Dynamic Voltage Scaling for Mixed Task Sets in Fixed-Priority Systems

Dongkun Shin and Jihong Kim
School of Computer Science and Engineering
Seoul National University
San 56-1, Shillim-dong, Kwanak-gu, Seoul, Korea
TEL : 02-880-1861
Email : sdk@davinci.snu.ac.kr, jihong@davinci.snu.ac.kr

Abstract - *We address the problem of dynamic voltage scaling (DVS) for real-time systems with both periodic and aperiodic tasks. Although many DVS algorithms have been developed for real-time systems with periodic tasks, the arbitrary temporal behaviors of aperiodic tasks make it difficult to use the algorithms for such a system with mixed tasks. We propose an off-line DVS algorithm and on-line DVS algorithms that are based on existing DVS algorithms but can utilize the execution behavior of bandwidth-preserving server which is a dedicated server to service aperiodic tasks. Experimental results show that the proposed algorithms reduce the energy consumption by 26% over the power-down method under the RM scheduling policy.*

Ⅰ. Introduction

Many practical real-time applications require aperiodic tasks as well as periodic tasks. For example, consider multimedia applications (e.g., MP3 or MPEG player) in which audio or video data is decoded periodically maintaining consistent output rates. These systems continue accepting user inputs that need prompt responses (e.g., volume control, playback control or playlist editing). While the decoding tasks are periodic tasks, the tasks to service user inputs are aperiodic tasks. Generally, periodic tasks are time-driven with hard deadlines but aperiodic tasks are event-driven (i.e., activated at arbitrary times) with soft deadlines. In this paper, we call a system with both periodic and aperiodic tasks as a mixed task system.

In mixed task systems, there are two design objectives. The first objective is to guarantee the schedulability of all periodic tasks under worst-case execution scenarios. That is, aperiodic tasks should not prevent periodic tasks from completing before their deadlines. The second objective is that aperiodic tasks should have "good" average response times. To satisfy these objectives, many scheduling algorithms such as deferrable server, sporadic server, total bandwidth server and constant bandwidth server had been proposed [7, 10, 1]. They are called "*bandwidth-preserving servers*". In this paper, we introduce the third design objective for the energy consumption in the mixed task system. That is, the third objective is to minimize the total energy consumption due to *both* periodic tasks and aperiodic tasks.

Dynamic voltage scaling (DVS) [5] is a good candidate to reduce the energy consumption of real-time systems. When the required performance of the target system is lower than the maximum performance, we can reduce the supply voltage and the clock speed to minimize the energy consumption. Recently, many voltage scheduling algorithms have been proposed for hard real-time systems [9, 2, 8, 4]. All of these algorithms assume that the system consists of periodic hard real-time tasks only and the task release times are known *a priori*.

Although the existing DVS algorithms can be effective for optimizing the energy consumption of periodic tasks, they cannot be used for mixed task systems. The arbitrary behaviors of aperiodic tasks prevent the DVS algorithms from identifying the slack times. Therefore, it is necessary to modify the existing DVS algorithms to be applicable to mixed task systems with aperiodic tasks.

Despite of many researches on dynamic voltage scheduling, there have been few studies to adapt DVS techniques to the aperiodic task scheduling. A recent work by W. Yuan and K. Nahrstedt [11] proposed a DVS algorithm for soft real-time multimedia and best-effort applications. The target of their algorithm is aperiodic task systems, not mixed task systems. Y. Doh *et al.* [3] investigated the problem of allocating both energy and utilization for mixed task systems. They used the total bandwidth server and considered the static scheduling problem only. Given the energy budget, their algorithm finds voltage settings for both periodic and aperiodic

tasks such that all periodic tasks are completed before their deadlines and all aperiodic tasks can attain the minimal response times.

We propose DVS algorithms that guarantee the first objective (i.e., timing constraints of periodic tasks) while making the best effort of satisfying the third objective (i.e., low energy) with a reasonable performance bound on the second objective (i.e., good average response time). We present new dynamic voltage scheduling algorithms by adding the slack estimation method for the bandwidth-preserving server to existing on-line voltage scheduling algorithms for a periodic task set.

The modified DVS algorithms utilize the execution behaviors of bandwidth-preserving server for aperiodic tasks to apply the key ideas of the existing DVS algorithms such as [9, 8, 2]. The task schedules generated by the proposed DVS algorithms can reduce the energy consumption by 25% over the task schedules which execute all tasks at full speed and power down at idle intervals (i.e., the power-down method).

To the best of our knowledge, our work is the *first* attempt to develop *on-line* DVS algorithms for the mixed task system. While Y. Doh *et al.*'s algorithm is an off-line static speed assignment algorithm under the EDF scheduling policy, our work in this paper considers both off-line and on-line algorithms under RM scheduling policy. Another difference is that we concentrate on minimizing the energy consumption under the constraint on the average response time.

The rest of this paper is organized as follows. In Section II, we introduce the problems of static speed assignment and dynamic speed assignment in mixed task systems. The dynamic speed assignment algorithms are presented in Sections III. In Section IV, the experimental results are discussed. We conclude with a summary and future works in Section V.

Ⅱ. Problem Formulation

We assume that a mixed task system T consists of n periodic tasks, $\tau_1, \cdots, \tau_n$, and an aperiodic task, σ. The aperiodic task σ is serviced by a bandwidth-preserving server S. The bandwidth -preserving server S is characterized by an ordered pair (Q_s, T_s) where Q_s is the maximum budget and T_s is the period of the server. During the execution of aperiodic tasks, the budget of S is consumed. We use q_s to denote the remaining budget of S. The budget q_s is set to Q_s at each replenishment time. S is scheduled together with periodic tasks in the system according to the given priority-driven algorithm. Once S is activated, it executes any pending aperiodic requests within the limit of its budget q_s.

A periodic task τ_i is specified by (C_{τ_i}, T_{τ_i}) where C_{τ_i} and T_{τ_i} are the worst-case execution time (WCET) and the period of τ_i, respectively. We assume that periodic tasks have relative deadlines equal to their periods. The j-th instance of τ_i and the k-th instance of σ are denoted by $\tau_{i,j}$ and σ_k, respectively. We assume that the aperiodic task instances $\sigma_1, \cdots, \sigma_m$ are executed during the hyper period H of periodic tasks.

We first consider the static speed assignment problem considering both the expected workload and the schedulability condition. Our static voltage assignment algorithm selects the operating speed S_p of periodic tasks and the operating speed S_s of scheduling server for aperiodic tasks, respectively. S_p and S_s should allow a real-time scheduler to meet all the deadlines for a given periodic task set minimizing the total energy consumption. Consequently, the problem of the static speed assignment can be formulated as follows:

Static Speed Assignment Problem

Given U_p, U_s, w, and ρ,
find S_p and S_s such that
$E = U_p \cdot w \cdot S_p^2 + \rho \cdot S_s^2$ is minimized
subject to $\frac{U_p}{S_p} + \frac{U_s}{S_s} \le U_{lub}$ and $0 \le S_p, S_s \le 1$.

where U_p is the worst case utilization of periodic task set, U_s $(= Q_s/T_s)$ is the server utilization, w is the average workload ratio of periodic tasks, and ρ is the average workload of aperiodic tasks. E is a metric reflecting energy consumption[1]. U_{lub}, which is the least upper bound of schedulable utilization, is $n(2^{1/n}-1)$ for n tasks at the RM scheduling. Using the Lagrange transform, we can get a following optimal solution for S_p and S_s.

$$S_p = \frac{1}{U_{lub}}\left(U_p + U_s \cdot \sqrt[3]{\frac{\rho}{U_s \cdot w}}\right),$$

$$S_s = \frac{1}{U_{lub}}\left(U_p \cdot \sqrt[3]{\frac{U_s \cdot w}{\rho}} + U_s\right)$$

1) Assuming the supply voltage and clock speed are proportional in DVS, the energy consumption is represented to be proportional to the square of clock speed.

Under the assumption that we can know the exact w and ρ values, we can get the optimal static speeds for periodic and aperiodic tasks.

Dynamic speed assignment problem is to find the speeds of each periodic task instances and aperiodic task instances at run time. Our objective is to minimize the total energy consumption of both periodic and aperiodic tasks using a DVS algorithm while satisfying the timing constraints of periodic tasks and bounding the response time delay.

If an aperiodic task σ_k can be serviced without any interference by periodic tasks or another aperiodic tasks, the response time of the aperiodic task σ_k is $c(\sigma_k)/s(\sigma_k)$ where $c(\sigma_k)$ and $s(\sigma_k)$ are the number of execution cycles and the clock speed of σ_k, respectively. However, the execution of the aperiodic task σ_k is delayed due to the following factors: (1) *Queueing delay*: σ_k should wait until the completion time of the aperiodic tasks released before σ_k. (2) *Budget delay*: σ_k should wait until the next replenishment time if q_s of the bandwidth-preserving server S is 0. (3) *Preemption delay*: σ_k should wait until the completion time of the periodic tasks which have higher priorities than the priority of S. We denote the delays due to the queueing, budget and preemption as $w(\sigma_k)$, $b(\sigma_k)$, and $p(\sigma_k)$, respectively. Then, the response time of σ_k can be represented as

$$c(\sigma_k)/s(\sigma_k) + w(\sigma_k) + b(\sigma_k) + p(\sigma_k).$$

The response time will be increased by a DVS algorithm because $s(\sigma_k)$, $w(\sigma_k)$, $b(\sigma_k)$ and $p(\sigma_k)$ are changed. When the response times of σ_k are t and $t+D$ in the non-DVS scheme and the DVS scheme respectively, we call the increase D in the response time as the *response time delay*. Therefore, the problem of dynamic speed assignment (DSA) can be formulated as follows:

Dynamic Speed Assignment Problem

Given $T=\{\tau_1,\cdots,\tau_n,\sigma\}$, S and δ,
find $s(\tau_{1,1}),\cdots,s(\tau_{n,H/T_n})$ and $s(\sigma_1),\cdots,s(\sigma_m)$ such that $E=\sum_{i=1}^{n}\sum_{j=1}^{H/T_i} E(\tau_{i,j}) + \sum_{k=1}^{m} E(\sigma_k)$ is minimized
subject to $\forall i,j, e(\tau_{i,j}) \le j \cdot T_{\tau_i}$ and $\forall k, D(\sigma_k) \le \delta$.

where $s(\tau_{i,j})$, $E(\tau_{i,j})$, and $e(\tau_{i,j})$ are the clock speed, the energy consumption and the completion time of the task instance $\tau_{i,j}$, respectively. $E(\sigma_k)$ denotes the energy consumption of the aperiodic task instance σ_k. $D(\sigma_k)$ represents the response time delay of σ_k. In this paper, we propose the DVS algorithms which provide solutions for the DSA problem when $\delta = T_s - Q_s$.

Existing on-line DVS algorithms such as [9, 2, 8, 4] are not directly applicable for the DSA problem. For example, consider the *stretching-to-NTA* technique used in [9]. It stretches the execution time of the periodic task ready for execution to the next arrival time of a periodic task when there is no another periodic task in ready queue. To use the *stretching-to-NTA* technique for a mixed task system, we should know the next arrival time of an aperiodic task as well as a periodic task. Though the arrival times of periodic tasks can be easily computed using their periods, we cannot know the arrival times of aperiodic tasks since they arrive at arbitrary times. If we ignore the arrivals of aperiodic tasks, there will be a deadline miss of periodic hard real-time task when an aperiodic task arrives before the next arrival time of a periodic task. Consequently, the *stretching-to-NTA* technique should assign the full speed to all tasks in the mixed task system.

Therefore, we need to modify on-line DVS algorithms to utilize the characteristics of bandwidth-preserving servers. In this paper, we handle only sporadic server [10] because it is more advanced algorithm for the RM scheduling policy.

Ⅲ. Dynamic Speed Assignment

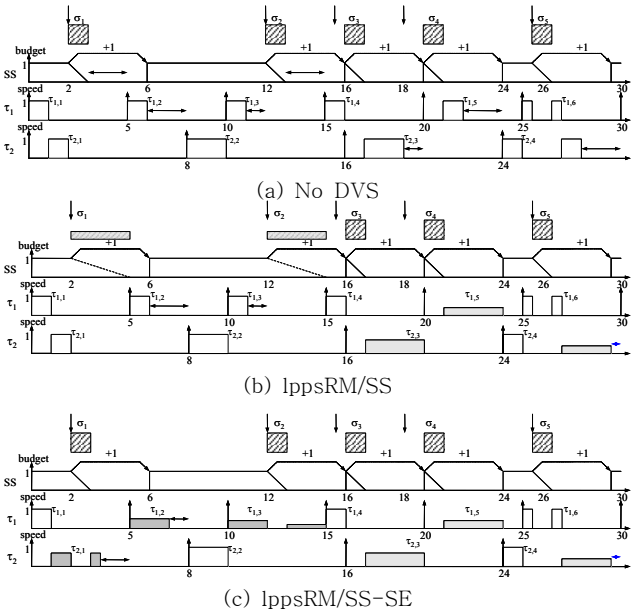
Figure 1(a) shows the task schedule using a sporadic server SS, assuming two periodic tasks, $\tau_1=(1,5)$ and $\tau_2=(2,8)$, and one SS =(1,4). The budget of SS, q_s, is set to Q_s at time 0. If an aperiodic task is executed during the time $[t_1, t_2]$, q_s is reduced by $t_2 - t_1$ at the time t_2. The budget q_s is replenished by the amount of $t_2 - t_1$ at the time $t_1 + T_s$. SS preserves its budget q_s if no requests are pending when released. An aperiodic request can be serviced at any time (at server's priority) as long as the budget of SS is not exhausted (e.g., task σ_1). If the budget is exhausted, aperiodic tasks should wait until the next replenishment time. For example, though the task σ_4 arrived at the time 19, it is serviced at the time 20.

Although we cannot know the arrival times of aperiodic tasks, the *stretching-to-NTA* method can be used if we utilize the execution behavior of SS. There are two cases the current ready task can be stretched: (1) **Rule for aperiodic task**: If there is no periodic task in the ready queue,

execute an aperiodic task at the speed of q_s /(min(next arrival time of a periodic task, next replenishment time)-t) where t is the current time. (2) **Rule for periodic task**: If there is only one periodic task in the ready queue and q_s is 0, stretch the periodic task to min(next arrival time of a periodic task, next replenishment time). This is because the arriving aperiodic task is delayed until the next replenishment time if q_s is 0. If q_s>0, we cannot scale down the speed of the periodic task even though there is only one periodic task in the ready queue.

Using these two rules, we modified existing on-line DVS algorithms. Figure 1(b) shows the task schedule using the lppsRM/SS algorithm which is the modified version of lppsRM [9] for SS. lppsRM uses the *stretching-to-NTA* method. The aperiodic tasks σ_1 and σ_2 are stretched to the next arrival times of periodic tasks (5 and 15) because there is no periodic task in ready queue. The periodic tasks $\tau_{1,5}$, $\tau_{2,3}$, and the latter part of $\tau_{2,4}$ are stretched to min(next arrival time, next replenishment time) because q_s is 0. We cannot stretch the tasks $\tau_{1,2}$ and $\tau_{1,3}$ because q_s is larger than 0.

The *preemption delays* in lppsRM and lppsRM/SS are same because periodic tasks are stretched only when q_s=0 by the stretching rule for periodic task. The *budget delays* are also same due to the stretching rule for aperiodic task. However, since the *queueing delay* and the clock speed of aperiodic task are changed, the response time of aperiodic task in lppsRM/SS is longer than that in lppsRM. Nevertheless, we can guarantee that $D(\sigma_k) \le T_s - Q_s$ for all σ_k. If σ_k is completed at t in lppsRM, the completion time of σ_k is smaller than $t + T_s - Q_s$ in lppsRM/SS because $R \le t + T_s - Q_s$ where R is the next replenishment time.



(a) No DVS

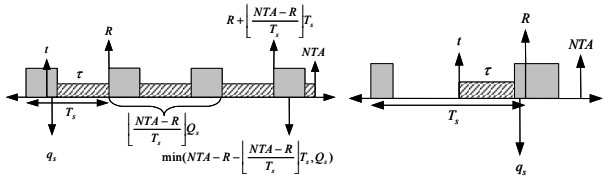
(b) lppsRM/SS

(c) lppsRM/SS-SE

Figure 1. Task schedules with a sporadic server.

Though we can reduce the energy consumption by lppsRM/SS algorithm, the algorithm can show poor performance when the workload of aperiodic tasks is small. In this case, since the budget q_s is larger than 0 at most of scheduling points, we cannot use the stretching rule for periodic task. Extremely, when there is no aperiodic request, there is nothing to do for the DVS algorithm. Therefore, we need a more advanced DVS algorithm which can be applicable to the mixed task system with a low aperiodic workload. For this purpose, we propose a new slack estimation method, *bandwidth-based slack-stealing*, which identifies the maximum slack time for a periodic task considering the bandwidth of sporadic server. Figure 1(c) shows the lppsRM/SS-SE algorithm, which is based on lppsRM/SS but uses the *bandwidth-based slack-stealing* method. When q_s is larger than 0 and there is only one periodic task in the ready queue, the slack estimation method calculates the maximum available time before the arrival time of next periodic task.

Figure 2 shows the *bandwidth-based slack-stealing* method. In Figure 2, T_τ is the period of τ, t is the current time, NTA is the next periodic task arrival time and R is the next replenishment time of SS. We should consider two different cases depending on the priority of SS. Figure 2(a) shows the case when $T_\tau > T_s$. In this case, the maximum blocking time by aperiodic tasks before the next task arrival time (NTA) should be identified. Figure 2(b) shows the case when $T_\tau < T_s$. In this case, the task τ is stretched to $\min(R, NTA) - q_s$. Although there is no deadline miss even when the periodic task τ is completed after R, the proposed DVS algorithm is designed to bound the response time delay. Under this policy, the *preemption delay* is increased but we can guarantee that $D(\sigma_k) \le T_s - Q_s$ for all σ_k because σ_k is not delayed above the replenishment time R.



(a) $T_\tau > T_s$ (b) $T_\tau < T_s$

Figure 2. Bandwidth-based slack stealing in lppsRM/SS-SE.

From Figure 2, the maximum available time MAT of a task τ can be calculated as follows:

$$\text{if } (T_\tau > T_s) \quad MAT = NTA - t - q_s - \left\lfloor \frac{NTA-R}{T_s} \right\rfloor Q_s - \min\left(NTA - R - \left\lfloor \frac{NTA-R}{T_s} \right\rfloor / T_s, Q_s\right)$$

$$\text{if } (T_\tau < T_s) \quad MAT = \min(R, NTA) - t - q_s$$

In Figure 1(c), the periodic tasks $\tau_{1,2}$, $\tau_{1,3}$ and $\tau_{2,1}$ are stretched by the *bandwidth-based slack-stealing* method. For example, at the time 5, the task $\tau_{1,2}$ has the available time 2 ($= NTA - t - q_s = 8 - 5 - 1$). A side effect of the *bandwidth-based slack-stealing* method is that aperiodic tasks tend to be executed at full speed. Due to the side effect, the DVS algorithm using the *bandwidth-based slack-stealing* method generates better average response times.

Ⅳ. Experimental Results

We have evaluated the performance of our DVS algorithms for sporadic server using simulations. The execution time of each periodic task instance was randomly drawn from a Gaussian distribution in the range of [BCET, WCET] where BCET is the best case execution time.

The interarrival times and service times of aperiodic tasks were generated from the exponential distribution using the parameters λ and μ where $1/\lambda$ is the mean interarrival time and $1/\mu$ is the mean service time. Then, the workload of aperiodic tasks can be represented by $\rho = \lambda/\mu$. If there is no interference between aperiodic tasks and periodic tasks, the average response time of aperiodic tasks is given by $(\mu - \lambda)^{-1}$ from the M/M/1 queueing model.

Table 1 shows the experimental results of the static speed assignment. The results show the energy consumption and response time normalized by the results of uniform speed assignment method, varying U_s with fixed values of U_p and ρ. In this experiments, BCET is assumed to be 50% of WCET. The uniform speed assignment method assigns the same speed to both periodic tasks and aperiodic tasks making the total utilization as U_{lub}. We assumed that if the system is idle it enters into the power-down mode. The proposed static speed assignment method reduced the energy consumption and the average response time up to 14% and 5%, respectively. Since the scheduling server gets a higher speed than the speed for periodic tasks when $w > \rho$, the static speed assignment reduces the average response time as well as the energy consumption.

Table 1. Experimental results of static speed assignment ($U_p = 0.3, \rho = 0.1$)

U_s	Normalized Energy Consumption	Normalized Response Time
0.15	0.98	0.97
0.20	0.88	0.96
0.25	0.91	0.95
0.30	0.88	0.96
0.35	0.86	0.97

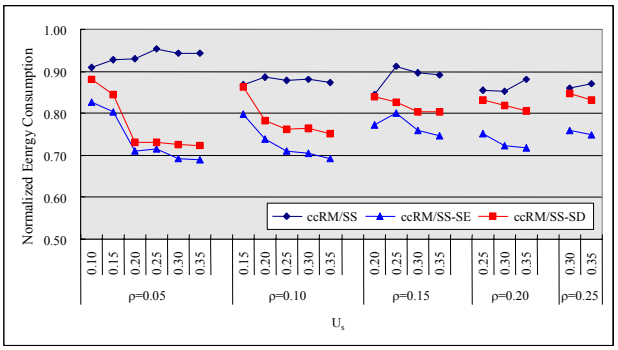
For the dynamic speed assignment algorithm, we observed the energy consumption of the total system and the average response time of aperiodic tasks varying the server utilization U_s and the workload of aperiodic tasks ρ under a fixed utilization U_p of periodic tasks. (Due to the limited space, we present the experimental results where U_s is controlled by changing the value of T_s with a fixed Q_s value and ρ is controlled by a varying λ with a fixed μ value.)

The periodic task set has three tasks with U_p =0.3. For all experiments including the non-DVS scheme, both periodic tasks and aperiodic tasks were given an initial clock speed $f_0 = (U_p + U_s) f_m / U_{lub}$, where f_m is the maximum clock speed. During run time, the speed is further reduced by on-line DVS algorithms exploiting the slack times. In the experiments, BCET is assumed to be 10% of WCET.

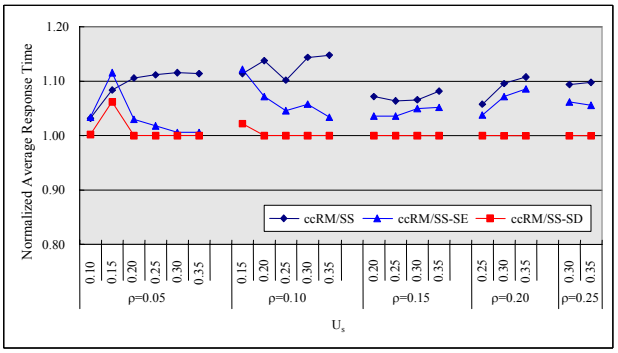
Figure 3(a) shows the energy consumptions of the ccRM/SS algorithm and the ccRM/SS-SE algorithm normalized by that of the power-down method. ccRM [8] also use the *stretching-to-NTA* method. ccRM/SS and ccRM/SS-SE use the proposed dynamic speed assignment algorithms additionally. We also evaluated the modified version of ccRM/SS-SE called ccRM/SS-SD. The ccRM/SS-SD algorithm uses a different slack distribution method. When slack times are identified, ccRM/SS-SD gives the slack times to only periodic tasks. Therefore, aperiodic tasks are always executed at the initial clock speed f_0. ccRM/SS-SD is good for a better response time.

The difference between the energy savings of ccRM/SS and ccRM/SS-SE decreases as ρ increases. This is because there are more chances for SS to have the zero budget when ρ is large. As U_s increases, ccRM/SS-SE shows a larger energy saving compared with ccRM/SS because ccRM/SS-SE performs well in the low aperiodic workload (over U_s). The ccRM/SS and ccRM/SS-SE reduced the energy consumption on average by 11% and 26% over the power-down method, respectively.

As shown in Figure 3(b), ccRM/SS and ccRM/SS-SE increase the response time on average by 10% and 5% over the power-down method, respectively. Due to the side effect on aperiodic tasks explained at Section III, ccRM/SS-SE shows better average response times. ccRM/SS-SD shows almost the same response time to that of power-down method because the execution speed of aperiodic task is always f_0 and the *preemption delay* is not increased except the case when T_s is larger than the periods of periodic tasks. However, it shows better energy performances than ccRM/SS.



(a) Energy Consumption



(b) Response Time

Figure 3. Experimental results using a sporadic server

V. Conclusions

We have proposed DVS algorithms for mixed task systems which have both periodic and aperiodic tasks. We presented the slack estimation methods for the bandwidth-preserving servers. Existing on-line DVS algorithms, which cannot be used for mixed task systems, were modified to use the proposed slack estimation methods. The modified DVS algorithms reduced the energy consumption by 26% over the power-down method. We also showed the effects of the slack distribution methods on the energy and the response time.

Our work in this paper can be extended in several directions. Though the proposed algorithm only guarantees that the response time delay is smaller than $T_s - Q_s$, it will be more useful if we can control the maximum response time delay with an arbitrary δ value. Furthermore, it will be interesting to use the DVS algorithm to utilize the temporal locality of aperiodic requests. When the aperiodic requests are sparse, we could use a larger δ value for a more energy-efficient schedule.

References

[1] L. Abeni and G. Buttazzo. Integrating Multimedia Applications in Hard Real-Time Systems. In *Proc. of IEEE Real-Time Systems Symp.*, pp. 4-13, 1998.

[2] H. Aydin, R. Melhem, D. Mosse, and P. M. Alvarez. Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems. In *Proc. of IEEE Real-Time Systems Symp.*, pp. 95-106, 2001.

[3] Y. Doh, D. Kim, Y.-H. Lee, and C. M. Krishna. Constrained Energy Allocation for Mixed Hard and Soft Real-Time Tasks. In *Proc. of Int. Conf. on Real-Time and Embedded Computing Systems and Applications*, pp. 533-550, 2003.

[4] W. Kim, J. Kim, and S. L. Min. A Dynamic Voltage Scaling Algorithm for Dynamic-Priority Hard Real-Time Systems Using Slack Time Analysis. In *Proc. of Design Automation and Test in Europe*, pp. 788-794, 2002.

[5] W. Kim, D. Shin, H.-S. Yun, J. Kim, and S. L. Min. Performance Comparison of Dynamic Voltage Scaling Algorithms for Hard Real-Time Systems. In *Proc. of IEEE Real-Time and Embedded Technology and Applications Symp.*, pp. 219-228, 2002.

[6] J. P. Lehoczky and S. Ramos-Thuel. An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed Priority Preemptive Systems. In *Proc. of IEEE Real-Time Systems Symp.*, pp. 110-123, 1992.

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

[8] P. Pillai and K. G. Shin. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *Proc. of ACM Symp. on Operating Systems Principles*, pp. 89-102, 2001.

[9] Y. Shin and K. Choi. Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems. In *Proc. of Design Automation Conf.*, pp. 134-139, 1999.

[10] B. Sprunt, L. Sha, and J. P. Lehoczky. Aperiodic Task Scheduling for Hard Real-Time Systems. *Journal of Real-Time Systems*, 1(1):27-60, 1989.

[11] W. Yuan and K. Nahrstedt. Integration of Dynamic Voltage Scaling and Soft Real-Time Scheduling for Open Mobile Systems. In *Proc. of Int. Workshop on Network and Operating Systems Support for Digital Audio and Video*, pp. 105-114, 2002.