Power-Aware Scheduling of Mixed Task Sets in Priority-Driven Systems*

Dongkun Shin and Jihong Kim

School of Computer Science and Engineering
Seoul National University, Korea

Abstract. We propose power-aware on-line task scheduling algorithms for mixed task sets which consist of both periodic and aperiodic tasks. The proposed algorithms utilize the execution behaviors of scheduling servers for aperiodic tasks. Since there is a trade-off between the energy consumption and the response time of aperiodic tasks, the proposed algorithms focus on bounding the response time degradation of aperiodic tasks while they use a more aggressive slack estimation technique for higher energy savings in mixed task sets. We also propose a new slack distribution method which gives better response times with slight energy increases. Experimental results show that the proposed algorithms reduce the energy consumption by 25% and 18% over the non-DVS scheme under the RM scheduling and the EDF scheduling, respectively.

1 Introduction

Many practical real-time applications require aperiodic tasks as well as periodic tasks. Generally, periodic tasks are time-driven with hard deadlines and aperiodic tasks are event-driven (i.e., activated at *arbitrary* times) with short response times. In this paper, we call a system with both periodic and aperiodic tasks as a *mixed* task system.

In implementing mixed task systems, there are two major design objectives. The first objective is to maintain the schedulability of (feasible) periodic tasks under the presence of aperiodic tasks. That is, aperiodic tasks should not prevent periodic tasks from completing before their deadlines. The second objective is to serve aperiodic tasks with reasonable average response times. To satisfy these two design objectives, many scheduling algorithms had been proposed, which are called *bandwidth-preserving servers* [1] because they set aside some portion of the system utilization for aperiodic tasks. In this paper, we consider as the third design parameter the energy consumption of mixed task sets. With the added energy consumption requirement, our overall design objective is to minimize the total energy consumption of *both* periodic tasks and aperiodic tasks while satisfying the previous two requirements.

* This work was supported by grant No. R01-2001-00360 from the Korea Science & Engineering Foundation and University IT Research Center Project.

L.T. Yang et al. (Eds.): EUC 2004, LNCS 3207, pp. 227–237, 2004.

Among many low-power design approaches, we focus on dynamic voltage scaling (DVS) [2] in this paper. Many on-line voltage scheduling algorithms exist for hard real-time systems [3,4,5,6]. Since most of these algorithms assume that the system consists of periodic hard real-time tasks only and the task release times are known *a priori*, they estimate slack times based on the known release times and stretch the task execution using the estimated slack times. Generally, the more slack times a DVS algorithm can estimate, the better energy efficiency the DVS algorithm can have.

However, when we apply a DVS algorithm to mixed task sets, the DVS algorithm should tackle the arbitrary behaviors of aperiodic tasks. Fortunately, since a bandwidth-preserving server limits the execution of aperiodic tasks within its allocated bandwidth, DVS algorithms can estimate slack times considering the characteristics of bandwidth-preserving servers. Though the energy efficiency of a DVS algorithm for mixed task sets is also related to how much slack times it can find, the presence of aperiodic tasks in the mixed task sets raises the trade-off between the energy consumption of the total system and the response time of aperiodic tasks. If we ignore the response time of aperiodic tasks, the most energy-efficient solution is not to serve aperiodic tasks, thus further reducing the energy consumption of periodic tasks. However, it is obvious such a solution will not be acceptable. Therefore, the main challenge in designing DVS algorithms for the mixed task sets is to bound the response times of aperiodic tasks while reducing the energy consumption of periodic tasks as well as aperiodic tasks.

In this paper, we first propose new DVS algorithms for deferrable server [7] and sporadic server [8], which achieve higher energy reductions than the existing DVS algorithms in [9]. Our DVS algorithms can reduce the energy consumption by 18∼29% over the existing algorithms. Second, we also propose the DVS algorithm for the constant bandwidth server [10] which is not handled in [9]. Lastly, we also propose a new slack distribution method for DVS algorithms. By assigning slack times only to periodic tasks and executing aperiodic tasks at the full speed, we can get better response times with small increases in the energy consumption.

The rest of this paper is organized as follows. In Section 2, we compare the related works for mixed task sets with our algorithms. We formulate the DVS scheduling problem for a mixed task set in Section 3. While the on-line DVS algorithms under fixed-priority systems are presented in Section 4, the algorithms under dynamic-priority systems are described in Section 5. In Section 6, the experimental results are discussed. Section 7 concludes with a summary.

2 Related Works

Recently, several researchers have proposed DVS algorithms for mixed task sets. W. Yuan *et al.* [11] proposed DVS algorithms for another kind of mixed task sets which consist of sporadic tasks and aperiodic tasks. The sporadic tasks[1] arrive

[1] Authors of [11] say that their target system is a mix of soft real-time (SRT) multimedia and best-effort applications. However, their definition for SRT tasks is same to sporadic tasks.

at arbitrary times and have soft deadlines. They handled only the constant bandwidth server.

Y. Doh *et al.* [12] investigated the problem of allocating both energy and utilization for mixed task sets which consist of periodic tasks and aperiodic tasks. They used the total bandwidth server and considered the static (off-line) 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 Y. Doh *et al.*'s algorithm is an off-line static speed assignment algorithm under the EDF scheduling policy, we propose on-line algorithms. Another difference is that we concentrate on minimizing the energy consumption under the constraint on the average response time.

The DVS problem tackled by D. Shin *et al.* [9] has the same problem formulation as one used in this paper. We improve the energy performance of the DVS algorithm using a more aggressive slack estimation method and propose a new slack distribution method.

3 Problem Formulation

We assume that a mixed task system $\mathcal{T}$ consists of n periodic tasks, $\tau_1, \cdots, \tau_n$, and an aperiodic task, σ. The aperiodic task σ is serviced by a scheduling server S. The scheduling server S is characterized by an ordered pair (Q_s, T_s). 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 cycles (WCEC) 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 operating speeds are the values between 0 and 1.

If the response time of σ_k is $t(\sigma_k)$ in the non-DVS scheme, the response time will be increased to $t(\sigma_k) + D(\sigma_k)$ by a DVS algorithm because the operating speeds of both periodic tasks and aperiodic tasks are changed. We call the increase $D(\sigma_k)$ in the response time as the *response time delay*. In this paper, we propose the DVS algorithms which minimizes the energy consumption satisfying the timing requirements of all periodic tasks while guaranteeing $D(\sigma_k) \leq T_s - Q_s$ for all σ_k.

Existing on-line DVS algorithms such as [3,4,5] are not directly applicable for the problem. As discussed in [2], most existing heuristics are based on three techniques: (1) *stretching-to-NTA*, (2) *priority-based slack-stealing*, and (3) *utilization updating*. For example, consider the *stretching-to-NTA* technique. The technique stretches the execution time of the periodic task ready for execution to the next task arrival time (NTA) of a periodic task when there is no an-

other 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 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. Consequently, the *stretching-to-NTA* technique should assign the full speed to all tasks in the mixed task system.

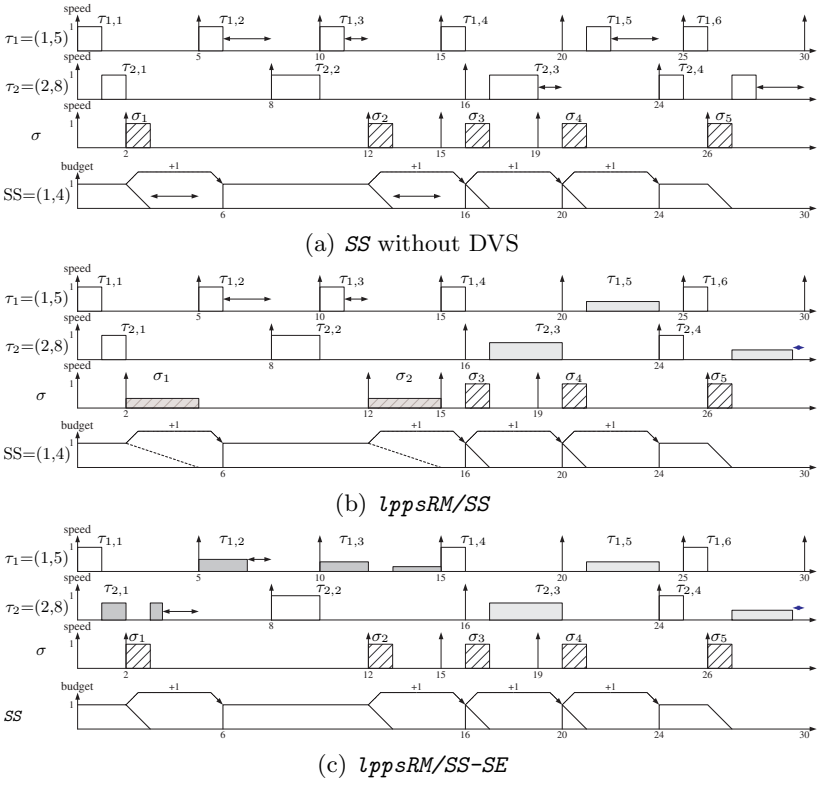
To use the *priority-based slack-stealing* method or the *utilization updating* method, we should be able to identify a slack time due to aperiodic tasks as well as periodic tasks. The slack time of a periodic task can easily be defined as the difference between the WCET and the real execution time of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

4 Scheduling Algorithms in Fixed-Priority Systems

For fixed-priority systems, we assume the RM scheduling policy. In [9], ***lppsRM/DS*** algorithm has been proposed to integrated the deferrable server with the DVS algorithm ***lppsRM*** [3]. If there is no periodic task in the ready queue, ***lppsRM/DS*** executes an aperiodic task at the speed of $max(1, q_s/(min(NTA, R) - t))$ where NTA, R and t are the next arrival time of a periodic task, the next replenishment time of DS and the current time (the start time of the aperiodic task), respectively. If there is only one periodic task in the ready queue and the remaining budget q_s is 0, the algorithm stretches the periodic task to $min(NTA, R)$.

We applied the idea of ***lppsRM/DS*** algorithm to the sporadic server. Figure 1(a) shows the task schedule using a sporadic server SS. There are two periodic tasks, $\tau_1 = (1, 5)$ and $\tau_2 = (2, 8)$, and one SS $= (1, 4)$. All periodic tasks and the SS are scheduled by the RM scheduler. The utilization of SS is 0.25 $(= \frac{Q_s}{T_s} = \frac{1}{4})$. 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 $t_2 - t_1$ until 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. Figure 1(b) shows the task schedule using the ***lppsRM/SS*** algorithm which is the modified version of ***lppsRM*** for SS.

Though we can reduce the energy consumption by ***lppsRM/SS*** algorithm, the algorithm can show poor performance when the workload of aperiodic tasks is



(a) *SS* without DVS

(b) *lppsRM/SS*

(c) *lppsRM/SS-SE*

Fig. 1. Task schedules with a sporadic server.

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 scheduling 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 limit the response time delay. Under this policy, we can guarantee that $D(\sigma_k) \leq T_s - Q_s$ for all σ_k because σ_k is not delayed above the replenishment time R. The detail proof is provided in [13].

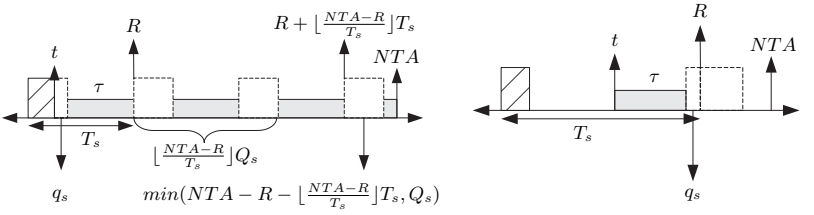


Fig. 2. Bandwidth-based slack stealing in *lppsRM/SS-SE*.

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

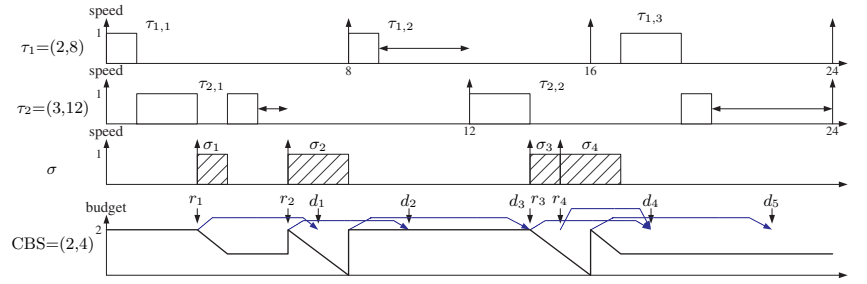
$$
\begin{aligned}
if\ (T_\tau > T_s) \quad MAT &= NTA - t - q_s - \lfloor \frac{NTA - R}{T_s} \rfloor Q_s \\
&\quad - min(NTA - R - \lfloor \frac{NTA - R}{T_s} \rfloor T_s, Q_s) \\
if\ (T_\tau < T_s) \quad MAT &= min(R, NTA) - t - q_s
\end{aligned}
$$

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.

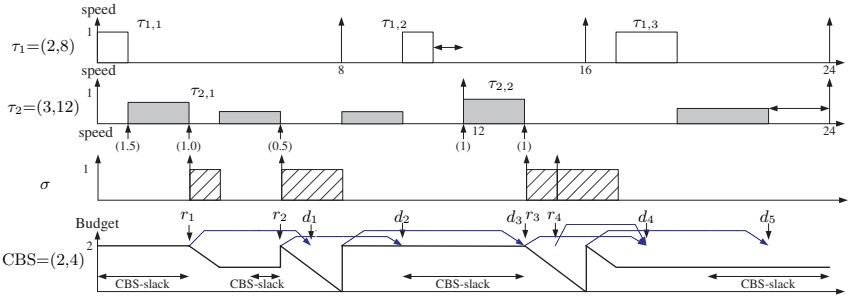
5 Scheduling Algorithms in Dynamic-Priority Systems

For dynamic-priority systems, we assume the EDF scheduling policy. We propose the slack estimation algorithm for constant bandwidth server (CBS). Figure 3(a) shows the task schedule using a CBS, assuming two periodic tasks, $\tau_1 = (2, 8)$ and $\tau_2 = (3, 12)$, and one CBS =(2,4). The maximum utilization of CBS (U_s) is 0.5(= 2/4). If $U_p + U_s \leq 1$, where U_p is the maximum utilization of periodic tasks, the task set is schedulable.

At each instant, a CBS deadline d_k is associated with CBS. At the beginning $d_0 = 0$. Each served aperiodic task σ_i is assigned a dynamic deadline equal to



(a) *CBS* without DVS



(b) *DRA/CBS*

Fig. 3. Task schedules with a constant bandwidth server.

the current server deadline d_k. Whenever a served task executes, the budget q_s is decreased by the same amount. When $q_s = 0$, the server budget is replenished to the maximum value Q_s and a new server deadline is generated as $d_{k+1} = d_k + T_s$. A CBS is said to be active at time t if there are pending jobs; that is, if there exists a served task σ_i such that $r(\sigma_i) \leq t < e(\sigma_i)$, where $r(\sigma_i)$ and $e(\sigma_i)$ are the arrival time and the completion time of the task σ_i. A CBS is said to be idle at time t if it is not active. When a task σ_i arrives and the server is active, the request is enqueued in a queue of pending jobs according to a given (arbitrary) non-preemptive discipline (e.g., FIFO).

When an aperiodic task σ_i arrives at $r(\sigma_i)$ and the server is idle (when CBS does not service aperiodic tasks), if $q_s \geq (d_k - r(\sigma_i))U_s$ the server generates a new deadline $d_{k+1} = r(\sigma_i) + T_s$ and q_s is replenished to the maximum value Q_s, otherwise the task is served with the last server deadline d_k using the current budget. When a job finishes, the next pending job, if any, is served using the current budget and deadline. If there are no pending jobs, the server becomes idle. At any instant, a job is assigned the last deadline generated by the server.

For example, when an aperiodic task σ_1 arrives at time 3, CBS sets its deadline d_1 to 7 ($= r(\sigma_1) + T_s = 3 + 4$) and σ_1 uses the deadline. When an aperiodic task σ_2 arrives at time 6, CBS sets σ_2's deadline to 10 ($= r(\sigma_2) + T_s = 6 + 4$) and q_s is replenished to 2 because $q_s = 1$ is greater than $(d_1 - r(\sigma_2))U_s = (7-6)0.5 = 0.5$. When a task σ_3 arrives at 14, CBS sets σ_3's deadline to 18 and σ_3 preempts

the task $\tau_{2,2}$. When an aperiodic task σ_4 arrives at 15, CBS sets σ_4's deadline to 18 ($= d_4$) because $q_s = 1$ is smaller than $(d_4 - r(\sigma_4))U_s = (18-15)0.5 = 1.5$. When $q_s = 0$ at time 16, CBS changes σ_4's deadline to a new deadline $d_5 = d_4 + T_s = 22$ and q_s is replenished to 2. In this manner, CBS maintains its bandwidth under U_s.

To use the *priority-based slack-stealing* [2] method for CBS, we should identify the slack times of CBS. We can estimate the slack time using the *workload-based slack-estimation* method. When the workload of CBS is lower than U_s, we can identify slack times.

Figure 4 shows the *workload-based slack-estimation* algorithm for CBS. The algorithm uses four variables, *release*, C_{slack}, C_{idle} and C_{active}. The *release* is a flag variable to know whether an aperiodic task is released. The C_{active} contains the number of execution cycles of the completed aperiodic tasks. When an aperiodic task is completed, C_{idle}, which is the number of idle cycles required to make the workload of CBS to be same to U_s, is calculated. During the idle period, the C_{idle} is decreased. When C_{idle} becomes to 0, the workload of CBS is equal to U_s. If the idle interval of CBS continues, the workload of CBS becomes to be smaller than U_s and C_{slack} is increased. The C_{slack} can be used for periodic tasks to stretch the execution time.

Initiation:
  release=F; $C_{slack} = 0$; $C_{idle} = 0$; $C_{active} = 0$;

upon aperiodic_task_release:
  release = T;

upon aperiodic_task_completion:
  C_{idle} += $C_{active} \cdot (1 - U_s)$ / U_s;
  release = F; $C_{active} = 0$;

during aperiodic_task_execution(t):
  increase C_{active} by t;

during CBS_idle(t):
  if (*release*==F and $C_{idle} == 0$) increase C_{slack} by $t \cdot U_s$;
  else decrease C_{idle} by t;

Fig. 4. Workload-based slack estimation in CBS.

Figure 3(b) shows the task schedule using the ***DRA/CBS*** algorithm which is modified from the ***DRA*** algorithm [4]. In Figure 3(b), the time intervals, where $C_{slack} > 0$, are marked with arrow lines. For example, when a task $\tau_{2,1}$ is scheduled at time 1, there is a slack time 1.5 (1 from the early completion of $\tau_{1,1}$ and 0.5 from CBS 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)). When the task $\tau_{2,1}$ is preempted at time 3, the slack time 1.0 from CBS is transferred to the remaining part of $\tau_{2,1}$. The guarantee of $D(\sigma_k) \leq T_s - Q_s$ for all σ_k is proved in [13].

6 Experimental Results

We have evaluated the performance of our DVS algorithms for scheduling servers 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. 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.

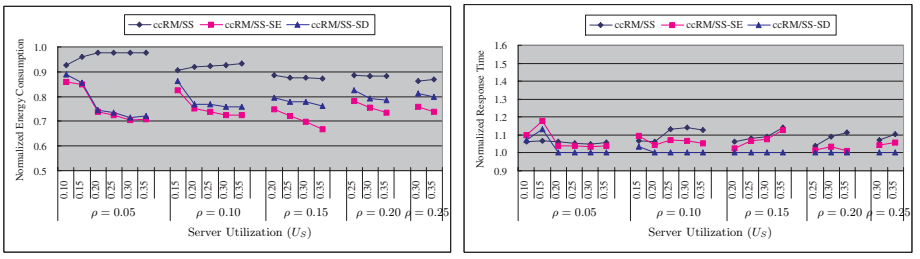
Varying the server utilization U_s and the workload of aperiodic tasks ρ under a fixed utilization U_p of periodic tasks, we observed the energy consumption of the total system and the average response time of aperiodic tasks. We present only the experimental results where U_s is controlled by changing 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 $U_p = 0.3$ and four tasks with $U_p = 0.4$ in the experiments of fixed-priority systems and dynamic-priority systems, respectively. For all experiments including the non-DVS scheme, both periodic tasks and aperiodic tasks were given an initial clock speed $s_0 = (U_p + U_s)s_m/U_m$, where s_m is the maximum clock speed and U_m is the upper bound of the schedulable utilization (1 in the EDF policy and $n(2^{1/n} - 1)$ for n tasks in the RM policy). During run time, the speed is further reduced by on-line DVS algorithms exploiting the slack times.

Figure 5(a) shows the energy consumptions of the *ccRM/SS* algorithm and the *ccRM/SS-SE* algorithm normalized by that of the power-down method. 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 s_0. *ccRM/SS-SD* is good for a better response time.

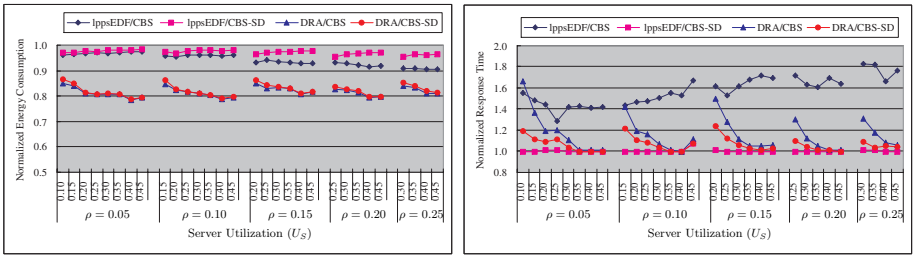
The difference between the energy savings of *ccRM/SS* and *ccRM/SS-SE* decreases as ρ increases. This is because there are more chances for SS to have the zero budget when ρ is large. As U_s increases, *ccRM/SS-SE* shows a larger energy saving compared with *ccRM/SS* because *ccRM/SS-SE* performs well in the low aperiodic workload (over U_s). The *ccRM/SS* and *ccRM/SS-SE* reduced the energy consumption on average by 9% and 25% over the power-down method, respectively. The *ccRM/SS-SE* reduced the energy consumption on average by 18% over *ccRM/SS*.

As shown in Figure 5(b), *ccRM/SS* and *ccRM/SS-SE* increase the response time on average by 8% and 6% over the power-down method, respectively. Due to the side effect on aperiodic tasks explained at Section 4, *ccRM/SS-SE* shows better average response times. *ccRM/SS-SD* shows almost the same response time



(a) Energy Consumption (b) Response Time

Fig. 5. Experimental results using sporadic servers.



(a) Energy Consumption (b) Response Time

Fig. 6. Experimental results using constant bandwidth servers.

to that of power-down method because the execution speed of aperiodic task is always s_0. However, it shows better energy performances than *ccRM/SS*.

For CBS, we observed the performances of *lppsEDF/CBS*, *lppsEDF/CBS-SD*, *DRA/CBS* and *DRA/CBS-SD*. *lppsEDF/CBS-SD* and *DRA/CBS-SD* assigns all aperiodic tasks the initial clock speed s_0. Figure 6(a) shows the energy consumption by each algorithm normalized by that of power-down method. The energy reductions are not significantly changed as ρ changes. This is because *DRA/CBS* does not utilize the zero budget of server as *ccRM/SS*. The average energy reductions by *DRA/CBS* and *DRA/CBS-SD* are 18%. Since most of slack times are generated by CBS and used by periodic tasks, *DRA/CBS* and *DRA/CBS-SD* show similar energy performances.

DRA/CBS increased the average response time on average by 16%. As U_s decreases (T_s increases)[2], the response time increases because the maximum response time delay is $T_s - Q_s$. However, the response time delay of aperiodic task is still smaller than $T_s - Q_s$. Since *DRA/CBS-SD* is similar to *DRA/CBS* in energy performances despite of its good response times, we can know that it is better to give slack times only to periodic tasks when the short response times are required.

[2] Note that we varied T_s to change U_s.

7 Conclusions

We have proposed the on-line DVS algorithms for mixed task systems. Considering the trade-off between the energy consumption and the response time, we modified the existing on-line DVS algorithms for periodic task sets to utilize the execution behaviors of various bandwidth-preserving servers. The proposed algorithms guarantee that the response time delay is no greater than $T_s - Q_s$. By using a more aggressive slack estimation method than the existing algorithms for the mixed task sets, the proposed algorithms reduced the energy consumption by 18% over the existing algorithm. We also proposed a new slack distribution method which provides better response times with slight energy overheads.

References

1. J. W. S. Liu. *Real-Time Systems*. Prentice Hall, 2000.
2. 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.
3. 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.
4. 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. D. Shin and J. Kim. Dynamic Voltage Scaling of Periodic and Aperiodic Tasks in Priority-Driven Systems. In *Proc. of Asia and South Pacific Design Automation Conf.*, 2004.
10. 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.
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*, pages 105–114, 2002.
12. 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.
13. D. Shin and J. Kim. Dynamic Voltage Scaling for Mixed Task Systems in Priority-Driven Systems. Technical report, Computer Architecture and Embedded Systems Laboratory, Seoul National University, 2004.