound $K$ of the total number of undominated states by

$$K = \frac{E_{max} - E_{min}}{r} = \frac{E_{max} - E_{min}}{E_{min}} \frac{n}{\epsilon}.$$

Replacing $K$ in the running time of the optimal algorithm, we get a time complexity in the order of $\sum_{i=1}^{n} m_i \frac{n}{\epsilon}$, which is polynomial in the number of tasks, speed levels, and $1/\epsilon$. We summarize the result in Theorem 4.2.

THEOREM 4.2. *The approximated solution in Algorithm 3 is a fully polynomial time approximation scheme of the energy optimization problem for periodic tasks with any relative performance ratio* $\epsilon \in (0, 1)$.

### 4.4 An Example

In this section, we present an example to simulate the execution of the proposed optimal and approximation algorithms and compare their performance with the algorithm CS-DVS from Jejurikar and Gupta [2004].

Consider a simple real-time system with four periodic tasks as shown in Table I. They have different execution time $C_i$ and utilization $U_i$ under the maximum processor speed. The accumulative system utilization is 70%. We assume task 1 uses CPU only. In addition to CPU, task 2 uses memory; task 3 requires both memory and flash drive to be standby during one half of its execution; task 4 requires memory and wireless interface to be standby during one half of its execution. Standby power consumption of the memory, flash drives, and wireless interface was set to 0.2, 0.4, and 1 W, respectively. The processor model was based on the Intel's XScale, as described in Section 3.

Table II shows the energy consumption $e_{ij}$ and utilization $u_{ij}$ starting from the critical speed of each task. Values from the optimal solution and the approximation scheme with $\epsilon = 0.5$ are shown. The group size $r$ of the 0.5-approximation is computed as $\frac{\epsilon E_{min}}{n} = \frac{0.5 \cdot (e_{11} + e_{21} + e_{32} + e_{42})}{4} = 0.9325$. Energy and

Table I. An Example Real-Time Workload with Four Periodic Tasks

| Tasks | $C_i$ | $T_i$ | $U_i$ | Required Resources | Normalized Critical Speed |
|---|---|---|---|---|---|
| Task 1 | 6.4 | 16 | 0.4 | CPU only | 0.4 |
| Task 2 | 1.6 | 20 | 0.08 | CPU + memory | 0.4 |
| Task 3 | 1.2 | 12 | 0.1 | CPU + memory + flash drive | 0.6 |
| Task 4 | 1.08 | 9 | 0.12 | CPU + memory + wireless interface | 0.6 |

Table II. Energy Consumption and Utilization under Different (Normalized) Processor Speeds

| | | Values in the Optimal Solution | | | | | Values in the 0.5-Approximation | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 0.4 | 0.6 | 0.8 | 1.0 | | 0.4 | 0.6 | 0.8 | 1.0 |
| Task 1 | $e_{ij}$ | 2.72 | 4.267 | 7.20 | 10.24 | $\lceil \frac{e_{ij}}{r} \rceil$ | 3 | 5 | 8 | 11 |
| | $u_{ij}$ | 1.00 | 0.667 | 0.50 | 0.40 | $u_{ij}$ | 1.00 | 0.667 | 0.50 | 0.40 |
| Task 2 | $e_{ij}$ | 1.48 | 1.60 | 2.20 | 2.88 | $\lceil \frac{e_{ij}}{r} \rceil$ | 2 | 2 | 3 | 4 |
| | $u_{ij}$ | 0.20 | 0.133 | 0.10 | 0.08 | $u_{ij}$ | 0.20 | 0.133 | 0.10 | 0.08 |
| Task 3 | $e_{ij}$ | n/a | 2.00 | 2.25 | 2.64 | $\lceil \frac{e_{ij}}{r} \rceil$ | n/a | 3 | 3 | 3 |
| | $u_{ij}$ | n/a | 0.167 | 0.125 | 0.10 | $u_{ij}$ | n/a | 0.167 | 0.125 | 0.10 |
| Task 4 | $e_{ij}$ | n/a | 1.26 | 1.62 | 2.052 | $\lceil \frac{e_{ij}}{r} \rceil$ | n/a | 2 | 2 | 3 |
| | $u_{ij}$ | n/a | 0.20 | 0.15 | 0.12 | $u_{ij}$ | n/a | 0.20 | 0.15 | 0.12 |

utilization values of approximation schemes with other performance bounds can be obtained in a similar way.

Figure 3 illustrates the process to producing the optimal solution to the example problem using tree structure. The $i$th level of the tree shows the partial solution to the first $i$ tasks. Each node in the $i$th level represents a state in list $L_i$. The root node represents the initial state with zero utilization and energy values. The algorithm starts with branching on all four speed levels of task 1, which generates four level-1 nodes. The left-most level-1 node can be eliminated, because it already has a utilization of 1.0 and cannot generate a feasible complete solution after enumeration of the remaining tasks. The minimum energy used by the first task among the other three nodes is 4.267; the maximum energy required by the remaining three tasks can be computed as $e_{25} + e_{35} + e_{45} = 7.572$. This leads to an energy upper bound of 11.839. Node 3 and 4 exceed this bound even if the remaining three tasks are set to their minimum speeds and they are removed. Node 2 becomes the only state in all level-1 nodes that will appear in the optimal solution. Its branching generates four level-2 nodes corresponding to different speed levels of task 2. They are pruned in a similar way to the level-1 nodes. This process continues until all four tasks are included. The optimal state is reached by the level-4 node (0.987, 11.159). Its corresponding speed assignment can be obtained by backtracking the tree structure. We can have a similar illustration about the solution process of the approximation scheme with scaled energy values. Details are omitted.

We present the energy minimization solutions from several different algorithms in Table III. The algorithm No-DVS always runs tasks at the highest speed without scaling; it is included for reference. The algorithm CS-DVS is effective in energy saving, with 25% less energy consumption than that of no DVS. However, the algorithm may incorrectly favor tasks with large energy increase per unit time. The resultant speed settings leads to a scaled system utilization

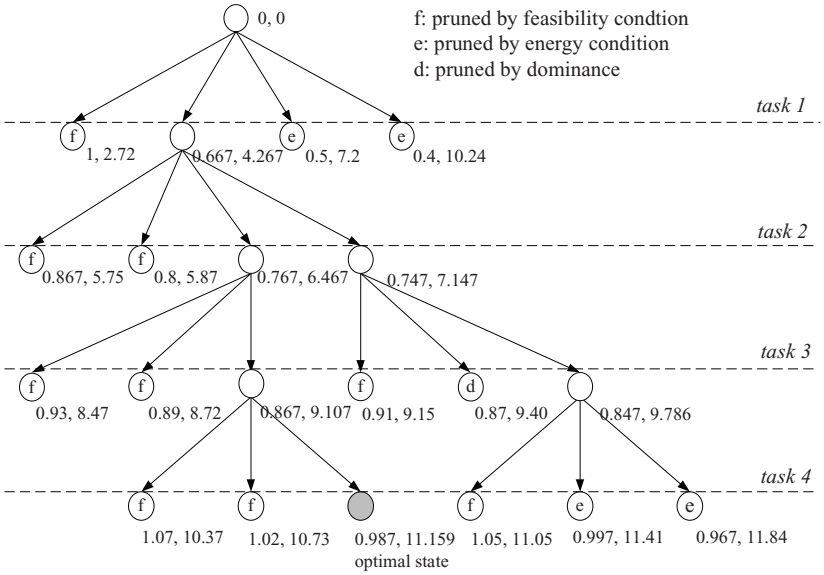


Fig. 3. An illustration of the optimal solution process in Algorithm 2.

Table III. Solutions to the Example Workload from Different Algorithms

| Algorithms | Assigned Speed | Scaled Utilization % | Run-Time (s) | Energy |
|---|---|---|---|---|
| No-DVS | [1 1 1 1] | 70 | 0 | 17.812 |
| CS-DVS | [0.8 0.6 1 1] | 85.3 | 0.016 | 13.492 |
| Optimal | [0.6 0.8 1 1] | 98.7 | 0.046 | 11.159 |
| 0.1-Approximation | [0.6 0.8 1 1] | 98.7 | 0.030 | 11.159 |
| 0.5-Approximation | [0.6 0.8 1 1] | 98.7 | 0.025 | 11.159 |
| 0.8-Approximation | [0.6 1 1 1] | 96.7 | 0.021 | 11.839 |

of only 85.3 and 21% more energy consumption than the optimal solution in this example. We list results of the approximation scheme with performance bounds 0.1, 0.5, and 0.8 also in Table III. It can be seen that both 0.1 and 0.5 approximations obtain the same optimal solution, although their energy values are scaled. The result of the 0.8 approximation is not optimal. The performance degradation, however, is only 6% relative to the optimal solution, much smaller than the predefined 80% bound. All algorithms complete quickly in less than 50 ms in a Pentium 4 with 2-GHz processor.

## 5. SPORADIC TASKS

### 5.1 Problem Formulation

Because of the irregular release times of sporadic tasks, we consider online scheduling with-task timing parameters known only after task releases. When there is a new task release, we first determine whether the task should be admitted by an acceptance test as suggested in [Hong et al. 1998]. Suppose there are $n$ ready tasks sorted in a nondecreasing order of deadlines. Let $\hat{C}_i$, $\hat{D}_i$ denote the remaining execution time and deadline of task $i$. We define the

maximum instantaneous utilization of all tasks $U_{max}$ as $U_i = \sum_{j=1}^{i} \frac{\hat{C}_j}{D_j}$ and $U_{max} = \max_{1 \le i \le n} U_i$. The task will not be admitted if $U_{max} > 1$.

Once the task is accepted, we will determine a speed assignment according to all ready tasks with the objective of minimizing their energy consumption. For brevity, we will consider a set of tasks released at time zero. In the general case for tasks released at different times, we can easily adapt the formulation by considering only ready tasks and by changing $C_i$, $D_i$ to residue execution time and deadline, in a similar way to [Dey et al. 1996]. The optimal speed assignment can be formulated as

$$\text{minimize} \sum_{i=1}^{n} E_i(S_i) \tag{10}$$

$$\text{subject to} \sum_{i=1}^{k} \frac{C_i}{S_i} \le D_k, 1 \le k \le n \tag{11}$$

$$S_i^{min} \le S_i \le 1, \ 1 \le i \le n. \tag{12}$$

The set of constraints in Equation (11) ensures that the schedule is feasible for all tasks. The formulation is involved with more constraints than Equations (4–6) for periodic tasks. The problem is also NP-hard, as shown in Theorem 5.1.

THEOREM 5.1. *The problem defined by Equations (10–12) is an MMKP with 0-1 variables, which is NP-hard.*

PROOF. Let $x_{ij} = 1$, if task $i$ is assigned to speed $j$, $j \in \Omega_i$; otherwise, $x_{ij} = 0$. The problem can be rewritten as:

$$\text{minimize} \sum_{i=1}^{n} \sum_{j \in \Omega_i} E_i(S_{ij}) x_{ij}$$

$$\text{subject to} \sum_{i=1}^{k} \sum_{j \in \Omega_i} \frac{C_i}{S_{ij}} x_{ij} \le D_k, 1 \le k \le n$$

$$\sum_{j \in \Omega_i} x_{ij} = 1, 1 \le i \le n.$$

Let $D_k$ be the capacity $c_k$, $-E_i(S_{ij})$ be the profit of item $j$ from class $i$, denoted by $p_{ij}$, with the corresponding weight $\frac{C_i}{S_{ij}}$, denoted by $w_{ijk}$. We then transform the problem to

$$\text{maximize} \sum_{i=1}^{n} \sum_{j \in \Omega_i} p_{ij} x_{ij}$$

$$\text{subject to} \sum_{i=1}^{n} \sum_{j \in \Omega_i} w_{ijk} x_{ij} \le c_k, 1 \le k \le n$$

$$\sum_{j \in \Omega_i} x_{ij} = 1, 1 \le i \le n.$$

It is a class of MMKP with $n$ dimensions, which is known to be NP-hard. □

### 5.2 Optimal and Approximated Solutions

MMKP is a generalization of MCKP and harder to solve. In fact, a typical MMKP is NP-hard in the strong sense, which means any optimal solutions will lead to strictly exponential computational times and there is no FPTAS [Parra-Hernandez and Dimopoulos 2005]. We will show that by investigating inherent properties of sporadic task scheduling, we are able to find a pseudopolynomial solution and an FPTAS with bounded performance degradation in polynomial running time.

We assume the tasks have been sorted in a nondecreasing order of their deadlines. Task $k$ will finish before its deadline as long as the execution time sum of the first $k$ tasks does not exceed $D_k$; it is not necessary to consider tasks completed later. This reduces the complexity of constraint checking in the optimization process. Combined with the iteration nature of the dynamic programming solution in Algorithm 2, it enables us to satisfy the constraint at each iteration step. For example, after task $i$ is added and we have a pruned list $L_i$, we find all feasible states in $L_i$ satisfying the constraint of task $i$ and its feasibility remains satisfied in later iterations. Therefore, after all the iterations, we get a list of states satisfying all the $n$ constraints.

Algorithm 4 lists principles of sporadic tasks speed assignment adapted from Algorithm 2. We use $t_{ij}$ to denote the execution time of task $i$ under speed $j$, $\bar{t}_{i*}$ to denote one of the execution time sums of the first $i$ tasks in list $L_i$. Unlike the periodic tasks case, we do not check feasibility including tasks from $i+1$ to $n$ during the $i$th iteration in line 8. This is because there are $n$ constraints according to Equation (11). The feasibility test needs to be performed for all the $n-i$ tasks and would increase the worst-case running time to $O(\sum_{i=1}^{n} m_i nK)$. The presented algorithm has the same time complexity as the case of periodic tasks, which is $O(\sum_{i=1}^{n} m_i K)$ in running time and $O(nK)$ in space.

Note that the speed assignment is optimal only in the sense that it minimizes the energy consumption in executing all tasks without assumption about future task releases. The energy consumption could be further reduced with more knowledge about future task release times, which would lead to an offline algorithm.

**Algorithm 4.** Energy minimization for sporadic tasks using dynamic programming.

1: sort the $n$ tasks in a non-decreasing order of deadlines
2: $L_0 = \langle (0,0,0) \rangle$
3: **for** $i = 1$ to $n$ **do**
4: **for all** speeds $j \in \Omega_i$ **do**
5: $L'_{ij} = L_{i-1} \oplus (t_{ij}, e_{ij}, j)$
6: **end for**
7: merge $L'_{ij}$ into a list $L'_i$ in non-decreasing order of energy
8: delete all states in $L'_i$ with $\bar{t}_{i*} > D_i$
9: delete all states in $L'_i$ with $\bar{e}_{i*} + \sum_{k=i+1}^{n} E_k(S_k^{min}) > \bar{e}_{i1} + \sum_{k=i+1}^{n} E_k(1)$
10: $L_i$ = Dominance-Prune($L'_i$)
11: **end for**
12: backtrack the lists from $L_n$ to $L_1$ to get the speed settings $S^* = [S_1^*, .., S_n^*]$
13: return $S^*$ and the smallest state in $L_n$

Algorithm 4 determines the lower bound for the NP-hard problem Equations (10–12). As the time complexity becomes significantly large with increased problem size, the algorithm may not get the solution in moderate running time. In addition, as the speed decision is made online, we expect it to be efficient with bounded performance degradation, rather than optimal. It is not difficult to see that Algorithm 4 can be extended to an FPTAS in a similar way to the approximation technique in Section 4.3. We omit details for brevity.

## 6. EXPERIMENTAL RESULTS

In this section, we evaluate the effectiveness of the proposed algorithms in energy savings for both periodic and sporadic tasks. Experiments are based on the Intel's XScale processor as described in Section 3. We assume negligible voltage switching overhead and CPU idle power. In addition to the processor, the system has three other resources with standby power consumption of 0.2, 0.4, and 1 W. These are typical values for memory, flash drives, and wireless interface. Standby time for the resources as a percentage of task execution time is assumed to be in the range of [20 and 60%], [10 and 25%], and [5, and 20%], respectively. The values of standby power and time periods were based on the experimental setting of Jejurikar and Gupta [2004].

### 6.1 Periodic Tasks

We first evaluate effectiveness of the algorithms for periodic tasks. Tasks were assigned random periods in the range of [10 and 120 ms]. We varied the processor utilizations from 0.1 to 1 in a step 0.1. For each utilization value, we generated 20 task sets, each containing five tasks. Utilization of each task was randomly chosen subject to the total utilization. Execution time of each task at the maximum processor speed is a multiplication of utilization and task period. We label the resources by integers starting from 1 and assume tasks have different resource requirements as $R_1 = \emptyset$, $R_2 = \{1\}$, $R_3 = \{1, 2\}$, $R_4 = \{1, 3\}$, and $R_5 = \{1, 2, 3\}$.

We performed experiments on the generated task sets and report normalized energy consumptions in Figure 4. We compare the proposed optimal algorithm for periodic tasks, referred to as OPT-P, with the heuristic algorithm CS-DVS [Jejurikar and Gupta 2004]. As critical speeds of all tasks are no lower than 0.4, it is not energ efficient to set the processor below the speed. Both policies set the processor speed to 0.4 for utilizations up to 40% by working at the critical speed. We have omitted those values to improve readability. The reported energy consumption is normalized with respect to OPT-P. For all utilization values, CS-DVS consumes up to 16% more energy than the optimal solution. The difference can in part, be, explained by the fact that CS-DVS terminates as soon as it finds a feasible solution. It may end up with a total processor utilization far less than 1. Although the solution can be possibly improved by a postprocessing step choosing other speeds or tasks to get an utilization closer to 1, there is no worst-case performance guarantee. Another observation is that the approximation schemes can bound the performance degradation well below their worst-case values. For example, with a given ratio 0.1, the average

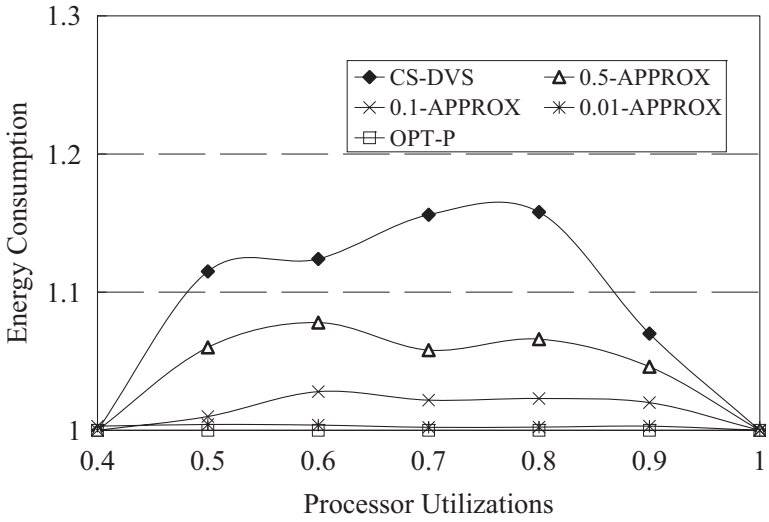


Fig. 4. System energy consumption for periodic tasks.

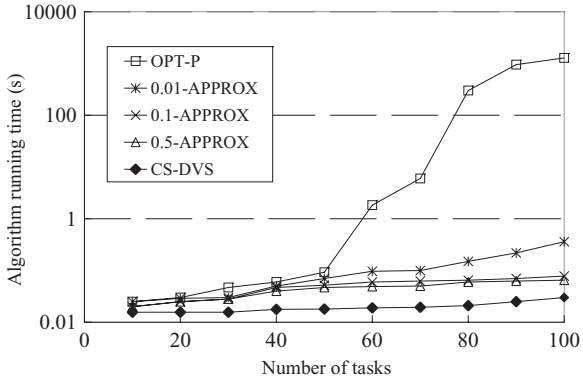


Fig. 5. Time complexity of the optimal and approximation algorithms for periodic tasks.

error relative to the optimal solution is no larger than 3%. A small ratio, such as 0.01, can provide a nearly optimal solution with a polynomial complexity.

Although both the optimal and approximation algorithms are more effective in energy minimization than CS-DVS, they come at the cost of a higher time complexity. We conducted experiments to measure the algorithm running time for different numbers of tasks. Figure 5 shows the running time collected in a Pentium 4 machine with a 2-GHz processor. It can be observed that running times of the proposed algorithms are below 0.09 ms for systems with up to 50 tasks. The time complexity of the optimal algorithm increases sharply when the task number exceeds 50. In all the cases, however, the approximation algorithms require no more than 0.4 s to finish. We conclude that the proposed optimal solution could be effectively used as the energy lower bound and the approximation would provide a better energy-time trade-off than CS-DVS.

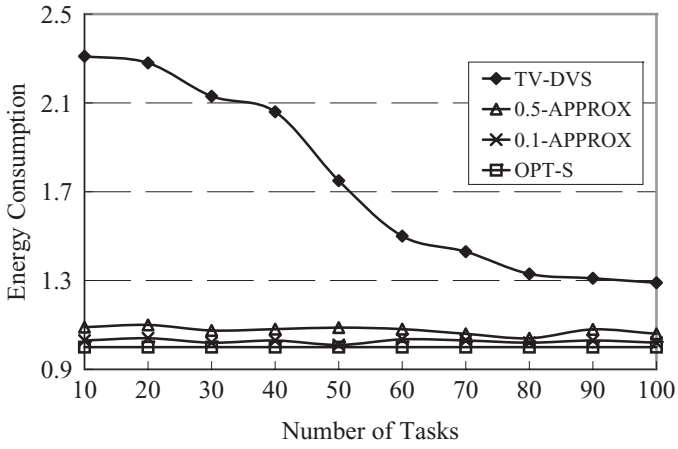


Fig. 6. System energy consumption for sporadic tasks.

### 6.2 Sporadic Tasks

Under the same energy model as the last experiment, we evaluate the proposed online scheduling algorithm for sporadic tasks. We include the results from a recent time-variant DVS algorithm (TV-DVS) for online sporadic tasks [Zhong and Xu 2007] for comparison. Since the competitor is based on a continuous speed level, we rounded the computed speeds up to their nearest neighbors. To compare the performance, we generated a group of sporadic tasks in a similar way [Zhong and Xu 2007]. The minimal interarrival time was set to 10 ms. We randomly set the maximum utilization of each task under the constraint that the accumulative utilization of all tasks at their maximum release rates does not exceed one. Task deadlines were drawn from [10 to 120 ms]. Execution time of a task is a multiplication of its maximum utilization and deadline. Interarrival times of a task were chosen from an exponential distribution with averages ranging from 20 to 50 ms. Each task was randomly assigned a subset of the available resources.

Figure 6 shows the results because of TV-DVS, the proposed optimal algorithm (OPT-S), and the approximation algorithms in a run of 10 min. It is expected that TV-DVS consumes much more energy without considering the impact of critical speed. When the number of tasks is small, energy consumption of TV-DVS can be as large as 2.3 times that of OPT-S. The performance difference is mainly because of the inefficiency of TV-DVS in running tasks at speeds lower than their critical speeds. With increased number of tasks, system utilization increases so that TV-DVS improves its performance by running tasks at higher speeds. As TV-DVS is optimal only for processors with continuous speed levels and assumes identical power characteristics of all tasks, the performance gap can still be as large as 30%.

We compare the overall execution time of the algorithms in generating speed schedules of the 10 min. run in Figure 7. Because TV-DVS has a linear time complexity, its running time is consistently smaller than the others. All algorithms have comparable complexities when the task number is small. The running time

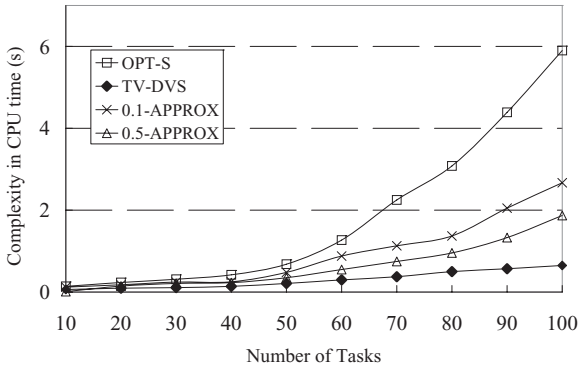


Fig. 7. Time complexities of the online algorithms for sporadic tasks.

of OPT-S grows faster with the number of tasks. However, with more tasks, the number of energy values in each approximation group increases, which makes the approximation scheme more effective in reducing the complexity. Figures 6 and 7 suggest that the approximation scheme provide a better trade-off between energy and time complexity than the time-variant algorithm.

## 7. CONCLUSION

We have presented solutions to system-wide energy optimization for a group of hard periodic real-time tasks. The energy minimization is proved to be an NP-hard MCKP with noninteger coefficients. We develop a pseudopolynomial dynamic programming algorithm in getting the optimal solution. A fully polynomial time approximation scheme (FPTAS) is proposed to provide bounded performance guarantee with moderate running time even for large problem size. Evaluation results in comparison with a heuristic approach [Jejurikar and Gupta 2004] show the energy efficiency of the exact solution as the theoretical lower bound and the effectiveness of the approximation schemes in providing bounded performance guarantee. For example, a relative performance ratio of 0.01 can improve over the heuristic algorithm up to 16% with a polynomial running time.

In addition to periodic tasks, we prove that energy minimization for sporadic tasks is an MMKP, which is known to have strictly exponential running time in getting the optimal solution. We exploit inherent properties of the problem and show that we can have a pseudopolynomial algorithm for exact solutions and an FPTAS with bounded performance guarantee. The exact solution is optimal in the sense it is online without any assumption about future task releases at the time of speed assignment decision. Experimental results compared with a recent approach [Zhong and Xu 2007] demonstrate superiority of the proposed algorithms in striking a better trade-off between energy savings and time complexity.

Considering the fact that task execution time does not scale at the same rate as the processor speed, we can extend the model and algorithms in this paper by splitting the task running time into on-chip and off-chip time, as

in Choi et al. [2004] and Seth et al. [2006]. We finally note that recent techniques considering task procrastination in extending shutdown intervals [Jejurikar and Gupta 2004], preemption control [Kim et al. 2004] [Zhuo and Chakrabarti 2005], nonpreemptive resources [Cheng and Goddard 2005], voltage transition overhead [Mochocki et al. 2005] can be applied to improve the policy. These issues are important but beyond the scope of this paper.

## ACKNOWLEDGMENTS

This research was supported in part by U.S. NSF grants ACI-0203592, CCF-0611750, and NASA grant 03-OBPR-01-0049. Parts of the results in this paper were presented in Zhong and Xu [2006].

## REFERENCES

AYDIN, H., MELHEM, R. G., MOSSÉ, D., AND MEJÍA-ALVAREZ, P. 2001. Optimal reward-based scheduling for periodic real-time tasks. *IEEE Trans. Comput. 50,* 2, 111–130.

CHEN, J.-J., KUO, T.-W., AND SHIH, C.-S. 2005. $(1+\epsilon)$ approximation clock rate assignment for periodic real-time tasks on a voltage-scaling processor. In *Proceedings of the International Conference on Embedded Systems.* 247–250.

CHEN, J.-J., HSU, H.-R., AND KUO, T.-W. 2006. Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium.* 408–417.

CHENG, H. AND GODDARD, S. 2005. Integrated device scheduling and processor voltage scaling for system-wide energy conservation. In *Proceedings of the International Workshop on Power-Aware Real-time Computing*. 24–29.

CHO, Y. AND CHANG, N. 2004. Memory-aware energy-optimal frequency assignment for dynamic supply voltage scaling. In *Proceedings of the International Symposium on Low-Power Electronics and Design*. 387–392.

CHOI, K., LEE, W., SOMA, R., AND PEDRAM, M. 2004. Dynamic voltage and frequency scaling under a precise energy model considering variable and fixed components of the system power dissipation. In *Proceedings of the International Conference on Computer-Aided Design*. 29–34.

DEY, J. K., KUROSE, J. F., AND TOWSLEY, D. F. 1996. On-line scheduling policies for a class of iris (increasing reward with increasing service) real-time tasks. *IEEE Trans. Comput. 45,* 7, 802–813.

DUDZINSKI, K. AND WALUKIEWICZ, S. 1987. Exact methods for the knapsack problem and its generalizations. *Europ. J. Operational Res. 28*, 3–21.

HONG, I., POTKONJAK, M., AND SRIVASTAVA, M. B. 1998. On-line scheduling of hard real-time tasks on variable voltage processor. In *Proceedings of the International Conference on Computer-Aided Design*. 653–656.

INTEL. *Intel-XScale Architecture*. http://www.intel.com/design/intelxscale/.

JEJURIKAR, R. AND GUPTA, R. K. 2004. Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems. In *Proceedings of the International Symposium on Low-Power Electronics and Design*. 78–81.

JEJURIKAR, R., PEREIRA, C., AND GUPTA, R. K. 2004. Leakage aware dynamic voltage scaling for real-time embedded systems. In *Proceedings of the Design Automation on Conference.* 275–280.

KELLERER, H., PFERSCHY, U., AND PISINGER, D. 2004. *Knapsack Problems*. Springer Verlag, New York.

KIM, W., KIM, J., AND MIN, S. L. 2004. Preemption-aware dynamic voltage scaling in hard real-time systems. In *Proceedings of the International Symposium on Low-Power Electronics and Design*. 393–398.

KNUTH, D. 1997. *The Art of Computer Programming, Volume 3: Sorting and Searching,* 3rd Ed. Addison-Wesley, Reading, MA. Chapter 5.2–5.3.

LEE, C.-H. AND SHIN, K. G. 2004. On-line dynamic voltage scaling for hard real-time systems using the edf algorithm. In *Proceedings of the IEEE International Real-Time Systematics Symposium*. 319–327.

LI, D. AND CHOU, P. H. 2005. Application/architecture power co-optimization for embedded systems powered by renewable sources. In *Proceedings of the Design Automation Conference*. 618–623.

MARTIN, S. M., FLAUTNER, K., MUDGE, T. N., AND BLAAUW, D. 2002. Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. In *Proceedings of the International Conference on Computer-Aided Design*. 721–725.

MEJÍA-ALVAREZ, P., LEVNER, E., AND MOSSÉ, D. 2004. Adaptive scheduling server for power-aware real-time tasks. *ACM Trans. Embedded Comput. Syst. 3,* 2, 284–306.

MOCHOCKI, B., HU, X. S., AND QUAN, G. 2005. Practical on-line dvs scheduling for fixed-priority real-time systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium*. 224–233.

PARRA-HERNANDEZ, R. AND DIMOPOULOS, N. J. 2005. A new heuristic for solving the multichoice multidimensional knapsack problem. *IEEE Trans. Syst. Man Cybern. A 35,* 5, 708–717.

PERING, T., BURD, T. D., AND BRODERSEN, R. W. 2000. Voltage scheduling in the iparm microprocessor system. In *Proceedings of the International Symposium on Low-Power Electronics and Design*. 96–101.

QADI, A., GODDARD, S., AND FARRITOR, S. 2003. A dynamic voltage scaling algorithm for sporadic tasks. In *Proceedings of the IEEE Real-Time Systems Symposium* 52–62.

QUAN, G., NIU, L., HU, X. S., AND MOCHOCKI, B. 2004. Fixed priority scheduling for reducing overall energy on variable voltage processors. In *Proceedings of the IEEE Real-Time Systems Symposium* 309–318.

RAKHMATOV, D. N. AND VRUDHULA, S. B. K. 2003. Energy management for battery-powered embedded systems. *ACM Trans. Embedded Comput. Syst. 2,* 3, 277–324.

SETH, K., ANANTARAMAN, A., MUELLER, F., AND ROTENBERG, E. 2006. Fast: Frequency-aware static timing analysis. *ACM Trans. Embedded Comput. Syst. 5,* 1, 200–224.

SWAMINATHAN, V. AND CHAKRABARTY, K. 2005. Pruning-based, energy-optimal, deterministic i/o device scheduling for hard real-time systems. *ACM Trans. Embedded Comput. Syst. 4,* 1, 141–167.

XIE, F., MARTONOSI, M., AND MALIK, S. 2005. Bounds on power savings using runtime dynamic voltage scaling: an exact algorithm and a linear-time heuristic approximation. In *Proceedings of the International Symposium on Low-Power Electronics and Design*. 287–292.

XU, R., XI, C., MELHEM, R. G., AND MOSSÉ, D. 2004. Practical pace for embedded systems. In *Proceedings of the International Conference on Embedded Syst.* 54–63.

YUAN, W. AND NAHRSTEDT, K. 2006. Energy-efficient cpu scheduling for multimedia applications. *ACM Trans. Comput. Syst. 24,* 3, 292–331.

ZHANG, F. AND CHANSON, S. T. 2005. Improving communication energy efficiency in wireless networks powered by renewable energy sources. *IEEE Trans. Vehicular Technol. 54,* 6, 2125–2136.

ZHONG, X. AND XU, C.-Z. 2006. System-wide energy minimization for real-time tasks: Lower bound and approximation. In *Proceedings of the International Conference on Computer-Aided Design*.

ZHONG, X. AND XU, C.-Z. 2007. Energy-aware modeling and scheduling for dynamic voltage scaling with statistical real-time guarantee. *IEEE Trans. Comput. 56,* 3, 358–372. (A preliminary version appeared in Proceedings of RTSS'05)

ZHU, D., MELHEM, R. G., AND MOSSÉ, D. 2004. The effects of energy management on reliability in real-time embedded systems. In *Proceedings of the International Conference on Computer-Aided Design*. 35–40.

ZHUO, J. AND CHAKRABARTI, C. 2005. System-level energy-efficient dynamic task scheduling. In *Proceedings of the Design Automation Conference* 628–631.

Received August 2005; revised October 2006; accepted December 2006