Real-Time Syst (2009) 41: 222–255
DOI 10.1007/s11241-009-9068-7

A DVS-assisted hard real-time I/O device scheduling algorithm

Edward T.-H. Chu · Tai-Yi Huang · Cheng-Han Tsai · Jian-Jia Chen · Tei-Wei Kuo

Published online: 12 February 2009

Abstract The I/O subsystem has become a major source of energy consumption in a hard real-time monitoring and control system. To reduce its energy consumption without missing deadlines, a dynamic power management (DPM) policy must carefully consider the power parameters of a device, such as its break-even time and wake-up latency, when switching off idle devices. This problem becomes extremely complicated when dynamic voltage scaling (DVS) is applied to change the execution time of a task. In this paper, we present COLORS, a composite low-power scheduling framework that includes DVS in a DPM policy to maximize the energy reduction on the I/O subsystem. COLORS dynamically predicts the earliest-access time of a device and switches off idle devices. It makes use of both static and dynamic slack time to extend the execution time of a task by DVS, in order to create additional switch-off opportunities. Task workloads, processor profiles, and device characteristics all impact the performance of a low-power real-time algorithm. We also identify a key

E.T.-H. Chu (✉) · C.-H. Tsai
Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan 300, ROC
e-mail: edwardchu@cs.nthu.edu.tw

C.-H. Tsai
e-mail: chtsai@cs.nthu.edu.tw

T.-Y. Huang
Google Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA
e-mail: tyhuang@google.com

J.-J. Chen
Swiss Federal Institute of Technology (ETH), Zurich, Switzerland
e-mail: jchen@tik.ee.ethz.ch

T.-W. Kuo
Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan 106, ROC
e-mail: ktw@csie.ntu.edu.tw

metric that primarily determines its performance. The experimental results show that, compared with previous work, COLORS achieves additional energy reduction up to 20%, due to the efficient utilization of slack time.

Keywords Real-time embedded systems · Dynamic power management · Dynamic voltage scaling

1 Introduction

A real-time monitoring and control system periodically senses its environment and takes action when an event occurs. Such a system, as shown in Fig. 1, consists of a processor and a set of I/O devices called sensors. The processor executes tasks that periodically poll sensors for data, run a control algorithm and activate the actuators accordingly. A task running on a hard real-time system must return response before its deadline to avoid any critical failure. Examples of hard real-time monitoring systems include medical-care systems (Khatib et al. 2006), critical environmental monitoring systems (McIntire et al. 2006; Alexandropoulos et al. 2005), and driver-vigilance monitoring systems (Bergasa et al. 2006). Many sensor devices consume more power and energy than a processor. For instance, a gas detector[1,2] consumes 660 mW, a temperature/humidity sensor[3] consumes 600 mW, while an ARM8 processor only consumes 380 mW (Im et al. 2004). Consequently, the problem of minimizing the energy consumption of the I/O subsystem has become an important issue in developing a hard real-time low-power system.

A number of real-time low-power scheduling algorithms have been developed recently. Many of them, such as Pillai and Shin (2001), Kim et al. (2002), apply the

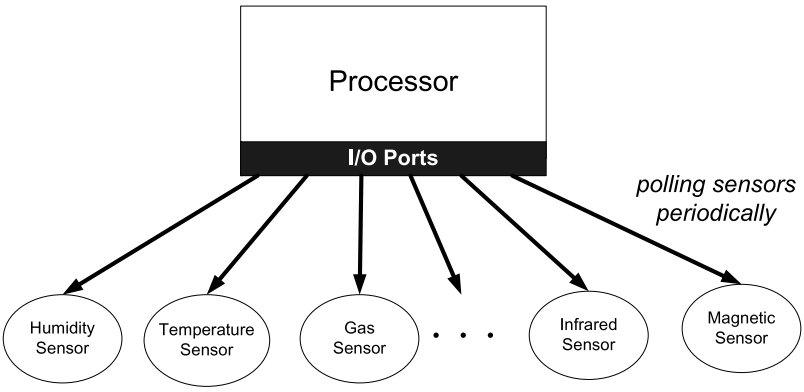


Fig. 1 A real-time monitoring system

[1] Axetris Microsystems Inc. (2006) http://www.axetris.com/downloads/f25irsstandard.pdf.

[2] Ohmic Instruments Inc. (2004) http://www.figaro.co.jp/en/pdf/825ProductInfo1104.pdf.

[3] Rixen Technology Inc. (2005) http://www.rixen.com.tw/product/trh300_e.htm.

technology of dynamic voltage scaling (DVS) to reduce the energy consumption of the processor. While meeting deadlines, these real-time DVS algorithms assume that every device is always switched on and no energy reduction is made in the I/O subsystem. A dynamic power management (DPM) policy switches off idle devices to save energy. Since the DVS technology can be used to extend the execution time of a non-I/O interval and make idle devices sleep longer, a real-time DPM policy that makes no use of DVS, such as Swaminathan and Chakrabarty (2003), Cheng and Goddard (2006), achieves limited energy reduction as well. Because it takes extra energy and time to change power states, a device can be switched off to save energy only when the length of its idle interval is longer than its break-even time, which is defined as the minimum length of an idle period to save energy by switching a device off and on Lu et al. (2000). Making a switching decision of a device becomes extremely difficult when DVS is applied to change the execution time of a task. Several approaches have been proposed to deal with the integration of real-time DVS and DPM. Kim and Ha (2001) proposed a slot-based approach for a one-device system. This approach did not consider the latency incurred in waking up and switching off a device. Both work of Jejurikar and Gupta (2004), Cheng and Goddard (2005) applied DVS on an I/O interval and executed it at a reduced speed to save energy. However, since a device is periodically accessed by tasks, extending an I/O interval may lead to a shorter idle period between accesses and may avoid a switch-off opportunity. Furthermore, the work of Jejurikar and Gupta (2004) assumed a fixed execution time for each task and cannot make use of dynamic slack when a task completes earlier.

In this paper, we present COLORS, a COmposite LOw-power Real-time Scheduling framework that includes DVS and a DPM policy. COLORS is designed to minimize the energy consumption of the I/O subsystem for a system where its CPU power is negligible. In other words, COLORS is better suited for a system on which the I/O subsystem dominates the energy consumption. The technology of DVS is primarily used in this framework to extend the length of idle intervals. We classify the execution of a job into a sequence of computational intervals and peripheral intervals. The real-time DPM policy of COLORS, called OPADS, carefully considers all power parameters of a device, including its break-even time and shutdown and wake-up latency. The DVS extension of OPADS makes use of both static and dynamic slack time to create additional switch-off opportunities as well as reduce the processor energy consumption. The efficient use of slack time allows COLORS to switch off idle devices for longer periods. COLORS determines a switching decision of a device according to its predicted earliest-access time. An accurate prediction requires large computational overhead. Finally, we present F-COLORS, a simplified version of COLORS, that significantly reduces its run-time complexity.

Many system parameters may impact the effectiveness of a real-time DPM policy. These parameters include task workloads, processor profiles, and device characteristics. Previous studies considered only a small subset of these parameters in their performance evaluation. Instead, we take a systematic approach to analyze all parameters and identify a key metric that primarily determines their performance. We compare COLORS with four algorithms, each of which is derived from an existing work, Swaminathan and Chakrabarty (2003), Jejurikar and Gupta (2004), Cheng and Goddard (2005, 2006). These algorithms well represent the state-of-the-art technology in the integration of DPM and DVS for hard real-time systems. The simulation

Table 1 The list of notations and definitions

Symbol	Definition	Symbol	Definition
τ_i	A real-time task	M	The number of devices
$\tau_{i,j}$	The j-th job of τ_i	N	The number of real-time tasks
r_i	The release time of τ_i	L_i	The peripheral interval list of τ_i
$r_{i,j}$	The release time of $\tau_{i,j}$	$L_{i,j}$	The j-th peripheral interval of τ_i
p_i	The period of τ_i	$\eta_{i,j}$	The device used by $L_{i,j}$
e_i	The WCET of τ_i	$\alpha_{i,j}$	The local waiting time of $L_{i,j}$
b_i	The BCET of τ_i	$\beta_{i,j}$	The length of $L_{i,j}$
η_i	A peripheral device	$W_{i,j}$	The predicted earliest-access time of $L_{i,j}$
w_i	The wake-up latency of η_i	t_c	The current scheduling instant
z_i	The shutdown latency of η_i	t_n	The next scheduling instant
v_i	The break-even time of η_i	t_f	The previous scheduling instant
ρ	The ready queue	$\varphi(\tau_{i,j})$	The priority of $\tau_{i,j}$
π	The expired queue		

results show that COLORS delivers significantly more energy reduction in all system configurations we tested. The reduction is as much as 20%. The performance of F-COLORS is only slightly worse than COLORS even at its largely reduced complexity.

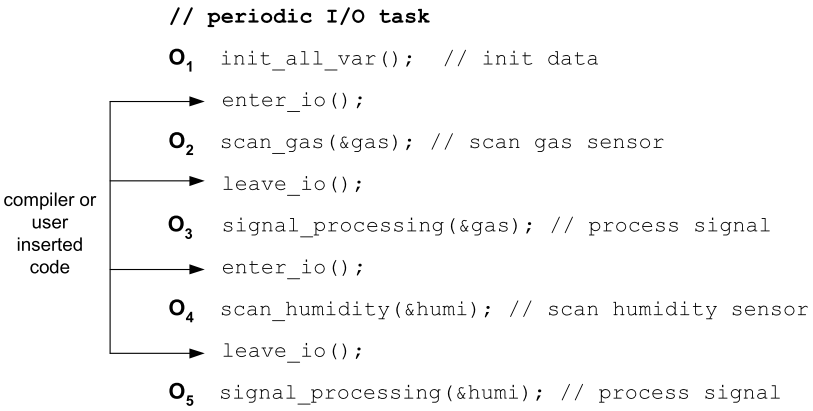
The rest of this paper is structured as follows. Section 2 describes the system model and gives an overview of a COLORS system. Section 3 presents our real-time DPM policy, called OPADS (Online Power-Aware Device Scheduling). Section 4 describes its DVS extension. Section 5 presents the complexity analysis and F-COLORS. The simulation results are described in Sect. 6. Section 7 discusses related work. Finally, Sect. 8 concludes this paper and discusses future work.

2 System model

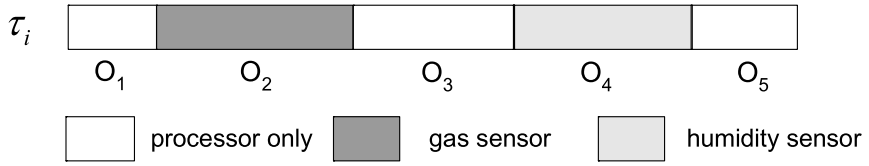
We design COLORS for a real-time task model on a system where the processor supports DVS and each peripheral device supports two operational modes: a high-power working mode and a low-power sleeping mode. In the following, we first describe the task model, the processor model, and the peripheral model. We next give an overview of a COLORS system.

2.1 Task model

The task model consists of a set of periodic real-time tasks. Let $\mathcal{T}$ denote the set of N periodic tasks, $\{\tau_1, \tau_2, \ldots, \tau_N\}$. Each periodic task τ_i is a sequence of jobs released at constant intervals called periods and $\tau_{i,j}$ denotes the j-th job of τ_i. All tasks and jobs are independent and preemptable. We use $(r_i, p_i, L_i, b_i, e_i)$ to denote a periodic task τ_i, where r_i is its release time and p_i is its period. In addition, b_i and e_i are the best-case execution time (BCET) and the worst-case execu-



(a) An environment monitoring task



(b) A sequence of computational and peripheral intervals

Fig. 2 A periodic task and its peripheral intervals

tion time (WCET) at the maximum processor speed. A number of timing analysis tools are available to obtain b_i and e_i, such as Engblom et al. (2001), Huang et al. (2003). The release time of $\tau_{i,j}$, denoted by $r_{i,j}$, is $r_i + (j-1) * p_i$. We classify the execution of a job into a sequence of *computational* intervals and *peripheral* intervals. A computational interval requires only the processor to execute. In contrast, a peripheral interval requires both the processor and a device to be switched on, as commonly assumed by previous work (Swaminathan and Chakrabarty 2003; Cheng and Goddard 2005, 2006; Jejurikar and Gupta 2004; Zhuo and Chakrabarti 2005). Finally, L_i is used to denote the set of peripheral intervals of τ_i.

Figure 2 shows an example of one task that is activated periodically to poll data from a gas sensor and a temperature sensor. Figure 2(a) gives the code of this task. We divide its execution into five intervals, as shown in Fig. 2(b). The second and fourth intervals are peripheral intervals and the other intervals are computational intervals. The beginning and the end mark of a peripheral interval can be automatically inserted by the compiler or manually by the user.

2.2 Processor model

Our system model makes use of a processor capable of DVS. Following previous work (Jejurikar and Gupta 2004; Cheng and Goddard 2005; Kim and Ha

2001), we assume that the processor has finite speed levels. In addition, the processor speed is linearly related to its supply voltage and the power consumption of a processor increases cubically with its processor speed (Lee and Shin 2004; Han and Li 2005). The overhead of changing processor speed, if not negligible, can be included for consideration when COLORS applies its DVS extension to determine a reduced speed. Because COLORS is designed to maximize the energy reduction on the I/O subsystem, peripheral intervals are always executed at the maximum speed, in order to switch off idle devices at an earliest instant. Depending on current workloads and available slack time, the processor speed of a computational interval may be reduced to create additional switch-off opportunities and reduce the processor energy consumption. In other words, our DVS scheme is an IntraDVS (Kim et al. 2002) one that we may change processor speeds at beginnings and ends of peripheral intervals, in addition to arrivals and completions of jobs.

2.3 Peripheral model

Let $\mathcal{H}$ denote the set of M peripheral devices, $\mathcal{H} = \{\eta_1, \eta_2, \ldots, \eta_M\}$. Each device supports a high-power working mode and a low-power sleeping mode. A device processes a request only at its working mode. We use w_i to denote the time to switch η_i from its sleeping mode to working mode, or its wake-up latency, and z_i to denote the time from its working mode to sleeping mode, or its shutdown latency. In addition, we use $P_{A,i}$ to denote the power consumption of η_i in working mode, and $P_{O,i}$ denote that in sleeping mode. Furthermore, $P_{F,i}$ represents the average power consumption of η_i from working to sleeping mode, and $P_{E,i}$ the average power consumption from sleeping to working mode. The break-even time of η_i, denoted by v_i, is defined as the minimum length of an idle period to save energy by switching η_i off and on Lu et al. (2000). According to the definition, we have

$$P_{A,i} \cdot v_i = (P_{F,i} \cdot z_i + P_{E,i} \cdot w_i) + P_{O,i} \cdot (v_i - z_i - w_i).$$

Also, v_i must be larger than $z_i + w_i$. Therefore, v_i is determined by

$$v_i = \max\left\{\frac{P_{F,i} \cdot z_i + P_{E,i} \cdot w_i - P_{O,i} \cdot (z_i + w_i)}{P_{A,i} - P_{O,i}}, z_i + w_i\right\}.$$

L_i is a list of entries, each of which represents a peripheral interval of τ_i. $L_{i,j}$ denotes the j-th entry of L_i, where $L_{i,j} = (\eta_{i,j}, \alpha_{i,j}, \beta_{i,j}, W_{i,j})$. $\eta_{i,j} \in \mathcal{H}$ is the accessed device and $\alpha_{i,j}$ is the starting time of this interval, relative to the starting time of this job. $\beta_{i,j}$ is initially set to the length of $L_{i,j}$ and is updated during run-time to record its remaining length. $W_{i,j}$ is the predicted earliest-access time of $L_{i,j}$, including preemptions from higher-priority jobs. $L_{i,j}$ is updated at each scheduling instant. Both $\alpha_{i,j}$ and $\beta_{i,j}$ can be statically bounded by timing analysis tools such as Engblom et al. (2001), Huang et al. (2003). The tighter this bound is, the more energy reduction COLORS achieves. Finally, we define a scheduling instant to be at arrival or completion of a job in order to reduce scheduling overhead.

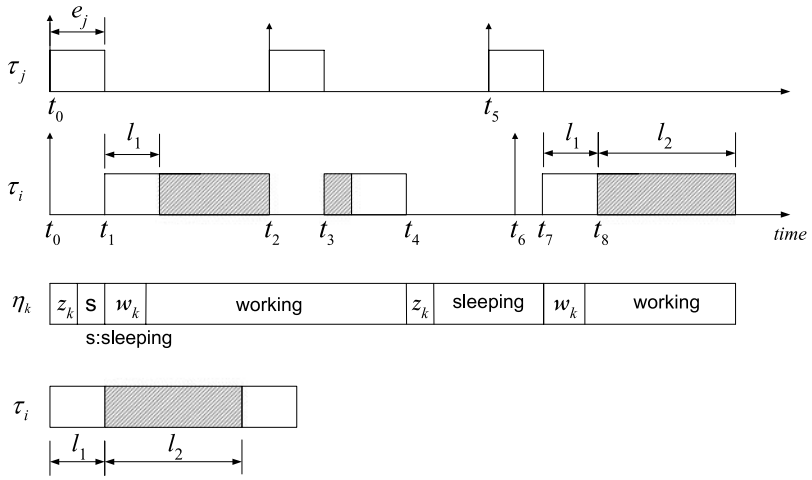


Fig. 3 A simple 2-task example

Table 2 The status of $L_{i,1}$ at each scheduling instant in Fig. 3

Time	Event	$(\eta_{i,1}, \alpha_{i,1}, \beta_{i,1}, W_{i,1})$
t_0	All jobs arrive	$(\eta_k, l_1, l_2, e_j + l_1)$
t_1	$\tau_{j,1}$ completes	(η_k, l_1, l_2, l_1)
t_2	$\tau_{j,2}$ arrives	$(\eta_k, 0, l_2 - (t_2 - t_1 - l_1), e_j)$
t_3	$\tau_{j,2}$ completes	$(\eta_k, 0, l_2 - (t_2 - t_1 - l_1), 0)$
t_4	$\tau_{i,1}$ completes	$(\eta_k, l_1, l_2, t_5 - t_4 + e_j + l_1)$

Example 1 We use a simple two-task example shown in Fig. 3 to illustrate the defined notations. The two tasks, denoted by τ_i and τ_j, execute periodically in a fixed-priority system with τ_j having a higher priority. τ_i has one peripheral interval $L_{i,1}$ and τ_j has no peripheral interval. $L_{i,1}$ accesses η_k. The status of $L_{i,1}$ at each scheduling instant between t_0 and t_4 is listed in Table 2. Let $\alpha_{i,1} = l_1$ and $\beta_{i,1} = l_2$ initially. For simplicity, we assume τ_j has a fixed execution time e_j. At t_0, $L_{i,1} = (\eta_k, l_1, l_2, e_j + l_1)$. At t_1, $\tau_{j,1}$ completes and we update $W_{i,1}$ to l_1 and keep $\alpha_{i,1}$ and $\beta_{i,1}$ unchanged. At t_2, $\tau_{j,2}$ is released and preempts $\tau_{i,1}$. Because $\tau_{i,1}$ is preempted inside $L_{i,1}$, we update $\alpha_{i,1} = 0$ and $\beta_{i,1} = l_2 - (t_2 - t_1 - l_1)$. In addition, because $\tau_{i,1}$ will resume its execution at completion of $\tau_{j,2}$, we set $W_{i,1} = e_j$. When $\tau_{j,2}$ completes at t_3, we update $W_{i,1}$ to 0. Finally, when $\tau_{i,1}$ completes at t_4, we set $\alpha_{i,1}$ and $\beta_{i,1}$ to their initial values and set $W_{i,1}$ to $(t_5 - t_4 + e_j + l_1)$.

2.4 COLORS architecture

Figure 4 shows the architecture of a COLORS system, built on top of *any* priority-driven real-time scheduler. COLORS consists of two components: a real-time DPM policy, called OPADS, and a DVS extension. OPADS is an on-line algorithm, so it does not take entire schedule as input; rather, it makes its device scheduling decision

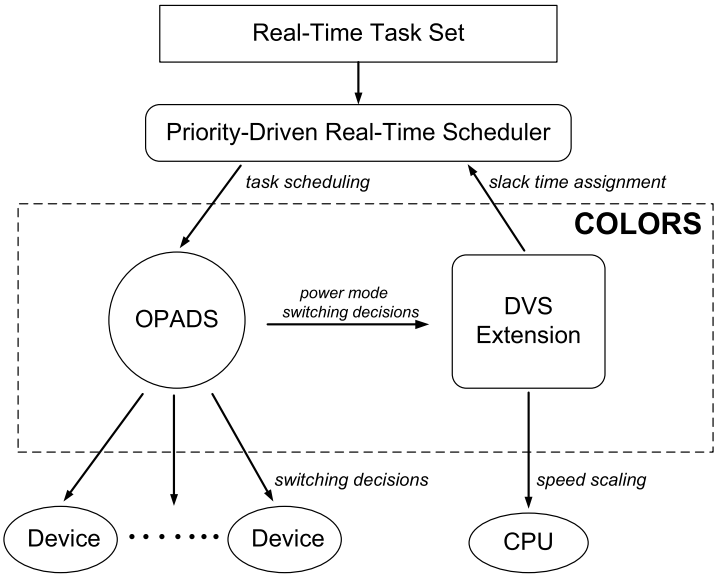


Fig. 4 The COLORS architecture

on-the fly, assuming that the underling schedule is feasible. The feasible schedule is produced by a priority-driven real-time scheduler such as Rate-Monotonic (RM) or Earliest-Deadline-First (EDF). At each scheduling instant, COLORS determines as output a set of switching decision for each device. If the next interval is a peripheral interval, we simply execute it at the maximum processor speed. Otherwise, COLORS determines the amount of available slack and applies its DVS to execute this computational interval at a reduced speed. Slack time is primarily used to create additional switch-off opportunities and secondarily for reducing the processor energy consumption. The DVS extension produces the final feasible real-time schedule which is taken by OPADS again to generate the final set of switching decisions.

3 Online power-aware device scheduling

Each invocation of OPADS first updates all peripheral lists $L_{i,j}$'s and, based on the latest status of local waiting time $\alpha_{i,j}$'s and predicted earliest-access time $W_{i,j}$'s, determines a switching decision for each device. The update of $\alpha_{i,j}$'s depends only on τ_i itself and can be determined locally. On the other hand, the update of $W_{i,j}$'s depends on not only τ_i but also higher-priority jobs that may preempt τ_i. In the following, we first describe how we determine a switching decision for each device based on latest $\alpha_{i,j}$'s and $W_{i,j}$'s. We later describe the update of $L_{i,j}$'s at each scheduling instant.

Let t_f and t_c denote the previous and the current scheduling instant, respectively. The system maintains one ready queue ρ and one expired queue π. The ready queue ρ keeps track of all released jobs sorted by their priority. When a job completes, its

Algorithm 1 OPADS: Online Power-Aware Device Scheduling

```
1:  Procedure OPADS()
2:  reset each device's decision to no-action;
3:  for all L_{i,j} do
4:      update α_{i,j} and W_{i,j};
5:      GenSwitchingDecision(L_{i,j});
6:  end for
7:  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
8:  Procedure GenSwitchingDecision(L_{i,j})
9:  W = W_{i,j}; η_k = η_{i,j};
10: if (η_k at power-mode transition) or (η_k's decision == continue-on or switch-on)
    then
11:     return;
12: end if
13: if (η_k at working mode) then
14:     if (W > υ_k and Eq. (1)) then
15:         η_k's decision = switch-off;
16:     else
17:         η_k's decision = continue-on;
18:     end if
19: else if (η_k at sleeping mode) then
20:     if (Eqs. (2) or (3)) then
21:         η_k's decision = switch-on;
22:     else
23:         η_k's decision = continue-off;
24:     end if
25: end if
```

task will be removed from ρ and inserted into the expired queue π. The tasks in π are sorted by their next release times.

3.1 Dynamic switching policy

OPADS is invoked at each scheduling instant and shown in Algorithm 1. Each invocation of OPADS first updates all $L_{i,j}$'s (line 4) and, based on the latest status of $\alpha_{i,j}$'s and $W_{i,j}$'s, determines a switching decision for each device (line 5). Let $L_{x,y}$ denote an interval of $\tau_{x,v}$ which is under consideration here. Let η_k denote the requested device $\eta_{x,y}$ (line 9). No action will be made for a device in power-mode transition (line 10). In addition, since a device may be accessed by more than one task, we always switch it on whenever a task determines a switch-on decision (line 10 to 12). If η_k is currently at working mode, we can switch it off at t_c to save energy as long as (1) $W_{x,y}$ is larger than its break-even time υ_k, and (2) there exists a scheduling instant to switch on η_k in time (line 14). The latter condition can be conservatively checked by

$$\exists i, j, \quad t_c + z_k \le r_{i,j} \le t_c + W_{x,y} - w_k. \tag{1}$$

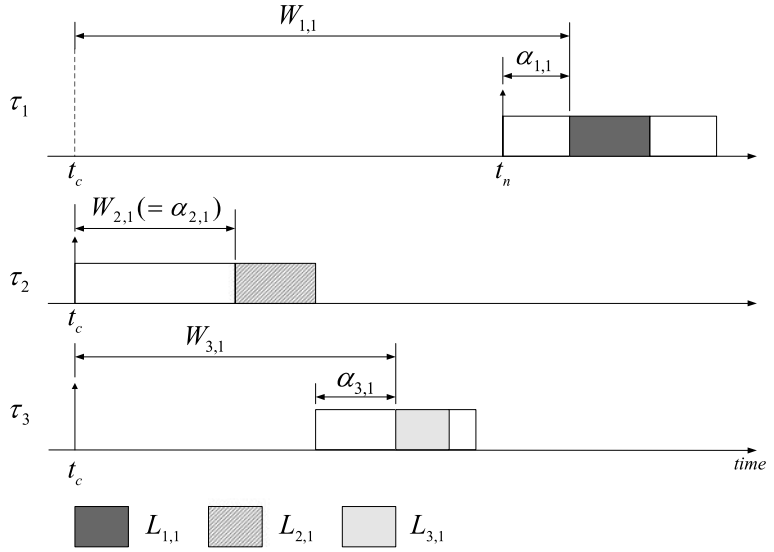


Fig. 5 The switch-on conditions for sleeping devices

Equation (1) represents that some job's arrival time is in $[t_c + z_k, t_c + W_{x,y} - w_k]$ and η_k can be switched off at current time t_c and wake up when that job is released. Otherwise, η_k must keep in working mode (line 17). On the other hand, if η_k is currently at sleeping mode, we continue to let it sleep if switch it on at the next scheduling instant will not violate any timing constraint (line 20 to 21). We obtain a conservative prediction of the next scheduling instant, denoted by t_n, from the first task in π (i.e. the arrival time of the next task, NTA, Kim et al. 2002). If $\tau_{x,v}$ starts to run at t_c, we switch η_k on when

$$W_{x,y} - (t_n - t_c) < w_k. \tag{2}$$

It means that $W_{x,y}$ is too short and we can not delay to switch on η_k at t_n. On the other hand, if $\tau_{x,v}$ is not executed at t_c, we switch η_k on when

$$W_{x,y} - (t_n - t_c) < w_k \wedge \alpha_{x,y} < w_k. \tag{3}$$

The latter condition in (3) indicates that the local waiting $\alpha_{x,y}$ is too short and η_k can not be delayed to switch on until its associated job starts to execution.

Figure 5 illustrates when to switch on sleeping devices in an example where τ_2 and τ_3 are released at t_c, and τ_1 is released at t_n. We use $\varphi(\tau_i)$ to denote the priority of τ_i in the priority-driven real-time schedule. Let $\varphi(\tau_1) > \varphi(\tau_2) > \varphi(\tau_3)$ in Fig. 5. By (2), we switch on $\eta_{2,1}$ at t_c because $W_{2,1} - (t_n - t_c) < w_{2,1}$. For $\eta_{3,1}$, we continue to let it sleep if $\alpha_{3,1} \geq w_{3,1}$ and switch it on at the completion of τ_2. Otherwise, by (3), we need to switch it on at t_c. The same principle is applied to any task in π, such as τ_1.

COLORS determines switching decisions at the arrival or the completion of a job, in order to reduce the scheduling overhead. If this overhead is not a concern, COLORS can incorporate a timer and define a scheduling instant at each timer tick.

The increased scheduling instants allow COLORS to safely sleep a device further longer, as long as a scheduled instant satisfies (1), (2) or (3).

3.2 The prediction of the earliest-access time

We modify the timing-demand analysis (Lehoczky et al. 1989; Joseph and Pandya 1986; Bril et al. 2004) to calculate $W_{x,y}$. The original analysis was used to calculate the worst-case or the best-case completion time of a job. Instead, we predict the earliest-access time of a device. In the following, we first discuss the calculation of $W_{x,y}$ at system initialization. We next generalize this calculation to any scheduling instant.

3.2.1 System initialization

We assume that all tasks are released at $t = 0$ for simplicity and all devices are switched on initially. We provide an iterative solution below to calculate $W_{x,y}$ for an interval $L_{x,y}$ of τ_x at its initialization. Intuitively, $W_{x,y}$, the predicted earliest-access time of $L_{x,y}$, is set to the sum of $\alpha_{x,y}$, the local waiting time of this request, and the BCET of each higher-priority job released in $[0, W_{x,y}]$.

$$W_{x,y} = \alpha_{x,y} + \sum_{\tau_{i,j} \in Q(\tau_{x,1}, [0, W_{x,y}])} b_i, \tag{4}$$

where

$$Q(\tau_{\mu,\nu}, [t_\iota, t_\gamma]) = \{\tau_{i,j} \mid \forall i, j,\ t_\iota \le r_{i,j} \le t_\gamma \wedge \varphi(\tau_{i,j}) > \varphi(\tau_{\mu,\nu})\}.$$

We use $Q(\tau_{\mu,\nu}, [t_\iota, t_\gamma])$ to denote the set of jobs that are released in $[t_\iota, t_\gamma]$ and have a higher priority than $\tau_{\mu,\nu}$. Equation (4) can be solved in an iterative manner as follows.

$$\begin{aligned} W_{x,y}^{(0)} &= \alpha_{x,y}, \\ W_{x,y}^{(1)} &= W_{x,y}^{(0)} + B(\tau_{x,1}, [0, W_{x,y}^{(0)}]), \\ W_{x,y}^{(n)} &= W_{x,y}^{(n-1)} + B(\tau_{x,1}, (W_{x,y}^{(n-2)}, W_{x,y}^{(n-1)}]), \quad n \ge 2, \end{aligned} \tag{5}$$

where

$$B(\tau_{\mu,\nu}, [t_\iota, t_\gamma]) = \sum_{\tau_{i,j} \in Q(\tau_{\mu,\nu}, [t_\iota, t_\gamma])} b_i.$$

The iterative computation stops when $W_{x,y}^{(n)} = W_{x,y}^{(n-1)}$.

We provide an example below that will be used throughout this section to illustrate the process of updating $W_{i,j}$'s and determining a switching decision in a RM system. Due to the limited space of the paper, similar examples can be easily developed for EDF.

Example 2(a) Table 3 shows a periodic task set consisting of three tasks. All three tasks are released at $t = 0$. τ_1 and τ_2 contain no peripheral interval and τ_3 contains

Table 3 A periodic task set

Task	r_i	p_i	e_i	b_i
τ_1	0	3	1	1
τ_2	0	14	2	1
τ_3	0	18	4	1

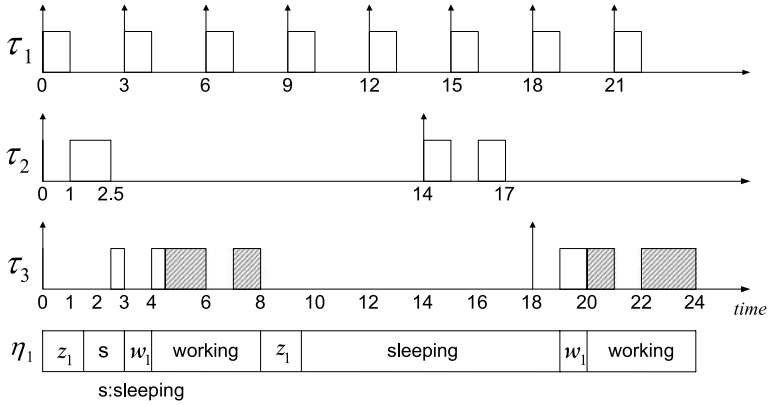


Fig. 6 A RM schedule for tasks in Table 3

one peripheral interval, $L_{3,1} = (\eta_1, 1, 3, W_{3,1})$. Figure 6 shows a RM schedule of this task set and Table 4 lists the status of $L_{3,1}$ at each scheduling instant. By (5), we obtain $W_{3,1}^{(0)} = \alpha_{3,1} = 1$ at $t = 0$. Because tasks are scheduled by RM, both τ_1 and τ_2 have higher priorities than τ_3. Accordingly, $W_{3,1}^{(1)} = 1 + 2 = 3$. A new job of τ_1 is released at $t = 3$. Thus, we obtain $W_{3,1}^{(2)} = 3 + 1 = 4 = W_{3,1}^{(3)}$. Finally, we set $W_{3,1} = 4$ at $t = 0$.

3.2.2 Updates at scheduling instants

Let $L_{x,y}$ be the discussed interval and $\tau_{x,v}$ be its associated job. Based on whether $L_{x,y}$ is executed in $[t_f, t_c)$, we divide the discussion of updating its $W_{x,y}$ at t_c into three cases: (1) $L_{x,y}$ is executed in $[t_f, t_c)$ and is preempted at t_c, (2) $L_{x,y}$ is executed in $[t_f, t_c)$ and completes at t_c, and (3) $L_{x,y}$ is not executed in $[t_f, t_c)$.

(1) If $L_{x,y}$ is executed in $[t_f, t_c)$ and a higher-priority job is released at t_c, we need to determine how long $L_{x,y}$ will be preempted and set $W_{x,y}$ accordingly. Thus,

$$W_{x,y} = \sum_{\tau_{i,j} \in Q(\tau_{x,v}, [t_c, t_c + W_{x,y}])} b_i. \tag{6}$$

Similarly, we can solve (6) in an iterative manner.

(2) When $L_{x,y}$ completes at t_c, we reset $W_{x,y}$ to the earliest-access time of the next job $\tau_{x,v+1}$. Let t_n denote the release time of $\tau_{x,v+1}$. We first determine its waiting

Table 4 The status of $L_{3,1}$ at each scheduling instant in Fig. 6

Time	Event	$(\eta_{3,1}, \alpha_{3,1}, \beta_{3,1}, W_{3,1})$	η_1's switch	Equation
0	All jobs arrive	$(\eta_1, 1, 3, 4)$	Switch off	(5)
1	$\tau_{1,1}$ completes	$(\eta_1, 1, 3, 3)$	No action	(9a)
2.5	$\tau_{2,1}$ completes	$(\eta_1, 1, 3, 2)$	No action	(9a)
3	$\tau_{1,2}$ arrives	$(\eta_1, 0.5, 3, 1.5)$	Switch on	(8)
4	$\tau_{1,2}$ completes	$(\eta_1, 0.5, 3, 0.5)$	No action	(9a)
6	$\tau_{1,3}$ arrives	$(\eta_1, 0, 1.5, 1)$	No action	(6)
7	$\tau_{1,3}$ completes	$(\eta_1, 0, 1.5, 0)$	No action	(9a)
8	$\tau_{3,1}$ completes	$(\eta_1, 1, 3, 12)$	Switch off	(7)

Algorithm 2 Calculation of $G(\tau_{x,v+1}, [t_c, t_n))$

1: $\mathcal{B}$ = the set of all higher priority jobs released in $[t_c, t_n)$;
2: m = the size of $\mathcal{B}$;
3: sort all jobs in $\mathcal{B}$ by their release times;
4: $G = b_1$;
5: **for** $i = 2$ to m **do**
6: $\quad G = b_i + \max\{G - (r_i - r_{i-1}), 0\}$;
7: **end for**
8: $G = \max\{G - (t_n - r_m), 0\}$;

time after t_n by solving

$$W'_{x,y} = \alpha_{x,y} + G(\tau_{x,v+1}, [t_c, t_n)) + \sum_{\tau_{i,j} \in Q(\tau_{x,v}, [t_n, t_n + W'_{x,y}])} b_i. \tag{7}$$

$G(\tau_{x,v+1}, [t_c, t_n))$ denotes the minimum amount of time $\tau_{x,v+1}$ is delayed due to higher-priority jobs released between $[t_c, t_n)$. We next set

$$W_{x,y} = W'_{x,y} + (t_n - t_c).$$

Algorithm 2 explains the calculation of $G(\tau_{x,v+1}, [t_c, t_n))$. It first determines the list of higher-priority jobs released in $[t_c, t_n)$ (line 1) and sorts them by their releasing times (line 3). It next adds the BCET of each job to obtain the minimum remaining workload at t_n (lines 5 to 8).

Figure 7 illustrates the calculation of $G(\tau_{x,v+1}, [t_c, t_n))$ in a 3-task example where $\varphi(\tau_1) > \varphi(\tau_2) > \varphi(\tau_x)$. There are two higher priority jobs released in $[t_c, t_n)$. τ_2 is preempted by τ_1, and both τ_1 and τ_2 execute at their BECTs to estimate $G(\tau_{x,v+1}, [t_c, t_n))$. Because there is no job released in $[t_n, t_n + G(\tau_{x,v+1}, [t_c, t_n)) + \alpha_{x,y}]$, we simply set $W_{x,y}$ to $(t_n - t_c + G(\tau_{x,v+1}, [t_c, t_n)) + \alpha_{x,y})$.

(3) If $L_{x,y}$ is not executed in $[t_f, t_c)$ and τ_x is executed in this period, we simply reduce $W_{x,y}$ as

$$W_{x,y} = W_{x,y} - (t_c - t_f). \tag{8}$$

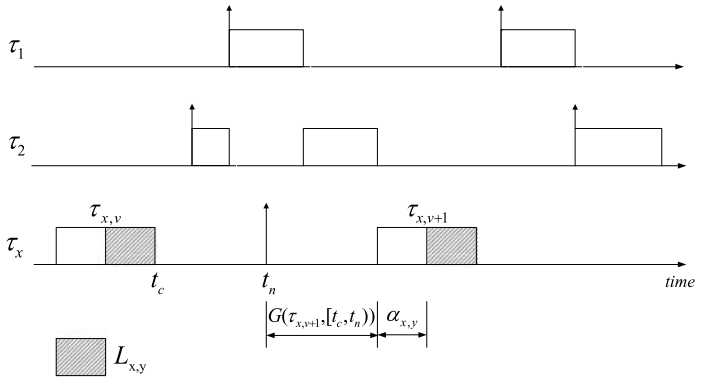


Fig. 7 The calculation of $G(\tau_{x,v+1}, [t_c, t_n))$

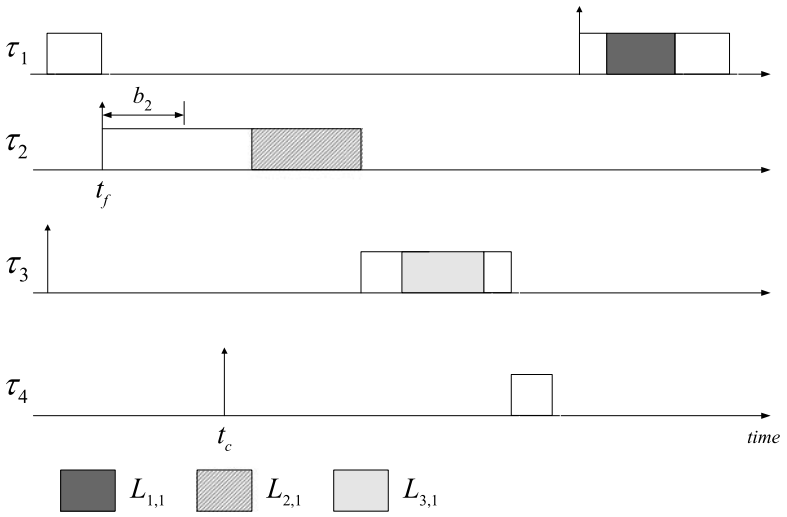


Fig. 8 The calculation of $L_{x,y}$ when it is idle in $[t_f, t_c)$

On the other hand, if τ_k is executed during $[t_f, t_c)$ and $k \neq x$, we reduce $W_{x,y}$ as

$$W_{x,y} = W_{x,y} - (\Delta t),$$

where

$$\Delta t = \begin{cases} b(\tau_k, [t_f, t_c)) & \text{if } \tau_x \in \rho \text{ in } [t_f, t_c), \quad (9a) \\ t_c - t_f & \text{otherwise.} \quad (9b) \end{cases}$$

$b(\tau_k, [t_f, t_c))$ denotes the amount of time in $[t_f, t_c)$ that contributes to the BCET of τ_k. Figure 8 illustrates the calculations of $L_{1,1}$, $L_{2,1}$, and $L_{3,1}$, where all of them are idle in $[t_f, t_c)$ and $\varphi(\tau_1) > \varphi(\tau_2) > \varphi(\tau_3) > \varphi(\tau_4)$. $W_{2,1}$ is updated by (8) as τ_2 is active in $[t_f, t_c)$. Because τ_3 is released and idle in $[t_f, t_c)$, by (9a),

we reduce $W_{3,1}$ by b_2. Because τ_1 is not released, we reduce $W_{1,1}$ by $(t_c - t_f)$, by (9b).

Example 2(b) We continue the explanation of $W_{3,1}$ in Table 4. The last column of Table 4 indicates the equations used to update $W_{3,1}$. At $t = 1$, $\tau_{1,1}$ completes. Since $L_{3,1}$ is idle in this period, we update $W_{3,1} = 4 - 1 = 3$ by (9a). We use the same equation to update $W_{3,1} = 3 - 1 = 2$ at $t = 2.5$ since $b(\tau_{2,1}, [1, 2.5)) = 1$. $L_{3,1}$ completes early at $t = 8$. Because $G(\tau_{3,2}, [8, 18)) = 0$ and $W^{(n)}_{x,y} = 1 + 1 = 2$, we update $W_{3,1} = 2 + (18 - 8) = 12$.

The switching decisions of η_1 in Table 4 and Fig. 6 are explained here. We assume that $z_1 = 1.5$, $w_1 = 1$, and $v_1 = 3$. At $t = 0$, because $W_{3,1} = 4$ and a scheduling instant $(r_{1,2})$ exists in [1.5, 3], we are safe to switch η_1 off by (1). At $t = 1$, η_1 is still in the process of being switched off and therefore no further operation is required. At $t = 2.5$, we keep η_1 in its sleeping mode by (2) because $W_{3,1} - (r_{1,2} - t_c) = 2 - (3 - 2.5) = 1.5 \not< w_1$. At $t = 3$, because both $W_{3,1} - (r_{1,3} - t_c) = -1.5 < w_1$ and $\alpha_{3,1} = 0.5 < w_1$, we switch η_1 on by (3). We keep η_1 in its working mode at $t = 6$ and $t = 7$ because $W_{3,1} < v_1$. At $t = 8$, $L_{3,1}$ completes and we switch η_1 off by (1), since $r_{1,5}$ is in [9.5, 19].

3.3 Design considerations

The overhead of context-switching a device is bounded by one transaction of this device. This overhead is currently ignored by COLORS. The correctness of COLORS still holds even if this overhead is not negligible: the calculation of $W_{i,j}$ will result in a value shorter than its runtime value and COLORS will switch on a device earlier. If more energy reduction is required, one can easily extend COLORS to include this overhead in the estimation of $W_{i,j}$'s.

Although polling I/O is commonly adopted in the work of low-power real-time device scheduling, COLORS is still flexible to handle interrupt-driven I/O under a conservative assumption. We can simply assume that COLORS keeps a device in its working mode as long as its associated peripheral interval is still in execution. In other words, no update on $W_{x,y}$ is made in this period of time. In addition, COLORS can be easily adjusted to work for a device that supports more than two operating modes. This is accomplished by modifying (1), where z_k is replaced by the transition latency from the current state to the desired sleeping mode and w_k is replaced by the wake-up latency of that sleeping mode.

Finally, OPADS favors a work-conserving schedule as it uses BCETs to estimate $W_{i,j}$. The advantage of this decision is to never let the processor idle such that all available slack time is fully utilized to reduce energy. However, when BCET is short, compared to its real execution time, we may switch on a device much earlier than it is needed to and therefore waste energy. Our DVS extension, described in the next section, intends to avoid such a waste by making use of slack time to extend computational intervals and create switch-off opportunities. The experimental results confirm that COLORS delivers significantly more energy reduction than related work (Swaminathan and Chakrabarty 2003; Cheng and Goddard 2006) that both estimate $W_{i,j}$ by WCETs.

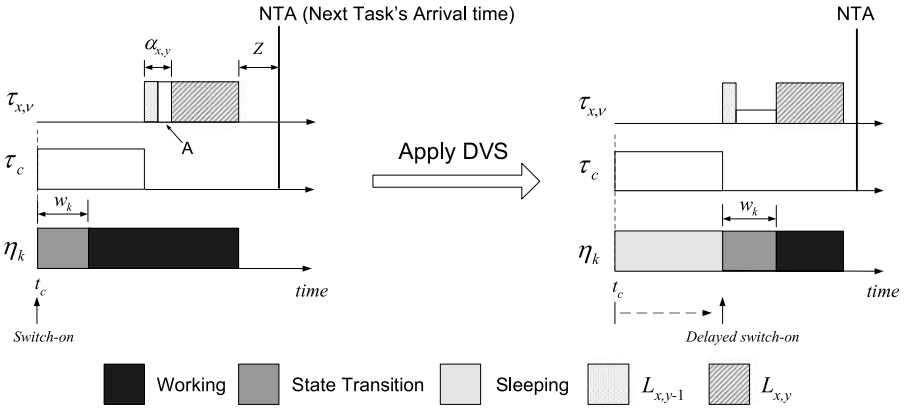


Fig. 9 A delayed switch-on scenario

4 The DVS extension

Slack time is an amount of time a job can be delayed without causing any job to miss its deadline. It is available when the sum of task utilizations e_i/p_i is less than its utilization upper bound or a job completes earlier than its WCET. The objective of this extension is to extend a computational interval with available slack such that peripheral intervals are delayed to create additional switch-off opportunities. In the following, we first explain two scenarios where we can use slack time to create additional switch-off opportunities. We next describe a novel DVS extension on top of OPADS to distribute slack time.

4.1 Delayed switch-on scenario

Figure 9 illustrates a delayed switch-on scenario where τ_c is the current task and $\tau_{x,v}$ is currently released but scheduled to execute later. Let η_k denote the requested device of $L_{x,y}$. OPADS switches η_k on at t_c because doing it at the next scheduling instant may cause $\tau_{x,v}$ to miss its deadline. Our intention here is to delay the earliest-access time of η_k so it can be kept off at t_c. We use the next task's arrival time (NTA) stretching approach (Kim et al. 2002) to calculate the dynamic slack Z between t_c and NTA. Let A denote a computational interval of τ_x and $L_{x,y}$ is executed after A. If there is enough slack to extend A and make $\alpha_{x,y} \geq w_k$, we can switch η_k off at t_c. Let $Len(A)$ denote the length of this computational interval and $S(A)$ denote its original processor speed. We calculate its new processor speed by

$$S(A) * \frac{Len(A)}{Len(A) + (w_k - \alpha_{x,y})}. \tag{10}$$

Because our processor model assumes finite speed levels, this reduced speed may not fall right into an available level. When this situation occurs, we simply reduce the speed further to the next available level, if enough slack time is available. Otherwise, such a speed reduction may miss a deadline and we ignore this delayed switch-on opportunity. Algorithm 3 summaries when a delayed switch-on decision is made.

Algorithm 3

1: Procedure **DelaySwitchOn**($L_{x,y}$, Z)
2: A = any computational interval in τ_x and executed before $L_{x,y}$;
3: η_k = the requested device of $L_{x,y}$;
4: **if** $\{\tau_{x,v} \neq$ current task and $Len(A) > 0$ and $Z \geq (w_k - \alpha_{x,y}) > 0\}$ **then**
5: $Z = Z - (w_k - \alpha_{x,y})$;
6: η_k's decision = *continue-off*;
7: update $\alpha_{x,y}$ and $S(A)$;
8: **end if**

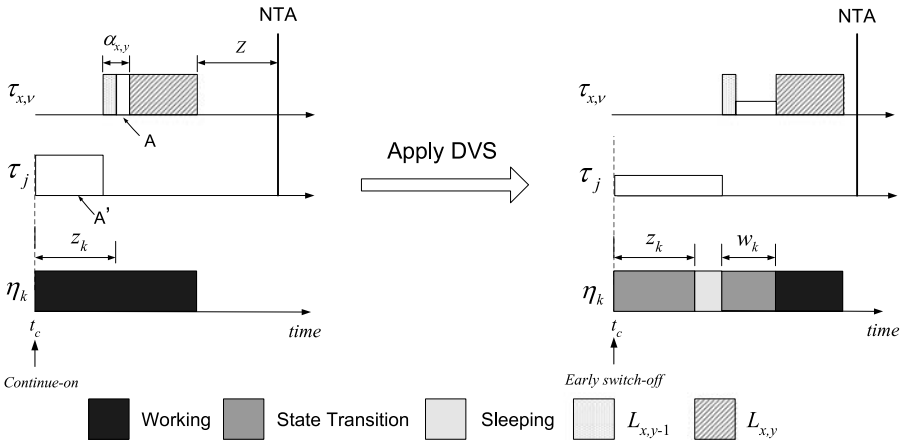


Fig. 10 An early switch-off scenario

4.2 Early switch-off scenario

Figure 10 shows an early switch-off scenario where η_k is originally switched on at t_c. Our intention here is similar: to delay the earliest-access time of η_k with available slack such that it can be switched off at t_c. We summarize when an early switch-off is made in Algorithm 4. It takes two steps to establish an early switch-off opportunity. First, we allocate $(w_k - \alpha_{x,y})$ slack, if necessary, to extend the computational interval before $L_{x,y}$ (line 8). Next, we extend any computational interval between t_c and the execution of $\tau_{x,v}$ such that there is at least $(v_k - w_k)$ apart between these two scheduling instants (line 11). If both conditions are satisfied, we can safely switch η_k off at t_c (line 13) and switch it on at the execution of $\tau_{x,v}$ to reduce its energy consumption.

4.3 Slack time distribution

Because the exact execution time and its device access pattern of a task cannot be known in advance, there exists no perfect slack time distribution. Our DVS extension, therefore, takes a heuristic way to distribute slack time to extend device idle intervals. There are two parts in our DVS extension: an offline and an online part.

Algorithm 4

1: Procedure **EarlySwitchOff**($L_{x,y}$, Z)
2: **if** (τ_x == current task) **then**
3: **return**
4: **end if**
5: A = any computational interval in τ_x and executed before $L_{x,y}$;
6: η_k = the requested device of $L_{x,y}$;
7: $r_1 = \max\{w_k - \alpha_{x,y}, 0\}$;
8: **if** ($r_1 == 0$ or ($Len(A) > 0$ and $Z \geq r_1$)) **then**
9: A' = any computational interval between t_c and $\tau_{x,v}$;
10: $r_2 = \max\{v_k - w_k - Len(A'), 0\}$;
11: **if** ($r_2 == 0$ or ($Len(A') > 0$ and $Z \geq r_1 + r_2$)) **then**
12: $Z = Z - r_1 - r_2$;
13: η_k's decision = *switch-off*;
14: update $\alpha_{x,y}$, $S(A)$, and $S(A')$;
15: **end if**
16: **end if**

Algorithm 5

1: Procedure **StaticDVS**()
2: $\mathcal{H}' = \mathcal{H}$; // the set of devices
3: **while** ($\mathcal{H}' \neq \emptyset$) **do**
4: η_k = the largest power device in $\mathcal{H}'$;
5: $L = \{L_{x,y} | \forall x, y, \eta_{x,y} = \eta_k\}$;
6: **for all** $L_{x,y} \in L$ **do**
7: A = any computational interval in τ_x and before $L_{x,y}$;
8: **if** $\{(A > 0)$ and $(\alpha_{x,y} < w_k)$ and $(e_x \leftarrow e_x + (w_k - \alpha_{x,y}))$ is feasible$\}$ **then**
9: update $\alpha_{x,y}$, b_x, e_x, and $S(A)$;
10: **end if**
11: **end for**
12: $\mathcal{H}' = \mathcal{H}' - \eta_k$;
13: **end while**

During the system initialization, the offline part, **StaticDVS**() summarized in Algorithm 5, utilizes static slack time to create delayed switch-on opportunities. Static slack is available when the sum of task utilizations e_i / p_i is less than its utilization upper bound. For RM scheduling, the feasibility check is done by the exact feasibility test algorithm (Lehoczky et al. 1989). For EDF scheduling, its utilization upper bound is 1. Let η_k denote the device of the largest power consumption rate (line 4). The slack time is first distributed to create delayed switch-on opportunities for intervals accessing η_k if an extended computational interval still results in a feasible schedule (line 8). We also need to set its new processor speed and adjust its BCET and WCET accordingly (line 9). The new processor speed and execution times are statically applied to each job of a task to fully utilize the static slack. Finally, the dis-

Algorithm 6

```
1: Procedure DynamicDVS()
2: Z = available dynamic slack;
3: H' = H;
4: while (H' ≠ ∅) and (Z > 0) do
5:    η_k = the largest power device in H';
6:    L_{x,y} = EarliestInterval(ρ, η_k);
7:    if η_k's decision = switch-on then
8:       DelaySwitchOn(L_{x,y}, Z);
9:    else if η_k's decision = continue-on then
10:      EarlySwichOff(L_{x,y}, Z);
11:   end if
12:   H' = H' − η_k;
13: end while
14: if (Z > 0) then
15:   Distribute Z to τ_c;
16: end if
17: OPADS(); // to determine final switching decisions
```

Algorithm 7 COLORS Execution Flow

```
1: Initialization (off-line):
2: StaticDVS(); // to distribute static slack
3: - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
4: On a job arrives or completes:
5: OPADS(); // to determine switching decisions
6: DynamicDVS(); // to distribute dynamic slack and determine final switching
                    decisions
```

tribution of slack continues to the device of the next largest power consumption rate until no additional delayed switch-on is possible (lines 4 to 13).

The online part, **DynamicDVS**(), is applied on each scheduling instant. We summarize it in Algorithm 6. We use the NTA-stretching approach (Kim et al. 2002) to calculate dynamic slack between the current scheduling instant and NTA. Let η_k denote the device of the largest power consumption rate (line 5). Let $L_{x,y}$ denote the earliest interval among released jobs that accesses η_k (line 6). If η_k's decision is switch-on, we check if there exists a delayed switch-on opportunity (line 8). If η_k's decision is continue-on, we check if there exists an early switch-off opportunity (line 10). The slack allocation continues to the device of the next largest power consumption rate (line 4 to 13). If no more switch-off opportunity is found, we simply distribute slack to the current task τ_c (line 14 to 16). The predicted earliest-access time of each interval in ρ is finally updated by (6) and final switching decisions are determined (line 17). Finally, we summarize each major step of COLORS in Algorithm 7.

Table 5 A periodic task set

Task	r_i	p_i	b_i	e_i
τ_1	0	7	2	2
τ_2 (before StaticDVS)	0	13	1	2
τ_2 (after StaticDVS)	0	13	2	3

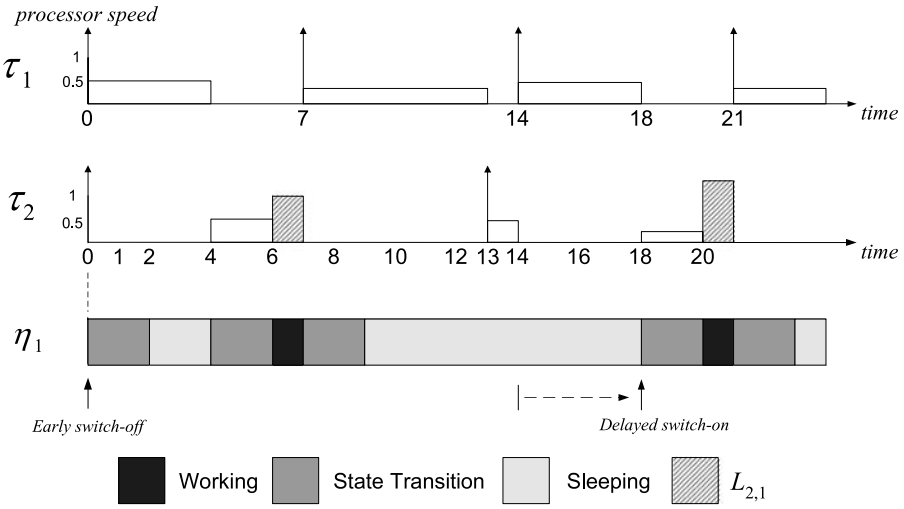


Fig. 11 The COLORS schedule for tasks in Fig. 5

Example 3 We use a two-task example shown in Table 5 to illustrate Algorithm 7. τ_1 contains no device request and τ_2 contains one device request $L_{2,1} = (\eta_1, 1, 1, W_{2,1})$. The parameters of η_1 are set at $z_1 = 2$, $w_1 = 2$, and $v_1 = 4$. All tasks are released at $t = 0$ and scheduled by the RM algorithm. For simplicity, each task executes at their WCET. At its initialization, StaticDVS() is invoked to allocate one unit of static slack to τ_2 for creating a switch-off opportunity. After StaticDVS(), the processor speed of τ_2's computational interval is reduced to 0.5, b_2 and e_2 are updated to 2 and 3, and $L_{2,1}$ becomes $(\eta_1, 2, 1, W_{2,1})$.

Figure 11 shows its COLORS schedule, where the processor speed is normalized to 1. Table 6 lists all dynamic slack assignments and the status of $L_{2,1}$ at each scheduling instant. At $t = 0$, OPADS() switches η_1 off. DynamicDVS() next calculates $(7 - 2 - 3 = 2)$ dynamic slack, allocates all slack to $\tau_{1,1}$, and updates $W_{2,1}$ to 6. At $t = 4$, $\tau_{1,1}$ completes, $W_{2,1}$ becomes 2, and η_1 is waked up by (2). At $t = 7$, there are $(6 - 2 = 4)$ slack, all allocated to $\tau_{1,2}$. At $t = 14$, OPADS() first determines to switch η_1 on. DynamicDVS() next calculates $(7 - 2 - 2 = 3)$ slack, enough to extend $\tau_{2,2}$ and create a delayed switch-on opportunity.

In summary, COLORS includes two components, OPADS and the DVS extension. OPADS is an on-line power-aware device scheduling algorithm. Without the DVS extension, the performance of OPADS in saving energy depends on the original

Table 6 The dynamic slack assignment and the status of $L_{2,1}$

Time	NTA	Slack	τ_1	τ_2	$(\eta_{2,1}, \alpha_{2,1}, \beta_{2,1}, W_{2,1})$	η_1's operation
0	7	2	2	0	$(\eta_1, 2, 1, 6)$	Switch off
4	7	0	0	0	$(\eta_1, 2, 1, 2)$	Switch on
7	13	4	4	0	$(\eta_1, 2, 1, 10)$	Switch off
13	14	0	0	0	$(\eta_1, 2, 1, 4)$	No action
14	21	3	2	1	$(\eta_1, 2, 1, 4)$	No action
18	21	0	0	0	$(\eta_1, 2, 1, 2)$	Switch on

real-time shcedule. To improve the energy-saving performance, the DVS extension is proposed to distribute both static and dynamic slack time so as to adjust the access time of each device. No optimal slack allocation could be made without the actual execution time of each task. Based on this observation, our DVS extension, both **StaticDVS**() and **DynamicDVS**(), favors the devices with larger power consumption. It intends to switch them off and keep them in sleeping mode as long as possible. In short, both OPADS and the DVS extension play important roles in COLORS.

5 Complexity analysis and performance improvement

The iterative equations of COLORS, although producing tight estimations on $W_{x,y}$'s, may require extensive computations in a large-scale system. Such computations can become an additional load during run-time. We begin with the time complexity analysis of COLORS. We next present a simplified version of COLORS, called F-COLORS, that intends to reduce its run-time complexity by looser $W_{x,y}$'s estimations. Our analysis assumes that there are N periodic tasks, M peripheral devices, and totally ξ ($\xi \geq M$) peripheral intervals in a fixed-priority system. Since each task has only one released job at a time, we obtain ξ by summing the number of peripheral intervals in L_i, $i = 1$ to N.

5.1 Complexity of COLORS

There are three major steps in COLORS: updating each $W_{x,y}$'s, determining switching decisions, and applying DVS for slack utilization. The update of $W_{x,y}$ is done by (6), (7), or (9). Both (8) and (9) take $O(1)$ complexity. On the other hand, we solve (6) and (7) in an iterative matter. At each iteration, we need to calculate $B(\tau_{\mu,\nu}, [t_\iota, t_\gamma])$. The calculation of $B(\tau_{\mu,\nu}, [t_\iota, t_\gamma])$ requires the analysis of priority assignments in the scheduling algorithm. For a fixed-priority scheduling algorithm such as Rate-Monotonic (RM), we can easily determine the number of releases of higher-priority tasks between t_ι and t_γ and calculate $B(\tau_{\mu,\nu}, [t_\iota, t_\gamma])$ in $O(N)$ time. For a dynamic-priority scheduling algorithm such a Earliest-Deadline-First (EDF), all released jobs except the last ones between t_ι and t_γ result in a preemption. Similarly, we can calculate $B(\tau_{\mu,\nu}, [t_\iota, t_\gamma])$ in $O(N)$ time. The number of iteration is bounded by the number of preemptions in a period. Let λ denote the ratio of the

largest period to the smallest period. Because a low-priority task may be preempted by a high-priority task λ times in one period, its maximum number of preemptions is bounded by $O(N\lambda)$. Consequently, the complexity of (6) is $O(N^2\lambda)$. For (7), we use Algorithm 2 to calculate $G(\tau_{x,v+1}, [t_c, t_n))$ first and apply iterations later. The time complexity of Algorithm 2 is bounded by sorting the list. Since the number of higher-priority jobs in one period is at most $N\lambda$, Algorithm 2 takes $O(N\lambda(\ln N\lambda))$ to get $G(\tau_{x,v+1}, [t_c, t_n))$. Thus, the complexity of (7) is

$$O(N\lambda(\ln N\lambda)) + O(N^2\lambda),$$

which also bounds the complexity of updating one $W_{x,y}$. Finally, by multiplying ξ, the sum of the number of peripheral intervals in all L_i's, we obtain the complexity of updating all $W_{x,y}$'s at $O(\xi N\lambda(\ln N\lambda)) + O(\xi N^2\lambda)$.

We use either (1), (2), or (3) to determine switching decisions. Equation (1) takes $O(N)$ to browse through all tasks. On the other hand, both (2) and (3) take $O(1)$ complexity. Since there are totally ξ requests, it takes $O(\xi N)$ to determine all switching decisions.

COLORS uses the NTA-stretching approach (Kim et al. 2002) to calculate dynamic slack time. Its complexity is bounded by $O(N)$. For each device, we next determine if a delayed switch-on or an early switch-off decision can be made. A delayed switch-on decision is made in $O(1)$. In addition, an early switch-off decision is made in $O(N)$ after browsing through all tasks. Since there are at most M requests to be examined, it takes totally $O(MN)$ complexity. Because computational intervals of released jobs may be extended by DVS, we use (6) to update all $L_{x,y}$'s associated to them and determine new switching decisions. The former takes $O(\xi N\lambda)$ and the latter takes $O(\xi N)$. Therefore, the complexity of DVS extension is bounded by

$$O(N) + O(MN) + O(\xi N\lambda) + O(\xi N) = O(\xi N\lambda),$$

and the complexity of COLORS is bounded by

$$O(\xi N\lambda(\ln N\lambda)) + O(\xi N^2\lambda) + O(\xi N) + O(\xi N\lambda) = O(\xi N\lambda(\ln N\lambda)) + O(\xi N^2\lambda).$$

In summary, the time complexity of COLORS is pseudo-polynomial.

5.2 Complexity of F-COLORS

The simplified version of COLORS, called F-COLORS, reduces its runtime complexity by limiting iterations and taking conservative estimations. F-COLORS takes the same three major steps as COLORS: updating each $W_{i,j}$'s, determining switching decisions, and applying DVS. For updating, F-COLORS uses $W_{x,y}^{(c)}$ to approximate $W_{x,y}$ in both (6) and (7), in which c is a constant defined by the designer. Namely, F-COLORS stops the computation of $W_{x,y}$ at the cth iteration. Also, the computation of $G(\tau_{x,v+1}, [t_c, t_n))$ in (7) is skipped and always set to 0. Accordingly, it takes $O(N)$ to update one $W_{x,y}$ and $O(\xi N)$ to update all $W_{x,y}$'s.

COLORS takes $O(N)$ to browse through all N tasks in (1) to determine a switching decision. F-COLORS simplifies this computation by examining only the task of

Table 7 The time complexity of COLORS and F-COLORS

Step	COLORS	F-COLORS
OPADS: Updates	$O(\xi N\lambda(\ln N\lambda)) + O(\xi N^2\lambda)$	$O(\xi N)$
OPADS: Decisions	$O(\xi N)$	$O(\xi)$
DVS	$O(\xi N\lambda)$	$O(\xi N)$
Total	$O(\xi N\lambda(\ln N\lambda)) + O(\xi N^2\lambda)$	$O(\xi N)$

Table 8 Three real-time task sets and their parameters

Task set type	Processor utilization
A	$0.1 \sim 0.3$
B	$0.3 \sim 0.5$
C	$0.5 \sim 0.7$

the earliest release time in π. Thus, it takes $O(1)$ to determine a switching decision and $O(\xi)$ to determine all switching decisions.

F-COLORS uses the same NTA-stretching method to calculate dynamic slack in $O(N)$. It reduces the complexity of making an early switch-off decision to $O(1)$. This improvement is realized by only checking whether the BCET of the highest-priority job is larger than z_k. Therefore, F-COLORS can determine if a delayed switch-on or an early switch-off decision can be made in $O(1)$. Since there are at most M requests, it takes $O(M)$ to examine all of them. Finally, it adopts the fast version to update all associated $L_{x,y}$'s in $O(\xi N)$ and determine final switching decisions in $O(\xi)$. Hence, the step of DVS is bounded by $O(N) + O(M) + O(\xi N) + O(\xi) = O(\xi N)$. Table 7 compares the complexity of COLORS and F-COLORS in each step.

The computation overhead is crucial for an on-line scheduling algorithm. Swaminathan et al. (2002) demonstrated that a polynomial time low-power scheduling algorithm can work well in RT-Linux at one hundred tasks. As a result, F-COLORS is a practical solution as well.

6 Simulation results

The energy-saving performance of a real-time device scheduling algorithm depends on many parameters such as task workloads, processor profiles, and device characteristics. We first identify a key metric that primarily determines the performance of an algorithm. We next use this metric to evaluate F-COLORS and several other algorithms. Finally, we compare COLORS and F-COLORS to show the effectiveness of F-COLORS.

6.1 Experimental setup

Table 8 lists three types of real-time task sets used in our simulations. A, B, and C represent a small, a medium, and a large workload, respectively. For each type of task sets, we randomly generate 30 task sets and report their average performance in

Table 9 The ARM8 processor and its performance parameters

Processor	Min Speed (MHz)/Power (mW)	Max Speed (MHz)/Power (mW)
ARM8	5/2.4	80/381

Table 10 Three sensor devices and their power-performance parameters

Device	P_A (W)	P_O (W)	P_E (W)	P_F (W)	z (s)	w (s)	v (s)
Gas Detector[a]	0.45	0	0.2	0.2	0.017	0.011	0.028
THR303 (Temperature/Humidity)[b]	0.6	0	0.3	0.3	0.07	0.04	0.11
EVS100K (CCD)[c]	0.11	0	0.05	0.05	0.12	0.03	0.15

P_A: working power, P_O: sleeping power, P_E: wakeup power, P_F: shutdown power

[a]Axetris Microsystems Inc. (2006) http://www.axetris.com/downloads/f25irsstandard.pdf

[b]Rixen Technology Inc. (2005) http://www.rixen.com.tw/product/trh300_e.htm

[c]ElecVision Inc. (2005) http://www.elecvision.com/pub/data_pub/Promotion_EVS100K.pdf

the following results. Each task set contains $5 \sim 15$ tasks. The period of each task is randomly selected in the range of [10 ms, 1000 ms]. The WCET of each task is also randomly generated according to the processor utilization. In addition, the number of peripheral intervals of each task is randomly selected from 0 to 3. The local waiting time and the length of a peripheral interval are also randomly generated according to the WCET of its associated task. All random numbers mentioned above are generated with the uniform distribution. Our simulations use ARM8 (Im et al. 2004) as the target processor and Table 9 lists its performance parameters. This processor can operate at a speed between 5 MHz using 1.2 V/2.4 mW and 80 MHz using 3.4 V/381 mW. In addition, Table 10 lists the parameters of each sensor device. The parameters of THR303 and EVS100K are obtained by measurements. The rest of parameters are retrieved from the manufacture's data sheets.

6.2 Evaluation metric

A system configuration consists of many parameters derived from tasks, processors, and devices. Instead of showing the performance of COLORS (or F-COLORS) in every combination of these parameters, we carefully analyze the relationship between these parameters and summarize them into a metric, the power-scaled total utilization, that primarily determines its performance.

Definition 1 We first define a pseudo utilization of a device η_i in an N-task set by

$$\overline{U_{\eta_i}} = \sum_{j=1}^{N} \frac{\beta_{i,j}}{p_j},$$

where $\beta_{i,j}$ denotes the length of peripheral intervals in which τ_j accesses η_i in one period. Let $p_{\min}$ denote the shortest period among all tasks accessing η_i. We define

its (actual) utilization by

$$U_{\eta_i} = \begin{cases} \overline{U_{\eta_i}} & \text{if } (p_{\min}(1 - \overline{U_{\eta_i}}) \geq v_i), \\ 1 & \text{otherwise.} \end{cases}$$

The device utilization U_{η_i} approximates the percentage of time the device η_i stays in its working mode. $\overline{U_{\eta_i}}$ is the percentage of time this device is accessed. However, to calculate its actual utilization, we need to determine first if the idle time of this device is larger than its break-even time. If this device can be switched off, its utilization is equal to $\overline{U_{\eta_i}}$. Otherwise, this device must be switched on always and its device utilization is 1.

Definition 2 We define the power-scaled total utilization by

$$U_D = \sum_{i=1}^{M} \delta_i U_{\eta_i}.$$

We use U_D to represent how busy the I/O subsystem is. As different devices require a wide range of power consumptions, we obtain U_D by multiplying each device's utilization U_{η_i} to its power weight δ_i. Intuitively, the larger U_D is, the more energy the I/O subsystem consumes and the less switch-off opportunities a DPM policy delivers.

6.3 Performance comparison

We compare F-COLORS with four real-time device-scheduling algorithms, **Static, EEDS, Static-DVS,** and **SYS-RM. Static**, an enhancement of Swaminathan and Chakrabarty (2003), is an off-line DPM policy. **EEDS** (Cheng and Goddard 2006) is an on-line DPM policy. **Static-DVS**, a combination of Swaminathan and Chakrabarty (2003) and Jejurikar and Gupta (2004), is an off-line algorithm integrating both DVS and DPM. **SYS-RM**, a porting of Cheng and Goddard (2005), is an online integrated algorithm. To our best knowledge, these algorithms represent the state-of-the-art technology in the integration of real-time DVS and DPM. We evaluate them at the same workload where each task has a variable execution time and tasks are scheduled by Rate-Monotonic (RM). Section 7 compares F-COLORS and these works (Swaminathan and Chakrabarty 2003; Jejurikar and Gupta 2004; Cheng and Goddard 2005, 2006) qualitatively.

Necessary modifications are made in each algorithm to work under our system model and the chosen workload. **Static** enhances the LEDES algorithm (Swaminathan and Chakrabarty 2003) by considering break-even time and relaxing the limit on device power-mode transition time. Without DVS, **Static** executes each task at the maximum processor speed. **Static-DVS** further enhances **Static** by adopting the DVS algorithm in Jejurikar and Gupta (2004). Because LEDES assumes fixed execution times, both **Static** and **Static-DVS** intentionally leave the processor idle when a job completes earlier than its WCET, to avoid missing any deadline. We develop **EEDS** by modifying (Cheng and Goddard 2006) to work for a RM schedule. We develop

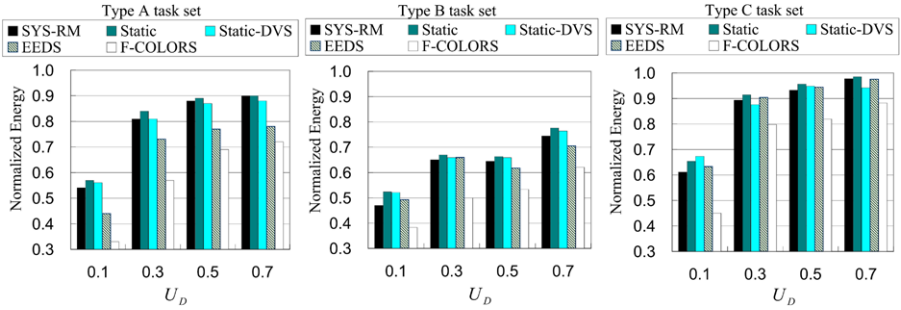


Fig. 12 Performance comparison at different U_D

SYS-RM by modifying the SYS-EDF algorithm (Cheng and Goddard 2005) to work for a RM schedule. We carefully made these modifications so their core algorithms are not altered.

6.3.1 Effect of U_D

This experiment evaluates the performance of an algorithm under U_D. We set BCET/WCET $= 0.5$. We randomly generate the peripheral intervals of each task to obtain a different U_D. Figure 12 shows the experimental results. We compare **Static, Static-DVS, SYS-RM, EEDS** and **F-COLORS** against **Basic**, one basic algorithm that consistently operates the processor at its maximum speed and keeps all devices on. The y axis shows the percentage of energy consumption in comparison to **Basic**. Generally speaking, all algorithms reduce more energy at a smaller U_D in which a device has more idle time to produce more switch-off. We present only the results at $U_D = 0.1$ to 0.7. When $U_D > 0.7$, the idle time of a device is often smaller than its break-even time and cannot be switched off. **Static** and **Static-DVS** deliver comparable performance. Although **Static-DVS** minimizes the energy consumption of a task by its DVS extension, it ignores the impact of DVS on DPM such that an extended task keeps a device on longer than **Static** and avoids many switch-off opportunities. **SYS-RM** delivers less energy reduction than F-COLORS because its DPM policy keeps a device switched on as long as its associated job has not completed. **EEDS** achieves less energy saving than F-COLORS because it takes a conservative estimation on the idle intervals of devices. In addition, **EEDS** uniformly distributes static slack to tasks without considering their device usage. This strategy becomes inefficient at C type of task sets (the largest processor utilization) where static slack is limited. On the other hand, F-COLORS outperforms all algorithms in every scenario. Particularly, with A type, F-COLORS saves energy by 28% at $U_D = 0.7$ and 67% at $U_D = 0.1$.

6.3.2 Effect of BCET/WCET ratio

This experiment evaluates an algorithm at the availability of slack time. The actual execution time of a task is randomly selected between BCET and WCET with the uniform distribution. We set $U_D = 0.1$. Figure 13 shows the results. A system provides

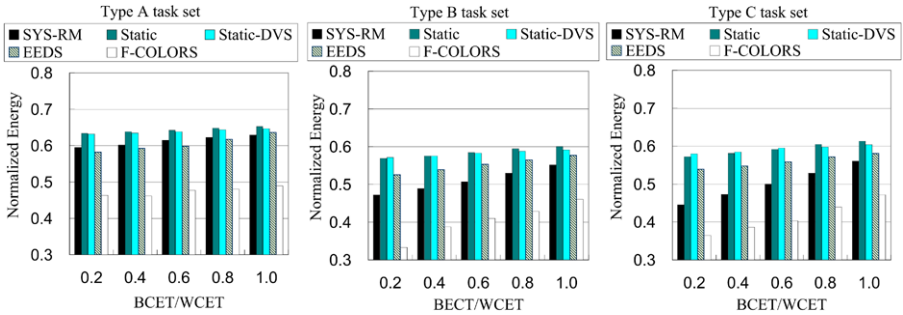


Fig. 13 Performance comparison at different BCET/WCET ratios

more dynamic slack at smaller BCET/WCET ratios. Both **Static** and **Static-DVS** deliver nearly identical performance at different ratios because both approaches assume fixed execution times (i.e., no dynamic slack). When BCET/WCET = 1, F-COLORS still achieves better energy saving than others because it is the first approach to use static slack for both processor speed-down and device switch-off while **Static-DVS** and **SYS-RM** use static slack only for processor speed-down. For this reason, The performance difference between F-COLORS and **Static-DVS** becomes the smallest at (Type C, BCET/WCET = 1), where static slack is limited. In addition, the performance difference between F-COLORS and **Static** or **Static-DVS** grows larger at a smaller ratio as F-COLORS efficiently makes use of dynamic slack. **EEDS** performs worse than F-COLORS at a smaller ratio in which the processor is blocked for a longer period. As a result, devices are kept on longer to waste energy. Comparing to **EEDS**, F-COLORS delivers the best improvement at the ratio of 0.2 on B: an additional 20% energy reduction.

6.3.3 Effect of power ratio

We define P_r as the ratio of the processor maximum power to the sum of P_A of all devices. All the experiments of Sects. 6.3.1, and 6.3.2 were conducted on a system where $P_r = 0.25$ (i.e., the I/O subsystem constitutes 80% of the total energy consumption). This experiment evaluates F-COLORS where P_r varies between 0.2 and 1. Figure 14 shows the experimental results. We assume BCET/WCET = 0.5 to allow dynamic slack and fix $U_D = 0.1$.

The energy-saving of **Static** becomes smaller at a larger P_r as it comes with no DVS support to speed down the processor. The performance difference between **Static** and **Static-DVS** grows larger at a larger P_r as the latter has DVS. The performance difference between **EEDS** and F-COLORS also grows larger at a larger P_r as F-COLORS is DVS-capable. F-COLORS achieves the best energy-saving performance when $P_r \leq 1$. Its OPADS and DVS extension successfully can create many switch-off opportunities when P_r is small. On the other hand, when P_r is large, its DVS extension still uses slack time to speed down the processor. When $P_r > 1$, **SYS-RM** or other DVS-centric real-time algorithms may perform better than F-COLORS.

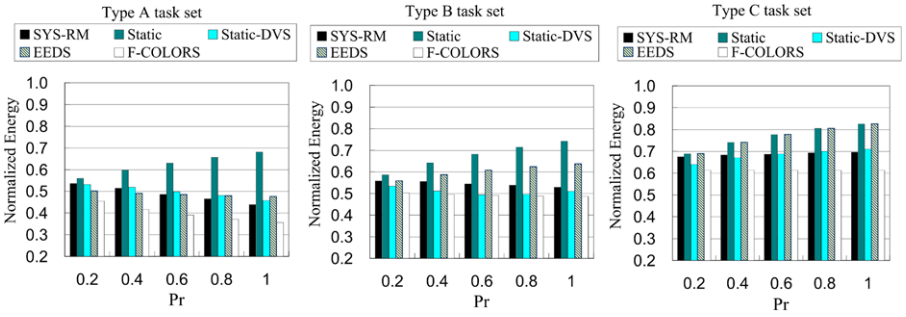


Fig. 14 Performance comparison at different P_r

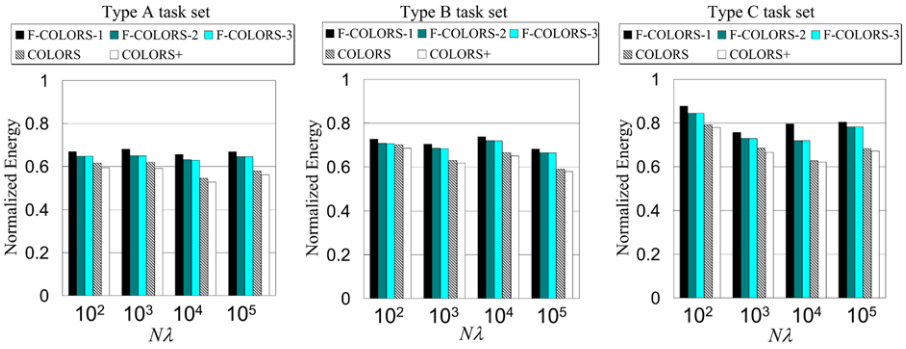


Fig. 15 Performance comparison of COLORS and F-COLORS

6.4 Performance comparison of COLORS and F-COLORS

This experiment studies the performance difference among COLORS, **COLORS+** and three versions of F-COLORS. **COLORS+** is a theoretical optimization of COLORS, in which it generates optimal switching decisions with the knowledge of actual workloads. The three versions of F-COLORS are F-COLORS-1, F-COLORS-2 and F-COLORS-3, each of which stops the calculation of $W_{x,y}$ at different iterations. F-COLORS-1 uses $W^{(1)}_{x,y}$ to approximate $W_{x,y}$ in both (6) and (7). Similarly, F-COLORS-2 uses $W^{(2)}_{x,y}$ to approximate $W_{x,y}$ and F-COLORS-3 uses $W^{(3)}_{x,y}$. We measure the performance of these algorithms at different $N\lambda$ because COLORS is more complicated than F-COLORS in the order of $O(N\lambda)$, when $N > \ln N\lambda$. As Fig. 15 shows, we set $N\lambda$ at 10^2, 10^3, 10^4, and 10^5, which are obtained by modifying task periods. Also, U_D is set at 0.1 and BCET/WCET is set at 0.5.

The results show that F-COLORS-1 performs worst due to its conservative approximation of $W_{x,y}$. The performance difference between F-COLORS-1 and COLORS grows larger at large $N\lambda$ or Type C task set because there is a larger number of preemption in these cases. F-COLORS-2 and F-COLORS-3 deliver almost the same performance since $W^{(2)}_{x,y}$ is similar to $W^{(3)}_{x,y}$. Also, the performance difference between F-COLORS-2 and COLORS or F-COLORS-2 and **COLORS+** is less than 10% in

most configurations even at its largely-reduced complexity and run-time overhead. This is because (1) the DVS extension captures many switch-off opportunities which are originally ignored by loose $W_{x,y}$ estimations in F-COLORS-2, and (2) the converging speed of (6) and (7) is faster than expected. Although F-COLORS-2 uses $W^{(2)}_{x,y}$ to approximate $W_{x,y}$ in both equations, $W^{(2)}_{x,y}$ is in fact close to its full-iterated estimation in most configurations.

7 Background and related work

A number of real-time DVS scheduling algorithms have been proposed recently to reduce the processor energy consumption without missing any deadline. Kim et al. conducted a performance comparison on several real-time DVS algorithms (Kim et al. 2002). The experimental results show that these algorithms (Aydin et al. 2004; Pillai and Shin 2001) deliver comparable performance to an optimal approach assuming the availability of future knowledge. However, all of them provide no solution in reducing the energy consumption of the I/O subsystem. Therefore, these algorithms are not suitable for a real-time monitoring system where sensor devices constitute a major source of energy consumption.

A real-time DPM policy switches off an idle device to save energy. One straightforward approach is to statically construct an exhaustive switching-decision table for each device in a hyper period. A simple table-lookup method is used to switch on/off a device at each scheduling instant. Swaminathan et al. proposed a similar tick-driven scheduler called LEDES that requires the look-ahead of only one task to determine a switching decision (Swaminathan and Chakrabarty 2003). This result is achieved at the assumptions that the power-mode transition time of a device is shorter than the execution time of a task and each task has a fixed execution time. In addition, it did not consider the impact of break-even time and requires the processor to run at a constant speed. EEDS (Cheng and Goddard 2006) is an on-line device scheduling. It uniformly distributes static slack to each task. Without considering the usage of devices, EEDS cannot create as many switch-off opportunities as COLORS. Furthermore, to avoid missing deadlines, both LEDES and EESD leave the processor idle intentionally even when dynamic slack is available. The experimental results shown in Sect. 6.3 evidently demonstrate that COLORS, making use of dynamic slack, significantly saves more energy than a static DPM policy and the EEDS algorithm.

The technique of DVS, although reducing processor energy consumption, extends the execution time of a task and increases the energy consumption of the I/O subsystem. Kim et al. proposed an approach considering both DVS and DPM (Kim and Ha 2001). It partitions the execution of a task into a sequence of time slots and switches off an idle device on a slot-by-slot basis. This approach is possible when the power-mode transition of a device is instantaneous. In addition, its discussion focused on an one-device system. Jejurikar et al. presented an approach considering the issue of system-wide energy reduction on a System-on-Chip (SOC) system (Jejurikar et al. 2004; Jejurikar and Gupta 2005). This work assumes that both the processor and peripheral devices are integrated on a single SOC chip and, therefore, need to be switched on or off at the same time. Under this model, a device may be switched on

even when it is not accessed. Because of its completely different system model, no direct comparison is made between this work and COLORS.

Jejurikar et al. and Zhuo et al. developed an off-line algorithm to consider both the processor energy leakage and the standby energy consumption of peripheral devices (Jejurikar and Gupta 2004; Zhuo and Chakrabarti 2005). However, both assume that all devices have instantaneous power-mode transition and zero energy penalty. Cheng et al. proposed a similar on-line algorithm (Cheng and Goddard 2005). These work (Jejurikar and Gupta 2004; Cheng and Goddard 2005; Zhuo and Chakrabarti 2005) applied DVS on a peripheral interval and executed it at a reduced speed where the energy consumption of an individual task is minimized. Failing to consider the impact of DVS on DPM, such an approach reduces the idle time of a device and avoids many switch-off opportunities. The experimental results, presented in Sect. 6, show that the energy reduction of this approach on the I/O subsystem system is limited. Furthermore, the work of Jejurikar and Gupta (2004) assumed fixed execution times and cannot make use of dynamic slack. COLORS, on the other hand, takes a practical approach to consider the energy and time for switching the power mode of a device and allows variable execution time.

Many parameters, such as those in Tables 8, 9, and 10, may impact its effectiveness. In previous studies, only one or two parameters are used due to the complexity of this problem. For example, Swaminathan and Chakrabarty (2003) and Cheng and Goddard (2005) evaluated their algorithms as a function of the processor utilization. However, a workload with a higher processor utilization has no direct implication on its device utilization. On the other hand, Krishnapura et al. (2004) and Kim and Ha (2001) used the device utilization and access probability as their evaluating function. Different levels of power consumptions in heterogenous devices are not considered. Consequently, their results and conclusions may not be generalized to apply on a system with different configurations. Lu et al. defined device utilization for general systems (Lu et al. 2002). Without including the information of tasks' periods, their definition cannot be directly applied on hard real-time systems. In this work, we identified a key metric, the power-scaled total utilization, that primarily determines the energy-saving performance. We evaluate COLORS through this metric and conclude that COLORS delivers significantly better performance than four representative approaches in all aspects of system configurations when the I/O subsystem constitutes 80% or more of the total energy consumption.

8 Conclusion and future work

In this paper, we presented COLORS that includes DVS in a DPM policy to maximize the energy reduction on the I/O subsystem. COLORS dynamically determines switching decisions at each scheduling instant. The DVS extension efficiently utilizes slack time to create additional switch-off opportunities. The evaluation of a real-time DPM policy is extremely complicated because many system parameters may impact its energy-saving performance. We define a key metric, the power-scaled total utilization, that primarily determines its performance. We used it to evaluate COLORS against four algorithms, each of which is derived from an existing work. The experimental results show that COLORS yields significantly more energy reduction.

Finally, although at its largely reduced complexity, F-COLORS still delivers comparable performance to COLORS and can be used on a system with limited computational capabilities.

In future work, we plan to further enhance the performance of COLORS from two aspects. One is to modify job priorities and group together jobs accessing a same device. Such an approach creates opportunities to switch off idle devices for a longer period of time. The other is to conduct an analysis on the optimal usage of slack time for the purpose of extending COLORS to maximize system-wide energy reduction. We also plan to investigate the impact of jitter on COLORS. Currently, each task is assumed to be released periodically without jitter. We would like to extend COLORS to the system, in which tasks may not be released at exact periodic intervals.

Acknowledgements We thank the anonymous reviewers for their feedback which greatly improved the content and the presentation of this article. This research was supported in part by National Science Council, ROC, under Grant NSC 95-2221-E-002-094-MY3.

References

Alexandropoulos T, Boutas S, Loumos V, Kayafas E (2005) Real-time change detection for surveillance in public transportation. In: AVSS '05: Proceedings of IEEE conference on advanced video and signal based surveillance. IEEE Press, New York

Aydin H, Melhem R, Mosse D, Alvarez PM (2004) Power-aware scheduling for periodic real-time tasks. IEEE Trans Comput 53(5):584–600

Bergasa LM, Nuevo J, Sotelo MA, Barea R, Lopez ME (2006) Real-time system for monitoring driver vigilance. IEEE Trans Intell Transp Syst 7(1):63–77

Bril RJ, Steffens EFM, Verhaegh WFJ (2004) Best-case response times and jitter analysis of real-time tasks. J Sched 7(2):133–147

Cheng H, Goddard S (2005) Integrated device scheduling and processor voltage scaling for system-wide energy conservation. In: PARC '05: Proceedings of the 2nd international workshop on power-aware real-time computing. ACM, New York

Cheng H, Goddard S (2006) Online energy-aware I/O device scheduling for hard real-time systems. In: DATE '06: Proceedings of the conference on design, automation and test in Europe. 3001 Leuven, Belgium, Belgium. European Design and Automation Association, pp 1055–1060

Engblom J, Ermedahl A, Sjoedin M, Gubstafsson J, Hansson H (2001) Worst-case execution-time analysis for embedded real-time systems. J Softw Tools Technol Transf

Han J-J, Li Q-H (2005) Dynamic power-aware scheduling algorithms for real-time task sets with fault-tolerance in parallel and distributed computing environment. In: Proceedings of the 19th IEEE international parallel and distributed processing symposium

Huang TY, Chou CC, Chen PY (2003) Bounding the execution times of DMA I/O tasks on hard-real-time embedded systems. In: Proceedings of the 9th international conference on real-time and embedded computing systems and applications, pp 516–529

Im C, Ha S, Kim H (2004) Dynamic voltage scheduling with buffers in low-power multimedia applications. ACM Trans Embed Comput Syst 3(4):686–705

Jejurikar R, Gupta RK (2004) Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems. In: ISLPED '04: Proceedings of international symposium on low power electronics and design. ACM, New York, pp 78–81

Jejurikar R, Gupta R (2005) Dynamic slack reclamation with procrastination scheduling in real-time embedded systems. In: DAC '05: Proceedings of the 42nd annual conference on design automation. ACM, New York, pp 111–116

Jejurikar R, Pereira C, Gupta R (2004) Leakage aware dynamic voltage scaling for real-time embedded systems. In: DAC '04: Proceedings of the 41st annual conference on design automation. ACM, New York, pp 275–280

Joseph M, Pandya P (1986) Finding response times in a real-time system. Comput J 29(5):390–395

Khatib IA, Bertozzi D, Poletti F, Benini L, Jantsch A, Bechara M, Khalifeh H, Hajjar M, Nabiev R, Jonsson S (2006) MPSoC ECG biochip: a multiprocessor system-on-chip for real-time human heart monitoring and analysis. In: CF '06: Proceedings of the 3rd conference on computing frontiers. ACM, New York, pp 21–28

Kim M, Ha S (2001) Hybrid run-time power management technique for real-time embedded system with voltage scalable processor. In: LCTES '01: Proceedings of the ACM SIGPLAN workshop on languages, compilers and tools for embedded systems. ACM, New York, pp 11–19

Kim W, Shin D, Yun HS, Kim J, Min SL (2002) Performance comparison of dynamic voltage scaling algorithms for hard real-time systems. In: RTAS '02: Proceedings of the 8th IEEE real-time and embedded technology and applications symposium. IEEE Computer Society, Los Alamitos, pp 219–228

Krishnapura R, Goddard S, Qadi A (2004) A dynamic real-time scheduling algorithm for reduced energy consumption. Technical Report TR-UNL-CSE-2004-0009, University of Nebraska Lincoln

Lee C-H, Shin KG (2004) On-line dynamic voltage scaling for hard real-time systems using the EDF algorithm. In: Proceedings of the 25th IEEE real-time systems symposium

Lehoczky J, Sha L, Ding Y (1989) The rate monotonic scheduling algorithm: exact characterization and average case behavior. In: Proc of 10th IEEE real-time systems symp, pp 166–171

Lu YH, Benini L, De Micheli G (2000) Low power task scheduling for multiple devices. In: CODES '00: Proceedings of the 8th international workshop on hardware/software codesign. ACM, New York, pp 39–43

Lu YH, Benini L, De Micheli G (2002) Power-aware operating systems for interactive systems. IEEE Trans Very Large Scale Integr Syst 10(2):119–134

McIntire D, Ho K, Yip B, Singh A, Wu W, Kaiser WJ (2006) The low power energy aware processing (LEAP) embedded networked sensor system. In: IPSN '06: Proceedings of the 5th international conference on information processing in sensor networks. ACM, New York, pp 449–457

Pillai P, Shin KG (2001) Real-time dynamic voltage scaling for low-power embedded operating systems. In: SOSP '01: Proceedings of the 18th ACM symposium on operating systems principles. ACM, New York, pp 89–102

Swaminathan V, Chakrabarty K (2003) Energy-conscious, deterministic I/O device scheduling in hard real-time systems. IEEE Trans Comput-Aided Des Integr Circ Syst 22(7):847–858

Swaminathan V, Schweizer CB, Chakrabarty K, Patel AA (2002) Experiences in implementing an energy-driven task scheduler in RT-linux. In: RTAS '02: Proceedings of the eighth IEEE real-time and embedded technology and applications symposium (RTAS'02). Washington, DC, USA. IEEE Computer Society, p 229

Zhuo J, Chakrabarti C (2005) System-level energy-efficient dynamic task scheduling. In: DAC '05: Proceedings of the 42nd annual conference on design automation. ACM Press, New York, pp 628–631

Edward T.-H. Chu received the B.S. and M.S. degrees in Mechanical Engineering from National Central University, Taiwan, in 1996 and 1998, respectively. He was a BIOS engineer in Acer Inc. from 1999 to 2000, and a senior BIOS engineer in Wistron Cooperation from 2000 to 2003. He is currently a Ph.D. candidate in the Department of Computer Science at National Tsing Hua University, Taiwan, and a visiting scholar at the School of Electrical and Computer Engineering, Purdue University, USA. His research interests include low-power embedded systems, real-time operating systems and energy profiling tools.

Tai-Yi Huang received the B.S. degree in computer science and information engineering from National Taiwan University in 1991. He received both the M.S. and Ph.D. degrees from the University of Illinois at Urbana-Champaign in computer science in 1994 and 1996, respectively. From 1996 to 2001, he was a software design engineer in Windows OS Kernel Performance group, Microsoft Inc. He joined the Computer Science Department at National Tsing Hua University, Taiwan, in 2002 as an assistant professor and was promoted to be an associate professor in 2006. He joined Google in 2007 as a member of technical staff. His research interests include low-power embedded systems, real-time operating systems, and high-performance clustered storages.

Cheng-Han Tsai received the B.S. and Ph.D. degree in computer science from National Tsing Hua University, Taiwan, in 2003 and 2007. He also received Microsoft Fellowship Award in 2005. His research is currently focusing on real-time scheduling, especially in providing real-time guaranty on disk I/O activities. His research interests also include automated fault localization and operating system performance analysis, such as synchronization.

Jian-Jia Chen received his Ph.D. degree from Department of Computer Science and Information Engineering at National Taiwan University in 2006. He received his B.S. degree from the Department of Chemistry at National Taiwan University 2001. Since 2008, after completing his compulsory military service, he has been a postdoc researcher at Computer Engineering and Networks Laboratory (TIK) Swiss Federal Institute of Technology (ETH) Zurich, Switzerland. His research interests include real-time systems, energy-efficient scheduling, power-aware designs, embedded systems, temperature-aware scheduling, distributed computing, and algorithmic analysis. He won the Best Paper Award on IEEE RTCSA 2005 and Institute of Information and Computing Machinery (IICM) Doctoral Dissertation Award in 2006.

Tei-Wei Kuo received the B.S.E. degree in Computer Science and Information Engineering from National Taiwan University in Taipei, Taiwan, in 1986. He received the M.S. and Ph.D. degrees in Computer Sciences from the University of Texas at Austin in 1990 and 1994, respectively. He is currently a Professor of the Department of Computer Science and Information Engineering, National Taiwan University. He served as the Chairman of his department from August 2005 to July 2008. Between Feb. 2006 and July 2008, he also served as a Deputy Dean of the College of Electrical Engineering and Computer Science, National Taiwan University. Prof. Kuo has served in the editorial board of many journals, including the Journal of Real-Time Systems and IEEE Transactions on Industrial Informatics. He was the Program Chair and General Chair of the IEEE Real-Time Systems Symposium in 2007 and 2008, respectively. Between 2005 and 2008, Prof. Kuo has also served as the Steering Committee Chair of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA). Prof. Kuo receives many research and teaching awards including the 2004 Ten Young Outstanding Persons Award of Taiwan, the Distinguished Research Award from the National Science Council, and the Distinguished Teaching Award from his university (Top 1%). His research interests include embedded systems, real-time task scheduling, real-time operating systems, flash-memory storage systems, and real-time database systems. He has over 170 technical papers published or been accepted in international journals and conferences and more than 6 patents in USA and Taiwan on the designs of flash-memory storage systems.