2010 10th IEEE International Conference on Computer and Information Technology (CIT 2010)

Leakage-Aware Scheduling for Real-Time Systems with (m,k)-Constraints

Linwei Niu
Math and Computer Science Department
Claflin University
linwei.niu@claflin.edu

Gang Quan
Electrical and Computer Engineering Department
Florida International University
gang.quan@fiu.edu

***Abstract*—In this paper, we study the problem of reducing both the dynamic and leakage energy consumption for real-time systems with (m,k)-constraints, which require that at least m out of any k consecutive jobs of a task meet their deadlines. Two energy efficient scheduling approaches incorporating both dynamic voltage scheduling (DVS) and dynamic power down (DPD) are proposed in this paper. The first one statically determines the mandatory jobs that need to meet their deadlines in order to satisfy the (m,k)-constraints, and the second one does so dynamically. The simulation results demonstrate that, with more accurate workload estimation, our proposed techniques outperformed previous research in both overall and idle energy reduction while providing the (m,k)-guarantee.**

I. Introduction

As transistor density continues to grow, the power/energy conservation problem becomes more and more critical in the design of pervasive real-time embedded systems. For CMOS circuits, the power consumption comes from the dynamic power consumption (mainly due to switching activities) and the static power consumption (mainly due to leakage current). As VLSI technology continues its evolution toward the sub-micron and nanoscale era, the rapidly elevating leakage power dissipation is becoming too prominent to be ignored [1].

The *dynamic voltage scaling* (DVS) strategy (e.g., [2], [3], [4]), *i.e.*, by dynamically changing the supply voltage as well as the working frequency, has long be recognized as an effective method to reduce the dynamic energy. However, as the leakage power continues to increase, the energy saving achievable via DVS alone is becoming severely limited. This is because DVS prolongs the active period of the processor and thus can increase the total leakage energy consumption [5]. The *dynamic power down* (DPD) strategy, on the other hand, dynamically turns the system on and off and is thus an effective way at the system level to control the leakage energy consumption. Therefore, to obtain the maximal reduction in the overall energy consumption, some researchers [5], [6], [7], [8] have proposed to combine DVS and DPD to reduce both the dynamic power and leakage power simultaneously. Most of the approaches have targeted the hard real-time systems, *i.e.*, the systems requiring that all the task instances meet their deadlines.

While the hard real-time model is the most basic model for real-time systems, many practical real-time applications exhibit more complicated characteristics that can only be captured with more complex requirements, generally called the *Quality of Service (QoS) requirements*. For example, Hamdaoui *et al.* [9] proposed a model, called the (m,k)-model [9], to provide certain QoS guarantee for real-time applications. According to this model, a repetitive task of the system is associated with an $(m,k)(0<m\leq k)$ constraint requiring that m out of any k consecutive job instances of the task meet their deadlines. A *dynamic failure* occurs, which implies that the temporal QoS constraint is violated and the scheduler is thus considered failed, if within any k consecutive jobs more than $(k-m)$ job instances miss their deadlines. The previous DVS scheduling techniques based on traditional hard real-time systems become inefficient or inadequate when QoS requirements such as the $(m,k)-$constraints are imposed on real-time systems.

Some previous works have been reported to reduce the energy for real time systems with (m,k)-constraints. In [10], Niu *et al.* introduced a hybrid approach which can reduce the energy consumption for real time systems with $(m,k)-$guarantee. With QoS requirements formulated as a tolerable statistical deadline miss rate, Hua *et al.* [11] introduced several techniques to reduce energy by exploiting processor slack time due to the missed deadlines. In [12], Alenawy *et. al.* proposed an approach to minimize the number of dynamic failures for (m,k)-firm systems with fixed energy budget constraint. In [13], a dynamic approach is proposed to minimize the energy consumption for *dual-voltage-mode* weakly hard real-time systems. In [14], Niu *et al.* proposed an approach to reduce energy for weakly hard real-time systems with peripheral devices. All of the techniques targeted dynamic power reduction and none of them considered reducing the dynamic and leakage energy consumption simultaneously.

In this paper, we study the problem of reducing the overall energy consumption for real-time systems with (m,k)-guarantee. A key aspect of this problem is to judiciously partition the jobs into *mandatory* jobs and *optional* jobs such that as far as all the mandatory jobs can meet their deadlines, the (m,k)-constraints can be ensured. We proposed two approaches to address this problem. In the first approach, the mandatory/optional partition is conducted statically. The mandatory jobs are carefully procrastinated with purpose of merging the idle intervals so that the processor can be shut down effectively. In the second approach, we dynamically adjust the mandatory/optional job partitioning to accommodate the dynamic nature of real-time embedded systems. A

978-0-7695-4108-2/10 $26.00 © 2010 IEEE
DOI 10.1109/CIT.2010.313

look-ahead strategy is proposed that can more accurately estimate the workload and therefore make better decision in the mandatory/optional partitioning. Moreover, our approaches will execute the jobs with speed below the critical speed when necessary and we will show that such strategy can be more energy efficient than the previous ones using the critical speed as the threshold for scaling. With a practical processor model and technology parameters [15], our experiment results show that our approach outperformed the existing research in energy saving performance while providing the (m,k)-guarantee.

The rest of the paper is organized as follows. Section II introduces the system models and some preliminaries. Section III introduces our static approach. Section IV introduces our dynamic approach. The effectiveness and energy efficiency of our approaches are evaluated in section V. In section VI, we offer the conclusions and future work.

II. Preliminary

In this section, we first give the system model and power model. Then we introduce some concepts and observations important to our research in this paper.

A. System model

The real-time system considered in this paper contains n independent periodic tasks, $\mathcal{T} = \{\tau_0, \tau_1, \cdots, \tau_{n-1}\}$, scheduled according to the earliest deadline first (EDF) policy [16]. Each task contains an infinite sequence of periodically arriving instances called *jobs*. 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. The QoS requirement for τ_i is represented by a pair of integers, i.e., (m_i, k_i) $(0 < m_i \leq k_i)$, which require that, among any consecutive k_i jobs of τ_i, at least m_i jobs must meet their deadlines. The j^{th} job of task τ_i is represented with J_{ij} and its arrival time, actual execution time and absolute deadline are represented by r_{ij}, c_{ij} and d_{ij}.

B. The power model and critical speed

The power consumption on a DVS processor can be divided into two parts: the speed-dependent part $P_{dep}(s)$ and the speed-independent part P_{ind}. Considering a job with workload w and total power function as $P_{act(s)} = P_{dep}(s) + P_{ind}$, the total energy $(E_{act}(s))$ consumed to finish this job with speed s can be represented as

$$E_{act}(s) = P_{act}(s) \times \frac{w}{s}. \quad (1)$$

Hence, to minimize the energy consumption, we have

$$P_{act}(s) = P'_{act}(s)s. \quad (2)$$

By solving equation (2), we can get the optimal speed to minimize the total energy when executing a job. We call this speed the *critical speed*, and denote it as s_{crit}. We can see from equation (2) that using a processor speed higher or lower than the critical speed will consume more energy than the one using the critical speed to complete the same workload.

We assume that the processor can be in one of the three states: *active*, *idle* and *sleeping* states. When the processor is idle, the major portion of the power consumption comes from the leakage which increases rapidly with the dramatic increasing of the leakage power consumption. Shutting-down strategy, *i.e.*, put the processor into its sleeping state, can greatly reduce the leakage energy. However, it has to pay extra energy and timing overhead to shut down and later wake up the processor. Assume that the power consumptions of a processor in its idle state and sleeping state are P_{idle} and P_{sleep}, respectively, and the energy overhead and the timing overhead of shutdown/wakeup is E_o and t_o. Then the processor can be shut down with positive energy gains only when the length of the idle interval is larger than $T_{th} = \max(\frac{E_o}{P_{idle} - P_{sleep}}, t_o)$. We call T_{th} as the *shut down threshold interval*.

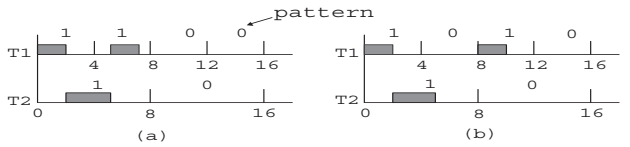

Fig. 1. (a) The schedule of a task set ($\tau_1 = (4,4,2,2,4)$; $\tau_2 = (8,8,3,1,2)$) partitioned with R-Pattern; (b) The schedule of the same task set partitioned with E-Pattern.

C. Meeting the (m,k)-constraints

A key problem for meeting the (m,k)-constraints is to judiciously partition the jobs into *mandatory* jobs and *optional* jobs [17]. The partition can be done statically or dynamically. Two well-known partition strategies proposed are the *deeply-red pattern* (or R-pattern) and *evenly distributed pattern* (or E-pattern) [10]. One example of mandatory/optional job partitioning with R-pattern and E-pattern for a given task set is shown in Figure 1. As shown, the R-pattern assigns the first m jobs as mandatory. It has been proved that as long as all mandatory jobs selected from R-pattern can meet their deadlines, the mandatory jobs selected from any other $(m,k)-$pattern can also meet their deadlines [10]. The mandatory/optional partitioning according to E-pattern has the property that it helps to spread out the mandatory jobs evenly in each task along the time. Interested readers can refer to [10] for more technical details about the R-pattern and E-pattern. Based on the mandatory/optional job partition, in next sections, we will introduce our static and dynamic approaches to reduce the energy consumption while ensuring the $(m,k)-$guarantee.

III. The static approach

One intuitive static approach is to partition the mandatory/optional jobs based on the E-pattern and then scale down the processor speed based on the mandatory job set. Since E-pattern in general helps to spread the workload evenly, it helps to better reduce the processor speed. Note that, using speed higher or lower than the critical speed will incur higher energy consumption when completing the same workload, we need to try to set the processor speed to the critical speed to achieve better energy saving performance.

The problem for this approach is that, by spreading the workload and also potentially using *higher than necessary* speed assignment for mandatory jobs can possibly lead to large number of idle intervals. Some of them will be too short for the processor to shut down. Even if some idle intervals are longer than the shut down threshold T_{th}, too many idle intervals will also increase the energy consumption significantly due to the overheads of frequently shutting down and waking up the processor.

One effective way to address this problem is to delay some of the mandatory jobs to extend or merge the idle intervals, such as the ones in [5], [18]. However, delaying the execution of the mandatory jobs might potentially cause some other mandatory jobs with lower priorities to miss their deadlines. On the other hand, delaying the mandatory jobs not sufficiently might generate new scatter idle intervals that can not be shut down. The problem then becomes how to determine the maximal delay to merge the idle intervals without causing any mandatory jobs to miss their deadlines. In the following, we introduce two sufficient conditions to help identify the delays for mandatory jobs. Before doing that, we first introduce the following theorem.

Theorem 1: Given task set $\mathcal{T}=\{\tau_0,\tau_1,...,\tau_{n-1}\}$ with tasks ordered by increasing value of D_i, let $\mathcal{E}$ be the mandatory jobs selected from $\mathcal{T}$ based on their E-patterns, and let L be the ending point of the first busy period when executing the mandatory jobs. Assuming the largest blocking factor for task τ_i be B_i, $\mathcal{E}$ is schedulable *if*

$$\sum_{D_i\le t}(\lceil\frac{m_i}{k_i}\lfloor\frac{t+T_i-D_i}{T_i}\rfloor\rceil)C_i+B_{l(t)}\le t \tag{3}$$

for all $t\le L$, $t=\lfloor p\frac{k_i}{m_i}\rfloor T_i+D_i, p\in Z, i=0,...,n-1$ and $l(t)=\max\{l|D_l\le t\}$.

(The proofs for theorems in this paper are provided in [19]).

Given a mandatory job set determined with E-pattern, with Theorem 1, it is possible to compute the maximal tolerable blocking factor B_i for each task τ_i by checking the accumulated work demand that must be finished before the absolute deadline d_i for each mandatory job J_i that arrives before L. Similar theorem can be established based on the R-patten as well and the blocking factor under R-pattern can be found in the same way. With B_i available, we can formulate the first sufficient condition as follows.

Theorem 2: Let $\mathcal{M}$ be the mandatory job set based on E-pattern or R-pattern, and let B_i be the corresponding largest blocking factor for each task τ_i. Let the current time $t=t_0$, and let the coming mandatory job set (i.e. with arrive time r_i later than t_0) be $\mathcal{J}'$. If the execution of $\mathcal{J}'$ is started at

$$T_{LS}(\mathcal{J}')=\min_{J_i\in\mathcal{J}'}(r_i+B_i), i=0,1,...,n-1, \tag{4}$$

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

Theorem 2 provides one method to calculating the latest time that the upcoming mandatory jobs can be delayed without incurring any deadline misses for any mandatory jobs. Note that, given that B_i is available, we can also develop another method (similar to the one in [6]) to identify the maximal delay for a mandatory job set, as stated in Theorem 3.

Theorem 3: Let $\mathcal{M}$ be the mandatory job set based on the R-pattern or E-pattern, and let B_i be the corresponding largest blocking factor for each task τ_i. Let the processor speed for each task τ_k be s_k. Assuming the current time is $t=t_0$, and let the coming mandatory job set (i.e. with arrive time r_i later than t_0) be $\mathcal{J}'$. Let the earliest deadline for the mandatory jobs in $\mathcal{J}'$ be T_B. Then no mandatory job in $\mathcal{J}'$ will miss its deadline if the execution of all mandatory jobs in $\mathcal{J}'$ is delayed to $T_{LS}(\mathcal{J}')$, where

$$T_{LS}(\mathcal{J}')=\min_{J_i\in\mathcal{J}_s}(d_i^*-\sum_{J_k\in hp(J_i)}\frac{c_k}{s_k}), \tag{5}$$

where $\mathcal{J}_s$ consists of mandatory jobs from $\mathcal{J}'$ with arrival times earlier than T_B but later than t_0, $hp(J_i)$ are the jobs with equal or higher priorities than J_i and

$$d_i^*=\min_p(d_i,r_p+B_p), \forall J_p\in\mathcal{J}', J_p\notin\mathcal{J}_s \text{ and } d_p>d_i. \tag{6}$$

The fundamental difference between our technique and the one in [6] is the way that the effective deadline d_i^* is defined. From equation (6), the effective deadline for a mandatory job is prolonged with the blocking factor B_p of the next mandatory jobs. This in turn will allow the mandatory jobs to be delayed further. While it might not be always energy efficient to delay the mandatory jobs to the maximal extent, having larger delay interval will provide us more flexibility in determining the most energy efficient speeds for jobs online. Note that since both Theorem 2 and Theorem 3 are sufficient conditions, the larger one from equation (4) and (5) can be used as the latest starting time t_d for the upcoming mandatory jobs with deadlines guarantee for them. Finally, our sufficient conditions in Theorem 2 and Theorem 3 can be used to compute the maximal delay for the upcoming mandatory job set not only when the processor is idle, but also when the processor is busy at time t_0, which will be very useful for our dynamic approach in Section IV.

With Theorem 2 and Theorem 3, our static approach can be implemented easily. Specifically, at current time t_{cur}, if the processor begins to idle, we compute the latest starting time t_d for the coming mandatory jobs and shut down the processor if $(t_d-t_{cur})>T_{th}$. Note that the difference between our static approach and the approach in [5] is that the approach in [5] can only procrastinate each job of the task individually upon its arrival time, while our approach is based on computing the latest starting time for the whole coming mandatory job set (interested readers can refer to [19] for more details). Generally our approach can predict the idle interval length and shut down the processor more precisely and thus save the idle energy more efficiently. Moreover, when mandatory jobs finish earlier than worst case, dynamic reclaiming techniques [19] can be exploited to reduce the energy further.

IV. The dynamic approach

The advantages introduced above are that the mandatory job set is the minimal, and the mandatory jobs are evenly distributed with respected to each task. However, even though the

mandatory jobs for each task are evenly distributed, the overall mandatory workload are not necessarily evenly distributed. Moreover, since E-pattern tends to separate mandatory jobs away from each other, it may generate a large number of idle intervals. It is thus desirable that the mandatory jobs be determined dynamically to improve the energy-saving efficiency.

A. The general algorithm

The algorithm of our dynamic approach is shown in Algorithm 1. In general, our dynamic approach consists of two phases: an off-line phase followed by an online phase. To ensure the (m,k)-constraints, the feasibility of the task set under R-pattern is tested according to Theorem 1 in [13]. For schedulable task sets, the speed for a mandatory task is scaled down but no lower than s_{crit}. The blocking factor B_i for each task under the R-pattern is computed using the method similar to that in Theorem 1.

During the on-line phase, two job ready queues are maintained, *i.e.*, the mandatory job queue (MQ) and the optional job queue (OQ), with jobs in MQ always having higher priority than those in OQ. Upon arrival, a job, *i.e.*, $J_{ip} \in \tau_i$ is determined as mandatory job or optional job based on the execution results of the $k_i - 1$ jobs in the most recent history. It is determined as mandatory only if one more deadline miss will incur dynamic failure. The mandatory jobs in MQ will generally be executed with their speeds predetermined during the off-line phases. However, whenever there is zero or only one mandatory job in MQ, opportunities exist to update the execution speed for the upcoming mandatory jobs and save energy consumption more aggressively. More details of this method can be found in Section IV-B.

In our dynamic approach, not only the mandatory jobs but also the optional jobs can be executed. The execution of optional job has great potential in help reducing the overall energy consumption. Some optional job with actual execution time much shorter than its worst case can also meet its deadline with speed lower than its predetermined speed. And that will help to reduce the possibility of having to run mandatory jobs at higher processor speeds in the future. However, executing the optional job might incur extra energy cost, which needs to be addressed carefully. Otherwise, the energy reduction achieved from executing the optional job might not be able to compensate the energy cost. Therefore, it is not only important to choose an appropriate optional job to execute, but also to determine the appropriate speed to execute the job. We introduce our method in choosing the optional job and its speed in section IV-C.

If no optional job is qualified to execute and the predicted idle interval is longer than the shut down threshold T_{th}, we shut down the processor and set the timer to be the idle interval length.

B. Update the predetermined speeds for mandatory jobs

When there exists only one mandatory job, i.e. J_i, in MQ, from Theorem 2 and Theorem 3, we can see that the speed of J_i can be scaled safely so long as J_i finishes no later than t_d.

Algorithm 1 The online phase. (Algorithm *LKDN*)

```
1:  Upon job completion:
2:  if MQ is empty then
3:     t_cur = the current time;
4:     t_d = the latest starting time for the upcoming mandatory
       jobs according to Theorem 2 and Theorem 3;
5:     if OQ is not empty then
6:        Select and run J_i ∈ OQ with energy efficient speed s'_i
          determined in Section IV-B non-preemptively;
7:     else if (t_d − t_cur) > T_th then
8:        Shut down the processor and set up the wake up timer
          to be (t_d − t_cur);
9:     end if
10: end if
11:
12: Upon job arrival or expiration of timer:
13: if MQ is not empty then
14:    if J_i is the only job in MQ then
15:       Run J_i with energy efficient speed s'_i determined in
          Section IV-B non-preemptively;
16:    else
17:       Run jobs in MQ with their current scaled speeds
          according to preemptive EDF;
18:    end if
19: end if
```

One intuitive strategy is to scale the speed of J_i as low as s_{crit}. If there is still available time, then the upcoming mandatory jobs are procrastinated and the processor is shut down if the length of idle time before t_d is larger than T_{th}. However, this strategy is not necessarily always the best strategy in saving energy.

Consider a task set consisting of two tasks ($\tau_1 = (20,20,5,2,4)$; $\tau_2 = (40,40,15,1,2)$). Assume the task set will be executed on the Intel XScale processor model [15]. According to [7], the power consumption function for Intel XScale [15] can be modeled approximately as $P_{act}(s) = 0.08 + 1.52s^3$ Watt by treating 1GHz as the reference speed 1. And the normalized critical speed in such a model is about 0.3 (at 297 MHz) with power consumption 0.12W. We assume the shut down overhead to be $E_o = 800\mu J$ as did in [7]. If the minimal processor speed is 0, the idle power consumption of the processor is 0.08 Watt and the corresponding shut down threshold will be $T_{th} = 10ms$.

The scaled speed for the task set under E-pattern is 0.5 and the schedule of the static approach is shown in Figure 2(a). The energy consumption under the static approach is $(1.52 \times 0.5^3 + 0.08) \times 40 + (1.52 \times 0.3^3 + 0.08) \times 16.67 + 0.8 = 13.61$. The scaled speed for the task set under R-pattern is 0.625. As shown in Figure 2(b), according to our optional job selection strategy in Section IV-C (for brevity here we set the job selection control coefficient κ to be 1), our dynamic approach will choose to schedule the optional job for τ_2 first in the interval [0,40] and scale its speed to be 0.375 (after completion, its dynamic pattern is updated from 0 to be 1). Then at time

$t = 40$, the dynamic approach will schedule the job J_{13} for τ_1. Since J_{13} is the only mandatory job in the ready queue and the latest starting time t_d for the future mandatory job(s) is 72, there is enough space to scale the speed for J_{13} to s_{crit} and shut down the processor upon completion of J_{13} at $t = 56.7$. Then at time $t = 72$, the processor will be waken up and continue to execute J_{14} at its predetermined speed 0.625. The energy consumption within the hyper period for the schedule in Figure 2(b) will be $(1.52 \times 0.375^3 + 0.08) \times 40 + (1.52 \times 0.3^3 + 0.08) \times 16.67 + 0.8 + (1.52 \times 0.645^3 + 0.08) \times 8 = 12.81$.

However, a different schedule in Figure 2(c) which, instead of shutting down, uses the available space to help scale the speed of J_{14} to $s_{crit} = 0.3$ and keeps the processor idle for the rest of the time, has energy consumption of $(1.52 \times 0.375^3 + 0.08) \times 40 + 0.12 \times 33.33 + 0.08 \times 6.67 = 10.92$, which is 15.6% lower than that in Figure 2(b). Moreover, another schedule in Figure 2(d) uses the idle time to further reduce the speeds for job J_{13} and J_{14} to 0.25, which is below the critical speed, can achieve an additional 3.3% energy reduction compared with the schedule in Figure 2(c) (the energy consumption in Figure 2(d) is $(1.52 \times 0.375^3 + 0.08) \times 40 + (1.52 \times 0.25^3 + 0.08) \times 40 = 10.56$).

In this example, we can see that the approach in Figure 2(b) considers only reducing the speed for the current job J_i. When looking ahead, it might be more urgent for the the future mandatory jobs to compete for the available idle time to scale their speeds instead of shutting down the processor. With this in mind, we proposed a look-ahead approach which incorporates the speed information of the future mandatory jobs in scaling the speed of the current job J_i and shutting down the processor when necessary.

Specifically, when the current job J_i is ready, we temporarily scale its speed as low as s_{crit} first, *i.e*, set its current speed $s_i' = max\{\frac{c_i s_i}{(\min\{t_d, d_i\} - t_{cur})}, s_{crit}\}$. Then by considering the speed information for the future mandatory jobs (for simplicity, in the following approach, we only look into the earliest arriving future mandatory job which is called $look-ahead$ job and denoted as J_{la}. And it can be easily extended to incorporate all incoming mandatory jobs arriving before t_d), this temporary speed s_i' might need to be updated depending on the expected finishing time $f_i (= t_{cur} + \frac{c_i s_i}{s_i'})$ of J_i under speed s_i'. Specifically, we need to consider three cases:

- 1) $(t_d - f_i) < T_{th}$. In this case, it is not energy beneficial to shut down the processor at all. Instead, we should use the idle time to scale the speed of J_i and J_{la} as low as possible.
- 2) $(t_d - f_i) \geq T_{th}$ and $s_{la} = s_{crit}$. In this case, J_{la}'s speed is not higher than s_{crit} and the new idle interval generated between $[f_i, t_d]$ can be shut down if we procrastinate the upcoming mandatory jobs to t_d. So keeping the current value of s_i' and shutting down the processor while delaying the future mandatory jobs to t_d is a good choice.
- 3) $(t_d - f_i) \geq T_{th}$ and $s_{la} > s_{crit}$. In this case, the idle interval generated between $[f_i, t_d]$ can be shut down if we procrastinate the upcoming mandatory jobs to t_d. However, we need to compare the beneficial between shutting down the processor and using the available time space to scale the speeds of J_i and J_{la}. In order to do so, we need to check whether both the speeds of J_i and J_{la} can be scaled to s_{crit} while the remaining idle time space within $[t_{cur}, t_d]$ is still long enough to shut down the processor. (This can be done by assuming the worst case execution time of J_{la} to be C_{la} and setting $s_{la}' = max\{\frac{C_{la} s_{la}}{(t_d - \max\{r_{la}, f_i\}) + C_{lq})}, s_{crit}\}$). If the remaining time space $t_{rem} = (t_d - f_i - (C_{la}' - C_{la}))$ is longer than T_{th}, then we shut down the processor at f_i and set the wake up timer to be t_{rem} (here C_{la}' is the worst case execution time of J_{la} under s_{la}'). Otherwise there is not enough space to shut down the processor if J_{la} is scaled and we should reduce the speed of J_i and J_{la} as low as possible within $[t_{cur}, t_d]$, as we did in Figure 2(d).

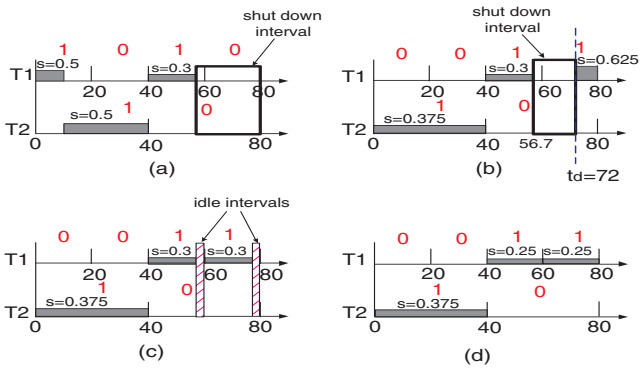

Fig. 2. (a) The schedule of task set (τ_1=(20,20,5,2,4); τ_2=(40,40,15,1,2)) under the static approach; (b) The schedule under dynamic pattern with the *Scale−to−Critical Speed and Shut−down* Strategy; (c) The schedule under dynamic pattern with look-ahead approach; (d) The schedule under dynamic pattern and with speed reduced below the critical speed.

Note that in case 1) and 3), the speeds of J_i and J_{la} can be scaled below the critical speed s_{crit} (*e.g.*, job J_{13} and job J_{14} in Figure 2(d)). However, the energy efficiency can be better than the ones without looking-ahead. The overhead of this approach mainly comes from computing t_d which has been shown to have very low online overhead in [6].

C. Executing optional jobs

To choose the appropriate optional job to execute, we first introduce the following two definitions.

Definition 1: The ***Task Energy Density*** of each task τ_i (denoted as TED_i) is defined as

$$TED_i = \frac{m_i E(s_i)}{k_i T_i} \tag{7}$$

where $E(s_i)$ is the energy consumption to execute a mandatory job of τ_i under its predetermined speed s_i.

Definition 2: The ***Job Energy Density*** of a job J_i (denoted as JED_i) is defined as

$$JED_i = \frac{E(s_i')}{T_i} \tag{8}$$

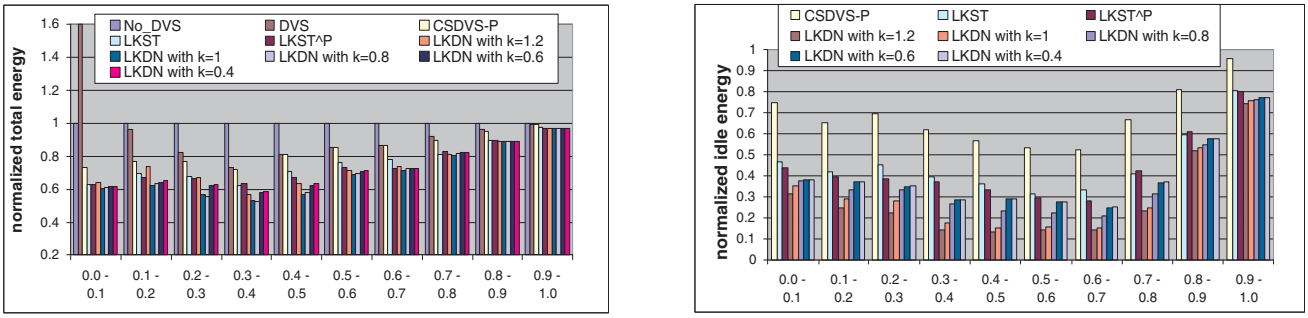

Fig. 3. (a) The total energy comparison by the different approaches. (b) The idle energy comparison by the different approaches.

where $E(s_i')$ is the energy consumption to execute job J_i under its current energy efficient speed s_i'.

Note that if the optional job cannot meet its deadline, it doesn't help in energy reduction or pattern adjustment. To guarantee the optional job can meet its deadline, the selected optional job should be executed non-preemptively. So the potential energy efficient speed s_i' for each optional job J_i can be computed at the online phase by treating it as the only ready job in the interval $[t_{cur}, t_d]$, similar to the speed determination approach in Section IV-B (in this case the speed for the future mandatory jobs should also be updated correspondingly in the same way. Moreover, when J_i is under consideration, the next upcoming mandatory job from the same task can be demoted to optional temporarily with the expectation that J_i will be selected for execution. If J_i is not selected, it should be restored.)

Whenever a job completes and the MQ is empty, we first check each optional job J_i in OQ by inspecting its job energy density JED_i under its potential energy efficient speed s_i' within the interval $[t_{cur}, t_d]$. Only those optional jobs with $JED_i < \kappa TED_i$ will be chosen as candidate jobs, where $\kappa(>0)$ is a user defined parameter to control the selection of optional jobs. Generally κ can be set to be 1 for energy saving purpose. But the user can also vary the value of κ to control the chances that optional jobs will be scheduled and thus provide the micro-tunable performance (QoS *versus* energy) levels that can be achieved without affecting the schedulability of the whole task set.

After that the candidate jobs are sorted according to their energy-gain/criticality ratio $\Delta E(J_i)/Cr_i$, where $\Delta E(J_i) = E(s_i) - E(s_i')$ and the criticality Cr_i is the number of deadline misses allowed before a dynamic failure occurs to J_i. Since the energy-gain $\Delta E(J_i)$ reflects the potential energy saving that can be achieved by executing the optional job and the criticality Cr_i reflects the relative urgency to schedule the optional job in order to meet the (m,k)-constraint, the candidate job with the maximal $\Delta E(J_i)/Cr_i$ will be chosen to be executed.

Compared with the static approach, the dynamic approach can lead to more aggressive energy reduction due to its adaptive pattern adjustment and speed determination with the run-time conditions. Moreover, when there are more than one jobs in the MQ, dynamic reclaiming techniques [19] can also be exploited to further reduce the energy. To ensure the (m,k)-constraints can be guaranteed, we have the following theorem:

Theorem 4: Let $\mathcal{T} = \{\tau_0, \tau_1, ..., \tau_{n-1}\}$ be executed in a variable voltage processor , where $\tau_i = \{T_i, D_i, C_i, m_i, k_i\}$. The dynamic approach, with complexity of $O(n)$, can ensure the (m,k)-requirements for $\mathcal{T}$ if $\mathcal{T}$ is schedulable with R-pattern.

V. Experimental results

Six different approaches are studied using experiments in this section. In the first approach, the task sets are statically partitioned with E-pattern, and the mandatory jobs are executed with the highest processor speed. We refer this approach as *NoDVS* and use its results as the reference results. In the second approach, the task sets are statically partitioned with E-pattern, but the speed of the tasks are scaled down at the task level as low as possible, We refer this approach as *DVS*. In the third approach, the task sets are statically partitioned with E-pattern, and the speed of the tasks are scaled as low as s_{crit} at the task level, then the approach in [5] based on individual task procrastination is applied, we call this approach $CSDVS-P$. The fourth approach, namely *LKST*, is our static approach introduced in Section III. The task sets are statically partitioned with $E-pattern$, and the speeds of the tasks are scaled as low as s_{crit}. When the system is idle, we tried to delay the mandatory jobs with the approaches introduced in Section III and shut down the processor whenever possible. Since the parameter controlled procrastination approach in [7] can also be applied to our static approach, in the fifth approach, we will also incorporate this strategy into our static approach with the parameter set correspondingly (according to [7], the best value should be 0.6 for our processor model), we refer this approach as $LKST^P$. The sixth approach *LKDN* is our dynamic approach as explained in Section IV. In this approach, we dynamically vary the job patterns according to the run time conditions and reduce the job speed (below the critical speed when necessary) as well as shut down the processor whenever possible. Since the value of κ, *i.e.*, the job selection coefficient in our dynamic approach, can affect the selection of optional jobs and thus

the performance of the algorithm, we also vary the value of κ within the range of (0.4, 0.6, 0.8, 1.0, 1.2) to compared their performance. For the processor model we will adopt the same processor model as used in [7], *i.e.* the Intel XScale processor. The parameters are the same as used in Section IV-B.

We first studied the energy consumption of these approaches based on the synthesized task sets. The periodic task sets tested in our experiments were randomly generated with the periods and the worst case execution times (WCET) of the tasks randomly chosen in the range of $[10ms, 100ms]$ and $[1ms, 30ms]$, respectively. The deadlines of the tasks were set to be less than or equal to their periods. The actual execution time of a job was randomly picked from [0.1WCET, WCET]. The m_i and k_i for the (m,k)-constraints were also randomly generated such that k_i is uniformly distributed between 2 to 10, and $m_i < k_i$. The total utilization, i.e., $\sum_i \frac{m_i C_i}{k_i T_i}$, is divided into intervals of length 0.1 and we randomly generate 50 feasible task sets for each interval. The energy consumption for each approach was normalized to that by *NoDVS*, and the results are shown in Figure 3(a) and (b).

From Figure 3(a), *DVS* will cause dramatic increase in the total energy consumption for low utilization intervals. For example, when the utilization is between [0.0, 0.1], the energy consumption by *DVS* is more than 60% of that by *NoDVS*. On the contrary, voltage scaling approaches with static energy and procrastination in mind can reduce the total energy consumption significantly. Moreover, with effective procrastination for the mandatory jobs and dynamic pattern adjustment, our static approach and dynamic approach can reduce the energy consumption further effectively. For example, compared with $CSDVS-P$, *LKST* can lead to a total energy reduction by around 8%. The $LKST^P$ has a marginal improvement over *LKST* and its energy consumption in some intervals can be slightly higher than that by *LKST*. Our dynamic approach *LKDN* can reduce the total energy more significantly. Compared with $CSDVS-P$, the maximal reduction can be around 23%. It is also noticed that during some low utilization interval the total energy consumption by *LKDN* is close to *LKST* and $LKST^P$ for small κ values. This is because, when the the system utilization is low and the value of κ is small, there is little chance to select candidate optional jobs and scale the speeds further. In this case, the procrastination and shut-down strategy will dominate. In some intervals, the energy performance of *LKDN* with very large value of κ is not the best either because in those cases *LKDN* optionally executed some redundant jobs. Although those jobs can help enhance the user perceivable QoS levels, they might also cost extra energy that cannot be completely compensated by the energy reduction from varying the job patterns. And it seems the overall energy is the lowest when κ is between [0.8, 1] and generally setting $\kappa = 1$ is reasonably good.

Due to the energy overhead of shut-down and the dramatic increase of the static power, the energy consumption during the processor idle time will also account for a significant part of the total energy consumption. We are therefore interested in investigating how our approach can help reduce this part of energy. Figure 3(b) shows the average idle energy consumptions by the different approaches. Note that, our approaches, *i.e.*, *LKST*, $LKST^P$ and *LKDN* can always lead to much better idle energy savings than the previous approaches. *LKST* and $LKST^P$ both consume much lower idle energy than $CSDVS-P$ due to effective idle extension and merging. The performance of *LKDN* is even better because it utilizes the idle intervals more efficiently to help reduce the job speed (below the critical speed when necessary) and to facilitate shut-down. Compared with $CSDVS-P$, *LKST* and $LKST^P$ can reduce the idle energy by 38% and *LKDN* can reduce the idle energy by up to 76%.

Next, we tested our techniques in a more practical environment. The test cases contained two real world applications: webphone [20], and INS (Inertial Navigation System) [21]. The timing parameters such as the deadlines, periods, and execution times were adopted from these practical applications. The actual execution times and the (m,k)-constraints were generated as we did for the synthesized task sets. Since the value of κ between [0.8, 1.0] tends to have better performance for our dynamic approach, this time we fixed the value of κ to be 0.9. The normalized total energy consumptions and idle energy consumptions are shown in Figure 4.

The experimental results based on the practical applications further demonstrate the effectiveness of our approaches in saving energy. As shown in Figure 4(a) and (b), for the webphone application, the static approach *LKST* and $LKST^P$ can reduce the total energy consumption by around 7% and idle energy consumption by about 30%. And the energy saving by the dynamic approach (*LKDN*) can be up to 12% for total energy consumption and up to 71% for idle energy consumption. For INS application, as shown in Figure 4(c) and (d), the static approach *LKST* and $LKST^P$ can reduce the total energy consumption by about 11% and idle energy consumption by about 36%. And the energy saving by *LKDN* can be up to 18% for total energy consumption and 78% for idle energy consumption.

VI. Conclusions and future work

Low power/energy consumption and QoS guarantee are two of the most critical factors for the successful design of pervasive real-time computing platforms. While the dynamic voltage scaling (DVS) techniques are efficient in reducing the dynamic power consumption for the processor, varying voltage alone becomes less effective for the overall energy reduction as the static power is growing rapidly. In this paper, we propose two approaches, a static one and a dynamic one, to reduce the overall energy consumption for real-time systems while guaranteeing the given QoS requirement in terms of (m,k)-constraints. The static approach determines the mandatory jobs statically and try to delay the mandatory jobs and shut down the processor whenever possible. The dynamic approach vary the job pattern dynamically during the runtime and determine the job speed in a more adaptive way. Through extensive simulations, our approaches outperformed previous research in both overall and idle energy reduction while providing the (m,k)-guarantee.

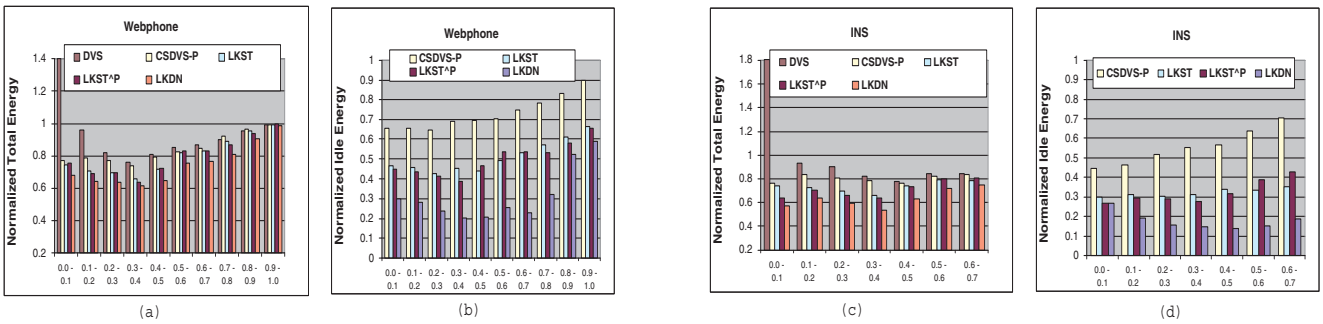

Fig. 4. (a) The total energy comparison for webphone. (b) The idle energy comparison for webphone. (c) The total energy comparison for INS. (d) The idle energy comparison for INS.

Future work As shown in [22], [23], there is a dependency relationship between leakage and temperature and it plays a critical role in both the power and thermal aware design. In this paper, the problem can be considered from two aspects:

- 1) Under (m,k) model, different from hard real-time case, soft task sets with relatively high (m,k)-utilization are often hard to be schedulable due to the (m,k)-constraint. That means, the schedulable processor utilization will be typically under-loaded for the general case. That will generate a lot of "internal" system idle intervals which, together with the static procrastination to facilitate processor shut down, provides us great opportunities to cool down the processor and thus reduce the effect of leakage/temperature dependency effectively;
- 2) With our dynamic pattern variation strategy, we are trying to reduce the job speed below the critical speed dynamically as well as shut down the processor whenever possible, which is also helpful in keeping the processor running in relatively low temperatures. Moreover, when executing the mandatory jobs, we can always incorporate the enhanced dual priority scheduling strategy in [13] by executing each mandatory job with a dual-speed mode, *i.e.*, the lowest speed before promotion and normal speed after promotion, which is also very helpful in preventing the processor temperature from increasing too fast.

However, in terms of reducing the temperature, what is proposed in this paper is still a best-effort approach. How to minimize the overall energy consumption under a given maximal temperature constraint while still ensuring the (m,k)-guarantee can be our future work.

ACKNOWLEDGE*

This work is supported in part by NSF under projects CNS-0545913 and CNS-0917021.

REFERENCES

[1] ITRS, *International Technology Roadmap for Semiconductors*. Austin, TX.: International SEMATECH, http://public.itrs.net/.
[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] 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.
[4] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced cpu energy," in *AFCS*, 1995, pp. 374–382.
[5] R. Jejurikar, C. Pereira, and R. Gupta, "Dynamic slack reclamation with procrastination scheduling in real-time embedded systems," *DAC*, 2005.
[6] L. Niu and G. Quan, "Reducing both dynamic and leakage energy consumption for hard real-time systems," *CASES'04*, Sep 2004.
[7] J.-J. Chen and T.-W. Kuo, "Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems," in *ICCAD*, 2007.
[8] J.-J. Chen, N. Stoimenov, and L. Thiele, "Feasibility analysis of on-line dvs algorithms for scheduling arbitrary event streams," in *RTSS*, 2009.
[9] M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m,k)-firm deadlines," *IEEE Transactions on Computes*, vol. 44, pp. 1443–1451, Dec 1995.
[10] 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*, pp. 717–729, July 2006.
[11] S. Hua, G. Qu, and S. Bhattacharyya, "Energy reduction techniques for multimedia applications with tolerance to deadline misses," *DAC*, pp. 131–136, 2003.
[12] T. A. AlEnawy and H. Aydin, "Energy-constrained scheduling for weakly-hard real-time systems," *RTSS*, 2005.
[13] L. Niu, "Energy-aware dual-mode voltage scaling for weakly hard real-time systems," *SAC'10 (RTS Track)*, 2010.
[14] L. Niu and G. Quan, "Peripheral-conscious scheduling on energy minimization for weakly hard real-time systems," *DATE*, 2007.
[15] INTEL-XSCALE. (2003). [Online]. Available: http://developer.intel.com/design/xscale/.
[16] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, vol. 17, no. 2, pp. 46–61, 1973.
[17] G. Quan and X. Hu, "Enhanced fixed-priority scheduling with (m,k)-firm guarantee," in *RTSS*, 2000, pp. 79–88.
[18] L. Niu and G. Quan, "Peripheral-conscious scheduling on energy minimization for weakly hard real-time systems," *under review by International Journal of Embedded Systems*, 2006.
[19] L. Niu, "Leakage-aware scheduling for real-time systems with (m,k)-constraint," in *Technical Report TR-2010-051, Claflin University*, 2010.
[20] D. Shin, J. Kim, and S. Lee, "Intra-task voltage scheduling for low-energy hard real-time applications," *IEEE Design and Test of Computers*, vol. 18, no. 2, March-April 2001.
[21] A.Burns, K. Tindell, and A. Wellings, "Effective analysis for engineering real-time fixed priority schedulers," *IEEE Transactions on Software Engineering*, vol. 21, pp. 920–934, May 1995.
[22] L. Yuan and G. Qu, "Alt-dvs: Dynamic voltage scaling with awareness of leakage and temperature for real-time systems," in *AHS*, 2007.
[23] G. Quan and Y. Zhang, "Leakage aware feasibility analysis for temperature-constrained hard real-time periodic tasks," in *ECRTS*, 2009.