System-Wide Dynamic Power Management for Multimedia Portable Devices

Abstract

Energy reduction is critical to increase the mobility and battery life for today's pervasive computing systems. At the same time, energy reduction must be subject to the real-time constraints and quality of service (QoS) requirements for multimedia applications running on these systems. This paper presents a novel run-time scheduling approach to reduce the system-wide *energy consumption for such systems. In this paper, the multimedia applications are modeled using a popular weakly hard real-time model, i.e., the (m,k)-model. Our experimental results show that, by judiciously scheduling the real-time tasks and shutting down the processor and/or peripheral devices, our approach can lead to significant energy savings while guaranteeing the $(m,k)-$firm deadlines at the same time.*

1 Introduction

Energy reduction is critical to increase the mobility for today's pervasive computing systems and thus becomes a wide spread research area. Power aware scheduling has been proven to be an effective way to reduce the energy consumption. Rooted in the traditional real-time scheduling technology, the power aware scheduling techniques change the system computing performance accordingly based on the dynamically varying computation demand. Most of the existing power aware scheduling techniques (e.g. [2, 12]) have been focused on reducing the processor energy consumption alone. While the processor is one of the major power hungry units in the system, other peripherals such as network interface card, memory banks, disks also consume significant amount of power. The empirical study by Viredaz and Wallach reveals that the processor core consumes around 28.8% of total power when playing a video file on a hardware testbed [25] for handheld devices, while the DRAM consumes about 28.4% of the total power. Note that this testbed [25] lacks disk storage and wireless networking capability, which may contribute as much power consumption as the processor core if not more [27, 6]. This implies that the techniques that attack the processor energy alone may not be overall energy efficient.

Two main types of techniques are reported in the literature. The first one is commonly known as the *dynamic power down* (DPD), i.e., to shut down a processing unit and save power when it is idle. The second one is called *dynamic voltage scaling* (DVS) which updates the processor's supply voltages and working frequencies dynamically. While DVS techniques can dramatically reduce the dynamic power consumption for the processor, DPD techniques seems to be more promising in reducing the system-wide overall energy when the energy consumptions of peripheral devices are taken into consideration. Even for the processor itself, the energy efficiency of DVS is becoming limited as IC technology continue its evolution [28], especially when the leakage power is increasing exponentially and will soon surpass the dynamic power consumption [7]. DPD, on the other hand, is one of the most intuitive and effective ways to control the leakage power consumption. Moreover, most peripheral devices do not support DVS at all. As a result, the research on employing DPD has regained its momentum to reduce the system-wide energy consumption.

Recently, several techniques (e.g. [10]) have been proposed to reduce the energy consumption for hard real-time systems consisting of both core processors and peripheral devices. However, few real-time applications are truly *hard* real-time, *i.e.*, many practical real-time applications can allow some deadline provided that user's perceived quality of service (QoS) constraints can be satisfied. The *weakly hard real-time model* is more suitable for this type of applications. In the weakly-hard real-time model, tasks have both firm deadlines (*i.e.*, deadline missing is useless) and a throughput requirement (i.e., *sufficient* task instances must meet deadlines to provide required quality levels). For example, Ramanathan *et. al.* [21] proposed a so-called $(m,k)-$model, with a periodic task being associated with a pair of integers, *i.e.*, (m,k), such that among any k consecutive instances of the task, at least m of the instances must finish by their deadlines for the system behavior to be acceptable. A *dynamic failure* occurs, which implies that the temporal QoS constraint is violated and the scheduler is thus considered failed, if within any consecutive k jobs more than $(k-m)$ job instances miss their deadlines.

In this paper, we study the problem of employing DPD to reduce the system-wide energy consumption with guaranteed QoS for a weakly hard real-time system. Specifically, we adopt the (m,k)-model to capture the QoS requirement for the real-time application. A key challenge for this problem has to do with the definition of which jobs are mandatory, i.e., whose deadlines have to be met to guarantee no dynamic failure occur, and which jobs can be optional. This problem has shows to be NP-hard even without the considerations of the power conservation [20]. In our approach, we employ a run-time technique and dynamically choose and execute the mandatory jobs in such a way that facilitates the system shut down. Our experiments shows that by judiciously choosing and merging the mandatory jobs, our techniques can lead to significant energy savings while still *guaranteeing* the (m,k)-firm deadlines.

The rest of the paper is organized as follows. We first introduce some previous work in Section 2. Then in Section 3, the system model and motivations are introduced. Section 4 describes a feasibility condition to guarantee the (m,k)-firm deadlines. Section 5 presents two methods to delay the job execution to extend the idle interval. Section 6 discusses how to judiciously execute the optional jobs. Section 7 presents our experimental results. Section 8 draws the conclusions.

2 Related Work

Most DVS real-time scheduling approaches focus on saving energy consumed by the processor only. Recently, a number of researches (e.g. [8, 11, 29]) are reported to reduce the energy consumption for systems consisting of DVS processors and peripheral devices. Kim and Ha [10] proposed a technique for *hard* real-time system, while scheduling decisions are made on a timeslot-by-timeslot basis. To facilitate a run-time mechanism, the processor speed for each task is determined by analyzing the energy savings based on a pre-determined set of execution times. Jejurikar and Gupta [8] introduced a heuristic search method to slow down the processor speed and optimize the energy usage by both the processor and peripheral devices. Zhuo and Chakrabarti [29] proposed a theoretical formulation of the optimal scaling factor and computed it numerically. Based on this factor, they introduced a dynamic scheduling technique that reduces the potential excessive preemptions among tasks to further reduce the system wide energy consumption.

As a traditional energy-saving technique, DPD has also been widely adopted in real-time scheduling [3]. A majority of DPD techniques (e.g. [23]) have been proposed for soft real-time systems, where task deadlines can be missed albeit with reduced quality levels. There are also a number of papers (e.g. [5, 24]) deal with the power optimization for hard real-time systems.

Few real-time applications are truly *hard* real-time, i.e., missing one task deadline does not necessarily crash the entire application or system. Many real-time applications, such as multimedia and communication applications, can often tolerate occasional deadline misses, but too much deadline misses cannot satisfy user's perceived quality of service (QoS) requirement. While the statistic information such as the average deadline miss rate is commonly used to quantify the system performance, this metric can be problematic. Note that even a very low average miss rate tolerance cannot prevent a large number of deadline misses from occurring in a very short period of time. This may cause the loss of critical information which cannot be reconstructed and therefore severely degrade the service quality from user's perspective.

We are more interested in developing scheduling techniques for weakly hard real-time systems. Several weakly-hard models have been introduced [21, 13, 26]. Ramanathan *et. al.* [21] proposed a so-called $(m,k)-$model, with a periodic task being associated with a pair of integers, i.e., (m,k), such that among any k consecutive instances of the task, at least m of the instances must finish by their deadlines for the system behavior to be acceptable. Koren *et. al.* [13] proposed a 'skip-over' model, which is a special case of (m,k) model with $m = k-1$. West *et. al.* [26] introduced another similar model, called the *window-constrained* model, which requires that within any *non-overlapped* and *consecutive* windows each of which containing k jobs, at least m of them can meet their deadlines. In [1], Alenawy and Aydinintroduced a scheduling technique to maximize (instead of guarantee) the quality of service level under energy constraints for real-time systems with (m,k)-constraints. Niu and Quan [19] presented a combined static/dynamic DVS scheduling method to reduce processor energy with (m,k)-guarantee. Both techniques only take the processor energy consumption into consideration. All these work target reducing the processor energy only.

3 Preliminary

In this section, we first introduce the system model and then discuss a motivation example.

3.1 System models

We model the multimedia applications with n independent periodic tasks, $\mathcal{T} = \{\tau_0, \tau_1, \cdots, \tau_{n-1}\}$, scheduled according to the earliest deadline first (EDF) policy [14], i.e., the scheduling policy that can best utilize the resource. Each task contains an infinite sequence of periodically arriving instances called *jobs*. We use J_{ij} to represent the jth job of task τ_i. Task τ_i is characterized using five parameters, *i.e.*, $(T_i, D_i, C_i, m_i, k_i)$. T_i, D_i ($D_i \leq T_i$), and C_i represent the period, the deadline and the worst case execution time for τ_i, respectively. A pair of integers, i.e., (m_i, k_i) $(0 < m_i \leq k_i)$, represent the QoS requirement for τ_i, requiring that, among any k_i consecutive jobs of τ_i, at least m_i jobs meet their deadlines.

The system architecture consists of two functional units: a core processor and a peripheral device. Both the processor and the peripheral device can be shut down and waken up later when idle time expired. We denote the processor power with P_{pact} when running a task, and P_{pidle} when the processor is idle (yet still *on*). When the processor is shut down, its power consumption is denoted as P_{psleep}. The peripheral device in our system can be in one of two states: *active* or *sleep*. When the processor is active, the peripheral devices must be also in active mode to provide timely service. We assume that the device consumes the same power during its active mode no matter whether it is idle or not. The power consumption for the device is denoted as P_{dact} and P_{dsleep} for its active mode and sleep mode, respectively.

Time and energy needed to be consumed to shut-down and later wake up the processor and device. It will not be feasible or beneficial to shut down the system if the idle interval is not long enough. We use T_{min} to represent the minimal idle interval that can be feasibly shut-down with positive energy-saving gains.

3.2 The motivations

Our goal is to shut down the processor and device efficiently and guarantee the (m,k)-constraints in the mean time. To schedule a real-time task set with (m,k)-firm deadline involves two sub-problems: (i) mandatory/optional partitioning problem, i.e., to determine if a job should be mandatory or optional, and (ii) scheduling problem, i.e., to schedule these jobs properly to guarantee their deadlines. Both problems are proven to be NP-hard [20].

The mandatory/optional partition decision can be made statically (off-line) or dynamically (on-line). Two known static mandatory/optional partitioning strategies are reported in literature. The first one is called *the deeply-red pattern* or *R-pattern*, which was proposed by Koren *et al.* [13]. According to this technique, let

$$\pi_{ij} = \begin{cases} 1 & 0 \leq j \ mod \ k_i < m_i \\ 0 & \text{otherwise} \qquad\qquad j = 0, 1, \cdots \end{cases} \tag{1}$$

Then job J_{ij} is market as mandatory if $\pi_{ij} = 1$, or optional otherwise. The second one is proposed by Ramanathan *et al.* [22] as follows.

$$\pi_{ij} = \begin{cases} 1 & \text{if } j = \lfloor \lceil \frac{j \times m_i}{k_i} \rceil \times \frac{k_i}{m_i} \rfloor \\ 0 & \text{otherwise} \qquad\qquad j = 0, 1, \cdots \end{cases} \tag{2}$$

The (m,k)-pattern defined with formula (2) has the property that mandatory jobs are marked evenly, and is therefore referred as the *evenly distributed pattern* (or *E-pattern*) [18].

The most significant advantage of applying static patterns is that they enable the application of theoretic real-time techniques to analyze the system feasibility and therefore can guarantee the desired requirements off-line. The problem, however, is its pessimism due to its worst case scenario assumption and the poor adaptivity in dealing with the run-time variations, which is inherent in many multimedia applications. For example, Figure 1(a) and (b) show the EDF schedules by determining mandatory jobs based on E-patterns and R-patterns, respectively, for a task set with three periodic tasks. As expected, Figure1(a) shows the mandatory jobs distributed evenly and causes a large number of idle intervals, i.e., as many as 11 idle intervals in time interval [0,96], which is not favorable for DPD mechanism. On the other hand, one tends to believe that the R-pattern assignments may help to reduce the number of idle intervals since the mandatory jobs from the same task are assigned consecutively. However, this is not necessarily true as shown in Figure1(b), i.e., as many as 12 idle intervals within the same time interval according to R-patterns. The reasons are two folds. First, from equation (1), an R-pattern always marks the first m_i jobs as mandatory jobs. The mandatory jobs from different tasks are likely to overlap for the first "window" but not necessarily for the following windows due to the differences of k's and periods from different tasks. Second, even though mandatory jobs and the time intervals in which they are supposed to run are overlapped (e.g., see interval [0,15] in Figure 1(b)), idle intervals still exist due to the deadline and arrival constraints for the tasks.

Figure 1(c) presents a schedule that can cut the number of idle intervals to as small as 4. A small number of idle intervals usually means longer idle interval length. As a result, the energy overhead for shutting down the processor and device can be reduced. In addition, some idle intervals that previously cannot be shut down because they are too short can now be done so. This can result in significant energy savings. A careful study of Figure 1(c) would reveal that such solution is obtained by employing an irregular mandatory/optional job pattern, i.e., neither E-pattern nor R-pattern, together with carefully delaying the execution of mandatory jobs. The challenges are then how to define appropriate mandatory jobs and how to delay the executions of these jobs effectively such that the idle intervals can be merged while the (m,k)-constraints can be guaranteed. In following sections, we propose an integrated run-time technique to attack these challenges.

4 Meeting the (m,k)-constraints

From the motivation example shown above, it is evident that the existing static (m,k)-patterns cannot effectively merge the idle intervals. How to devise new static (m,k)-patterns that can cluster mandatory jobs for this purpose is

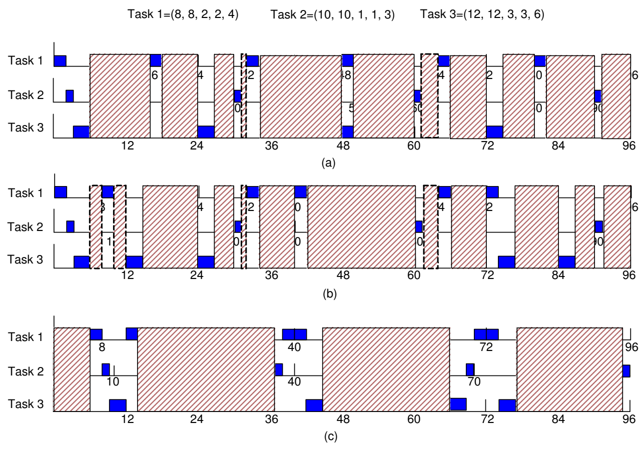


Figure 1. (a) The EDF schedule for three tasks according to E-patterns; (b) The EDF schedule for same tasks based on R-patterns; (c) A better schedule for the same task set.

an interesting problem and needs further study. Nonetheless, the static patterns are usually based on worst case scenarios and less adaptive. Judiciously exploiting the variations, inevitable in the runtime environment, dynamically can be extremely beneficial. The problem is how to determine the patterns dynamically and ensure that no dynamic failure will ever occur.

A number of dynamic mandatory/optional partitioning heuristics are proposed (e.g. [21, 1]) with no guarantee for the deadlines of mandatory jobs at all. Currently, two dynamic techniques published can ensure the (m,k)-guarantee. Niu *et. al.* [18] proposed to shift the E-pattern dynamically when an optional job meets its deadline. Since E-pattern tends to distributed the mandatory jobs evenly, it is advantageous to schedule task sets with high utilizations but not to reduce the number of idle intervals. Bernat *et. al.* [4] proposed a Bi-Modal Scheduler under fixed-priority scheduling policy based on the worst case timing analysis. However, a well-formulated analytical worst case timing analysis is not available under the EDF scheduling policy. In what follows, we develop a sufficient condition to ensure the feasibility when choosing mandatory jobs dynamically in our approach (The proof is omitted due to page limit.)

Lemma 1 *Given system $\mathcal{T}$, let $\mathcal{M}$ be the mandatory job set according to their R-patterns. Then if $\mathcal{M}$ is EDF-schedulable, a job (i.e. J_p) can be marked as mandatory and meet its deadline if for each $\tau_i \in \mathcal{T}, i = 0, 1, ..., n-1$, no more than m_i jobs (including J_p) among any consecutive k_i jobs are marked as mandatory.*

The necessary and sufficient condition to test the schedulability for a mandatory job set according to R-patterns is formulated in [17]. Based on this condition, Lemma 1 implies that as long as a task set is schedulable under R-patterns, we can flexibly choose a job as mandatory provided we do not choose more than m_i among k_i consecutive jobs from same task τ_i. Therefore, when the system is idle, we can intentionally delay the *assignment* of mandatory jobs in such a way that they can be congregated. However, recall that in the motivation example, even though the mandatory jobs are allocated closely, large number of idle intervals may still exist due to their arrival and deadline constraints. In next section, we solve this problem by carefully delaying the *execution* of mandatory jobs.

5 Delaying the execution of mandatory jobs

When the processor is idle, delaying the execution of mandatory jobs helps to extend the idle intervals. However, it may also potentially cause mandatory jobs to miss their deadlines and thus cause dynamic failure. A number of papers published [5, 9] proposed to compute the job delay amount for a hard real-time task set based on its utilization factor. These approaches cannot be applied for real-time system with weakly hard real-time constraints since the famous condition, i.e., $U \leq 1$ is not necessary for a weakly hard real-time system to be feasible under EDF. In this section, we develop two sufficient conditions for delaying the execution of mandatory jobs as late as possible without causing any dynamic failure. Before we introduce these sufficient conditions, we first introduce the following definition.

Definition 1 *Assume that $\mathcal{M}$ is the mandatory job sets from $\mathcal{T}$ according to R-pattern and schedulable, and let R_i be the worst case response time (i.e., the time from a job arrival to its finish). The* delay factor *for τ_i (denoted as Y_i) is defined as*

$$Y_i = (D_i - R_i). \tag{3}$$

Even though there is no analytical method to compute the worst case response time under EDF for mandatory jobs determined according to R-pattern, we can always scan through the interval from $[0, LCM(k_i T_i)]$, $i = 0, ..., n-1$ to find the exact value for each task. With Definition 3, our first sufficient condition is formulated in the following Theorem.

Theorem 1 *Let $\mathcal{M}$ be the mandatory job set such that no more than m_i mandatory jobs assigned for any k_i consecutive jobs from $\tau_i \in \mathcal{T}$. Assume that processor is idle at $t = t_0$, and let the arrival time for mandatory job J_i from τ_i immediately after t_0 be r_i. Then if the processor resumes its execution at*

$$T_{LS}(\mathcal{M}) = \min_i (r_i + Y_i), i = 0, 1, ..., n-1, \tag{4}$$

no mandatory job in $\mathcal{M}$ will miss its deadline.

Theorem 1 allows us to determine the maximal delay for mandatory jobs based on worst case response time, which is available off-line. The advantage of this approach is its small run-time overhead. Unfortunately, same as any other off-line strategy, it suffers the pessimistic estimation due to its assumption of the worst case scenario, as exemplified in Figure 2. Figure 2(a) shows the schedule of a task set of three tasks according to their static R-patterns. We can readily identify that $Y_1 = 4, Y_2 = 0$, and $Y_3 = 2$. Assume a dynamically determined mandatory job sets shown in Figure 2(b). (We can see that the job execution intervals are largely overlapped.) Since $Y_2 = 0$, the mandatory job from Task 2 cannot be delayed according to Theorem 1, and there is one idle interval between [28,36]. On the other hand, however, if we delay the processor execution till $t = 27$ (as shown in Figure 2(d)), all jobs can meet their deadline and no idle interval exists. This is because that Theorem 1 computes the maximal delay assuming the job always takes its worst case response time. When a job has a much smaller response time, it can be delayed further and may thus be more effective in reducing the idle intervals.

Mochocki *et. al.* [15] introduced a method to compute the latest starting time (LST) for a real-time job set. Their method is based on the following lemma.

Lemma 2 *[15] Let job set $\mathcal{J}$=$\{J_0, J_1, ..., J_s\}$ and $J_i = \{r_i, d_i, c_i\}$, where r_i, d_i, and c_i refer to the arrival time, deadline, and execution time of J_i, respectively. Let*

$$t_{LS}(J_i) = d_i - \sum_{J_k \in hp(J_i)} c_k, \tag{5}$$

where $hp(J_k)$ is the jobs with the same or higher priorities than that of J_k. Then the latest starting time (LST) of $\mathcal{J}$, i.e., $T_{LS}(\mathcal{J})$, without violating deadline constraints is

$$T_{LS}(\mathcal{J}) = \min_i t_{LS}(J_i). \tag{6}$$

Lemma 2 helps to compute the LST for a given job set. However, this method cannot be readily applied in our dynamic approach where the job set is not statically determined. Niu *et al.* [16] later extended Lemma 2 and compute LST based on information from only a subset of the jobs. This approach has a much lower complexity and hence is more suitable for on-line purpose. We use Figure 2(c) to illustrate this approach.

Assume the processor is idle before $t = 19$ in Figure 2(c). Since the LST for a job set is bounded by the earliest deadline of the jobs (so called *delay bound* and denoted as T_B), and is usually known on-line (i.e. $T_B = 32$ in this case), it is desirable to estimate LST for the entire job set based on the jobs arriving before the delay bound,i.e., J_{11} and J_{21}. As pointed out in [16], the LST computed by employing equation (5) directly for J_{11} and J_{21} may not be valid since the validity of LST in Lemma 2 is ensured by employing (5) for *every* job in the job set. In this regard, Niu *et al.* proposed to use the *effective deadline* of a job (i.e. the time before which a job has to be finished such that it will not cause any other job to miss deadline) in place of the deadline in (5). To keep low complexity of the algorithm, they simply defined the effective deadline for a job by its own deadline or the earliest arrival time of the coming low priority job, whichever is smaller. In Figure 2(c), both the effective deadlines for J_{11} and J_{21} happen to be 32. Therefore, based on equation (6), $T_{LS} \min(t_{LS}(J_{11}), t_{LS}(J_{11})) = 23$.

The approach in [16] delays mandatory jobs further than the one applying Theorem 1 and shorten the idle interval in Figure 2(b). However, it fails to eliminate the idle interval. In what follows, we present another method to estimate the LST for mandatory jobs. Our method maintains the same computational complexity as that in [16] but with a more accurate estimation. Specifically, our method is formally formulated in Theorem 2.

Theorem 2 *Let $\mathcal{M}$ be the mandatory job set such that no more than m_i mandatory jobs assigned for any k_i consecutive jobs from $\tau_i \in \mathcal{T}$. Assume that processor is idle at $t = t_0$, and let the delay bound (i.e., the earliest deadline for the coming mandatory jobs) be T_B for $\mathcal{M}$. Then no mandatory job in $\mathcal{M}$ will miss its deadline if the processor resumes its execution at $\tilde{T}_{LS}(\mathcal{M})$, where*

$$\tilde{T}_{LS}(\mathcal{M}) = \min_{J_i \in \mathcal{J}_s}(d_i^* - \sum_{J_k \in hp(J_i)} c_k), \tag{7}$$

where $\mathcal{J}_s$ consists of mandatory jobs from $\mathcal{M}$ with arrival times earlier than T_B but later than t_0, and

$$d_i^* = \min_p(d_i, r_p + Y_p), \forall J_p \in \mathcal{M}, J_p \notin \mathcal{J}_s \text{ and } d_p > d_i. \tag{8}$$

The fundamental difference between our technique and the one in [16] is the way that effective deadlines are defined. From equation (8), the effective deadline for a mandatory job is relaxed from the earliest arrival time of the next lower priority job further with its delay factor. This in turn will allow mandatory jobs to delay further to merge the idle interval. As such, the effective deadline for J_{21} becomes 34 instead of 32, and thus we have $\tilde{T}_{LS} = 27$, which is the case shown in Figure 2(d). Note that, since Y_i is available off-line, our technique based on Theorem 2 has the same on-line complexity as that in [16]. Also, it is not difficult that the LST computed based on Theorem 2 is never worse than that by the technique in [16]. Finally, it is worthy to mention that both Theorem 1 and Theorem 2 are sufficient conditions. Therefore, the larger one from equation (4) and (7) can be used as LST and guarantee the deadlines for all mandatory jobs.

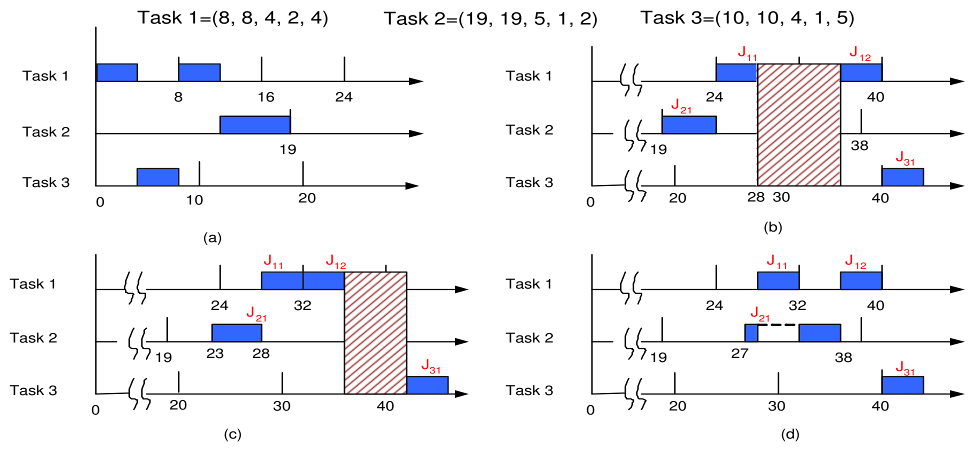


Figure 2. (a) Three tasks scheduled based on their R-Pattern; (b) J_{21} cannot be delayed according to Theorem 1; (c) Delaying the mandatory jobs to $t = 23$ ([16]) cannot remove the idle interval; (d) Delaying the mandatory jobs to $t = 27$ and eliminating the idle interval.

6 Execution of the optional jobs

When the system are idle, Lemma 1 helps us to *assign* a mandatory job as late as possible, and Theorem 1 and Theorem 2 can further delay the idle intervals by delaying the execution of mandatory jobs. When the predicted idle interval is long enough (i.e. greater than T_{min}), it will be beneficial to shut down the processor and devices. One missing piece in our approach is, however, what if the idle interval is still not long enough?

We have two choices when the idle interval is not long enough to accommodate the timing and energy overhead: (1) we can simply keep the system idle (but active); (2) we can opt to run some optional jobs. For the first case, the processor consumes a little less power (as $P_{pidle} < P_{pact}$) while the device consumes nearly the same power. At the first sight, running optional jobs does not seem to be energy efficient since $P_{pact} > P_{pidle}$. However, executing optional jobs may potentially lead to positive energy saving gain because (1) some mandatory jobs become optional and do not have to be executed; and (2) more importantly, some short idle intervals in the future can be merged to longer ones and enable system to shut down if appropriate mandatory jobs are demoted to optional. The problem is how to select the *right* optional jobs.

To make a precise analysis of the trade off in executing the optional jobs is a challenging problem, especially from the dynamic scheduling perspective. In considering this, we resort to a heuristic approach in solving this problem. In our heuristic approach, an optional job is executed, non-preemptively, only when it can finish within the idle intervals as predicted. This helps to avoid the execution of too many optional jobs, which would not be energy efficient. When there are more than one candidate optional jobs, we devise a function to evaluate the fitness of an optional job. The fitness function, i.e. $\mathcal{F}$, is determined by two parameters, i.e., the flexibility (F) and criticality (Cr). An optional tends to have higher energy-saving potential if its corresponding mandatory jobs are more flexible to be moved around and/or it is closer to dynamic failure. Therefore, for optional job J_{ij}, we define

$$F(J_{ij}) = (Y_i + D_i) \times \frac{k_i T_i}{m_i C_i}, \tag{9}$$

and

$$Cr(J_{ij}) = \frac{m_i'}{k_i - m_i}, \tag{10}$$

where m_i' is the currently allowed deadline misses of τ_i without causing dynamic failure. Note that $F(J_{ij})$ can be computed off-line but $C(J_{ij})$ is computed on-line.

The rationale behind equation (9) is that, from Theorem 1 and Theorem 2, large Y_i and D_i tend to make future mandatory jobs from τ_i more flexible to be delayed. On the other hand, $\frac{m_i C_i}{k_i T_i}$ indicates the average mandatory workload for task τ_i. The higher the value is, the more difficult it is to shift the workload and thus merge idle intervals. Equation (10) measures the number (normalized) of deadline misses that can still be tolerated. The higher the value, the less urgent that J_{ij} needs to be executed in order not to cause a dynamic failure. Note that if J_{ij} is optional, $C(J_{ij})$ cannot be zero. Therefore, based on equation (9) and (10, we define $\mathcal{F}$ as

$$\mathcal{F}(J_{ij}) = \frac{F^*(\tau_i)}{Cr(J_{ij})}, \tag{11}$$

where $F^*(\tau_i)$ is the normalized value of $F(J_{ij})$ (based on the largest value) for consistency.

7 Experiments

In this section, we evaluate the performance of our approach using simulations. We implemented five approaches in our experiments. In the first approach, the mandatory jobs were statically determined using the R-patterns. We refer this approach as DPD_R and use its results as the reference results. The second approach adopts E-patterns instead of R-patterns and hereby is referred as DPD_E. In the third approach, referred as DPD_{ND}, we marketed mandatory jobs as late as possible, but the execution of the mandatory jobs were not delayed. The fourth approach, referred as DPD_{NTA}, determines the mandatory job dynamically and delays the mandatory job executions, with delay amount computed based on the approach in [16]. The final approach, denoted by DPD_{DYN}, is the complete implementation of our approach presented in this paper.

The periodic task set in our experiments consisted of five tasks. Each task set were randomly generated with the periods randomly chosen in the range of $[10, 50]ms$. We assumed that the deadlines for the tasks were the same as their periods. The worst case execution time (WCET) of a task was set to be uniformly distributed from $1ms$ to its deadline, and the actual execution time for a job was evenly distributed from [0.4WCET, WCET]. The m_i and k_i for the (m,k)-constraints were randomly generated such that k_i is uniformly distributed between 2 to 10, and $1 \leq m_i < k_i$. The total (m,k)-utilization, i.e., $\sum_i \frac{m_i C_i}{k_i T_i}$, is divided into intervals of length 0.1, each of which contains at least 20 schedulable task sets, or at least 5000 task sets within each interval have been generated. For the processor considered in our experiments, we assume that $P_{pact} = 1.0W$, $P_{pidle} = \frac{1}{3} P_{pact}$. We assume that the power consumption for the processor and device during the sleep mode are negligible. We also assume the minimal idle interval length to be $3ms$. Since the active power of the peripheral devices can be comparable to or even much larger than that of the core processor, we collect the data of power consumption for these two cases separately: (i) $P_{dact} = P_{pact}$ and (ii) $P_{dact} = 10P_{pact}$.

We first study the number of idle intervals by the five different scheduling strategies. A large number of idle intervals is undesirable in DPD since it either incurs higher transition overhead due to more frequent transitions or has to keep system busy due to shorter idle interval length. Figure 3(a) compares the normalized (with respect to DPD_R) number of idle intervals within LCM($k_i T_i$) by different approaches.

Figure 3 clearly shows that our proposed technique (i.e. DPD_{DYN}) can effectively reduce the number of idle intervals. It is interesting to see that the numbers of idle intervals by DPD_E and DPD_R are quite close to each other, which shows that both static approaches are not effective in merging the idle intervals. When compared with our approach, the number of idle intervals by DPD_R and DPD_E can be as nearly 3.5 times higher than our approach. In addition, we can observe from Figure 3 that, if we only dynamically change the mandatory job assignment without delaying the execution of the mandatory jobs (i.e. DPD_{ND}), it may help

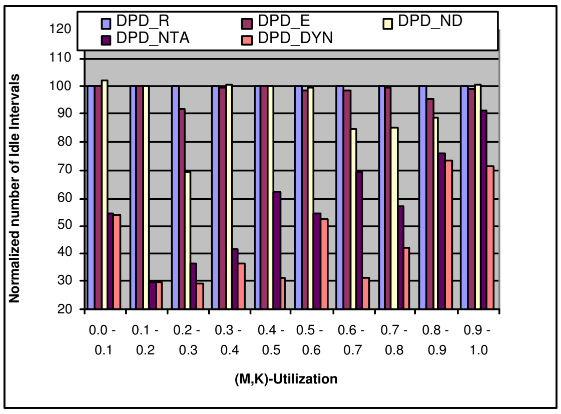
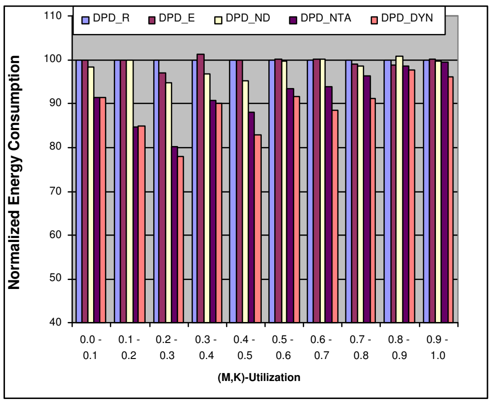


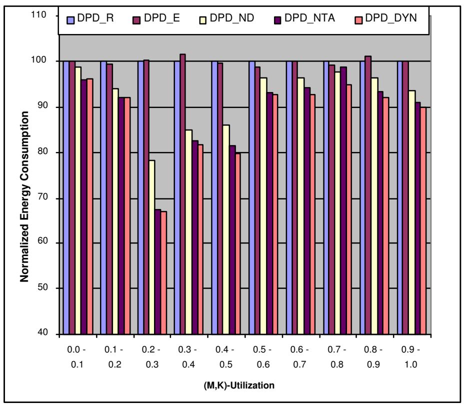
Figure 3. The average number of idle intervals by different approaches.

to merge the idle intervals in some cases but not always. And the number of idle intervals can be much larger than those by delaying the processor execution. Furthermore, compared with DPD_{NTA}, i.e., the approach that adopts a different way to delay the mandatory jobs [16], our approach can cut its idle interval number as many as a half. These results favorably demonstrate the strength of the two sufficient conditions presented in section 5.

The reduction of idle intervals has a strong correlation with the reduction of energy as shown in Figure 4. From Figure 4 (a) and (b), it is not surprising to see that the energy savings obtained with DPD_{DYN} varies according to the (m,k)-utilization. When (m,k)-utilization is very high (e.g. within [0.8,1.0]), the system is busy most of the time and cannot be shut down. Under this scenario, all the approaches have the similar energy savings. When the (m,k)-utilization is small, we can see that DPD_{DYN} can save energy more efficiently. As shown in Figure 4(a), when the active power of the peripheral devices (P_{dact}) is comparable to that of the core processor (P_{pact}), DPD_{DYN} can reduce the energy consumption by up to 18% when compared with DPD_{ND} , and by up to 6% when compared with DPD_{NTA}, *without* increasing the on-line complexity. The energy conservation is more significant when compared with the conventional and naive approaches (DPD_E and DPD_R), i.e., up to over 23%. When the active power of the peripheral devices (P_{dact}) is much larger than that of the core processor (P_{pact}), more energy saving can be observed. As shown in Figure 4(b), DPD_{DYN} can reduce the energy consumption by up to 30% when compared with DPD_{ND}, and by up to 5% when compared with DPD_{NTA}. In summary, the experiment results have shown that our approach can significantly reduce the idle intervals, and hence achieve better energy savings than the conventional approaches.



(a)



(b)

Figure 4. The average total energy consumption by different approaches when (a) $P_{dact} = P_{pact}$; (b) $P_{dact} = 10P_{pact}$.

8 Conclusions

Energy consumption is critical in the design of pervasive real-time computing platforms. The power consumption for peripheral devices, as a significant part of the overall power consumption, must be taken into consideration to reduce the system-wide power consumption. On the other hand, most of these real-time systems are not hard real-time but exhibit complex QoS behaviors that can only be modeled by more complicated constraints. In this paper, we present a dynamic DPD approach to reduce the system-wide energy consumption while guaranteeing the QoS requirements, which are modeled as the (m,k)-constraints. Our approach ensures the (m,k)-firm guarantee by taking advantage of static analysis. The energy saving performance of our approach comes from the fact that we dynamically change the mandatory/optional job settings, and effectively merge the idle intervals by delaying the execution for mandatory jobs. Our experimental results demonstrate that our approach can greatly reduce the number of idle intervals and thus the power consumption, while still providing the (m,k)-firm guarantee.

References

[1] T. A. AlEnawy and H. Aydin. Energy-constrained scheduling for weakly-hard real-time systems. *RTSS*, 2005.

[2] H. Aydin, R. Melhem, D. Mosse, and P. Alvarez. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *ECRTS01*, June 2001.

[3] L. Benini, A. Bogliolo, and G. Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Trans. on VLSI*, 8(3):299–316, June 2000.

[4] G. Bernat and R. Cayssials. Guarantted on-line weakly-hard real-time systems. In *RTSS*, 2001.

[5] H. Cheng and S. Goddard. Online energy-aware i/o device scheduling for hard real-time systems. *DATE*, 2006.

[6] L. Doherty, B. Warneke, B. Boser, and K. Pister. Energy and performance considerations for smart dust. *International Journal of Parallel Distributed Systems and Networks*, 4(3):121–133, 2001.

[7] ITRS. *International Technology Roadmap for Semiconductors*. International SEMATECH, Austin, TX., http://public.itrs.net/.

[8] R. Jejurikar and R. Gupta. Dynamic voltage scaling for system-wide energy minimization in real-time embedded systems. *ISLPED*, 2004.

[9] R. Jejurikar, C. Pereira, and R. Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. *DAC*, pages 275 – 280, 2004.

[10] M. Kim and S. Ha. Hybrid run-time power management technique for real-time embedded system with voltage scalable processor. *OM'01*, pages 11–19, 2001.

[11] W. Kim, J. Kim, and S. Min. Preemption aware dynamic voltage scaling in hard real time systems. *ISLPED*, 2004.

[12] W. Kim, J. Kim, and S.L.Min. A dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack analysis. *DATE*, 2002.

[13] G. Koren and D. Shasha. Skip-over: Algorithms and complexity for overloaded systems that allow skips. In *RTSS*, 1995.

[14] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 17(2):46–61, 1973.

[15] B. Mochocki, X. Hu, and G. Quan. A realistic variable voltage scheduling model for real-time applications. *ICCAD*, 2002.

[16] L. Niu and G. Quan. Reducing both dynamic and leakage energy consumption for hard real-time systems. *CASES'04*, Sep 2004.

[17] L. Niu and G. Quan. Energy-aware scheduling for realtime systems with (m,k)-guarantee. *Technical Report TR-2005-05, Department of Computer Science and Engineering, University of South Carolina*, 2005.

[18] L. Niu and G. Quan. A hybrid static/dynamic dvs scheduling for real-time systems with (m, k)-guarantee. *RTSS*, 2005.

[19] L. Niu and G. Quan. Energy minimization for real-time systems with (m,k)-guarantee. *IEEE Trans. on VLSI, Special Section on Hardware/Software Codesign and System Synthesis*, pages 717–729, July 2006.

[20] G. Quan and X. Hu. Enhanced fixed-priority scheduling with (m,k)-firm guarantee. In *RTSS*, pages 79–88, 2000.

[21] K. Ramamritham and J. A. Stankovic. Scheduling algorithms and operating system support for real-time systems. *Proceedings of the IEEE*, 82(1):55–67, January 1994.

[22] P. Ramanathan. Overload management in real-time control applications using (m,k)-firm guarantee. *IEEE Trans. on Paral. and Dist. Sys.*, 10(6):549–559, Jun 1999.

[23] P. Rong and M. Pedram. Hierarchical power management with application to scheduling. *ISLPED*, 2005.

[24] V. Swaminathan and K. Chakrabarty. Pruning-based, energy-optimal, deterministic i/o device scheduling for hard real-time systems. *Trans. on Embedded Computing Sys.*, 4(1):141–167, 2005.

[25] M. A. Viredaz and D. A. Wallach. Power evaluation of a handheld computer. *IEEE Micro*, 23(1):66–74, 2003.

[26] R. West and K. Schwan. Dynamic window-constrained scheduling for multimedia applications. In *ICMCS*, 1999.

[27] J. Zedlewski, S. Sobti, N. Garg, F. Zheng, A. Krishnamurthy, and R. Wang. Modeling hard-disk power consumption. *FAST '03*, pages 217–230, 2003.

[28] B. Zhai, D. Blaauw, D. Sylvester, and K. Flautner. Theoretical and practical limits of dynamic voltage scaling. *DAC*, pages 868–873, 2004.

[29] J. Zhuo and C. Chakrabarti. Systemlevel energyefficient dynamic task scheduling. *DAC*, 2005.