Dynamic Voltage Scaling of Periodic and Aperiodic Tasks in Priority-Driven Systems*

Dongkun Shin
School of CSE
Seoul National University
Seoul, Korea 151-742
Tel: +82-2-880-1861
e-mail: sdk@davinci.snu.ac.kr

Jihong Kim
School of CSE
Seoul National University
Seoul, Korea 151-742
Tel: +82-2-880-8792
e-mail: jihong@davinci.snu.ac.kr

Abstract— We describe dynamic voltage scaling (DVS) algorithms for real-time systems with both periodic and aperiodic tasks. Although many DVS algorithms have been developed for real-time systems with periodic tasks, none of them can be used for the system with both periodic and aperiodic tasks because of arbitrary temporal behaviors of aperiodic tasks. We propose an off-line DVS algorithm and on-line DVS algorithms that are based on existing DVS algorithms. The proposed algorithms utilize the execution behaviors of scheduling server for aperiodic tasks. Experimental results show that the proposed algorithms reduce the energy consumption by 12% and 32% under the RM scheduling policy and the EDF scheduling policy, respectively.

I. Introduction

Dynamic voltage scaling (DVS) [3] is one of the most effective approaches in reducing the power consumption of real-time systems. When the required performance of the target system is lower than the maximum performance, supply voltage can be dynamically reduced to the lowest possible extent that ensures a proper operation of the system. Recently, many voltage scheduling algorithms have been proposed for hard real-time systems [9, 2, 8, 5]. 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*. For periodic tasks, these algorithms assign the proper speed to each task dynamically while guaranteeing all their deadlines.

However, many practical real-time applications require aperiodic tasks as well as periodic tasks. While periodic tasks are time-driven with hard deadlines, aperiodic tasks are usually event-driven (i.e., activated at arbitrary times) with soft deadlines. An aperiodic task set is specified by the mean arrival rate λ and the mean service rate μ. In this paper, we call a system with 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 based on the "server" concept had been proposed [12, 10, 11, 1].

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. 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.

In this paper, 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). First, we describe an off-line static voltage scaling algorithm which considers the expected workload of aperiodic tasks. Second, we present on-line dynamic voltage scaling algorithms by modifying existing on-line voltage scaling algorithms for a periodic task set.

The modified DVS algorithms utilize the execution behaviors of each scheduling server for aperiodic tasks to apply the key ideas of the existing DVS algorithms such as [9, 2]. The task schedules generated by the proposed DVS algorithms can reduce the energy consumption by 12%∼32% over the task schedules which execute all tasks at full speed and power down at idle intervals (i.e., the power-down mode). To the best of our knowledge, our work is the *first* attempt to develop *on-line* DVS algorithms for the mixed task set.

The rest of this paper is organized as follows. In Section II, we summarize the related works on aperiodic task scheduling and the recent efforts to integrate dynamic voltage scheduling into aperiodic task scheduling. The proposed static DVS algorithm is described in Section III while the dynamic DVS algorithms are presented in Section IV. In Section V, the experimental results are discussed. Section VI concludes with a summary and future works.

II. Related Works

In this section, we review the main approaches for scheduling a mixture of aperiodic tasks and periodic hard real-time

*This work was supported in part by the Ministry of Information & Communications, Korea, under the Information Technology Research Center (ITRC) Support Program.

tasks.

The easiest way to prevent aperiodic tasks from interfering with periodic hard real-time tasks is to schedule them as *background* tasks. In this approach, aperiodic tasks are scheduled and executed only at times when there is no periodic task ready for execution. Though this method guarantees the schedulability of periodic task, the execution of aperiodic tasks may be delayed and their response times are prolonged unnecessarily.

Another approach is to use a dedicated server which handles aperiodic tasks. The server is characterized by an ordered pair (T_s, C_s), where T_s is the period of the server and C_s is the maximum budget. The simplest server is the *Polling Server* (PS). PS is ready for execution periodically at integer multiplies of T_s and is scheduled together with periodic tasks in the system according to the given priority-driven algorithm. Once PS is activated, it executes any pending aperiodic requests within the limit of its budget C_s. If no aperiodic requests are pending, PS immediately suspends its execution until the start of its next period. Since PS is exactly same to a periodic task which has the period T_s and the worst case execution time (WCET) C_s, we can test the schedulability of the system using the traditional RM or EDF schedulability test.

The *Deferrable Server* (DS) [12] was introduced to solve the poor performance of background scheduling and PS. Unlike PS, DS can service an aperiodic request at any time as long as the budget is not exhausted. Though this feature of DS provides better performance than that of PS, a lower priority task could miss its deadline even if the task set seemed to be schedulable by the schedulability test because DS can defer its execution. To solve this problem, the *Sporadic Server* (SS) [10] was proposed. SS ensures that each SS with period T_s and budget C_s never demands more processor time than the periodic task (T_s, C_s) in any time interval. Consequently, we can treat a SS exactly likely the periodic task (T_s, C_s) when we check for the schedulability of the system.

Though there are the modified DS and SS algorithms for EDF scheduling, DS and SS are mainly used for RM scheduling due to the complexity of the modified algorithms. For EDF scheduling, the *Total Bandwidth Server* (TBS) [11] is more suitable. TBS is characterized by U_s which is the utilization of TBS. When an aperiodic task arrives, TBS assigns a deadline to the task such that the utilization of the aperiodic task is equal to U_s. Since TBS assigns the deadline using the WCET of the aperiodic task, there can be overrun when the real execution time is longer than the WCET. (This situation could occur for aperiodic tasks.) Recently, the *Constant Bandwidth Server* (CBS) [1] was proposed to solve the overrun problem of TBS.

A different approach for scheduling aperiodic tasks is the *Slack Stealing* technique [7]. It steals all available slack from periodic tasks and gives it to aperiodic tasks. Though it provides better performance than the server approaches, i.e., minimizes response times of aperiodic requests, its complexity is very high. In addition, since the main idea of the slack stealing is to give as much as possible time to aperiodic tasks executing periodic tasks at full speed, the slack stealing is improper to be integrated with DVS algorithms. So, we concentrate on the server techniques in this paper.

Despite of many researches on aperiodic task scheduling, there have been few studies to adapt the DVS technique to aperiodic task scheduling. A recent work by W. Yuan and K. Nahrstedt [13] proposed a DVS algorithm for soft real-time multimedia and best-effort applications. They handled only the constant bandwidth server. The target of their algorithm is aperiodic task systems, not mixed task systems.

Y. Doh *et al.* [4] also investigated the problem of allocating both energy and utilization for mixed task sets. 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. While their algorithm is an off-line static speed assignment algorithm under the EDF scheduling policy, our work in this paper considers both static and dynamic algorithms under both RM and EDF scheduling policies. Another difference is that we concentrate on minimizing the energy consumption under the constraint on the average response time.

III. Static Scheduling for Mixed Task Sets

Pillai and Shin [8] proposed the static voltage scheduling algorithms using the RM and EDF schedulability tests. Their static scheduling algorithm finds a clock speed of periodic tasks for a hard real-time system. The clock speed is set statically, and is not changed unless the task set is changed. For the mixed task set using a scheduling server such as DS or TBS, Pillai's static scheduling algorithms can also be used with the utilization of the scheduling server. For example, in EDF scheduling using TBS, if the worst case utilization of periodic tasks is 0.3 and the utilization of TBS is 0.4 at 100 MHz clock speed, the static scheduling algorithm determines the clock speed as 70 MHz ($= 100\ \mathrm{MHz} \cdot (0.3 + 0.4)$).

However, the scheduling server for aperiodic tasks generally occupies a large utilization compared with the workload of aperiodic tasks to provide a good responsiveness. If the real utilization of aperiodic tasks is 0.2 rather than 0.4, it is better to use a lower clock speed for periodic tasks and a higher clock speed for aperiodic tasks than 70 MHz. This is because TBS has many idle intervals. However, we cannot use the clock speed 50 MHz ($= 100\ \mathrm{MHz} \cdot (0.3 + 0.2)$) because it can produce deadline misses when the real utilization of aperiodic task is larger than 0.2.

Therefore, in static voltage scheduling, we should consider both the expected workload and the schedulability condition. Our static voltage scheduling 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 scheduling can be formulated as follows:

Static Speed Assignment Problem

Given $U_p, U_s, \omega,$ and ρ,

find S_p and S_s such that

$E = U_p \cdot \omega \cdot S_p^2 + \rho \cdot S_s^2$ is minimized

subject to $\frac{U_p}{S_p} + \frac{U_s}{S_s} \leq U^*$ and $0 \leq S_p, S_s \leq 1$.

U_s	Energy consumption (mJ)			Response time (msec)		
	UNI	OPT	Reduction(%)	UNI	OPT	Reduction(%)
0.15	52.02	50.35	3	1.65	1.38	16
0.20	60.73	54.50	10	1.01	0.75	26
0.25	65.81	58.81	11	0.94	0.75	20
0.30	71.19	63.41	11	0.89	0.75	16
0.35	76.66	72.32	6	0.84	0.75	11
0.40	87.28	77.86	11	0.79	0.75	5

$U_p = 0.4, \omega = 0.75, \rho = 0.15$

TABLE I
STATIC SPEED ASSIGNMENT FOR TOTAL BANDWIDTH SERVER.

, where U_p is the worst case utilization of periodic task set, U_s is the server utilization, ω is the average workload ratio of periodic tasks, and ρ is the average workload ratio of aperiodic tasks ($\rho = \lambda/\mu$). E is a metric reflecting energy consumption[1]. U^*, which is the least upper bound of schedulable utilization, is 1 at the EDF scheduling and $\ln(2)$ at the RM scheduling[2], respectively. Using the Lagrange transform, we can get a following optimal solution for S_p and S_s.

$$S_p = \frac{1}{U^*}\left(U_p + U_s \sqrt[3]{\frac{\rho}{U_s \cdot \omega}}\right), \quad S_s = \frac{1}{U^*}\left(U_p \sqrt[3]{\frac{U_s \cdot \omega}{\rho}} + U_s\right)$$

Under the assumption that we can know the exact ω and ρ values, we can get the optimal static speeds for periodic and aperiodic tasks. Table I shows the experimental results of the optimal static speed assignment. The results show the reduction of energy consumption and response time varying U_s with fixed values of U_p, ω and ρ. Aperiodic tasks are assumed to be serviced by the total bandwidth server. We assumed that if the system is idle it enters into the power-down mode. We compared our optimal speed assignment method (OPT) with Pillai's uniform speed assignment method (UNI) which assigns the same speed to both periodic tasks and aperiodic tasks making the total utilization as U^*. The optimal speed assignment method reduced the energy consumption and the average response time up to 11% and 26%, respectively. Since the scheduling server gets a higher speed than the speed for periodic tasks when $\omega > \rho$, the optimal speed assignment reduces the average response time as well as the energy consumption.

From the result, we can see if a higher U_s is used, the average response time of aperiodic tasks decreases and total energy consumption increases. Since two objectives of the response time and the energy consumption conflict with each other, it is recommended to use the constraint on the response time. We can determine the minimum value of U_s which satisfies the constraint minimizing the energy consumption. For example, if we have the constraint that the average response time should be lower than 1 msec, then we can select 0.2 for the server utilization from the results in Table I.

IV. DYNAMIC SCHEDULING FOR MIXED TASK SETS

There are some problems using existing on-line DVS algorithms such as [9, 2, 8, 5] for mixed task sets. They use three kinds of slack estimation methods [6]: (1) *stretching-to-NTA*, (2) *priority-based slack-stealing*, and (3) *utilization updating*. The *stretching-to-NTA* technique 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 in a mixed task set, 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 arrival 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. Therefore, we cannot use the *stretching-to-NTA* method directly for mixed task sets.

To use the *priority-based slack-stealing* method or the *utilization updating* method, we should be able to identify a slack time due to aperiodic tasks as well as periodic tasks. The slack time of a periodic task can easily be defined as the difference between the WCET and the real execution time of the task. However, for the slack time from aperiodic tasks, we should be concerned about the scheduling server rather than aperiodic tasks because the scheduling server is related with the schedulability condition.

Therefore, we need to modify on-line DVS algorithms to utilize the characteristics of scheduling servers. In this paper, we handle only DS and TBS because they are simple and representative algorithms for the RM scheduling policy and the EDF scheduling policy, respectivly.

A. Deferrable Server

Figure 1(a) shows the task schedule with a deferrable server. There are two periodic tasks, $\tau_1 = (5, 1)$ and $\tau_2 = (8, 2)$, and one DS $= (4, 1)$. Each periodic task and the DS is scheduled by the RM scheduler. The utilization of DS is 0.25 ($= \frac{C_s}{T_s} = \frac{1}{4}$). We assume that periodic tasks have relative deadlines equal to their periods. DS preserves its budget 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 DS is not exhausted (e.g., task a_1). If the budget is exhausted, aperiodic tasks should wait until the next replenishment time. For example, though the task a_4 arrived at the time of 19, it is serviced at the time of 20.

Although we have no clairvoyant power to know the arrival times of aperiodic tasks, the *stretching-to-NTA* method can be used if we utilize the execution behavior of DS. There are two cases the current ready task can be stretched:

- **Rule for aperiodic task**: If there is no periodic task in the ready queue, stretch an aperiodic task to min(next arrival time of a periodic task, next replenishment time).

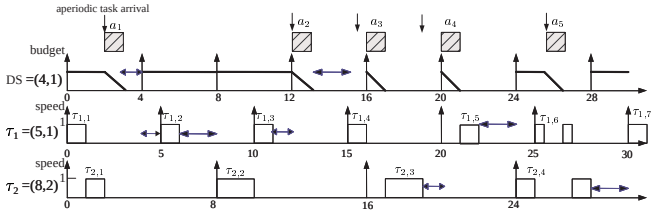
- **Rule for periodic task**: If there is only one periodic task in the ready queue and the budget of DS is 0, stretch a periodic task to min(next arrival time of a periodic task, next replenishment time). This is because the arriving aperiodic task is delayed until the next replenishment time if the budget is 0. If budget > 0, we cannot scale down the

[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.

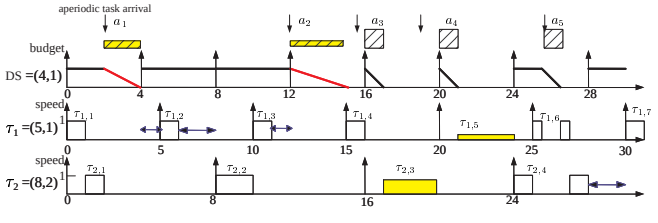
[2] When a deferrable server is used, the utilization bound is 0.6518 [12].

speed of the periodic task even though there is only one periodic task in the ready queue.

Figure 1(b) shows the task schedule using the modified `lppsRM` algorithm [9] which uses the *stretching-to-NTA* method. The aperiodic task a_2 is stretched to the next arrival time of periodic task (15) because there is no periodic task in ready queue. The aperiodic task a_1 is stretched to the next replenishment time (4) because the replenishment time is earlier than the arrival time of a periodic task (5). Though there is no deadline miss even if a_1 is stretched to 5, we limit the stretching bound by the replenishment time to bound the delay of response time of aperiodic tasks. Using this policy, we can guarantee that the maximum increase of the average response time is $T_s - C_s$. The tasks $\tau_{1,5}$ and $\tau_{2,3}$ are stretched to min(next arrival time, next replenishment time) because the remaining budget of DS is 0. We cannot stretch the tasks $\tau_{1,2}$ and $\tau_{1,3}$ because the remaining budget of DS is larger than 0.



(a) No DVS



(b) `lppsRM`

Fig. 1. Task schedules with a deferrable server.

B. Total Bandwidth Server

The total bandwidth server is proposed for the EDF scheduling policy. Figure 2(a) shows the task schedule with a TBS. There are two periodic tasks, $\tau_1 = (8, 2)$ and $\tau_2 = (12, 3)$, and one TBS with $U_s = 0.5$. U_s is the utilization of TBS. If $U_p + U_s \leq 1$, the periodic tasks are schedulable. When an aperiodic task a_k arrives, TBS sets the deadline of a_k to $d_k = max(r_k, d_{k-1}) + C_k/U_s$, where C_k, r_k and d_k is the WCET, the release time, and the deadline of a_k, respectively. For example, when an aperiodic task a_2 with $C_2 = 2$ arrives at 6, TBS sets a_2's deadline to 11 ($= max(6, 7) + 2/0.5$). When a task a_3 arrives at 14, it preempts the task $\tau_{2,2}$ because a_3's deadline is 18 and $\tau_{2,2}$'s deadline is 24.

With TBS, we cannot employ the *stretching-to-NTA* technique used for DS because it is not controlled by the budget. Instead, we can make use of the fact that TBS sustains the utilization of aperiodic tasks as U_s. If we can endure a little degradation of aperiodic tasks, we can delay an aperiodic task a_i until d_{i-1} when $r_i < d_{i-1}$. This delay does not affect the utilization of TBS and does not cause the deadline miss of periodic task. Delaying an aperiodic task until d_{i-1} is identical with assuming the earliest arrival time of the aperiodic task a_i as d_{i-1}. Figure 2(b) shows the task schedule using the modified `lppsEDF` algorithm [9] which uses the *stretching-to-NTA* method. For example, the remaining part of the task $\tau_{2,1}$ at the time of 4 can be stretched to $d_1 = 7$. When an aperiodic task a_2 arrives at the time of 6, it preempts $\tau_{2,1}$ because its priority (i.e., deadline) is higher than $\tau_{2,1}$ but produces no deadline miss. If the priority of $\tau_{2,1}$ is higher than a_2, the start of a_2 will be delayed until 7. In such cases, the maximum delay of an aperiodic task a_i is $d_{i-1} - r_i$.

To use the *priority-based slack-stealing* method for TBS, we should identify the slack times of TBS. There are two types of slack times available when TBS is used:

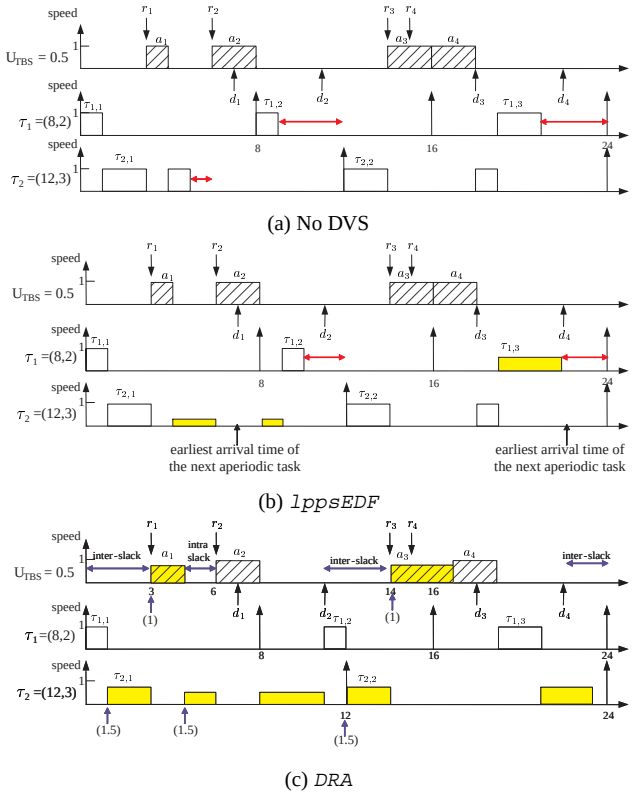
- **Inter-slack**: If an interval $[t_1, t_2]$ in TBS is not overlapped with any active interval of aperiodic tasks $[r_k, d_k]$, there is $(t_2 - t_1) \cdot U_s$ amount of slack time. This is because the total utilization does not exceed 1 even if an aperiodic task with the execution time of $(t_2 - t_1) \cdot U_s$ is executed during the interval $[t_1, t_2]$.
- **Intra-slack**: When an aperiodic task, whose WCET is C_k, consumes only the time of c, there is $(C_k - c)$ amount of slack time.

Figure 2(c) shows the task schedule using the modified `DRA` algorithm [2] which uses the *priority-based slack-stealing*. Originally, in the `DRA` algorithm, when a task τ is to be executed, the slack times due to the early completions of tasks which have the higher priorities than the priority of τ are computed and the speed of τ is determined using the slack times. The modified `DRA` algorithm for TBS uses the same technique except that it considers the inter-slack as well as the intra-slack of TBS.

For example, in Figure 2(c), when a task $\tau_{2,1}$ is scheduled at the time of 1, there is a slack time 1.5 (1 from the early completion of $\tau_{1,1}$ and 0.5 from the inter-slack of TBS during the time interval [0,1]). Using the slack time, the task $\tau_{2,1}$ is scheduled with the speed of 0.67 (=3/(3+1.5)). The aperiodic task a_1 is also scheduled with the speed of 0.67 (=2/(2+1)) exploiting the inter-slack of TBS, 1, during the time interval [2,3]. When the task a_1 is completed consuming only the time of 1.5, the remaining slack 1.5 is transferred to the remaining part of the task $\tau_{2,1}$ lowering its clock speed. Using the modified `DRA` algorithm, we can get a better energy efficiency than that of the modified `lppsEDF` algorithm because `DRA` exploits more slack times. But, the average response time of aperiodic tasks is longer in `DRA` than `lppsEDF`.

V. Experimental Results

We have evaluated the performance of our DVS algorithms for DS and TBS using simulations. In each experiment, we first assigned the static speed to periodic tasks and aperiodic tasks using the static speed assignment algorithm described in Section III. During run time, the operating speed is further reduced by on-line DVS algorithms exploiting the slack times. The execution time of each periodic task instance was

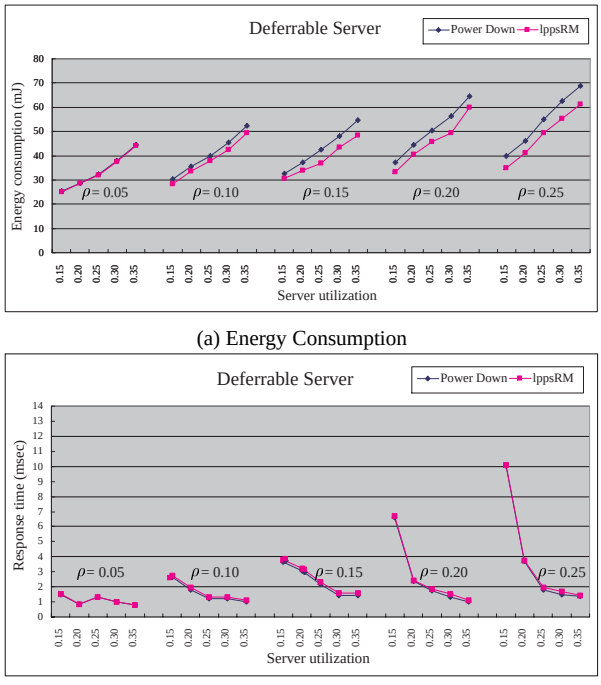


(a) No DVS

(b) `lppsEDF`

(c) `DRA`

Fig. 2. Task schedules with a total bandwidth server.



(a) Energy Consumption

(b) Response Time

Fig. 3. Experimental results using a deferrable server.

randomly drawn from a Gaussian distribution in the range of [BCET, WCET] where BCET is the best case execution time. In the experiments, BCET is assumed to be 10% of WCET.

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.

Varying the server utilization U_s and the workload of aperiodic tasks ρ under a fixed utilization U_p of periodic tasks, we observed the interactions between the energy consumption of the total system and the average response time of aperiodic 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 C_s value and ρ is controlled by a varying λ with a fixed μ value.)

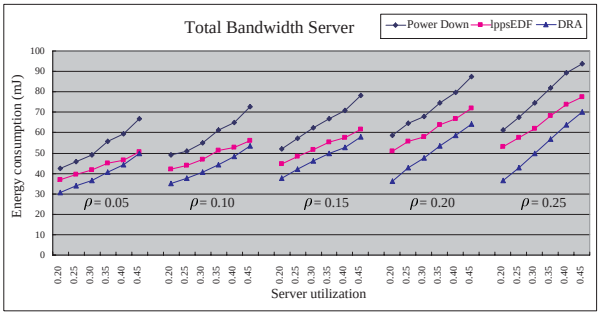
Figure 3(a) compares the energy consumption of the modified `lppsRM` algorithm over that of the power-down method when a deferrable server is used. In this experiment, U_p was fixed to 0.3. As U_s and ρ increase, the energy consumption also increases as with the results of the static speed assignment. The energy saving from the modified `lppsRM` algorithm increases as ρ increases. This is because there are more chances for DS to have the zero budget when the workload of aperiodic tasks is large. When ρ is 0.25, `lppsRM` reduces the energy consumption by 12% over the power-down method.

Figure 3(b) shows how the average response times of aperiodic tasks change. As U_s increases, the response time decreases, converging on the average response time of M/M/1 because the number of interferences by periodic tasks is reduced. As shown in Figure 3(b), the modified `lppsRM` algorithm does not significantly increase the response time. This is because there are few cases when tasks can be stretched (as we can see from Figure 3(a)) and the delay due to the stretching is smaller than $T_s - C_s$.

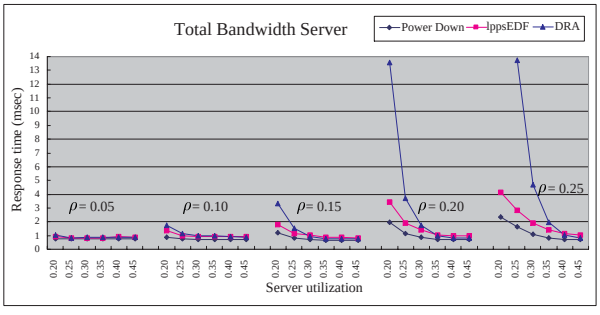
For TBS, we observed the performance of the modified `lppsEDF` algorithm and the modified `DRA` algorithm with U_p set to 0.4. Figure 4(a) shows the energy consumption by each algorithm. `DRA` consumes less energy than `lppsEDF`. As ρ increases, the energy reduction patterns of `DRA` and `lppsEDF` (over the power-down method) do not change significantly. Since the number of cases when `lppsEDF` can stretch the execution time of periodic task is determined by the workload of periodic tasks rather than ρ, the relative energy savings from `lppsEDF` are similar irrespective of ρ. However, the energy savings from `lppsEDF` increase as U_s increases. This is because, in the static speed assignment, a higher speed is assigned to the aperiodic tasks when a large U_s is used, thus the energy saving from aperiodic tasks increases.

The relative energy savings from `DRA` also show little variations depending on ρ because `DRA` utilizes the inter-slack as well as the intra-slack. Unlike `lppsEDF`, the energy savings from `DRA` are not different depending on U_s. Since `DRA` identifies slack times more aggressively during run time, its energy efficiency is less dependent on the static speed assignment. When ρ is 0.25, `lppsEDF` and `DRA` reduce the energy consumption by 16% and 32% on average over the power-down method, respectively.

Figure 4(b) shows the average response times of aperiodic tasks in TBS. Both `lppsEDF` and `DRA` have longer response



(a) Energy Consumption



(b) Response Time

Fig. 4. Experimental results using a total bandwidth server.

times than the power-down technique because their operating speed is lower. As U_s decreases and ρ increases, the response times are further increased. Since the higher U_s is, the nearer the deadline to aperiodic tasks in TBS, the delay times of aperiodic tasks by `lppsEDF` are inversely proportional to U_s.

When U_s is high, the average response time of `DRA` is better than that of `lppsEDF` because `lppsEDF` may delay the execution of the aperiodic task a_i by d_{i-1} when a_i arrives before d_{i-1} while `DRA` does not delay the start time of aperiodic task. However, `DRA` increases the response times very quickly when ρ is close to U_s. For example, when $\rho = 0.2$ and $U_s = 0.25$, the response time increases to 13.5 msec. In this case, `DRA` stretches each task to complete its execution near its deadline because `DRA` can exploit most available slack times. As shown in Figure 4(b), `DRA` becomes unusable when $\rho \geq U_s$ although its energy efficiency is high. When $\rho = 0.25$ and $U_s = 0.2$, the response time is 44.8 msec.

From the results in Figure 4, we can observe that the on-line DVS algorithm and the server utilization should be carefully selected to satisfy the response time requirement. For example, assume that ρ is 0.2 and TBS is used for aperiodic task scheduling. If the average response time should be less than 4 msec, `DRA` with the server utilization 0.25 is the best choice because it minimizes the energy consumption while satisfying the response time constraint.

VI. Conclusions

We have proposed DVS algorithms for mixed task systems, which have both periodic and aperiodic tasks. For the static voltage scheduling algorithm, we proposed the optimal speed assignment algorithm considering the workload of aperiodic tasks. For the dynamic voltage scheduling algorithms, we presented the slack identification methods for the servers dedicated to aperiodic tasks. Existing on-line DVS algorithms, which cannot be used for mixed task systems, were modified to use the proposed slack identification methods. The modified DVS algorithms reduced the energy consumption by 12% and 32% under the RM scheduling policy and the EDF scheduling policy, respectively.

Our work in this paper can be extended in several directions. For example, we plan to develop DVS algorithms for other scheduling servers such as *Sporadic Server* and *Constant Bandwidth Server*. Furthermore, although we focused on the slack identification based on the characteristics of server algorithms, it will be an interesting future work to study the effect of the slack distribution methods on the average response time.

References

[1] L. Abeni and G. Buttazzo. Integrating Multimedia Applications in Hard Real-Time Systems. In *Proc. of IEEE Real-Time Systems Symp.*, pages 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.*, pages 95–106, 2001.

[3] T. Burd, T. Pering, A. Stratakos, and R. Brodersen. A Dynamic Voltage Scaled Microprocessor System. In *Proc. of IEEE Int. Solid-State Circuits Conf.*, pages 294–295, 2000.

[4] 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*, pages 533–550, 2003.

[5] 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*, pages 788–794, 2002.

[6] 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.*, pages 219–228, 2002.

[7] 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.*, pages 110–123, 1992.

[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*, pages 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.*, pages 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] M. Spuri and G. Buttazzo. Scheduling Aperiodic Tasks in Dynamic Priority Systems. *Journal of Real-Time Systems*, 10(2):179–210, 1996.

[12] J. K. Strosnider, J. P. Lehoczky, and L. Sha. The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments. *IEEE Transactions on Computers*, 44(1):73–91, 1995.

[13] 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*, pages 105–114, 2002.