國立交通大學

網路工程研究所

碩 士 論 文

在混合負載即時系統下之

高能源效率動態電壓調整演算法

Energy Efficient Dynamic Voltage Scaling for

Mixed Workload Real-time Systems

研 究 生：陳蒸民

指導教授：王國禎　教授

中 華 民 國　九 十 六　年 六 月

在混合負載即時系統下之高能源效率動態電壓調整演算法

Energy Efficient Dynamic Voltage Scaling for Mixed Workload Real-time Systems

研 究 生：陳蒸民　　Student：Jheng-Ming Chen

指導教授：王國禎　　Advisor：Kuochen Wang

國 立 交 通 大 學

網 路 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Network Engineering

College of Computer Science

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年六月

在混合負載即時系統下之
高能源效率動態電壓調整演算法

學生：陳蒸民　　指導教授：王國禎 博士

國立交通大學 資訊學院 網路工程研究所

摘　要

近年來，由於無線通訊的快速成長，使個人數位助理與手機等手持裝置的能源消耗成為一個值得重視的研究問題。儘管有許多動態電壓調整演算法被提出來降低即時系統上的能源消耗，但絕大部分的動態電壓調整演算法僅針對只有週期性工作的即時系統，甚少動態電壓調整演算法適用於週期性工作與非週期性工作同時存在的混合負載即時系統。一個混合負載即時系統下的動態電壓調整演算法，不僅要節省能源消耗，同時也必須考慮到非週期性工作的反應時間。本論文提出一個在速率單調(Rate Monotonic)排程法下針對混合負載即時系統之基於工作需求估計寬裕時間(slack time)的高能源效率動態電壓調整演算法(WSS)。WSS 估計寬裕時間是依據非週期性工作排程伺服器的執行行為和原本只適用於純週期性工作即時

系統下的短期工作需求分析法。除此之外，當非週期性的工作量大於非週期性工作排程伺服器所能服務的量時，WSS 會利用所估計的寬裕時間來提前服務非週期性工作，因此可縮短非週期性工作的反應時間。模擬結果顯示，我們所提出的方法相較於現存的 SNRT (BSS) 方法，節省了 23% (13%) 的能源消耗，縮短了 38% (31%) 的非週期性工作的反應時間，同時也減少 52% (40%) 的能源消耗與反應時間乘積。

關鍵詞：實際工作量、任務間動態電壓調整方法、混合負載即時系統、寬裕時間、最差執行時間。

Energy Efficient Dynamic Voltage Scaling for Mixed Workload Real-time Systems

Student: Jheng-Ming Chen Advisor: Dr. Kuochen Wang

Department of Computer Science

National Chiao Tung University

Abstract

Recently the wireless communication is rapidly growing up and the energy consumption on mobile devices, such as personal digital assistants (PDAs) and cellular phones, becomes a critical issue. In spite of numerous inter-task dynamic voltage scaling (DVS) algorithms have been brought up for energy saving of real-time systems with only periodic tasks or only aperiodic tasks, few of them were aimed at the mixed workload of periodic tasks and aperiodic tasks. A DVS algorithm for mixed workload real-time systems should not only focus on energy saving, but also consider low response time of aperiodic tasks. In this thesis, we develop an on-line energy efficient scheduling, *Work-demand-based Slack-Stealing scheme* (WSS), to reduce CPU energy consumption for mixed workload real-time systems under the Rate Monotonic (RM) scheduling policy. The WSS calculates the available slack time by the execution behaviors of scheduling servers for aperiodic tasks and by short-term work-demand analysis, which was originally designed for real-time systems with periodic tasks only. Moreover, the WSS also utilizes the concept of slack stealing to service aperiodic tasks and to reduce the response time when the actual workload of aperiodic tasks is close to the server utilization. Simulation results shows that the proposed WSS can effectively reduce the energy consumption, response time, and energy consumption * response time, in average, by 23%, 38%, 52% compared to the SNRT and 13%, 31% , 40% compared to the BSS, respectively.

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

Acknowledgements

Many people have helped me with this thesis. I deeply appreciate my thesis advisor, Dr. Kuochen Wang, for his intensive advice and instruction. I would like to thank all the classmates in *Mobile Computing and Broadband Networking Laboratory* for their invaluable assistance and suggestions. This work was supported by the NCTU EECS-MediaTek Research Center under Grant Q583.

Finally, I thank my Father and Mother for their endless love and support.

Contents

List of Figures

List of Tables

Chapter 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 increasing the response time. It's not effortless to determine when and which device should be shut down and woken up [1]. 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 commercially available processors provide the DVS feature, including Intel Xscale [2] and Xeon [3], Transmeta Crusoe [4], AMD Mobile Athlon [5], and IBM PowerPC 405LP [6].

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 $E = C_{eff} \cdot V_{dd}^2 \cdot C$ [7], 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 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 [8][9]. Thus, a DVS algorithm can be used without modifications for real variable-voltage processors [8]. Additionally, [10] 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 [11], 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 thesis, we consider inter-task scheduling. Most of the existing inter-task 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 [8]. Another example, a partition of jobs in a robot control application such as sensory acquisitions, data processing, path planning and low level control loops arrive periodically and must be serviced before each deadline to ensure robot stability. Other jobs such as operator requests or displaying activities occur randomly and usually have soft deadlines or no deadlines at all [12]. The flight systems and the automatic memory reclamation in real-time systems also have both periodic and aperiodic tasks [8]. 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 [13].

Therefore, we reduce the energy consumption while satisfying the requirements of periodic tasks. Although the aperiodic tasks are soft real-time and can be executed with the minimum speed to minimize the energy consumption, the response time of aperiodic tasks may become long. Contrarily, using the maximum speed wastes energy. It's not a suitable approach especially when we want battery-powered devices to live longer. So while reducing energy consumption on mixed workload real-time systems one has to consider the response time of aperiodic tasks.

The rest of this thesis is organized as follows. In chapter 2, we classify aperiodic real-time tasks scheduling, existing on-line inter-task DVS strategies, and describe the low power work-demand analysis. Chapter 3 reviews related work. Chapter 4 presents the target system model and describes the proposed WSS algorithm. In chapter 5, simulation results are evaluated and discussed. Some practical issues are also addressed. Finally, chapter 6 concludes with a summary and future work.

Chapter 2
Preliminaries

2.1 Categories of Aperiodic Real-Time Tasks Scheduling

Approaches to serve aperiodic and periodic tasks in real-time systems can be classified into four types:

(1). Background [14]

This is the simplest approach to service aperiodic tasks. In order to execute hard real-time tasks (i.e., periodic tasks) without deadline miss, the periodic tasks need to be serviced first. While there are no periodic tasks in the ready queue waiting for executing, aperiodic tasks can be serviced. The advantage of this approach is simple and with low overhead, but the response time of aperiodic tasks is large.

(2). Polling Server [14]

It creates a polling server as a periodic task, and aperiodic tasks are served during the activation of the polling server. It is characterized by Q_s and T_s, where Q_s is the maximum budget and T_s is the period of the server. The polling server is up periodically at integer multiplies of T_s, and it executes aperiodic tasks until the budget is exhausted or no aperiodic task is in the queue. Then the polling server goes down and waits for the next wake-up time. Thus the budget is the WCET of the polling server, and the slack time is the interval that no aperiodic task is in the queue and the polling server goes down early. However, aperiodic tasks have to wait for next integer multiplies of T_s if they arrive when the polling server just went down. As a result, the response time of aperiodic tasks becomes long.

(3). Bandwidth-Preserving Servers [12][14][15][16]

It is similar to the polling server. The concept of the bandwidth preserving servers is

creating a server with execution budget which is a time amount for executing aperiodic tasks. It is also characterized by an ordered pair (T_s, Q_s). Q_s / T_s is the server utilization (U_s). 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 execution budget. If the execution budget is exhausted, it will stop or delay serving the aperiodic tasks. For instance, the deferrable servers (DS) algorithm [14] is the simplest bandwidth-preserving servers. The execution budget q_s of DS is consumed while aperiodic tasks execute. An aperiodic task can execute as long as the DS has the highest priority and the execution budget is larger than zero. If the execution budget is exhausted, aperiodic tasks are serviced at background priority until the next replenishment time (NRT). At each replenishment time, the execution budget is replenished to a maximum budget Q_s. The replenishment of DS is at time kT_s, for k = 0, 1, 2….

(4).Slack Sealing [17][18]

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 approach, the response time of aperiodic tasks is the lowest, but the complexity of such a real-time system is the highest among the above four approaches.

2.2 Categories of On-Line Inter-Task DVS Strategies

Existing on-line DVS algorithms for periodic tasks in real-time systems can be classified into four strategies [11][19]:

(1).*Stretching-to-NTA* [9]

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* [20]

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.

(3). *Utilization Updating* [21]

The worst case processor utilization can be computed by the WCETs of tasks off-line. However, the actual execution time can be obtained on-line when a task completes its operation. That is, the actual processor utilization can be computed during run time, and it is less than the worst case processor utilization. Hence, the operating frequency can be decreased according to the new actual processor utilization.

(4). *Short-Term Work-Demand Analysis* [22]

The slack time is estimated by using the short term work-demand analysis. It enlarges the available slack time of the scheduled task by delaying the schedule of lower-priority tasks in near future as late as possible.

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 and the utilization updating 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 predict processor utilization, the deadline

miss will occur just like that occurs in the stretching to NTA. In the short-term work-demand analysis strategy, it's hard to calculate the amount of work of the aperiodic tasks required to be processed in a period. 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 approaches.

2.3 Low Power Work-Demand Analysis (lpWDA)

The lpWDA [22], which was originally designed for real-time systems with periodic tasks only, is an efficient on-line slack estimation heuristic for the RM scheduling. The slack estimation procedure uses the short-term work-demand analysis. The goal of the lpWDA is to extend the available slack time of the scheduled task by delaying the schedule of lower-priority tasks in near future as late as possible.

The slack time, $slack_i(t)$ of a periodic task T_i at time t can be computed as $D_i - t - load_i(t)$, where D_i is the deadline of T_i and $load_i(t)$ is the amount of work required to be processed in $[t, D_i]$. $load_i(t)$ consists of three types of work: (1) $w_i^{rem}(t)$: the remaining WCET of T_i at time t for T_i itself, (2) $H_i(t)$: the work from the higher-priority tasks, and (3) $L_i(t)$: the work from the lower-priority tasks. $w_i^{rem}(t)$ is the known value at each scheduling point, but $H_i(t)$ and $L_i(t)$ should be computed from a complex analysis. In the lpWDA, it computes approximate estimates of $H_i(t)$ and $L_i(t)$, $\tilde{H}_i(t)$ and $\tilde{L}_i(t)$, where $\tilde{H}_i(t) \geq H_i(t)$ and $\tilde{L}_i(t) \geq L_i(t)$, for a safe estimation on available slack times. Therefore, $slack_i(t)$ can be computed by estimating an approximate value of $load_i(t)$, $\tilde{l}oad_i(t)$, where $\tilde{l}oad_i(t) = w_i^{rem}(t) + \tilde{H}_i(t) + \tilde{L}_i(t)$. Here, $slack_i(t)$ may be a negative value when

$\tilde{l}oad_i(t)$ is overestimated. Fig. 1 are the lpWDA algorithm, where P_i and W_i are the period and WCET of T_i, ε is the infinitesimal, ud_i is the *upcoming deadline* of T_i, w_{done} is the amount of work to be done between T_i being scheduled for execution and being preempted, H_i^{past} of T_i is the work required by uncompleted higher-priority tasks before *t*, and *CalcSlackTime*() is a procedure of the lpWDA used to calculate the slack time $slack_i(t)$ of T_i [22].

For instance, there are two periodic tasks in Table 1 and their periods are 6 and 8, respectively. The WCET of tasks are 1 and 2 time units, respectively. Fig. 2 (a) shows the non-DVS scheme, and Fig. 2 (b) shows the load estimation of each task at $t = 0$. While the first instance of T_1, $T_{1,1}$, is scheduled for execution, $slack_1(0)$ has to be calculated. The analysis scope of the lpWDA is [0, 8] according to the latest upcoming deadline, and $\tilde{H}_1(0) = 0$ and $\tilde{H}_2(0) = 2$ can be derived. Now $w_1^{rem}(0)$ and $\tilde{H}_1(0)$ are known values, but $\tilde{L}_1(0)$ has to be determined by T_2 which has the earliest upcoming deadline among tasks whose priorities are lower than that of T_1. According to the lpWDA, $\tilde{L}_1(0) = max(0,\Delta)$, where $\Delta = \tilde{l}oad_2(0) - w_1^{rem}(0) - \tilde{H}_1(0) - (ud_2(0) - D_1)$, and $ud_2(t) = 8$ is the upcoming deadline of task T_2. Once $\tilde{l}oad_2(0)$ is computed, we can estimate $slack_1(0)$ as follows,

$$\tilde{l}oad_2(0) = w_2^{rem}(0) + \tilde{H}_2(0) + \tilde{L}_2(0) = 2 + 2 + 0 = 4$$

$$\tilde{L}_1(0) = max(0, 4 - 1 - 0 - (8 - 6)) = 1$$

$$slack_1(0) = 6 - 0 - 1 - 0 - 1 = 4$$

Therefore, the available execution time $A_1(0)$ for T_1 can be estimated as

$$A_1(0) = max(0, slack_1(0)) + \ w_1^{rem}(0) \ = 5$$

And the clock speed can be adjusted to

$$S_{clk} = (w_1^{rem}(0) / A_1(0)) \times S_{max} = 1/5 \times S_{max}$$

lpWDA Algorithm :

IF system start **then**

FOR each task T_x

$ud_x = P_x$

$H_x(t) = H_x^{past}$

$$+ \sum_{i=1}^{x-1} \left(\left\lfloor \frac{ud_x - \varepsilon}{P_i} \right\rfloor - \left\lceil \frac{t+\varepsilon}{P_i} \right\rceil + 1 \right) \times W_i$$

IF T_x is activated **THEN**

$w_x^{rem}(t) = W_x$

END IF

IF T_x is completed or preempted **THEN**

Call UpdataLoadInfo()

END IF

IF T_x is scheduled for execution **THEN**

Call CalcSlackTime() to get slack time $slack_x(t)$

Set the clock frequency and voltage accordingly.

END IF

END IF

Function CalcSlackTime()

Identify the task T_y that has the earliest upcoming deadline among tasks whose priorities are not higher than that of T_x

$L_y(t)$ = CalcLowerPriorityWork(T_y)

$load_y(t) = w_y^{rem}(t) + H_y(t) + L_y(t)$

$slack_x(t) = max\ (0, ud_y - t - load_y(t))$

return ($slack_x(t)$)

Fig. 1. lpWDA algorithm [22]

Function CalcLowerPriorityWork()

IF T_y is identical to T_n **THEN** return 0 **END IF**

Identify the task T_z that has the earliest upcoming deadline among tasks whose priorities are lower than that of T_y

$L_z(t)$ = CalcLowerPriorityWork(T_z)

$load_z(t) = w_z^{rem}(t) + H_z(t) + L_z(t)$

$L_y(t) = max\ (0, load_z(t) - w_y^{rem}(t) - H_y(t) - ud_z + ud_y)$

return ($L_y(t)$)

Function UpdateLoadInfo()

IF (COMPLETION) **THEN**

$ud_x = ud_x + P_x$

$$H_x(t) = \sum_{i=1}^{x-1}\left(\left\lfloor \frac{ud_x - \varepsilon}{P_i} \right\rfloor - \left\lceil \frac{t+\varepsilon}{P_i} \right\rceil + 1\right) \times W_i$$

LOOP each task T_i, $i = x + 1$ until $i = n$

$H_i(t) = H_i(t) - w_x^{rem}(t)$

END LOOP

$w_x^{rem}(t) = 0$

ELSE (PREEMPTION)

$w_x^{rem} = w_x^{rem} - w_{done}$

LOOP each task T_i, $i = x + 1$ until $i = n$

$H_i(t) = H_i(t) - w_{done}$

END LOOP

END IF

Fig. 1. lpWDA algorithm [22] (cont.)

Table 1. Example task set

Task	WCET(ms)	Period(ms)
T_1	1	6
T_2	2	8

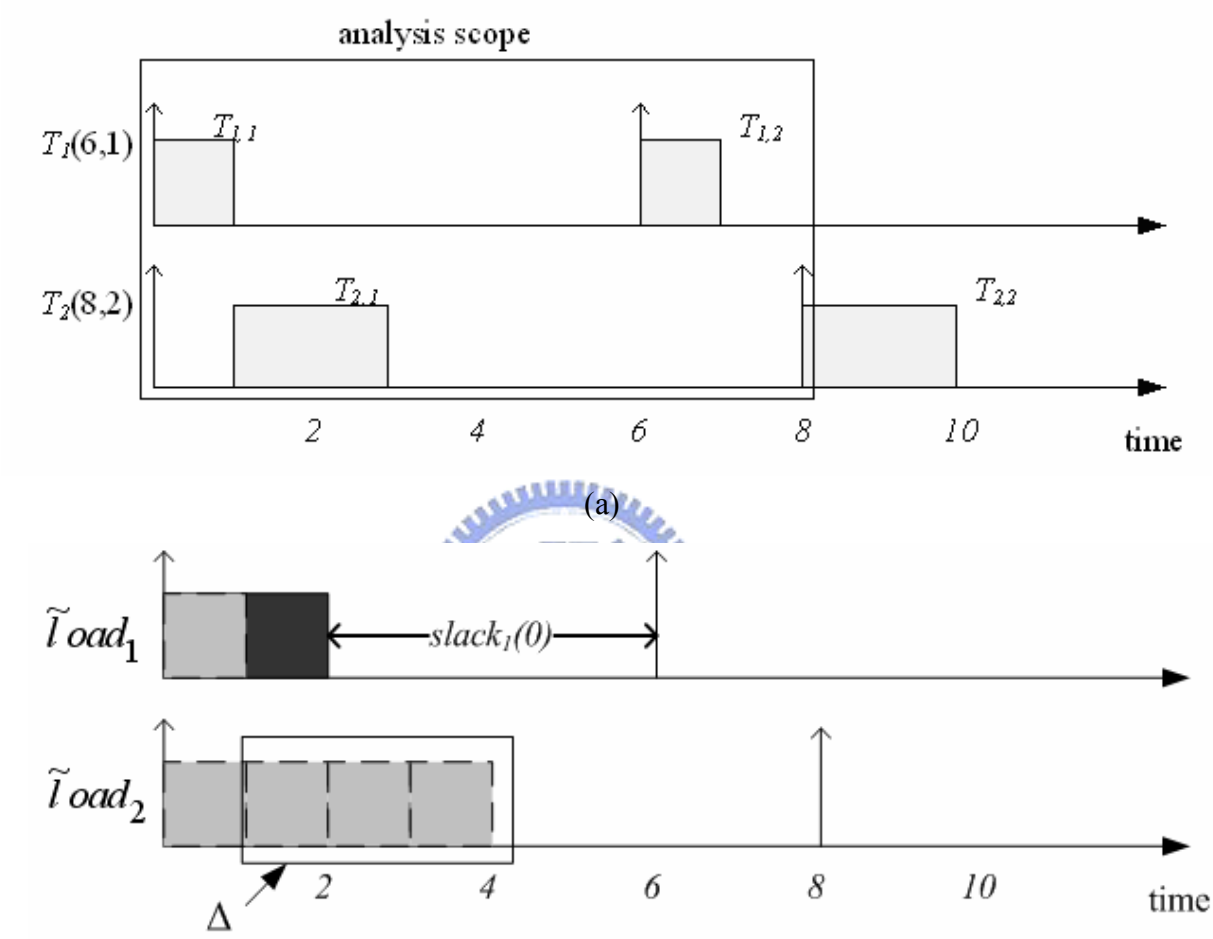


Fig. 2. Short-term work-demand analysis example: (a) shows the non-DVS scheme before using lpWDA. (b) shows the load estimation of each task at t = 0 using lpWDA. (The gray-boxes mean the amount of work from the higher or same priority tasks, and the black-boxes mean the amount of work from the lower priority tasks.)

Chapter 3
Related Work

3.1 Existing DVS Algorithms for Mixed Workload

Real-Time Systems

Recently, several researchers proposed DVS algorithms for mixed workload real-time systems. Under the EDF (Earliest Deadline First) (or EDF*) scheduling policy, most of these algorithms integrate the bandwidth preserving servers and priority-based slack stealing strategy.

Doh et al. [23] 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 Total Bandwidth Servers (TBS) [12], 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 Dynamic Reclaiming Algorithm (DRA) [20] under the EDF* scheduling policy. In the Basic Reclaiming Scheme (BRS) [24] 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) [24] 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) [24] is to solve the problem of the actual aperiodic workload that is relatively lower than the predict aperiodic workload. In the

BSS when the TBS is idle, the algorithm will create a ghost job *J* to produce more earliness to aggressively reduce the clock speed. But it will increase the response time of aperiodic tasks if an actual aperiodic task arrives right after creating the ghost job.

In [26], Shin et al. merged the TBS and two DVS algorithms, lppsEDF [9] and DRA, respectively, under the EDF* scheduling policy. They also proposed an enhanced approach called Workload-based Slack Estimation (WSE) [8], which integrates Constant Bandwidth Servers (CBS) [16] and DRA. The WSE is almost the same as the MRS (as indicated in [8]).

All of the above approaches focused on the EDF (or EDF*) scheduling policy. Few of existing DVS for mixed workload real-time systems were proposed under the RM scheduling policy. The RM scheduling policy has been adopted in most real-time schedulers of practical interest due to its low overhead and predictability [25].

Under the RM scheduling policy, most of these algorithms integrate the bandwidth preserving servers and stretching-to-NTA strategy. The *Stretching-To-NRT* (SNRT) [26] scheme is the first proposed DVS algorithm for mixed workload real-time systems under the RM scheduling policy by Shin et al. The SNRT is a modified stretching-to-NTA algorithm with the Deferrable Servers (DS) and Sporadic Servers (SS). Because the arrival time of an aperiodic task is unknown, the stretching rules are only applied when the budget of bandwidth preserving servers is exhausted. But it is inefficient when the workload of aperiodic tasks is small and the budget is larger than 0. They also proposed *Bandwidth-Based Slack-Stealing* (BSS) [8] scheme which considers the bandwidth of a scheduling server and identifies the maximum available time (MAT) for a periodic task. Even when the execution budget is larger than 0, the MAT can be calculated before the arrival time of next periodic task. Therefore, the clock speed of the periodic task can be slow down based on the MAT.

One problem of existing approaches is that they only calculate the MAT from the schedule point to NTA. To have long MAT, in this thesis, we propose an on-line DVS

algorithm named *Work-demand-based Slack-Stealing* (WSS) scheme for mixed workload real-time systems under the RM scheduling policy. The WSS integrates the bandwidth preserving servers and lpWDA [22] to compute the MAT by delaying the schedule of lower-priority periodic tasks. As a result, the MAT of high periodic tasks can be extended and the overall energy consumption can be reduced. Another problem of the existing approaches is that a periodic task with the highest priority may still run slowly even if there are some aperiodic tasks, without the execution budget, waiting in the ready queue. The WSS use the concept of slack stealing, which was originally used in mixed workload real time systems without DVS, to service aperiodic tasks when there are some aperiodic tasks, without the execution budget, waiting in the ready queue.

3.2 Comparison of Existing Inter-Task DVS Algorithms for Mixed Workload Real-time Systems

Table 2 shows the qualitative comparison of several existing inter-task DVS algorithms for mixed workload real-time systems along with the proposed WSS. The *scheduling policy* indicates an DVS algorithm uses RM or EDF scheduling policy, which are the two most popular real-time schedulers. 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. Among the tasks whose deadlines and arrival times are the same, the task with the lowest index has the highest priority. The *scaling decision* describes that the decision of the clock speed is calculated on-line or off-line. The *on-line DVS strategy* indicates that the DVS algorithms for mixed workload real-time systems belong to which on-line DVS strategies that were described in Chapter 2. The "*bandwidth-preserving servers*" indicates it combined which bandwidth-preserving servers to schedule aperiodic tasks. The metric of *energy consumption* indicates the CPU energy consumption of each DVS algorithm. The metric of *response time* represents the time interval between the arrival and completion of an

aperiodic task. We will compare the proposed WSS with the non-DVS scheme, SNRT, BSS quantitatively in chapter 5.

Table 2. Qualitative comparison of existing on-line DVS algorithms for mixed workload real-time systems.

Algorithm	Scheduling Policy	Scaling Decision	On-line DVS Strategy	Bandwidth-Preserving Servers	Energy Consumption	Response Time
EBA [23]	EDF	Off-Line	None	TBS	High	Low
WSE [8]	EDF*	On-Line	(2)	CBS	Medium	Medium
BRS [24]	EDF*	On-Line	(2)	TBS	Medium	Medium
MRS [24]	EDF*	On-Line	(2)	TBS	Medium	Medium
BSS [24]	EDF*	On-Line	(2)	TBS	Low	High
SNRT [26]	RM	On-Line	(1)	SS/DS	High	High
BSS [8]	RM	On-Line	(1)	SS/DS	Medium	Medium
WSS (proposed)	RM	On-Line	(4)	SS/DS	Low	Low

Chapter 4
Proposed WSS Algorithm

4.1 System Model, Assumptions and Notations

The target processor can change its supply voltage (V) and clock speed (S_{clk}) (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. J_i is the i^{th} aperiodic task. T_i has higher priority than T_j if $i < j$. A periodic task T_i can be specified as $T_i(P_i, W_i)$, where P_i and W_i are the period and WCET of T_i. Based on related work [8][12][14][15][16][24][26], the arrival time, period and WCET of periodic tasks are known in advance, but those of aperiodic tasks are made available only when they arrive. The relative deadline (D_i) of each periodic task instance is assumed equal to its period. All tasks are assumed to be independent.

4.2 Basic Idea

The basic idea of the proposed WSS (*Work-demand-based Slack-Stealing)* is to compute the slack time under the existence of the bandwidth preserving servers by using short-term work-demand analysis. Moreover, the WSS uses two modes of operation to reduce the response time of aperiodic tasks. It uses the concept of slack stealing to service aperiodic tasks. If there are some aperiodic tasks waiting in the ready queue, and there is no execution budget and the slack time is large than 0, the WSS will service the aperiodic tasks first and will not cause any deadline miss of periodic tasks.

4.3 Two Modes of the WSS

There are two modes of operation in the WSS: *power saving mode* and *non-power saving mode*. When the execution budget is exhausted and there are still some aperiodic tasks in the ready queue, the mode is set to the non-power saving mode. When no aperiodic task is in the ready queue or the execution budget is replenished, the mode is switched back to the power saving mode.

4.4 The Stretching Rules in Power Saving Mode

In the real-time systems with mixed workload, the slack time can still be computed by assuming that the bandwidth preserving server is an additional periodic task T_{BPS}. The WCET of T_{BPS} is Q_s and the period is T_s. The priority of T_{BPS} is according to T_s. However, T_{BPS} isn't like other periodic tasks. T_{BPS} will not be executed even if its execution time is not zero when there is no aperiodic task in the ready queue. The algorithm of the lpWDA with the DS/SS has to be modified as follows in order to be applicable to *mixed workload* real time systems:

- When a periodic tasks T_i finishes, $H_i(t)$ has to be recalculated in the lpWDA. If the bandwidth preserving server (BPS), which has the remaining execution budget q_s, has a higher-priority than T_i, the calculation of $H_i(t)$ must include q_s when T_i finishes.
- When an aperiodic task is completed, it is just seen as a preemption of T_{BPS}.
- At the deadline D_{BPS} of T_{BPS}, $H_i(t)$ of each lower priority task T_i (for $i > BPS$) has to be recomputed as $H_i(t) = H_i(t) - q_s$. This implies that T_{BPS} is completed at D_{BPS}.
- When an aperiodic task is scheduled for execution, no computation of slack time is needed.

Therefore, in the power saving mode, we calculate the clock speed by following two stretching rules:

- Stretching rule for a periodic task T_i: The clock speed of T_i can be calculated as

follows:

$$\frac{w_i^{rem}(t)}{max(0, slack_i(t)) + w_i^{rem}(t)} \times S_0$$

where S_0 is the initial clock speed.

- Stretching rule for an aperiodic task: If there is no periodic task in the ready queue, execute the aperiodic task at the clock speed of

$$\frac{q_s}{max(min(NTA, R, D_{BPS}) - t, q_s)} \times S_0$$

where the next periodic task arrival time (NTA), the next replenishment time (R) of the bandwidth preserving server, and the deadline (D_{BPS}) of T_{BPS} are known in advance and t is the start time of the aperiodic task.

If there is any periodic task in the ready queue, the clock speed of aperiodic task is S_0.

In this way, $H_i(t)$ of periodic task T_i is still an overestimated value for a safe estimation on available slack time, because the consumed execution budget of bandwidth preserving server will not exceed Q_s in a period of T_s. Fig. 3 is the modified UpdateLoadInfo procedure of Fig. 1 for the WSS, where $\Delta budget$ is the amount of consumed execution budget.

We give an example to illustrate the operation of WSS. Assume there are two periodic tasks $T_a(6,1)$ and $T_b(8,2)$. Fig. 4 (a) shows two aperiodic tasks are serviced by a DS(5,1) without DVS. At $t = 1$, an aperiodic task J_1 arrives. Because the execution budget is large than zero and the DS has the highest priority, J_1 can be serviced immediately and the execution budget of DS is consumed at a rate of the clock speed per unit time. At $t = 5$, the execution budget is replenished. If any aperiodic task arrives between $t = 2$ and $t = 5$, the aperiodic task will be serviced by the background priority which was described in Chapter 2. Fig. 4 (b) and (c) show how to use the short-term work-demand analysis with a DS. When a periodic task is scheduled for execution, the calculation of slack time has to consider the execution budget of the DS. Because T_s of the DS is 5, T_{BPS} has the highest priority. Therefore, we let $T_{BPS} = T_1$, T_a

= T_2, and T_b = T_3. At t = 0 , $H_0(0)$ = 0, $H_1(0)$ = 2, $H_2(0)$ = 4, and then $slack_2(0)$ = 2 can be derived by the CalcSlackTime() of Fig. 1. And the clock speed of T_2 is 1 / (2 + 1) × S_0 = 1/3 × S_0. If $T_{2,1}$ completes at t =1, $H_2(1)$ and $H_3(1)$ will be recalculated. Thus, $H_2(1)$ = 2 + 1 = 3 (1 from q_s) and $H_3(1)$ = 3 according to Fig. 3. At this time, an aperiodic task J_1 arrives, and it will complete at t = 2 as shown in Fig. 4(d). Then $H_2(2)$ and $H_3(2)$ are 2 and 2 according to Fig. 3.

Function UpdateLoadInfo()

IF (PERIODIC TASK COMPLETION) **THEN**

$ud_x = ud_x + P_x$

IF x < BPS **THEN**

$$H_x(t) = \sum_{i=1}^{x-1}\left(\left\lfloor \frac{ud_x - \varepsilon}{P_i} \right\rfloor - \left\lceil \frac{t+\varepsilon}{P_i} \right\rceil + 1\right) \times W_i$$

ELSE IF x > BPS **THEN**

$$H_x(t) = \sum_{i=1}^{x-1}\left(\left\lfloor \frac{ud_x - \varepsilon}{P_i} \right\rfloor - \left\lceil \frac{t+\varepsilon}{P_i} \right\rceil + 1\right) \times W_i \quad + q_s$$

END IF

LOOP each task T_i, $i = x + 1$ until $i = n$

$H_i(t) = H_i(t) - w_x^{rem}(t)$

END LOOP

$w_x^{rem}(t) = 0$

ELSE IF(PERIODIC TASK PREEMPTION) **THEN**

$w_x^{rem} = w_x^{rem} - w_{done}$

LOOP each task T_i, $i = x + 1$ until $i = n$

$H_i(t) = H_i(t) - w_{done}$

END LOOP

ELSE IF(APERIODIC TASK PREEMPTION **OR** COMPLETION) **THEN**

$w_{BPS}^{rem} = w_{BPS}^{rem} - \Delta budget$

LOOP each task T_i, $i = BPS + 1$ until $i = n$

$H_i(t) = H_i(t) - \Delta budget$

END LOOP

ELSE IF(D_{BPS}) **THEN**

LOOP each task T_i, $i = BPS + 1$ until $i = n$

$H_i(t) = H_i(t) - q_s$

END LOOP

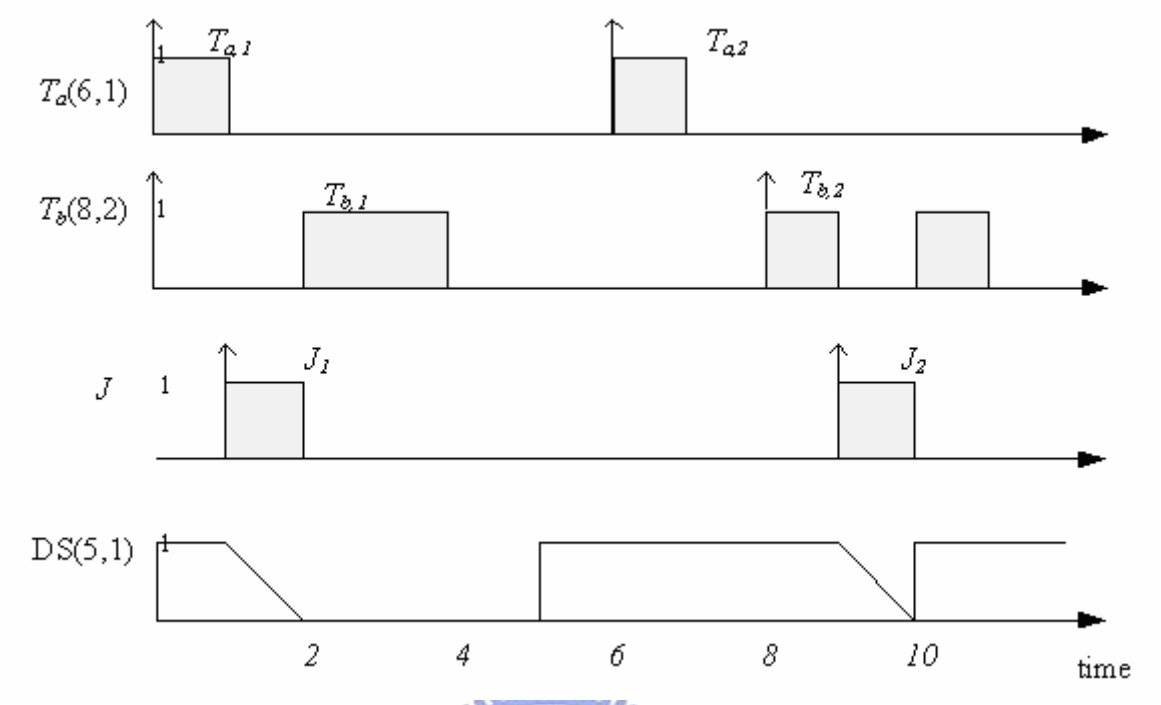
$ud_{BPS} = ud_{BPS} + P_{BPS}$

$$H_{BPS}(t) = \sum_{i=1}^{BPS-1}\left(\left\lfloor \frac{ud_{BPS} - \varepsilon}{P_i} \right\rfloor - \left\lceil \frac{t+\varepsilon}{P_i} \right\rceil + 1\right) \times W_i$$

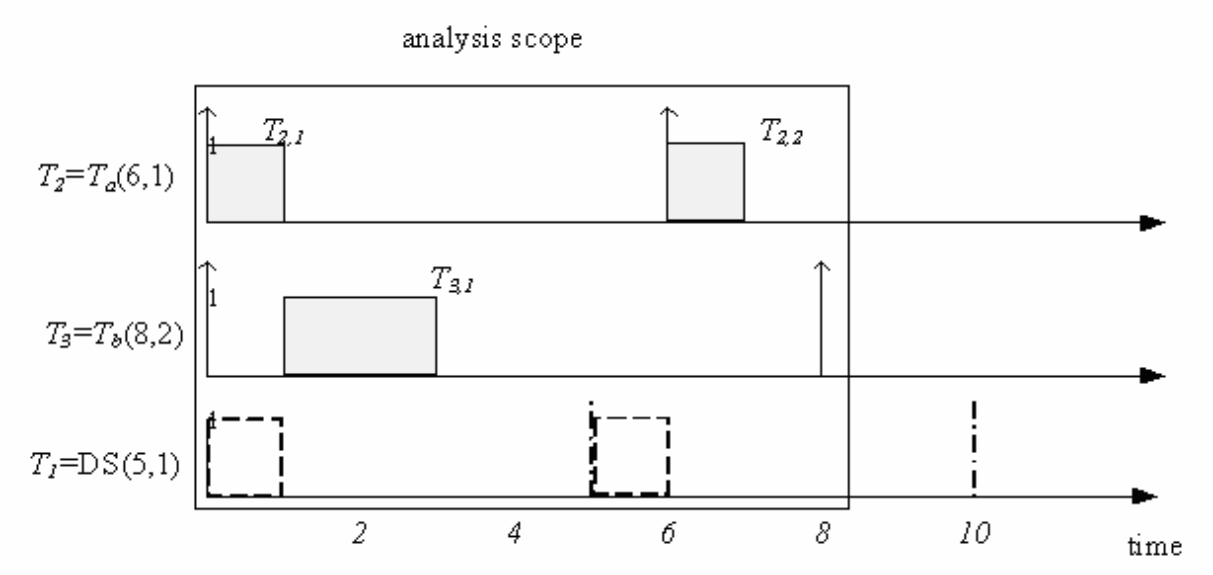
$w_{BPS}^{rem}(t) = Q_s$

END IF

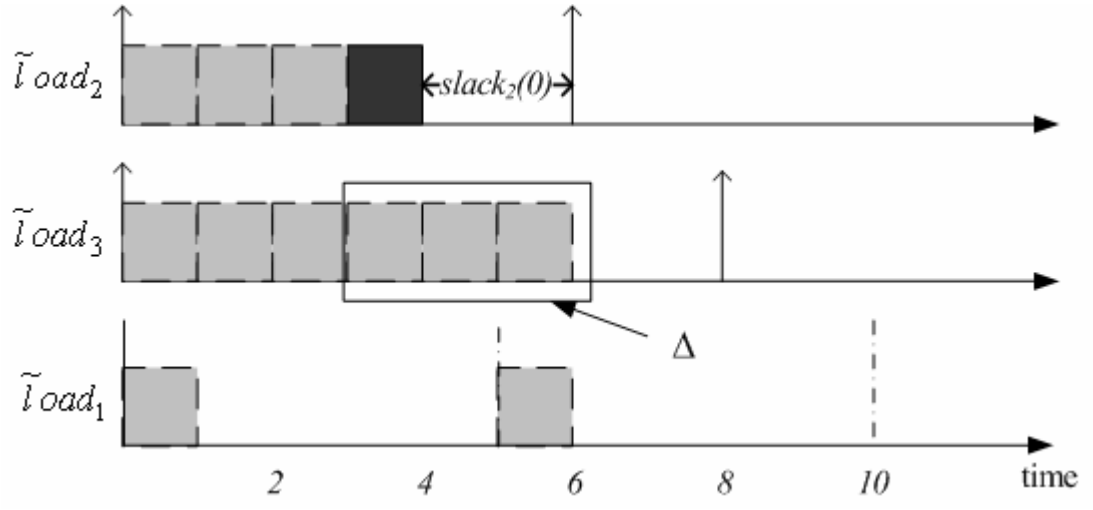
Fig. 3. The UpdataLoadInfo procedure in the WSS.


$T_a(6,1)$
$T_{a,1}$
$T_{a,2}$
$T_b(8,2)$
$T_{b,1}$
$T_{b,2}$
J
J_1
J_2
DS(5,1)
2
4
6
8
10
time

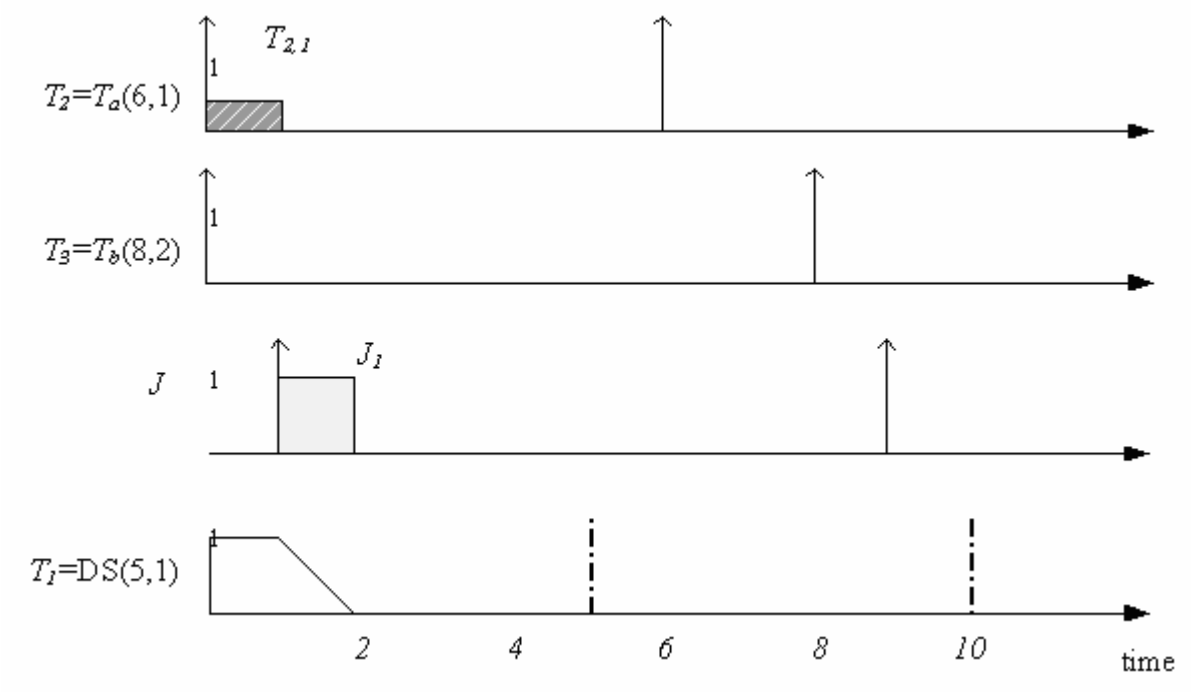
(a)


analysis scope
$T_2=T_a(6,1)$
$T_{2,1}$
$T_{2,2}$
$T_3=T_b(8,2)$
$T_{3,1}$
T_1=DS(5,1)
2
4
6
8
10
time

(b)



(c)



(d)

Fig. 4. An example of work-demand-based slack-stealing scheme: (a) shows that tasks are scheduled with a DS without DVS. (b) shows the analysis scope with a DS. (c) shows the load estimation of each task at $t = 0$. (The gray-boxes mean the amount of work from higher or same priority tasks, and the black-boxes mean the amount of work from lower priority tasks.) (d) shows an aperiodic task completed at $t = 2$.

4.5 The Serviced Rules in Non-Power Saving Mode

If an aperiodic task arrives and no other aperiodic task is in the ready queue in the non-power saving mode, the WSS computes the slack time of a periodic task which has the highest priority in the ready queue and services the aperiodic task according to the slack time. For instance, if J_y arrives at time t without the execution budget and no other aperiodic task in the ready queue, $slack_i(t)$ of the periodic task T_i, which has the highest priority in the ready queue has to be calculated. If $slack_i(t)$ *is* greater than zero, it means the schedulability of T_i will not be affected even if the execution time of T_i is deferred by the amount of $slack_i(t)$. Hence, we set $[t, min(t + slack_i(t), NRT)]$ as an time interval allowed to execute aperiodic tasks. If no aperiodic task is in the ready queue, $slack_i(t)$ can be reclaimed for T_i and the mode can be switched back to the power saving mode. In other words, aperiodic tasks can be executed first if $slack_i(t)$ is larger than zero in the non-power saving mode, and in this way the response time can be reduced when the actual workload of aperiodic tasks is large than the execution budget. On the other hand, if $slack_i(t)$ is smaller than or equal to zero, T_i will be executed using clock speed S_0. If T_i completes early, J_y has an opportunity to be serviced according to the slack time of the new scheduled periodic task.

If J_y arrives at time t without execution budget but another aperiodic task is already in the ready queue, then task J_y will be placed in a queue of pending tasks according to the FIFO (first in, first out) scheduling policy. In addition, if another higher priority task T_j ($j < i$) arrives during a time interval allowed to execute aperiodic tasks, this time interval has to be recalculated.

Moreover, if an aperiodic task is serviced in the non-power saving mode, the $H_i(t)$'s of the other lower priority tasks ($BPS < i$) need not be modified. The period of servicing aperiodic tasks looks like an idle period of the schedule in the non-power saving mode.

Chapter 5
Simulation Results and Discussion

5.1 Simulation Model

Aperiodic tasks were generated by the exponential distribution with interarrival time ($1/\lambda$) and service time ($1/\mu$). We used a fixed value μ and varied λ to control the workload ($\rho = \lambda/\mu$) of aperiodic tasks under a fixed utilization U_p of periodic tasks [8]. There are three periodic tasks in Table 3. The period of each task is 6, 8, and 14, respectively and the WCET of each task is 0.5, 1.0, and 1.283, respectively. The utilization U_p of periodic tasks is 0.3 ((0.5 / 6) + (1.0 / 8) + (1.283 / 14)) [8].

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 [24]. In the experiments, the voltage scaling overhead is assumed negligible both in the time delay and power consumption [8]. The total amount of U_s and U_p must be smaller than U_{lub}, which is the least upper bound of schedulable utilization. U_{lub} is *1* with EDF scheduling and $n(2^{1/n}-1)$ for n tasks with RM scheduling.

In order to experimentally evaluate the performance of the proposed algorithms, WSS, we implemented the following existing schemes for performance evaluation:

(1) PD scheme [8]: Aperiodic tasks were assumed to be serviced by the SS. It was also 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 [8].

(2) Stretching-to-NRT (SNRT) scheme [25]: It was described in Chapter 3.

(3)Bandwidth-Based Slack-Stealing (BSS) scheme [8]: It was described in Chapter 3.

For all experiments, all tasks were assigned an initial clock speed $S_0 = (U_p + U_s) S_m / U_{lub}$, where S_m is the maximum clock speed [8]. In the following, we evaluated the performance of each scheme in terms of energy consumption and response time. The energy consumption and response time are normalized to those of PD [8].

Table 3. Periodic task set description.

Task Set (millisecond)		
Task	Period	WCET
T_1	6	0.5
T_2	8	1.0
T_3	14	1.283

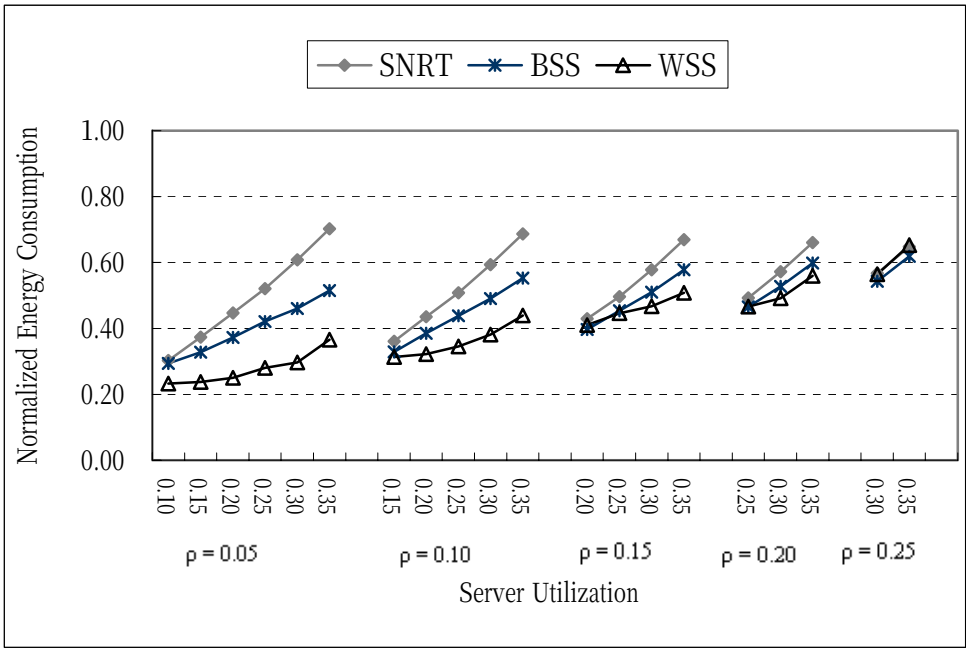
5.2 Effects of Different Workloads of Aperiodic Tasks on Performance

BCET is assumed to be 10% of WCET, and ρ is ranging from 0.05 to 0.25 (λ = 0.05 ~ 0.25 and μ = 1.0) [8]. The server utilization U_s is set from 10% ~ 35%, where U_s is controlled by changing the value of T_s with a fixed Q_s value [8]. The WSS is compared with the others three schemes under different workloads (server utilization) of aperiodic tasks. Fig. 5 shows the effects of different workloads (server utilization) on (a) normalized energy consumption, (b) normalized response time, and (c) normalized energy * response time of all schemes under different server utilization. From the simulation results, we have the

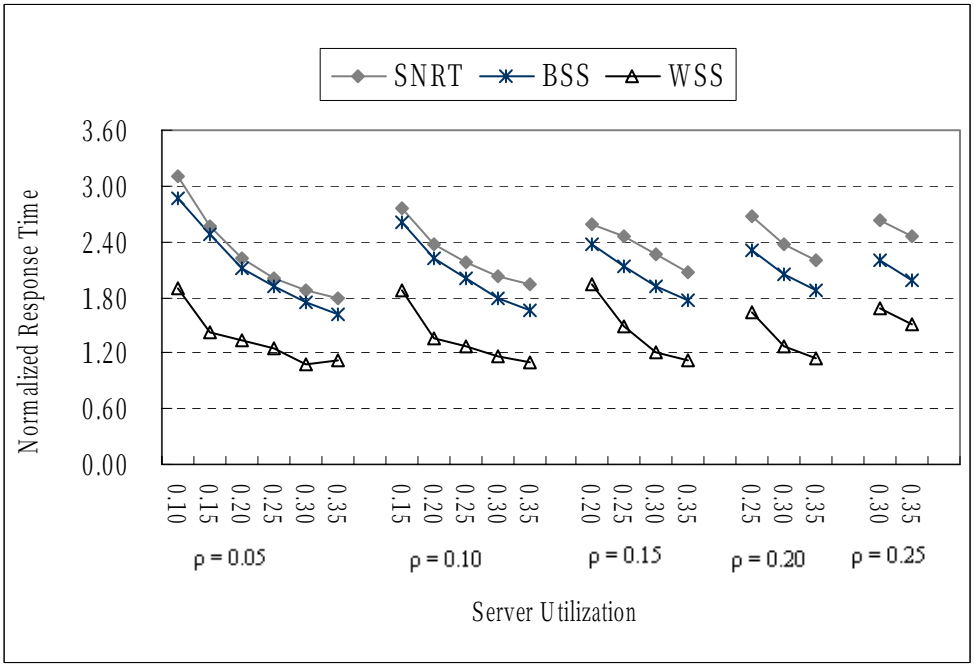
following observations:

- As the server utilization (U_s) increases, the energy consumption of all schemes increases because the initial clock speed S_0 increases.
- When the actual workload of aperiodic tasks is close to the server utilization, the energy consumption of the WSS is close to that of the BSS. The reason is that the slack time of the WSS will be used to service aperiodic tasks when the execution budget is exhausted.
- When the workload of aperiodic tasks is close to server utilization, the response time of all schemes increases.
- The WSS reduces the energy consumption, in average, by 58%, 22%, and 12% compared with the PD, SNRT, and BSS, respectively.
- The WSS reduces the response time, in average, by 38% and 31% compared with the SNRT and BSS, respectively. The PD has the smallest response time.

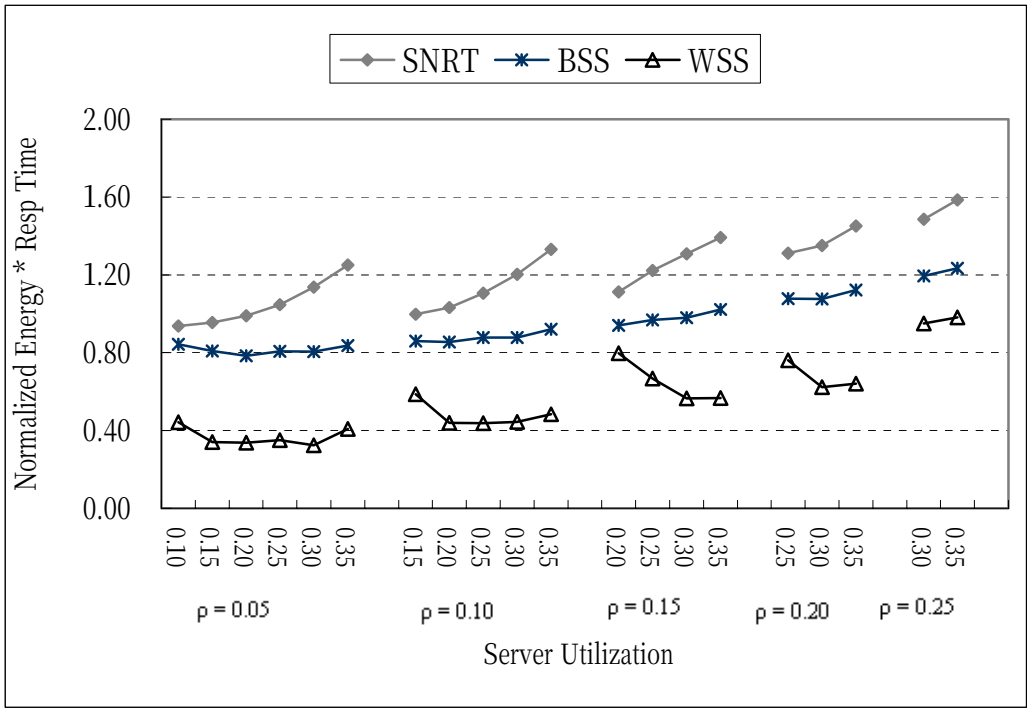
The normalized energy * response time [24] is a performance metric that combines the two important dimensions, energy consumption and response time, of the mixed workload real-time systems. As Fig. 5 (c) shown, the WSS has better performance than the other three schemes in terms of normalized energy * response time.


SNRT
BSS
WSS
Normalized Energy Consumption
1.00
0.80
0.60
0.40
0.20
0.00
0.10 0.15 0.20 0.25 0.30 0.35
0.15 0.20 0.25 0.30 0.35
0.20 0.25 0.30 0.35
0.25 0.30 0.35
0.30 0.35
ρ = 0.05
ρ = 0.10
ρ = 0.15
ρ = 0.20
ρ = 0.25
Server Utilization

(a)


SNRT
BSS
WSS
Normalized Response Time
3.60
3.00
2.40
1.80
1.20
0.60
0.00
0.10 0.15 0.20 0.25 0.30 0.35
0.15 0.20 0.25 0.30 0.35
0.20 0.25 0.30 0.35
0.25 0.30 0.35
0.30 0.35
ρ = 0.05
ρ = 0.10
ρ = 0.15
ρ = 0.20
ρ = 0.25
Server Utilization

(b)



(c)

Fig. 5. Effects of the workload of aperiodic tasks on (a) normalized energy consumption (b) normalized response time (c) normalized energy * response time

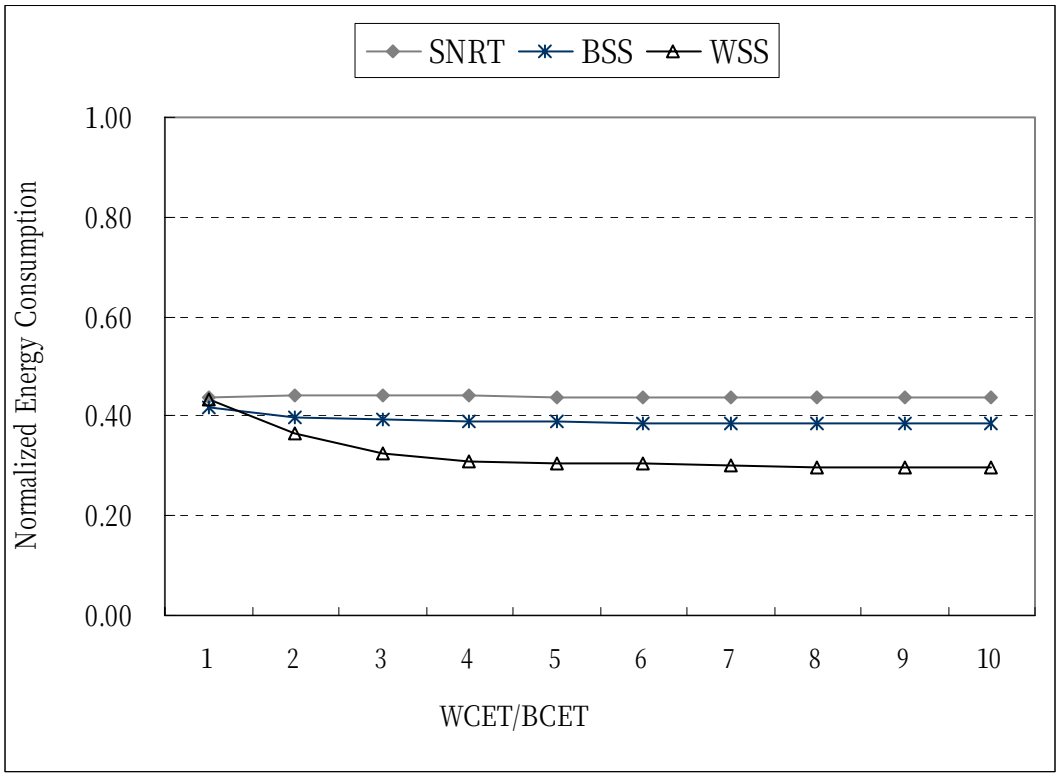
5.3 Effects of Different WCET/BCET Ratios of Periodic Tasks on Performance

Assume that the workload of aperiodic tasks ρ is 0.1 [24], server utilization U_s is 0.2, and the WCET/BCET ratio varies from 1 to 10 [20]. Fig. 6 shows the effects of different workloads (server utilization) on the (a) normalized energy consumption, (b) normalized response time, and (c) normalized energy * response time of all schemes. From these simulation results, we have the following observations:

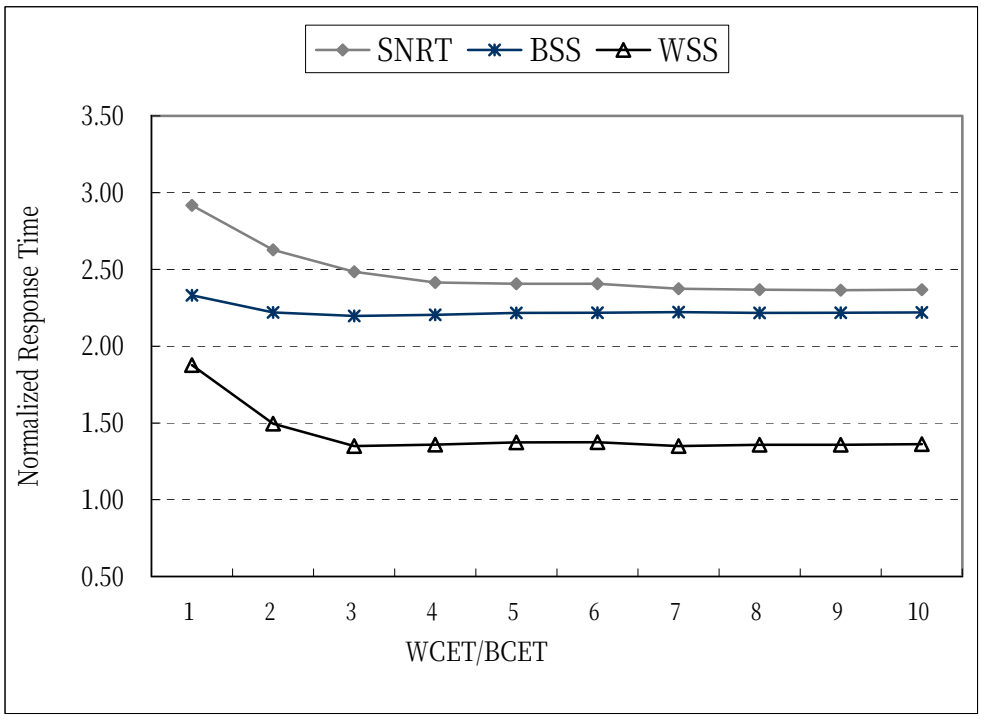
- The normalized energy consumption of the SNRT and BSS decreases very little with respect to the increase of the WCET/BCET ratio of periodic tasks.
- As the WCET/BCET ratio increases, the normalized energy consumption of the

WSS decreases. The reason is that the slack time increases as WCET/BCET increases, and the WSS is more efficient in utilizing the slack time than the other three schemes.

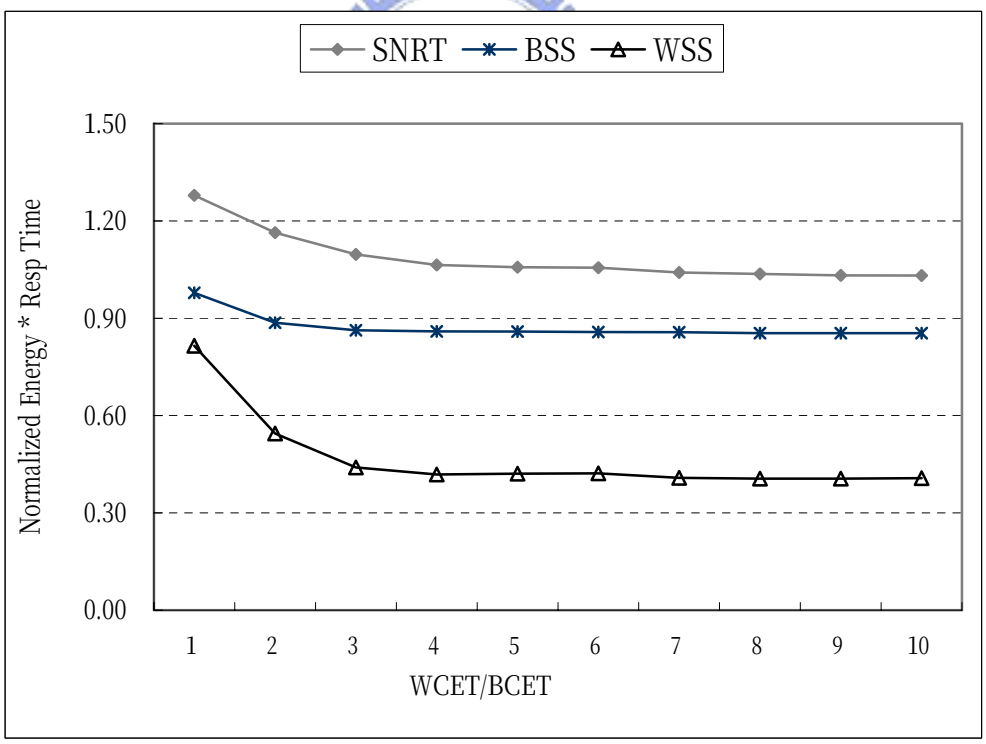
- The WSS reduces the energy consumption by an average of 66%, 23%, and 13% compared with the PD, SNRT, and BSS, respectively.
- The WSS reduces the response time by 38%, and 31% compare with the SNRT and BSS, respectively. The PD has the smallest response time.
- The WSS has better performance than the other three schemes in terms of normalized energy * response time.



(a)



(b)



(c)

Fig. 6. Effects of the WCET/BCET ratio of periodic tasks on (a) normalized energy consumption (b) normalized response time (c) normalized energy * response time.

5.4 Practical Issues

In this thesis, we assumed that the CPU speed can be changed continuously for DVS. However, real CPUs only support a finite number of speed levels. The WSS can be adapted to real CPUs by choosing the lowest speed level that is equal to or greater than the speed value suggested by the WSS [20]. The more speed levels CPUs provide, the energy consumption is more close to that of ideal CPUs [20]. In the proposed WSS scheme, the time complexity to estimate slack time varies from $O(1)$ (in the best case) to $O(n)$ (in the worst case) with respect to the priorities of the scheduled tasks [22]. This is the overhead of the proposed algorithm.

Chapter 6
Conclusions and Future Work

6.1 Concluding Remarks

In this thesis, we have presented an on-line dynamic voltage scaling (DVS) algorithm, called WSS, for mixed workload real-time systems. The WSS not only addresses the energy consumption for mixed workload real-time systems, but also consider the response time of aperiodic tasks. The WSS integrates the low power work-demand analysis (lpWDA [22]) to set a suitable clock speed and uses slack time to service aperiodic tasks when the actual workload of aperiodic tasks is close to the server utilization. The WSS can use the slack time more efficient than existing approaches, because it enlarges the slack time by delaying the schedule of lower-priority periodic tasks in near future as late as possible. Simulation results have shown that the WSS can effectively reduce the energy consumption, response time, and energy consumption * response time, in average, by 23%, 38%, 52% compared to the SNRT and 13%, 31% , 40% compared to the BSS, respectively.

6.2 Future Work

In the future, we may take more factors, such as number of preemptions, extra time and energy consumption overhead for voltage transitions and the realistic CPU speed levels, into consideration to make the proposed algorithm more feasible to realistic systems. The influences of these factors on energy consumption and response time deserve to further study.

Bibliography

[1] L. Benini, A. Bogliolo, and G. De Micheli, “A survey of design techniques for system level dynamic power management,” I*EEE Transactions on Very Large Scale Integration Systems*, Vol.8, No.3, pp. 299-316, 2000.

[2] Intel XScale® Technology, “Intel. PXA270 processor electrical, mechanical, and thermal specification,“ http://www.intel.com/design/intelxscale/

[3] Intel® Xeon® Processor, http://www.intel.com/products/processor/xeon/

[4] Trasmeta Corporation, “TN5400 processor specification,” http://www.transmeta.com/crusoe/

[5] Mobile AMD AthlonTM 64 processor, http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_10220_10221,00.html

[6] G. Carpenter, “Low power SOC for IBM's PowerPC information appliance platform,” in http://www.research.ibm.com/arl.

[7] B. Moyer, “Low-power design for embedded processors,” in *Proceedings of IEEE*, Volume 89, Issue 11, pp. 1576-1587, November 2001.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] T.S. Tia, “Utilizing slack time for aperiodic and sporadic request scheduling in Real-Time Systems,” Technical Report No. UIUCDCS-R-95-1906, University of Illinois, April, 1995.

[19] C. Im, and S. Ha, "Dynamic voltage scaling for real-time multi-task scheduling using buffers," in *Proceedings of the 2004 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems*, pp. 88-94, 2004.

[20] 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.

[21] 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.

[22] W. Kim, J. Kim, and S. L. Min, "Dynamic voltage scaling algorithm for fixed-priority real-time systems using work-demand analysis," in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 396-401, 2003.

[23] 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.

[24] 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.

[25] X.S. Hu, G. Quan, B. Mochocki, "A realistic variable voltage scheduling model for real-time applications," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 726-731, 2002.

[26] 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.