Energy Efficient Scheduling for Real-Time Systems with Mixed Workload[1]

Jheng-Ming Chen, Kuochen Wang and Ming-Ham Lin
Department of Computer Science
National Chiao Tung University
Hsinchu 300, Taiwan
jmchen@cs.nctu.edu.tw kwang@cs.nctu.edu.tw travelaman958@cs.95g.nctu.edu.tw

Abstract. In spite of numerous inter-task dynamic voltage scaling (DVS) algorithms of real-time systems with either periodic tasks or aperiodic tasks, few of them were aimed at the mixed workload of both kind of tasks. A DVS algorithm for mixed workload real-time systems should not only focus on energy saving, but also should consider low response time of aperiodic tasks. In this paper, we develop an on-line energy efficient scheduling, called *Slack Stealing for DVS* (SS-DVS), to reduce CPU energy consumption for mixed workload real-time systems under the earliest deadline first (EDF) scheduling policy. The SS-DVS is based on the concept of slack stealing to serve aperiodic tasks and to save energy by using the dynamic reclaiming algorithm (DRA). Unlike other existing approaches, the SS-DVS does not need to know the workload and the worst case execution time of aperiodic tasks in advance. Experimental results show that the proposed SS-DVS obtains better energy reduction (17% ~ 22%) while maintaining the same response time compared to existing approaches.

Keywords: mixed workload real-time system, inter-task dynamic voltage scaling, slack time, actual workload, worst case-execution time.

1 Introduction

In order to conserve energy for battery-powered real-time systems, some techniques were proposed in the past. Such as shutting down systems parts while they are not in use is one of the techniques for portable devices. However, restarting the hardware takes time and increases the response time. It's not effortless to determine when and which device should be shut down and woken up [8]. Another approach, called dynamic voltage scaling (DVS), to conserve power is by scaling down the processor voltage and frequency when some unused idle periods exist in the schedule at run time. The voltage scheduler determines which voltage to use by analyzing the state of the system. That is, the voltage scheduler of the real-time system supplies the lowest possible level voltage without affecting the system performance. Several

[1] This work was supported by the NCTU EECS-MediaTek Research Center under Grant Q583 and the National Science Council under Grant NSC96-2219-E-009-012.

commercially available processors provide the DVS feature, including Intel Xscale [11] and Xeon [12], Transmeta Crusoe [13], AMD Mobile Athlon [14], and IBM PowerPC 405LP [15].

It is known that the energy consumption E of a CMOS circuit is dominated by its dynamic supply voltage and is proportional to the square of its supply voltage, which is defined as $E = C_{eff} \cdot V_{dd}^2 \cdot C$ [10], where C_{eff} is the effective switched capacitance, V_{dd} is the supply voltage, and C is the number of execution cycles. Degrading the supply voltage also drops the maximum operating frequency proportionally ($V_{dd} \propto f$). Thus E could be approximated as being proportional to the operating frequency squared ($E \propto f^2$). Therefore, lowering operating frequency and according supply voltage is an effective technique for reducing energy consumption. However, reduction of the operating frequency leads to long service time. For this reason, applying DVS algorithms for real-time tasks to reduce energy consumption should still meet all requirements of real-time systems. For hard real-time tasks, DVS algorithms which lower operating frequency have to ensure no task missing its deadline. In the same way, DVS algorithms have to ensure reasonable response time of soft real-time tasks while reducing the operating frequency.

Despite several obvious advantages by using DVS algorithms, it also causes more preemptions which increase energy consumption in memory subsystems, and extra energy consumption and extra time for voltage transitions. However, these overheads have been generally ignored because the overhead can be included into the worst case execution time (WCET) of a task [3] [16]. Thus, a DVS algorithm can be used without modifications for real variable-voltage processors [3]. Additionally, [17] provides a technique that takes the task preemption into account while adjusting the supply voltage using the delayed preemption technique. The DVS algorithms have been proposed in growing numbers to minimize energy consumption in the past decade. In [9], it classifies existing DVS algorithms for real-time systems into two categories. One is *intra-task DVS* algorithms, which uses the slack time when a task is predicted to complete before its WCET. The other is *inter-task DVS* algorithms, which allocates the slack time between the current task and the following tasks. The basic difference between them is that intra-task algorithms adjust the supply voltage during an individual task boundary, while inter-task algorithms adjust the supply voltage task by task.

In this paper, we consider inter-task DVS scheduling. Most of the existing inter-task DVS algorithms were targeted at periodic tasks, and could get all tasks information in advance including arrival time, deadline, and WCET at the maximum processor speed. However, practical real-time applications involve both periodic and aperiodic tasks. For instance, in multimedia applications such as MPEG players, some tasks such as decoding frames periodically have stringent periodic performance, and some aperiodic user requests (e.g., volume control) should be with reasonable response times [3]. Periodic tasks are time driven with absolute hard deadlines, in general, and aperiodic tasks are event driven with soft deadline. Moreover, a portion of aperiodic tasks could not know the actual workload in advance [7].

2 Related work

2.1 Aperiodic Real-Time Task Scheduling Schemes

Two schemes to serve aperiodic and periodic tasks in real-time systems, which were used in the proposed approach, are described:

(1). Bandwidth Preserving Server [18][19][20][21]: It is similar to the polling server. The concept of the bandwidth preserving server is creating a server with execution budget which is a time amount for executing aperiodic tasks. It is also characterized by an ordered pair (Q_s, T_s). Q_s/T_s is the server utilization. The difference from the polling serve is that it serves aperiodic tasks anytime while the budget isn't zero. It will execute aperiodic tasks according to the budget. If the budget is exhausted, it will stop or delay serving the aperiodic tasks.

(2). Slack Stealing [5]: The slack stealing is executing aperiodic tasks by using the available slack times of periodic tasks. If there is available slack time from periodic tasks, aperiodic tasks could be serviced first without causing any deadline miss of period tasks. In this scheme, the response time of aperiodic tasks is the lowest, but the complexity of such a real-time system is the highest among the possible approaches.

2.2 On-Line Inter-Task DVS Strategies for Period Tasks

Two on-line inter-task DVS strategies for periodic tasks in real-time systems, which were used in the proposed approach, are depicted [9]:

(1). Stretching-to-NTA [16]: This strategy is based on that the scheduler already knows the next task arrival time (NTA) of periodic tasks. The scheduler will stretch the execution time to the NTA, if it doesn't cause deadline miss in this way. Therefore, the operating frequency and supply voltage can be decreased.

(2). Priority-Based Slack Stealing [2]: Because not all the execution time of tasks are in the worst cases, the slack time remains on the schedule if high priority tasks complete earlier than their WCETs. Consequently, the allowed execution time of low priority tasks can be extended.

Note that most DVS algorithms used these strategies for real-time systems with periodic tasks only, and directly using these algorithms for mixed workload real-time systems is not appropriate. If we directly use the stretching to NTA for mixed workload real-time systems, it is hard to know the next arrival time of an aperiodic task. As a result, there will be deadline miss of hard real-time periodic tasks when high priority aperiodic tasks abruptly arrive during the stretching period.

In the priority-based slack stealing strategy [2] and the utilization updating [22] strategy, although getting the slack time of a periodic task is easy, it's not easy to decide the slack time of an aperiodic task. Especially, we even don't know the arrival time and the WCET of each aperiodic task ahead. If utilizing the slack time from an aperiodic task is too aggressive or the actual workload of aperiodic tasks is higher than the predicted processor utilization, the deadline miss will occur just like that

occurs in the stretching to NTA. Therefore, for mixed workload real-time systems, we should adapt these strategies to satisfy the timing constraints of periodic tasks and the short response time requirement of aperiodic tasks. That is, we need to modify the on-line DVS algorithms for periodic tasks and integrated them with the previous mentioned aperiodic real-time tasks scheduling schemes.

2.3 Dynamic Reclaiming Algorithm

The Dynamic Reclaiming Algorithm (DRA) [2] is a kind of priority-based slack stealing technique, which is based on detecting early completions and adjusting the speeds of periodic tasks in order to provide additional power saving while still meeting the deadlines of periodic tasks. The DRA is the basis of most existing DVS algorithms for mixed workload real time systems. First of all, the optimal constant speed $\overline{S}$ could be calculated [2][22], at which speed every instance completes its worst case before the deadline. However, if the task finishes earlier than the worst case, the amount of remaining CPU time the dispatched task can safely use to reduce its speed. The DRA keeps and updates a data structure, α-queue, to calculate the earliness belonging to the executing tasks at run time. The information of the α-queue includes the identity, arrival time, deadline and remaining execution time rem_i of each periodic task T_i. The rem_i field of the head of the α-queue decreases with a rate equal to that of the passage of time. The DRA is under the EDF* policy. EDF* is almost the same as EDF. The difference is that in EDF* among the tasks whose deadlines are the same, the task with the earliest arrival time has the highest priority (a FIFO policy). Among the tasks whose deadlines and arrival times are the same, the task with the lowest index has the highest priority. The key notation for the DRA is as follows [2]:

- S^{can} : The canonical schedule in which each periodic task finishes before its deadline
- $\hat{S}_i$: The nominal speed of periodic task T_i
- $rem_i(t)$: The remaining execution time of periodic task T_i at time t in S^{can}
- $w_i^S(t)$: The remaining WCET of periodic task T_i under speed S at time t in the actual schedule
- $\varepsilon_i(t)$: The earliness of periodic task T_i at time t in the actual schedule, defined as :

$$\varepsilon_i(t) = \sum_{j|D_j^* < D_i^*} rem_j(t) + rem_i(t) - w_i^{S_i}(t)$$
$$= \sum_{j|D_j^* \le D_i^*} rem_j(t) - w_i^{S_i}(t)$$

where D_i is the deadline of T_i

Note that the nominal speed is the default speed it has whenever it is dispatched by the operating system prior to any dynamic adjustment.

2.4 Existing Inter-task DVS Algorithm for Mixed Workload Real-time Systems

Recently, several researchers proposed DVS algorithms for mixed workload real-time systems. Under the EDF (or EDF*) scheduling policy, most of these algorithms integrate the bandwidth preserving server and priority-based slack stealing strategies. Doh et al. [6] proposed an approach which leads to proper allocation of energy budgets for hard periodic and soft aperiodic real-time tasks. Given an energy budget, it computes a proper voltage setting for attaining an improved performance for aperiodic tasks while meeting the deadline requirements of periodic tasks. It used TBS (total bandwidth server) [18], which is a kind of bandwidth preserving servers, and only focused on the off-line static scheduling problem. Aydin et al. proposed three separate on-line schemes with mixed workload under a power consumption constraint. It also used TBS and DRA [2] under the EDF* scheduling policy. In the Basic Reclaiming Scheme (BRS) [1] the earliness of aperiodic tasks is only used for reclaiming the coming aperiodic tasks, and the earliness of periodic tasks is only used for reclaiming the coming periodic tasks. The Mutual Reclaiming Scheme (MRS) [1] was developed from BRS. The main difference between MRS and BRS is that in the MRS both periodic and aperiodic tasks can mutually reclaim their unused computation times. The Bandwidth Sharing Scheme (BSS) [1] is to solve the problem of the actual aperiodic workload that is relatively lower than the predicted aperiodic workload. In BSS when TBS is idle, the algorithm will create a ghost job *J* to produce more earliness to aggressively reduce the operating speed. But it will increase the response time of aperiodic tasks if an actual aperiodic task arrives right after creating the ghost job.

In [4], Shin et al. merged the TBS and two DVS algorithms, lppsEDF [16] and DRA, respectively, under the EDF* scheduling policy. They also proposed an enhanced approach called Workload-based Slack Estimation (WSE) [3], which integrates CBS [19] and DRA. The WSE is almost the same as the MRS (as indicated in [3]), except that it uses C_{slack} to stretch the execution time of periodic tasks by using the slack time from CBS and it guarantees the constrained response time. But C_{slack} decreases to zero rapidly as aperiodic tasks arrive. In this case, the slack time of CBS is wasted. Note that both the MRS and WSE need to know the workload of aperiodic tasks in advance. In addition, the MRS needs to know the WCETs of aperiodic tasks. However, in most cases the workload of aperiodic tasks of some real-time systems is with large variance or unpredictable [7]. If the given budget of the bandwidth preserving server is not suitable for the current schedule point, it will waste unnecessary energy consumption or result in long response time. Another problem is that a periodic task with the highest priority may run slowly even if there are some aperiodic tasks waiting in the queue.

3 System Model, Assumptions and Notations

The target processor can change its supply voltage (V) and operating speed (S) (or frequency) continuously within its operational ranges, [V_{min} , V_{max}] and [S_{min} , S_{max}]. There are two components of mixed workload real-time systems: a set of T = { $T_1 \ldots T_n$ } of n periodic tasks with hard deadlines, and a set of J aperiodic tasks arriving randomly with soft deadlines. Based on related work [1][3][4][5][18][19][20][21], the arrival time and the worst case requirements of periodic tasks are known in advance, but those of aperiodic tasks are made available only when they arrive. The relative deadline of each periodic task instance is assumed equal to its period. All tasks are assumed to be independent. We used the following notation:

- $T_{i,j}$: The j^{th} instance of T_i
- C_i : The worst case CPU execution cycles per instance of T_i
- D_i : The deadline of T_i
- J_i : The i^{th} aperiodic task
- r_i : The arrival time of J_i
- d_i : The deadline of J_i
- f_i : The finish time of J_i
- E_a : The average CPU execution cycles per aperiodic task
- N_a : The number of aperiodic tasks in the queue
- A_i : The largest amount of aperiodic processing possible of J_i

4 Proposed SS-DVS Algorithm

In this paper, we propose an on-line DVS algorithm for mixed workload real-time systems, named *Slack Stealing for DVS* (SS-DVS). The SS-DVS doesn't need to know the workload and the WCET of aperiodic tasks. It used the concept of slack stealing, which was originally used in mixed workload real time systems without DVS, to service aperiodic tasks. It used the DRA to serve aperiodic tasks and to reclaim the operating speed. In SS-DVS, all allowed execution time belongs to periodic tasks. In this way, the periodic tasks can execute slowly to save energy. It will increase the operating frequency once aperiodic tasks arrive and serve the aperiodic tasks as soon as possible. Although the slack stealing approach was considered to have high computation overhead to derive the slack time of the schedule to service aperiodic tasks, the proposed SS-DVS has low computation overhead compared to other approaches. The reason is that the SS-DVS obtains slack time from the periodic tasks completed earlier or generates slack time by raising the operation speed of the current task. That is, the SS-DVS not only collects the slack time for reclaiming the speed to save energy, but also for servicing aperiodic tasks to reduce their response time.

The basic idea of the proposed SS-DVS is using the slack time to reduce the operating speed of periodic and aperiodic tasks and also using the slack time to service aperiodic tasks based on the DRA. If no aperiodic task arrives, obtaining low

energy consumption is to operate periodic tasks with low speed. And if any aperiodic task arrives, it uses the collected slack time to service aperiodic tasks. First we have to construct a data structure call *α'-queue* which is almost the same as α-queue with an additional flag, *ModeFlag*, in the α'-queue. The *ModeFlag* records the nominal speed of each periodic task. If *ModeFlag* is set to *H*, it means the nominal speed is S_H and if *ModeFlag* is set to *L*, it means the nominal speed is S_L. There are four operations in our algorithm.

(1). Setup: Compute the static optimal speed $\overline{S}$. Then set S_L, to a value large than or equal to $\overline{S}$, and set S_H to a value between $S_{\max}$ and S_L. Then, initialize the α'-queue to the empty list and set the nominal speed of periodic tasks to S_L.

(2). Basic stealing: Using the slack time to service aperiodic tasks is the basic idea of SS-DVS. First of all, when an aperiodic task arrives at the active state, it means that there are pending aperiodic tasks in the queue at time *t* and there already exists at least one task J_i in the queue such that $r_i \le t < f_i$. Otherwise, it is at the idle state. When a task J_y arrives at the active state, the task is placed in a queue of pending tasks according to the FIFO, and the allowed execution time and the deadline of J_y are set according to the preceding task J_{y-1}. When a task J_y arrives at the idle state and no periodic tasks being serviced or in the queue, the deadline of J_y is set to NTA, and the allowed execution time of J_y is set to $NTA - t$. As a task J_y arrives at the idle state and some periodic tasks are being serviced or in the queue, the deadline of J_y depends on the earliness of periodic task T_x with the highest priority. Hence the deadline of J_y is $t + \varepsilon_x(t)$, where $\varepsilon_x(t)$ is the earliness of periodic task T_x at time t ,and A_x is set to $\varepsilon_x(t)$. For example, in Fig. 1(a), when a task J_y arrives at the idle state and there are no other periodic tasks in the queue, the A_x of J_y is $NTA - t$. And in Fig. 1(b), when a task J_y arrives at the idle state and periodic task T_x is executing, the A_x of J_y is set to $\varepsilon_x(t)$.

(3). Aggressive stealing: The speed S_L is the nominal speed of periodic tasks. However, if we execute a periodic task at S_L and the actual execution time of the periodic task is in its worst case, there will be no earliness left. Consequently, the speed is raised to S_H to create more slack time for serving aperiodic tasks. In Fig. 1(c), J_y arrives at t_1, but there is no earliness of T_x left. Therefore, the operating speed is changed to S_H to create more slack time. As we can see, T_x finishes at t_2 and the slack time is available

(4). Swapping stealing: In the aggressive stealing, a periodic task executes at high speed S_H and produces some slack time. However, we still have to service periodic tasks first. Therefore, we could more aggressively service aperiodic tasks first to reduce the response time without causing any deadline miss of hard real-time

periodic tasks. Swapping the order of periodic and aperiodic tasks not only can reduce the response time, but also could lower the speed of periodic tasks if aperiodic tasks completes early than expects. As shown in Fig. 1(d), we change the order of execution between T_x and J_y.

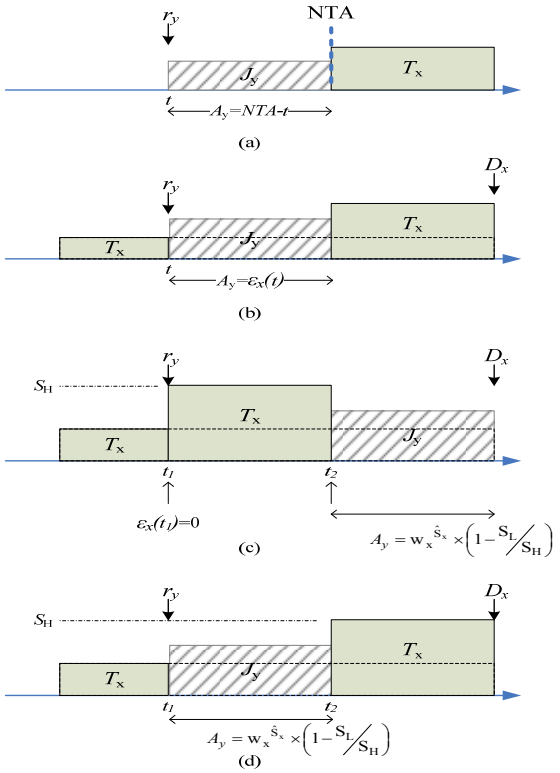

Fig. 1. The SS-DVS scheme (a) the aperiodic task arrives with no other periodic task in the queue (b) the aperiodic task arrives with some periodic tasks in the queue, and the highest priority task has some earliness (c) aggressive stealing(d) swapping stealing.

5 Simulation Results

5.1 Simulation Model

We use the same simulation environment as that of WSE [3]. Aperiodic tasks were generated by the exponential distribution using with inter arrival time (1/λ) and service time (1/μ) with parameters λ and μ. We used a fixed value μ and varied λ to control the workload (ρ = λ/μ) of aperiodic tasks under a fixed utilization U_p of periodic tasks [3]. There are four periodic tasks in Table 1. The period of each task is 6, 8, 14, and 18, respectively, and the WCET of each task is 0.5, 1.0, 2.1, and 3.1,

respectively. The utilization U_p of periodic tasks is 0.4 ((0.5 / 6) + (1.0 / 8) + (2.1 / 14) + (3.1 / 18)).

Table1. Periodic task set descriptin.

Task Set (millisecond)		
Task	Period	WCET
T_1	6	0.5
T_2	8	1.0
T_3	14	2.1
T_4	18	3.1
U_p	0.4	

The actual execution time of each periodic task instance was generated by a normal distribution function in the range of [BCET, WCET], where BCET is the best-case execution time. The mean and the standard deviation were set to (WCET+BCET)/2 and (WCET-BCET)/6, respectively [1]. In the experiments, the voltage scaling overhead was assumed negligible both in the time delay and power consumption. For easiness to compare with other approaches, we used the fixed S_L and S_H in this experiments. S_L is 60% of the maximal speed (must be larger than or equal to the static optimal speed) and S_H is the maximal speed.

In order to experimentally evaluate the performance of the proposed algorithm, SS-DVS, we implemented the following schemes for performance evaluation: (1) PD/CBS [3]: Aperiodic tasks are serviced by CBS (constant bandwidth servers). It was assumed that if the system is idle, it enters into the power-down mode (PD). The power consumption in the PD mode is assumed to be zero. (2) Mutual Reclaiming Scheme (MRS) [1]: We set S_a (the nominal speed of aperiodic tasks) equal to S_p (the nominal speed of periodic tasks). (3) Workload-based Slack Estimation scheme (WSE) [3]. (4) Bandwidth Sharing Scheme (BSS) [1]: We set S_a equal to S_p.

In the following, the average energy consumption and response time are normalized to those of PD/CBS, under different conditions.

5.2 Effect of the Workload of Aperiodic Tasks on Energy Consumption and Response Time

BCET is assumed to be 10% of WCET, and ρ is ranging from 0.05 to 0.25 ($\lambda = 0.05 \sim 0.25$ and $\mu = 1.0$) [3]. As shown in Fig. 2, by different workload of aperiodic tasks, SS-DVS is compared with others approaches under varied server utilization. Fig. 2(a) shows that SS-DVS has lower normalized energy consumption than that of the other three existing algorithms, and it also has comparable normalized response time with MRS and WSE, as shown in Fig. 2(b). Although BSS has less energy consumption than SS-DVS when the server utilization is close to the workload of aperiodic tasks, BSS has the highest response time. The normalized energy * response time [1] is a performance metric that combines the two important dimensions, energy consumption and response time, of the mixed workload real-time systems. As Fig. 2 (c) shown, SS-

DVS has better performance than the three other approaches in terms of normalized energy * response time.

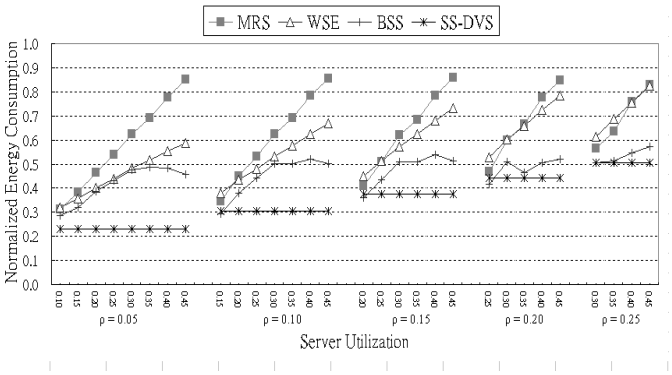

Fig. 2(a) Normalized energy consumption.

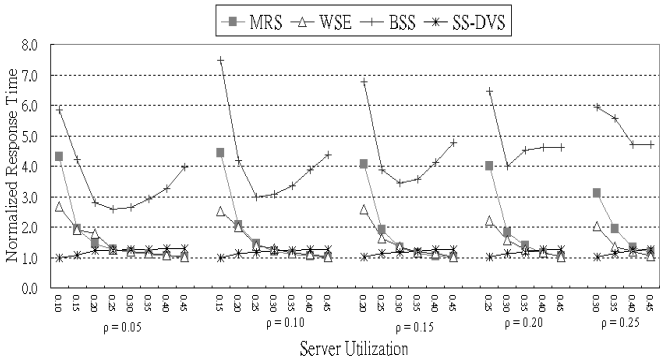

Fig. 2(b) Normalized response time.

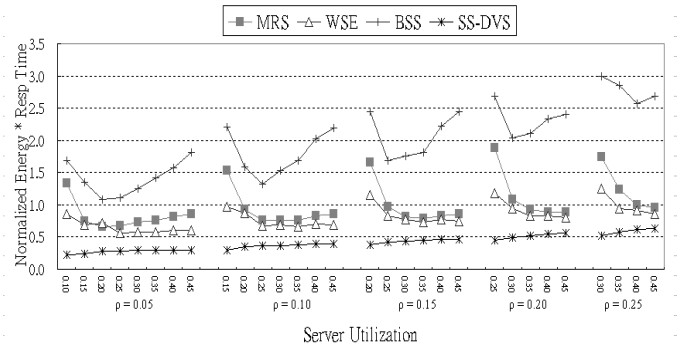

Fig. 2(c) Normalized energy * response time.

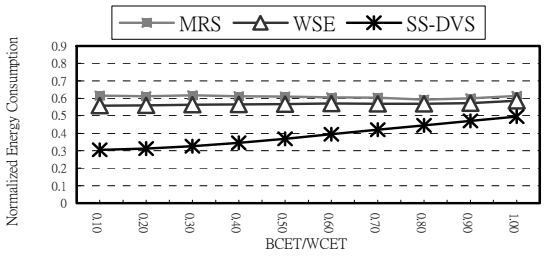

Fig. 3. The normalized energy consumption base on same response time with different BCET/WCET.

5.3 Effect of BCET/WCET Ratio of Periodic Tasks on Energy Consumption

The workload of aperiodic tasks was set to 0.1 (λ = 0.1 and μ = 1.0), and the BCET/WCET ratio was from 0.1 to 1.0 with an increment of 0.1 0. Fig. 3 shows the normalized energy consumption based on the same response time of all algorithms with different BCET/WCET ratios. We have the following observations:

- The normalized energy consumption of SS-DVS increases as BCET/WCET increases. This is because aperiodic tasks are served by the slack time from periodic tasks. The less slack time from periodic tasks may cause the periodic and aperiodic tasks to run at a higher speed.
- SS-DVS reduces the energy consumption by an average of 21% and 17% compared with the MRS and WSE algorithms, respectively.

6 Conclusion

In this paper, we have presented an on-line dynamic voltage scaling (DVS) algorithm, called SS-DVS, for mixed workload real-time systems. SS-DVS not only addresses the energy consumption for mixed workload real-time systems, but also considers the response time of aperiodic tasks. SS-DVS integrates the slack time stealing concept to service aperiodic tasks and the dynamic reclaiming algorithm (DRA) to set a suitable operating speed. SS-DVS can use the slack time more efficiently than existing approaches, because it doesn't reserve any time for aperiodic workload. Simulation results have shown that SS-DVS can effectively reduce the average energy consumption by 48%, 22%, 18% compared with the PD/CBS, MRS, and WSE algorithms, respectively, under the same response time.

7 References

1. H. Aydin, and Q. Yang, "Energy-responsiveness tradeoffs for real-time systems with mixed workload," in *Proceedings of 10th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp 74-83, 2004.

2. H. Aydin, R. Melhem, D. Moose, and P. Mejia-Alvarez, “Power-aware scheduling for periodic real-Time tasks,” *IEEE Transactions on Computers*, Volume 53, Issue 5, pp. 584-600, May 2004.
3. D. Shin, and J. Kim, “Dynamic voltage scaling of mixed task sets systems in priority-driven systems,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Volume 25, Issue 3, pp. 438-453, March 2006.
4. D. Shin and J. Kim, “Dynamic voltage scaling of periodic and aperiodic tasks in priority-driven systems,” in *Proceedings of Asia and South Pacific Design Automation Conference*, pp. 653-658, 2004.
5. J. P. Lehoczky, and S. Ramos-Thuel, “An optimal algorithm for scheduling soft-aperiodic tasks in fixed priority preemptive systems,” in *Proceedings of the IEEE Real-Time Systems Symposium*, pp. 110-123, Dec. 1992.
6. Y. Doh, D. Kim, Y.-H. Lee, and C. M. Krishna, “Constrained energy allocation for mixed hard and soft real-time tasks,” in *Proceedings of 9th International Conference on Real-Time and Embedded Computing Systems and Applications*, pp. 533-550, 2003.
7. C. Rusu, X. Ruibin, R. Melhem, D. Mosse, "Energy-efficient policies for request-driven soft real-time systems," in *Proceedings of Euromicro Conference on Real-Time Systems*, pp.175-183, July 2004.
8. L. Benini, A. Bogliolo, and G. De Micheli, “A survey of design techniques for system level dynamic power management,” *IEEE Transactions on Very Large Scale Integration Systems*, Vol.8, No.3, pp. 299-316, 2000.
9. W. Kim, D. Shin, H. S. Yun, S. L. Min, and J. Kim. ”Performance comparison of dynamic voltage scaling algorithms for hard real-time systems,” in *Proceedings of the IEEE Real-Time and Embedded Technology and Application Symposium,* pp. 219-228, September 2002.
10. B. Moyer, “Low-power design for embedded processors,” in *Proceedings of IEEE*, Volume 89, Issue 11, pp. 1576-1587, November 2001.
11. Intel XScale® Technology, “Intel. PXA270 processor electrical, mechanical, and thermal specification,“ http://www.intel.com/design/intelxscale/
12. Intel® Xeon® Processor, http://www.intel.com/products/processor/xeon/
13. Trasmeta Corporation, “TN5400 processor specification,” http://www.transmeta.com/crusoe/
14. Mobile AMD Athlon™ 64 processor, http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_10220_10221,00.html
15. G. Carpenter, “Low power SOC for IBM's PowerPC information appliance platform,” in http://www.research.ibm.com/arl.
16. Y. Shin, and K. Choi, “Power conscious fixed priority scheduling for hard real-time systems,” in *Proceedings of the Design Automation Conference*, pp. 134-139, 1999.
17. W. Kim, J. Kim, and S. L. Min, “Preemption-aware dynamic voltage scaling in hard real-time systems,” in *Proceedings International Symposium Low Power Electronics and Design*, pp. 393–398, 2004.
18. M. Spuri and G. Buttazzo, “Scheduling aperiodic tasks in dynamic priority systems,” *Journal of Real-Time Systems*, vol. 10, no. 2, pp. 179-210, 1996.
19. L. Abeni and G. Buttazzo, “Integrating multimedia applications in hard real-time systems,” in *Proceedings of the IEEE Real-Time Systems Symposium*, pp. 4-13, 1998.
20. 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*, vol. 44, no. 1, pp. 73-91, Jan. 1995.
21. B. Sprunt, L. Sha, and J. P. Lehoczky, “Aperiodic task scheduling for hard real-time systems,” *Journal of Real-Time Systems*, vol. 1, no. 1, pp. 27–60, 1989.
22. P. Pillai and K. G. Shin, “Real-time dynamic voltage scaling for low power embedded operating systems,” in *Proceedings of the ACM Symposium on Operating Systems Principles*, pp. 89-102, 2001.