Computing and Informatics, Vol. 36, 2017, 765–792, doi: 10.4149/cai_2017_4_765

ENERGY-EFFICIENT CONCURRENCY CONTROL FOR DYNAMIC-PRIORITY REAL-TIME TASKS WITH ABORTABLE CRITICAL SECTIONS

Jun Wu

Department of Computer Science and Information Engineering
National Pingtung University
900 Pingtung, Taiwan, R.O.C.
e-mail: `junwu@mail.nptu.edu.tw`

Abstract. In this paper, we are interested in energy-efficient concurrency control for real-time tasks on a non-ideal DVS processor. Based on well-known ceiling-based concurrency control protocols (such as priority ceiling protocol (PCP) and stack resource policy (SRP)), researchers have proposed energy-efficient approaches to mange concurrent accesses to shared resources so that the energy consumption can be reduced. However, ceiling-based protocols have a problem of ceiling blocking which imposes a great impact on the performance of real-time systems. In order to achieve sufficient performance, we propose a new protocol, called *conditional abortable stack resource policy* (CA-SRP), to resolve the ceiling blocking problem for dynamic-priority real-time tasks by incorporating a conditional abort rule into SRP. Based on the schedulability analysis of CA-SRP, we also propose a method, called *dynamic speed assignment* (DSA), to dynamically calculate and assign proper processor speeds for task execution so that the energy consumption can be reduced further. The capabilities of our proposed CA-SRP and DSA have been evaluated by a series of experiments, for which we have encouraging results.

Keywords: Real-time systems, dynamic voltage scaling, energy efficiency, concurrency control, task scheduling

Mathematics Subject Classification 2010: 68-N25

This article is a revised and expanded version of a paper presented at 6th International C* Conference on Computer Science and Software Engineering, Porto, Portugal, July 10–12, 2013.

1 INTRODUCTION

Reducing energy consumption is a challenge issue in the design of embedded real-time systems, which has received more and more attention from researchers. A *dynamic voltage scaling* (DVS) processor can operate at different speeds by varying its supply voltage. An effective way to reduce energy consumption is to slow down the execution speed of tasks on a DVS processor. Unfortunately, it will decrease the performance of the entire system because of the late completion of tasks. However, it provides a strong driving force in energy-efficient scheduling of real-time tasks due to the fact that the late completion is allowed as long as the real-time tasks meet their timing constraints.

In the past decades, energy-efficient real-time task scheduling has been studied extensively for minimizing the energy consumption without violating tasks' timing constraints (comprehensive surveys can be found in [22, 3]). In the literature, most energy-efficient scheduling algorithms were proposed for independent real-time tasks (i.e., tasks share nothing but CPU), relatively little work has been done for dependent real-time tasks. However, independent real-time tasks are rarely observed in real-world applications. On the contrary, dependent real-time tasks are most commonly found. Therefore, we are interested in energy-efficient scheduling of dependent real-time tasks so that the meaningful research results can be obtained for real-world applications. Real-time tasks are considered to be dependent when they may make requests simultaneously for accessing the same set of shared resources during their executions. To maintain data consistency, shared resources do not allow concurrent accesses by competing tasks, but require their mutual exclusion. Note that the code fragment of a task for accessing a shared resource is called *critical section*. In this paper, we assume that a critical section is guarded by a semaphore, and it has to be executed in a mutually exclusive manner.

Based on well-known ceiling-based real-time concurrency control protocols (such as *priority ceiling protocol* (PCP) [23] and *stack resource policy* (SRP) [1]), researchers have proposed several approaches to reduce the energy consumption for scheduling dependent real-time tasks [32, 33, 15, 7, 11, 13, 29, 30, 28, 27]. Unfortunately, those ceiling-based energy-efficient approaches have a problem of ceiling blocking which imposes a great impact on the performance of real-time systems. Note that the ceiling blocking problem is the situation in which a task is blocked by a lower-priority task even they do not require the same shared resource. This is possible in ceiling-based concurrency control protocols when a lower-priority task is accessing a shared resource which is also required for other higher-priority tasks. In this case, the system ceiling will be equal to the highest priority of the higher-priority tasks, thus, medium-priority tasks will be blocked.

In this paper, we consider tasks as periodic and preemptible (for both critical sections and non-critical part of tasks), and they can access one or more multiunit shared resources during their executions[1]. Note that each multiunit shared resource has a fixed number of units in the system[2]. Different from the past work, we also assume that the critical sections of tasks are abortable. Originally, making critical sections abortable [9, 26, 24, 25, 14] is a strategy developed to resolve ceiling blocking problem so that sufficient performance can be achieved. When critical sections are abortable, a higher-priority task is allowed to abort another lower-priority task if they are conflicting in the same shared resource. Note that the execution of the aborted task has to be re-executed from the beginning of the aborted critical section.

We will propose a new energy-efficient concurrency control protocol, called *conditional abortable stack resource policy* (CA-SRP), to resolve the ceiling blocking problem for dynamic-priority real-time tasks. Our proposed CA-SRP achieves sufficient performance by incorporating a conditional abort rule into SRP. When tasks are assumed to be executed on a non-ideal DVS processor, we also propose a method, called *dynamic speed assignment* (DSA), to dynamically calculate proper processor speeds for task executions so that the energy consumption can be reduced. Note that the execution speeds of tasks are calculated dynamically based on the schedulability condition of CA-SRP. As a result, the energy consumption can be reduced without violating tasks' timing constraints. The capabilities of our proposed CA-SRP and DSA have been evaluated by a series of experiments, for which we have encouraging results.

The rest of this paper is organized as follows: Section 2 summarizes the related work. Section 3 presents the system and task models. Section 4 and Section 5 present our proposed CA-SRP and DSA method, respectively. Section 6 is dedicated to the properties of our proposed approach. Section 7 reports the experimental results. Section 8 is the conclusion.

2 RELATED WORK

In the field of energy-efficient scheduling, Yao et al. [31] are considered the pioneers where they have for the first time proposed an optimal offline scheduling algorithm for aperiodic tasks on an ideal DVS processor[3]. Later energy-efficient scheduling has

[1] A multiunit resource has more than one units (i.e. instances), and each of them can provide the same services to tasks. Typical examples are input/output buffers and communication channels.

[2] When the number of units of shared resources are all set to one, all of the research results presented in this work can be also applied to a system which only has single-unit shared resources.

[3] Two types of DVS processors have been considered in the literature: *ideal* and *non-ideal*. An ideal DVS processor can operate at any speed in the range from the minimum to the maximum available speed, while a non-ideal DVS processor has only discrete speeds. Nowadays, most DVS processors are non-ideal while the ideal DVS processors are only for theoretical analysis purpose.

been studied extensively for general periodic real-time tasks on an ideal or a non-ideal DVS processor. In the literature, thousand of algorithms have been proposed to minimize the energy consumption without violating tasks' timing constraints (comprehensive surveys can be found in [22, 3]). However, most of those scheduling algorithms were proposed for independent real-time tasks, relatively little work has been done for dependent real-time tasks in the presence of task synchronization with shared resources.

In the recent decades, some energy-efficient approaches had been proposed for scheduling of dependent real-time tasks under various assumptions on the system and task models [32, 33, 15, 7, 11, 13, 29, 30, 28, 27]. Most of them use a simple strategy, called *two-speed strategy* (TSS), to execute tasks at a low speed initially and it switches to a high speed as soon as a blocking occurs. When critical sections are considered to be non-preemptible, Zhang and Chanson [32, 33] first proposed a TSS-based scheduling algorithm, called *dual speed* (DS), for dynamic-priority tasks based on the *earliest deadline first* (EDF) algorithm [16] and the *stack resource policy* (SRP) [1]. Since DS calculates the low speed and the high speed based on the sufficient schedulability condition of EDF, the timing constraints of tasks can be met while energy consumption is reduced. A time interval that the processor is operating at the high speed is called *high speed interval.* Under DS, a high speed interval starts from the beginning of a blocking and ends at the deadline of the blocking task. Lee et al. [15] explored the same problem with a tighter schedulability analysis, and proposed a multi-speed extension of DS, call *multi-speed* (MS) algorithm, to achieve further energy savings. In particular, MS assigns different high speeds for each task by considering different blocking scenarios of a task instance. MS reduces more energy consumption than that of DS, and the schedulability of the tasks is still guaranteed. Later Elewi et al. [8, 7] proposed two extensions of DS and MS, called *enhanced dual speed* (EDS)[8] and *improved multi-speed* (IMS) algorithm [7]. Their proposed EDS and IMS obtain more energy saving by shortening the high speed intervals, i.e., they end a high speed interval at the deadline of the blocked tasks (instead of the deadline of the blocking task) or the earliest time that the processor becomes idle. However, EDS and IMS cannot provide the schedulability guarantee.

When critical sections are considered to be preemptible, Jejurikar and Gupta [11] first proposed a TSS-based scheduling algorithm, called *critical section maximum speed* (CSMS) algorithm, for fixed-priority real-time tasks which are scheduled and synchronized by *rate monotonic* (RM) [16] and *priority ceiling protocol* (PCP) [23], respectively. CSMS always executes tasks at a low speed and then it switches to the maximum processor speed when tasks being executed in a critical section. Jejurikar and Gupta also proposed another TSS-based algorithm, called *dual mode* (DM) algorithm [12], to derive the operating speeds for fixed-priority real-time tasks in two different modes: *independent* and *synchronization* modes. Under DM, the system starts in the independent mode and tasks are executed at the low speed. Whenever a lower-priority task blocks other higher-priority tasks in the independent mode, the system enters the synchronization mode and tasks are executed at the high speed.

Note that the basic idea of DM is quite similar to that of DS. However, DM and DS are different in the task model, i.e., DM and DS assume critical sections are preemptible and non-preemptible, respectively.

Later Jejurikar and Gupta extended their work to dynamic-priority real-time tasks. They proposed an algorithm, called *uniform slowdown with frequency inheritance* (USFI) algorithm [13]. Under USFI, each task is assigned to be executed at a static speed which is calculated based on the schedulability condition of EDF with the consideration of tasks' worst-case blocking times. Whenever a task blocks other tasks, it will inherit and be executed at the highest speed of the blocked tasks during the blocking. They showed that the blocking time of tasks can be reduced without violating tasks' timing constraints under their proposed USFI algorithm. However, the energy consumption under CSMS and USFI are much higher than that of DS and MS. To achieve more energy saving, Wu [28] proposed a TSS-based algorithm, called *blocking-aware two-speed* (BATS) algorithm, for both dynamic-priority and fixed-priority real-time tasks. Different from DS and DM, BATS is more energy-efficient because the processor has less chances to execute tasks at a high speed. In particular, BATS starts a high speed interval from the beginning of a blocking and ends at the deadline of the blocked task or the time when the processor becomes idle. In comparison with the existing work, the high speed calculated by BATS is also lower than other approaches because of a tighter schedulability analysis. Hence, BATS is the most energy-efficient approach for dependent real-time tasks.

Recently, some research results were obtained for tasks with abortable critical sections. Originally, making critical section abortable is a strategy proposed to reduce the number of priority inversions for tasks with higher priorities. When critical sections are abortable, a higher-priority task is allowed to abort another lower-priority task if they are conflicting in the same shared resource. Note that the execution of the aborted task has to be re-executed from the beginning of the aborted critical section. Huang et al. [9] first proposed an approach, called *priority abort scheme* (PAS), in which a critical section can be aborted by any other task whose priority is higher than that of the task currently accessing the critical section. However, PAS may cause unnecessary aborts such that the schedulability of tasks is degraded. Later Takada and Sakamura [25] have proposed the *ceiling abort protocol* (CAP) to make critical sections abortable under PCP. In CAP, a critical section might have an abortable section which is allowed to be aborted by higher-priority tasks. However, CAP has a serious problem of uncontrollable aborts. In particular, unbounded number of aborts may occur during the execution of a task such that the performance of the entire system will be reduced seriously. To resolve such a problem, Lam and Ng [14] proposed a PCP-based protocol, called *conditional abortable priority ceiling protocol* (CA-PCP). Under CA-PCP, a condition is defined to control the number of aborts with the attempt to reduce impact of aborting a task on the schedulability, and at the same time, to reduce the number of ceiling blockings.

In the literature, there are few results in providing energy-efficient scheduling for tasks with abortable critical sections. To the best of our knowledge, the closest related work was proposed by Wu and Ke [29]. When tasks are scheduled by RM and PCP, they proposed an energy-efficient concurrency control protocol for fixed-priority real-time tasks with abortable critical sections. However, in this paper, we are interested in dynamic-priority scheduling because it allows a better usage of the available processor power and significantly improves the performance [2]. In this paper, we will explore the energy-efficient scheduling of dynamic-priority real-time tasks with abortable critical sections so that the energy consumption can be reduced further while providing sufficient performance.

3 SYSTEM MODEL AND PROBLEM DEFINITIONS

3.1 DVS Processor Models

We consider a *non-ideal* DVS processor which supports a set of K discrete speeds $\mathcal{S} = \{s_1, s_2, \ldots, s_K\}$, where $s_1 < s_2 < \cdots < s_K$. Note that the processor speed is approximately proportional to the supply voltage. The supply voltage of a processor speed s_i is denoted as V_i, for $1 \leq i \leq K$. Let s_{min} and s_{max} denote the lowest and highest speeds (i.e. $s_{min} = s_1$ and $s_{max} = s_K$), respectively. Without lose of generality, we assume the s_{max} is 1 and all other speeds are normalized with respect to the maximum speed s_{max}.

The power consumption function of a DVS processor can be defined as a function $PC(s_i)$ of the adopted processor speed s_i. It can be divided into two parts: dynamic and static power [13, 32, 7]. The dynamic power consumption $PC_{dynamic}(s_i)$, due to gate switching at processor speed s_i, can be modeled as a convex function of the processor speed: $PC_{dynamic}(s_i) = C_{ef}V_{dd}^2 s_i$, where $s_i = \kappa\frac{(V_{dd}-V_i)^2}{V_{dd}}$, and C_{ef}, V_i, V_{dd} and κ denote the effective switch capacitance, the threshold voltage, the supply voltage, and a hardware-design-specific constant, respectively. The static power consumption PC_{static}, due to leakage currents I_{leak}, can be represented as: $PC_{static} = I_{leak}V_{dd}$. Since the supply voltage V_{dd} is the dominant source of the dynamic and the static power consumption. Lowering the supply voltage can reduce energy consumption which is the power dissipated over time.

The amount of CPU cycles completed in a time interval $(t_1, t_2]$ can be represented as $\int_{t_1}^{t_2} s(t)\,\mathrm{d}t$, where $s(t)$ is the processor speed at time t. Hence, the energy consumption $EC_{(t_1,t_2]}$ in a time interval $(t_1, t_2]$ is $\int_{t_1}^{t_2} PC(s(t))\,\mathrm{d}t$. Note that, in this paper, we ignore some overheads incurred for scheduling tasks, such as the time and energy consumption for speed-switching and context-switching (due to preemptions), because the overheads are relatively low when compared with the cost of executing tasks. For example, Rajan et al. [21] have shown that the speed-switching overhead is about 1–2 % of the total energy consumption. However, when the overheads are significant and cannot be ignored, researchers have proposed excellent approaches for reducing or eliminating the speed-switching and/or the preemptions [17, 20, 18].

3.2 Task and Resource Models

We consider a real-time system which consists of a set of n periodic hard real-time tasks $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$. A task τ_i is a template of its instances and each instance will be created for each request of the task. The requests of a task τ_i will arrive regularly for every period T_i. The worst-case computation amount and the relative deadline of a task τ_i are defined by C_i and D_i, respectively. When a task τ_i is executed at a processor speed s_j, the worst-case execution time of τ_i is $\frac{C_i}{s_j}$. Note that the worst-case execution time of the task could be C_i if the task is executed at the maximum processor speed s_{max}.

Let $\tau_{i,j}$ denote the j^{th} instance of task τ_i. The execution of each task instance $\tau_{i,j}$ must be completed no later than its deadline which is defined as its arrival time plus the relative deadline D_i. We consider well formed tasks which satisfy $0 \leq C_i \leq D_i \leq T_i$, $\forall \tau_i \in \mathcal{T}$. We also assume that the relative deadline is equal to the period, i.e. $D_i = T_i$. The computation amount C_i can be divided into two parts: cC_i and nC_i, where cC_i is the total computation amount of its critical sections, i.e. $cC_i = \sum_{z_{i,j} \in \mathcal{Z}_i} |z_{i,j}|$, and nC_i is the computation amount of its non-critical part, i.e. $nC_i = C_i - cC_i$.

We assume that a set of m multiunit resources $\mathcal{R} = \{r_1, r_2, \ldots, r_m\}$ can be accessed in the system [1]. For each multiunit resource r_j there is a fixed number of units in the system, denoted as N_{r_j}. A task τ_i may make one or more requests for accessing multiunit resources during its execution. Let $\mu_{r_j}(\tau_i)$ be the number of units of resource r_j requested by task τ_i, and $\mu_{r_j}(\tau_i) < N_{r_j}$ for $1 < i < n$. To maintain data consistency, resources do not allow simultaneous accesses but require mutual exclusion among competing tasks. We assume that resources are guarded by semaphores, and the time interval during the accessing of a resource is called a *critical section*.

Let $\mathcal{Z}_i =< z_{i,1}, z_{i,2}, \ldots, z_{i,n_i} >$ be the list of critical sections of a task τ_i, where $z_{i,j}$ is the j^{th} critical section of τ_i. Each critical section $z_{i,j}$ is a request which needs $\mu(z_{i,j})$ units of the resource $R(z_{i,j})$. We also use $|z_{i,j}|$ to denote the computation amount of the critical section. If a critical section $z_{i,j}$ is executed at a processor speed s_k, then its execution time is $\frac{|z_{i,j}|}{s_k}$. Any task that needs to enter a critical section must wait until there is a sufficient number of units of resources and the access right has been granted. A task τ_i is said to be *blocked* by the critical section of task τ_j if τ_j has a lower-priority than τ_i but τ_i has to wait for τ_j to exit its critical section in order to resume its execution. Note that such a situation is also called *priority inversion.*

3.3 Abortable Critical Sections

In this paper, we also assume that critical sections are abortable. Each critical section consists of an *abortable segment* followed by an *unabortable segment.* Let $z^a_{i,j}$ and $z^u_{i,j}$ be the abortable segment and the unabortable segment of the j^{th} critical section of τ_i, respectively. Note that the execution of a task can be aborted when it is executed in the abortable segment of a critical section. Then the execution will be restarted from the beginning of the abortable segment. On the contrary, once a task enters the unabortable segment, the critical section cannot be aborted and it will be executed to the end (the preemption is allowed). Figure 1 shows a task with its two critical sections, each consists of an abortable segment and an unabortable segment. Figure 2 shows another example with two nested critical sections. In particular, task τ_i has two critical sections $z_{i,1}$ and $z_{i,2}$. Note that $z_{i,2}$ is executed fully inside the $z_{i,1}$. In this case, only the first critical section, i.e. $z_{i,1}$, is allowed to have an abortable segment; while the inside one is not allowed to have an abortable segment.

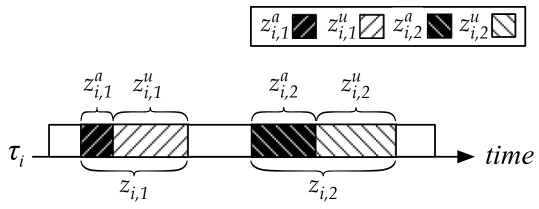

Figure 1. An example of a task with two critical sections

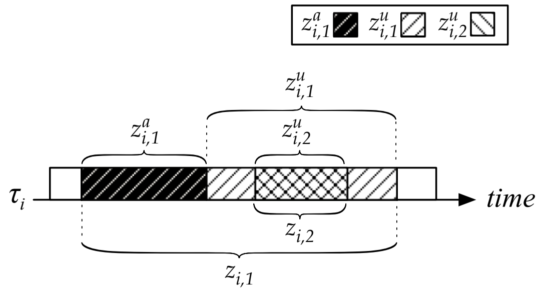

Figure 2. An example of a task with two nested critical sections

3.4 Problem Definitions

The research problem is given as follows:

Problem 1 (DVS Scheduling of Real-Time Tasks with Abortable Critical Sections)**.** Given a set of n periodic hard real-time tasks $\mathcal{T}$ which have abortable critical sections. The problem is to schedule $\mathcal{T}$ and to synchronize their accesses of a set of m shared multiunit resources $\mathcal{R}$ on a non-ideal DVS processor which supports a set of K discrete speeds $\mathcal{S}$. During a given time interval $(0, lcm]$, the objective of this problem is to generate a schedule such that all tasks can meet their timing constraints (i.e. their deadlines) and the energy consumption $\int_0^{lcm} PC(s(t))\,\mathrm{d}t$ is minimized.

This problem is an optimization problem which is to generate a schedule for the given task set with minimal energy consumption. Where lcm is the *least common multiple* of all tasks' periods (it is also called *hyperperiod*). We only need to examine the time interval $(0, lcm]$ for analyzing the performance and the schedulability of the entire schedule because the given task set repeats an identical execution trace for every hyperperiod [16]. The NP-hardness of the problem is given by the following theorem.

Theorem 1. The Problem 1 is NP-hard.

Proof. The NP-hardness of Problem 1 can be proven by considering the following special case. Assuming that the DVS processor only supports a single speed (i.e. $K = 1$) so that all feasible schedules consume the same energy (it is because the computation amount is fixed.). We also assume that the number of units of each multiunit resource is 1 and the length of any abortable segment $z^a_{i,j}$ is 0. Such a special case of Problem 1 is equivalent to the well-known uniprocessor real-time scheduling and synchronization (URT-SS) problem[19] which is NP-hard. Problem 1 is NP-hard due to the fact that one of its special cases is NP-hard. $\square$

4 CONDITIONAL ABORTABLE STACK RESOURCE POLICY

In this section, a new concurrency control protocol, called *conditional abortable stack resource policy* (CA-SRP), is proposed for dynamic-priority real-time tasks with abortable critical sections.

4.1 The CA-SRP

CA-SRP uses *earliest deadline first* (EDF) algorithm for task scheduling, where the priorities of tasks are assigned dynamically according to their absolute deadlines. In other words, tasks with earlier absolute deadlines will receive higher priorities. It always executes the task with the highest priority (ties can be broken arbitrarily).

When tasks might have concurrent access to shared resources, CA-SRP uses the *stack resource policy* (SRP)[1] for task synchronization. Under SRP, each task τ_i is required to have a *preemption level* π_i. The values of tasks' preemption levels are assigned inversely proportional to their relative deadlines, i.e. $\pi_i > \pi_j \Leftrightarrow D_i < D_j$. In addition, each resource r_j is required to have a *current ceiling* $CL_{r_j}(n_{r_j})$ which represents the maximum preemption level of the tasks which may be blocked by resource r_j. The value of $CL_{r_j}(n_{r_j})$ can be calculated as $CL_{r_j}(n_{r_j}) = \max_{\tau_i \in \mathcal{T}}\{\{0\} \cup \{\pi_i | n_{r_j} < \mu_{r_j}(\tau_i)\}\}$, where n_{r_j} is the number of units of r_j that are currently available. We also define the *system ceiling* π_s as the maximum value of all current ceilings of resources, i.e. $\pi_s = \max_{r_j \in \mathcal{R}}\{CL_{r_j}(n_{r_j})\}$. The rules for concurrency control of CA-SRP are the same as those of SRP: a task τ_i is permitted to preempt the current task if its priority is the highest among all ready tasks, and its preemption level is higher than the system ceiling (i.e. $\pi_i > \pi_s$).

Different from SRP, CA-SRP are designed for tasks with abortable critical sections. It can abort lower-priority tasks to improve the responsibility of tasks with higher priority. A new rule, called *conditional abort* rule, is introduced: a task τ_i can abort a lower-priority task τ_x if τ_x is executing in the abortable segment of a critical section $z_{x,y}$ and $\pi_i > \pi_s^{x,y}$, where $\pi_s^{x,y}$ is the system ceiling when the critical section $z_{x,y}$ is aborted. Where the value of $\pi_s^{x,y}$ is calculated as follows:

$$\pi_s^{x,y} = \max\left\{\max_{r_k \in \mathcal{R}-R(z_{x,y})}\{CL_{r_k}(n_{r_k})\}, \max_{\tau_k \in \mathcal{T}}\{0, \pi_k | (n_{R(z_{x,y})} + \mu(z_{x,y})) \leq \mu_{R(z_{x,y})(\tau_k)}\}\right\}. \tag{1}$$

Our proposed CA-SRP is given in Algorithm 1.

Algorithm 1 CONDITIONAL ABORTABLE STACK RESOURCE POLICY (CA-SRP)

```
Priority Assignment and Task Scheduling:
  Tasks with earlier absolute deadlines will be assigned higher priorities. It always
  executes the task with the highest priority (ties can be broken arbitrarily).

Concurrency Control and Conditional Abort:
  When $\tau_{i,j}$ becomes the highest priority task among all ready tasks,
  if $\pi_i > \pi_s$ then
      $\tau_{i,j}$ preempts the current task.
  else
      if $\exists\tau_x$ which is executing in $z^a_{x,y}$ and $\pi_i > \pi_s^{x,y}$ then
          $\tau_{i,j}$ aborts $\tau_x$'s critical section $z_{x,y}$ and starts its execution.
      else
          $\tau_{i,j}$ is blocked.
      end if
  end if
```

4.2 Worst-Case Blocking Time and Re-Execution Time

Since CA-SRP uses SRP for concurrency control, the following theorem follows directly from the theories of SRP [1].

Theorem 2. [1] When a set of tasks is scheduled under CA-SRP, then

1. no task can be blocked after it starts,
2. a task can be blocked for at most the duration of one critical section.

Let B_i be the worst-case blocking time of a task τ_i. Based on Theorem 2, the value of B_i can be calculated as follows[1]:

$$B_i = \max_{\tau_j \in \mathcal{T} \wedge \pi_j < \pi_i} \{0, |z_{j,k}| \mid z_{j,k} \in \mathcal{Z}_j, CL_{R(z_{j,k})}(0) > \pi_i\}. \tag{2}$$

When a task τ_x has been aborted, it has to be re-executed from the beginning of the aborted critical section. Such a re-execution time is obviously no more than the length of the aborted segment. Let A_i be the worst-case re-execution time of the task aborted by τ_i. It can be calculated by the following equation:

$$A_i = \max_{\substack{\tau_j \in \mathcal{T} \\ \wedge \pi_j < \pi_i}} \{0, |z^a_{j,k}| \mid z_{j,k} \in \mathcal{Z}_j, CL_{R(z_{j,k})}(0) > \pi_i\}. \tag{3}$$

4.3 Example

The following example shows the schedule and the corresponding energy consumption of tasks scheduled by CA-SRP. Note that the schedule is given by assuming that tasks are executed at the maximum processor speed s_{max}.

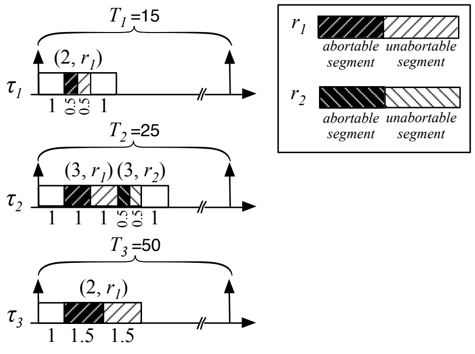

Figure 3. Example task set

Example 1. Consider three tasks τ_1, τ_2 and τ_3 as shown in Figure 3. The parameters of the tasks and shared resources are given in Table 1 a) and b). Note that the abortable segment and the unabortable segment of r_1 and r_2 in Figures 3 to 6 are represented as ▨ and ▧, respectively. Also note that the abortable segment and the unabortable segment of r_2 are represented as ▩ and ▤, respectively. In other words, those boxes are the critical sections of tasks. The label on the top of the boxes represents the requirements for the resources, i.e., (a, r_b) indicates that the critical section requires a units of resource r_b. Also note that the white boxes represent the non-critical part of tasks. In Figure 3, the labels on the bottom of the boxes represent their execution time.

	π_i	C_i	$D_i = T_i$	B_i	A_i
τ_1	3	3	15	3	1.5
τ_2	2	5	25	3	1.5
τ_3	1	4	50	0	0

a) Parameters of the task set

	N_{r_j}	$\mu_{r_j}(\tau_1)$	$\mu_{r_j}(\tau_2)$	$\mu_{r_j}(\tau_3)$
r_1	3	2	3	2
r_2	3	0	3	0

b) Parameters of the shared resources

Table 1. Parameter settings of the example

When the three tasks are executed at the maximum processor speed, Figure 4 shows the schedule of the tasks scheduled by CA-SRP. Note that we only consider the first instances of the three tasks, i.e., $\tau_{1,1}$, $\tau_{2,1}$ and $\tau_{3,1}$, and the value of the system ceiling is indicated as the red bold-lines, as shown in the bottom of Figure 4.

At time 0, $\tau_{3,1}$ arrives and starts its execution immediately because it is the only one ready task in the system. At time 1, $\tau_{3,1}$ requests 2 units of resource r_1 successfully. Later, at time 2, $\tau_{2,1}$ arrives with an earlier deadline than $\tau_{3,1}$ for which $\tau_{2,1}$ becomes the highest-priority task. According to CA-SRP's conditional abort rule, $\tau_{2,1}$ aborts the execution of $\tau_{3,1}$ and starts its execution. At time 3 and 5, $\tau_{2,1}$ requests 3 units of r_1 and r_2 successfully, respectively. Next, at time 6, $\tau_{1,1}$ arrives and preempts the current task (i.e. $\tau_{2,1}$) because $\tau_{1,1}$ has the earliest deadline and $\pi_1 > \pi_s$. At time 9, $\tau_{1,1}$ completes its execution and $\tau_{2,1}$ resumes. At time 10, $\tau_{2,1}$ completes and $\tau_{3,1}$ resumes its execution from the beginning of its aborted critical section. Finally, $\tau_{3,1}$ completes its execution at time 13.

Assume that tasks are executed on a non-ideal DVS processor which supports 10 discrete speeds $\mathcal{S} = \{s_1 = 0.1, s_2 = 0.2, \ldots, s_{10} = 1\}$. Let the power consumption function of processor speeds be $PC(s_i) = (0.08 + 1.52s_i^3)$ Watt ([4]). The energy consumption of this schedule is 20.8 Watt.

5 DYNAMIC SPEED ASSIGNMENT METHOD

In this section, we shall analyze the schedulability of CA-SRP and derive a lowest processor speed, called *base processor speed*, for task executions so that the energy consumption can be reduced without violating tasks' timing constraints. Furthermore, we also propose a method, called *dynamic speed assignment*, to dynamically calculate lower execution speeds for task execution for saving more energy.

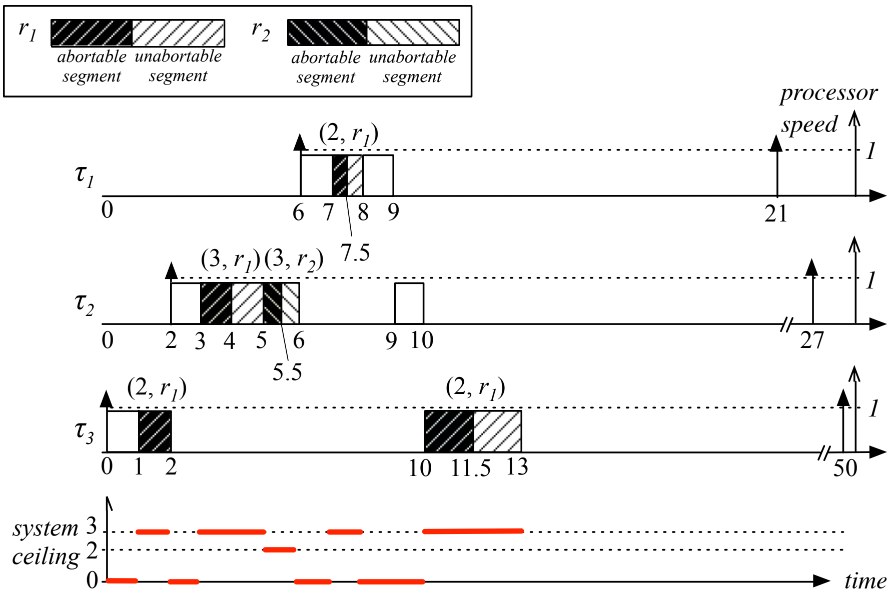

Figure 4. Example CA-SRP schedule (when tasks are executed at s_{max})

5.1 Schedulability and Base Processor Speed

For independent tasks, Liu and Layland [16] have shown that a set of n tasks is schedulable under EDF if $\sum_{i=1}^{n} C_i/D_i \leq 1$. Such a condition has to be modified for CA-SRP because we assume that tasks might have a concurrent access to shared resources. In particular, a task τ_i might be blocked by lower-priority tasks or abort lower-priority tasks according to the concurrency control and conditional abort rules of CA-SRP. The following theorem provides the sufficient schedulability condition for CA-SRP when tasks are executed at the maximum processor speed s_{max}:

Theorem 3. When a set of n tasks (sorted in non-decreasing order of their preemption levels) is executed at the maximum processor speed, it is schedulable under CA-SRP if the following conditions are satisfied:

$$\sum_{i=1}^{n} \frac{(C_i + B_i)/s_{max}}{D_i} = \sum_{i=1}^{n} \frac{C_i + B_i}{D_i} \leq 1. \tag{4}$$

Proof. According to the concurrency control and conditional abort rules of CA-SRP, a task instance of task τ_i might be blocked by a lower-priority task or aborts a lower-priority task's critical section. Hence, the utilization of any task instance of τ_i is no higher than $(C_i + \max\{B_i, A_i\})/D_i$. Note that A_i is less than B_i for every task τ_i according to Equations (2) and (3). Therefore, the utilization of τ_i never exceeds $(C_i + B_i)/D_i$, $\forall \tau_i \in \mathcal{T}$. Based on the schedulability condition of EDF and $s_{max} = 1$, this theorem is correct by considering the worse-case blocking time B_i as a part of tasks' computation. $\square$

The following corollary shows that the tasks remain schedulable when they are executed at the *base processor speed* s^b. Note that the base processor speed is the lowest processor speed for executing tasks without violating their timing constraints.

Corollary 1. If a set of n tasks (sorted in non-decreasing order of their preemption levels) satisfies the sufficient schedulability condition of CA-SRP (i.e. Equation (4) in Theorem 3), then the task set remains schedulable under CA-SRP by using the base processor speed s^b for task execution, where

$$s^b = \min_{s_j \in S} \left\{ s_j \;\middle|\; \sum_{k=1}^{n} \frac{C_k + B_k}{D_k} \le s_j \right\}. \tag{5}$$

Proof. This corollary is correct based on the Theorem 3 and the fact that $s^b \le s_{max}$. $\square$

The following example shows the schedule of the tasks given in Example 1 when they are executed at the base processor speed.

Example 2. Consider the same tasks given in Example 1. When the tasks are executed at the base processor speed s^b, Figure 5 shows the schedule of the tasks under CA-SRP. Note that the base processor speed is 0.8 which is calculated according to Equation (5). In this schedule, $\tau_{2,1}$ aborts $\tau_{3,1}$ and $\tau_{1,1}$ aborts $\tau_{2,1}$ at time 2 and 6, respectively. This schedule completes at time 16 which is later than that in Example 1 because those two abortings create additional re-execution time for the schedule. However, the energy consumption of this schedule is less than that in Example 1 because tasks are executed at a speed lower than the maximum processor speed, i.e. $s^b \le s_{max}$. In particular, the energy consumption of this schedule is 13.73184 Watt according to the same assumptions on the power consumptions in Example 1. Compared with Example 1, the tasks remain schedulable and it saves more than 33.98 % of energy.

5.2 Dynamic Speed Assignment (DSA) Method

As we mentioned previously, CA-SRP uses EDF and SRP for scheduling tasks and synchronizing their accesses to shared resources. When a DVS processor is considered, one of the more important issues of CA-SRP is how to calculate and assign proper processor speeds for task execution so that the energy consumption can be reduced further. In the previous subsection, the base processor speed is derived for

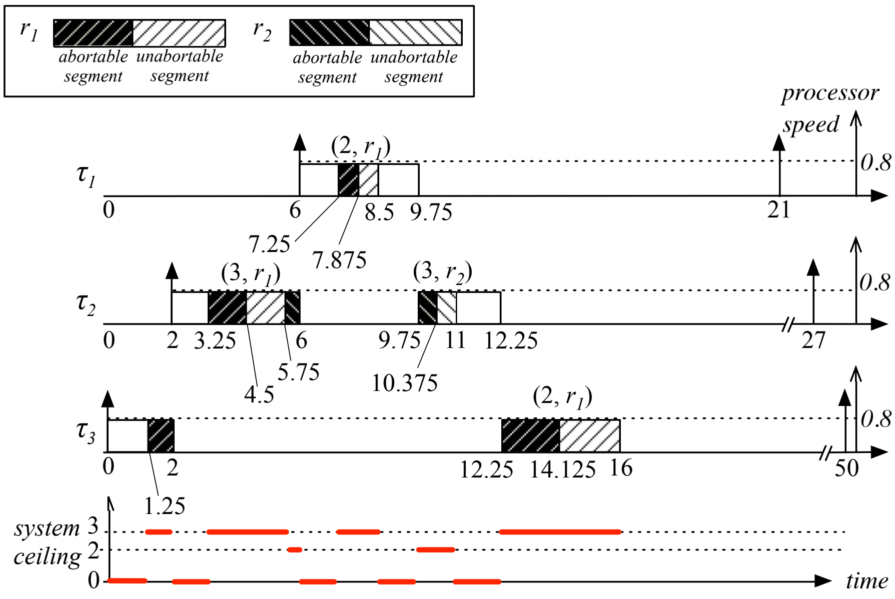

Figure 5. Example CA-SRP schedule (when tasks are executed at s^b)

guaranteeing the schedulability of tasks, which is calculated based on the consideration of the worst cases (e.g., tasks are blocked with worst-case blocking time). However, the worst-case blocking time and re-execution time are overestimated due to the fact that the worst case of a task will not happen every time. We shall propose a method, called *dynamic speed assignment* (DSA), to adjust the execution speeds of tasks dynamically so that the energy consumption can be reduced further.

When tasks are scheduled by CA-SRP, the execution speeds of tasks are assigned according to the DSA method which as follows:

- Any task τ_i's critical sections (i.e. the cC_i part) are assigned to be executed at the base processor speed s^b. It ensures that the actual blocking time and the re-execution cost of any task τ_i does not exceed B_i/s^b and A_i/s^b, respectively. Note that the s^b is calculated offline according to Equation (5) with a $O(Kn)$ time complexity, where K and n are the number of available processor speeds and the number of tasks, respectively. Furthermore, K can be considered as a constant.
- The execution speed of every task instance $\tau_{i,j}$'s non-critical part (i.e. nC_i) is assigned to a *dynamic speed* $s^*_{i,j}$. Note that $s^*_{i,j} \leq s^b$ and it is calculated dynamically whenever $\tau_{i,j}$ is permitted to start its execution.

We now describe the derivation of the dynamic speed for a task instance. The following lemma provides the worst-case computation time of a task instance under CA-SRP.

Lemma 1. If a set of n tasks is schedulable by CA-SRP, the worst-case computation time of a task instance $\tau_{i,j}$ is never exceeded $(nC_i + cC_i + B_i)/s^b$ when tasks are executed at the base processor speed s^b.

Proof. According to Corollary 1, a set of n tasks are schedulable if Equation (4) is satisfied and all tasks are executed at the base processor speed s^b. Because Corollary 1 remains correct even if all task instances are the worst cases, it is implied that the worst-case computation time of any task instance $\tau_{i,j}$ is never exceeded $(C_i + B_i)/s^b = (nC_i + cC_i + B_i)/s^b$. □

Based on the concurrency control and conditional abort rules of CA-SRP, the dynamic speed $s^*_{i,j}$ is calculated by considering the following two cases:

Case 1 (when $\pi_i > \pi_s$): In this case, $\tau_{i,j}$ is allowed to preempt the current task immediately. Note that $\tau_{i,j}$ will not be blocked because $\pi_i > \pi_s$. According to CA-SRP's rules for execution speed assignment, the non-critical part and the critical sections of $\tau_{i,j}$ are executed at $s^*_{i,j}$ and s^b, respectively. Hence, the computation time of $\tau_{i,j}$ can be calculated as $\frac{nC_i}{s^*_{i,j}} + \frac{cC_i}{s^b}$. For guaranteeing the schedulability of tasks, $\tau_{i,j}$'s computation time must be no longer than its worst-case computation time (which can be obtained by Lemma 1), i.e. $\frac{nC_i}{s^*_{i,j}} + \frac{cC_i}{s^b} \leq \frac{nC_i+cC_i+B_i}{s^b}$. Hence, the dynamic speed $s^*_{i,j}$ can be calculated as follows:

$$s^*_{i,j} = \min_{s_k \in S} \left\{ s_k \,\middle|\, \frac{s^b nC_i}{nC_i + B_i} \leq s_k \right\}. \tag{6}$$

Case 2 (when $\pi_i \leq \pi_s$): In this case, $\tau_{i,j}$ might abort a lower-priority task or be blocked. If there exists a task τ_x which is executing in the abortable segment of a critical section $z_{x,y}$ and $\pi_i > \pi_s^{x,y}$, $\tau_{i,j}$ will abort $z_{x,y}$. Because the re-execution cost is considered as a part of $\tau_{i,j}$'s computation, the computation time of $\tau_{i,j}$ can be calculated as $\frac{nC_i}{s^*_{i,j}} + \frac{cC_i+|z^a_{x,y}|}{s^b}$. For guaranteeing the schedulability, $\tau_{i,j}$'s computation time must be no longer than its worst-case computation time, i.e. $\frac{nC_i}{s^*_{i,j}} + \frac{cC_i+|z^a_{x,y}|}{s^b} \leq \frac{nC_i+cC_i+B_i}{s^b}$. Therefore, the dynamic speed $s^*_{i,j}$ can be calculated as follows:

$$s^*_{i,j} = \min_{s_k \in S} \left\{ s_k \,\middle|\, \frac{s^b nC_i}{nC_i + B_i - |z^a_{x,y}|} \leq s_k \right\}. \tag{7}$$

Suppose that a task instance $\tau_{i,j}$ becomes the highest-priority task among all ready tasks at time $t^h_{i,j}$.

On the other hand, $\tau_{i,j}$ will be blocked if such a task τ_x does not exist. Let $t^h_{i,j}$ be the time that $\tau_{i,j}$ becomes the highest-priority task among all ready tasks, and $t^s_{i,j}$ be the time that $\tau_{i,j}$ is permitted to start its execution. Similarly to the above, $\tau_{i,j}$'s computation time must be no longer than its worst-case computation time, i.e. $\frac{nC_i}{s^*_{i,j}} + \frac{cC_i}{s^b} + (t^s_{i,j} - t^h_{i,j}) \le \frac{nC_i+cC_i+B_i}{s^b}$, for guaranteeing the schedulability of tasks. Note that $(t^s_{i,j} - t^h_{i,j})$ is the actual blocking time of $\tau_{i,j}$. The dynamic speed $s^*_{i,j}$ can be calculated for this case by the following equation:

$$s^*_{i,j} = \min_{s_k \in S} \left\{ s_k \;\middle|\; \frac{s^b nC_i}{nC_i + B_i - s^b(t^s_{i,j} - t^h_{i,j})} \le s_k \right\}. \tag{8}$$

We now formally define our proposed DSA method in Algorithm 2.

Algorithm 2 Dynamic Speed Assignment Method for CA-SRP

```
Task Execution Speed :
  For all task instance $\tau_{i,j}$ of $\tau_i \in \mathcal{T}$, its critical sections and non-critical parts are
  assigned to be executed at the base processor speed $s^b$ and a dynamic speed $s^*_{i,j}$,
  respectively, where
      $s^b = \min_{s_k \in \mathcal{S}} \left\{ s_k \;\middle|\; \sum_{l=1}^{n} \frac{C_l + B_l}{D_l} \le s_k \right\}.$
Dynamic Speed Assignment:
  When $\tau_{i,j}$ becomes the highest priority task among all ready tasks,
  if $\pi_i > \pi_s$ then
      $\tau_{i,j}$ preempts the current task and $s^*_{i,j} = \min_{s_k \in \mathcal{S}} \left\{ s_k \;\middle|\; \frac{s^b nC_i}{nC_i+B_i} \le s_k \right\}$
  else
      if $\exists \tau_x$ which is executing in $z^a_{x,y}$ and $\pi_i > \pi^{x,y}_s$ then
          $\tau_{i,j}$ aborts $\tau_x$'s critical section $z_{x,y}$ and $s^*_{i,j} = \min_{s_k \in \mathcal{S}} \left\{ s_k \;\middle|\; \frac{s^b nC_i}{nC_i+B_i-|z^a_{x,y}|} \le s_k \right\}$.
      else
          $\tau_{i,j}$ is blocked and $s^*_{i,j} = \min_{s_k \in \mathcal{S}} \left\{ s_k \;\middle|\; \frac{s^b nC_i}{nC_i+B_i-s^b(t^s_{i,j}-t^h_{i,j})} \le s_k \right\}$.
      end if
  end if
```

The following examples illustrate our proposed CA-SRP with DSA:

Example 3. Consider the same tasks and the DVS processor given in Example 1. Figure 6 shows the schedule of the tasks when they are scheduled by CA-SRP with DSA method. Under CA-SRP, all critical sections are executed at the base processor speed which is calculated based on Equation (5) and the value is 0.8. The non-critical part of a task instance $\tau_{i,j}$ is executed at $s^*_{i,j}$ which is calculated dynamically based on the blocking time and the re-execution time.

At time 0, $\tau_{3,1}$ arrives and starts its execution immediately because it is the only one ready task. The execution speed of $\tau_{3,1}$'s non-critical part is $s^*_{3,1} = 0.8$ which is calculated according to Equation (6). Note that the computation time of a task τ_i is $\frac{C_i}{s_j}$ when it is executed at speed s_j. It is because the original value of C_i is given by assuming that the task is executed at the maximum processor speed. At time 1.25, $\tau_{3,1}$ completes the execution of its non-critical part and it requests 2 units of resource r_1 successfully because $\pi_3 = 1 > \pi_s = 0$. Note that the system ceiling $\pi_s = 3$ after time 1.25.

At time 2, $\tau_{2,1}$ arrives (with an earlier deadline than $\tau_{3,1}$) and aborts the execution of $\tau_{3,1}$ because $\tau_{3,1}$ is executing in an abortable segment $z^a_{3,1}$ and the value of $\pi_2 > \pi^{3,1}_s = 0$ ($\pi^{3,1}_s$ is calculated by Equation (1)). The non-critical parts of $\tau_{2,1}$ are executed at speed $s^*_{2,1} = 0.5$ which is calculated according to Equation (7). Since $\tau_{3,1}$ has been aborted, the system ceiling π_s becomes 0. At time 4, $\tau_{2,1}$ requests 3 units of r_1 successfully and the system ceiling π_s becomes 3. Note that every critical section has to be executed at the base processor speed $s^b = 0.8$.

At time 6, $\tau_{1,1}$ arrives with an earlier deadline than other two tasks. However, $\tau_{1,1}$ is blocked because its preemption level is not higher than the system ceiling (i.e. $\pi_1 = \pi_s = 3$) and there is no task currently executing in an abortable segment. At time 6.5, $\tau_{2,1}$ completes the execution of its first critical section, i.e. $z_{2,1}$, $\tau_{1,1}$ preempts $\tau_{2,1}$ and is executed at speed $s^*_{1,1} = 0.4$ which is calculated according to Equation (8). At time 12.75, $\tau_{1,1}$ finishes its execution and $\tau_{2,1}$ resumes. At time 16, $\tau_{2,1}$ finishes its execution and $\tau_{3,1}$ resumes. Note that the aborted critical section of $\tau_{3,1}$ has to be re-executed from the beginning. Finally, $\tau_{3,1}$ finishes its execution at time 19.75.

Based on the same power consumption function of Example 1, the energy consumption is 11.6216 Watt. Compared with Example 1, the tasks remain schedulable and it saves more than 44.12 % of energy.

6 PROPERTIES

The purpose of this section is to provide the properties of our proposed CA-SRP with DSA method for dynamic-priority real-time tasks with abortable critical sections.

Lemma 2. For any task instance $\tau_{i,j}$, the value of the dynamic speed $s^*_{i,j}$ is no more than the value of the base processor speed s^b.

Proof. This lemma is correct according to Equations (6), (7), and (8). □

Theorem 4. A set of n tasks (sorted in non-decreasing order of their preemption levels) is schedulable under CA-SRP with DSA method, if the following conditions are satisfied:

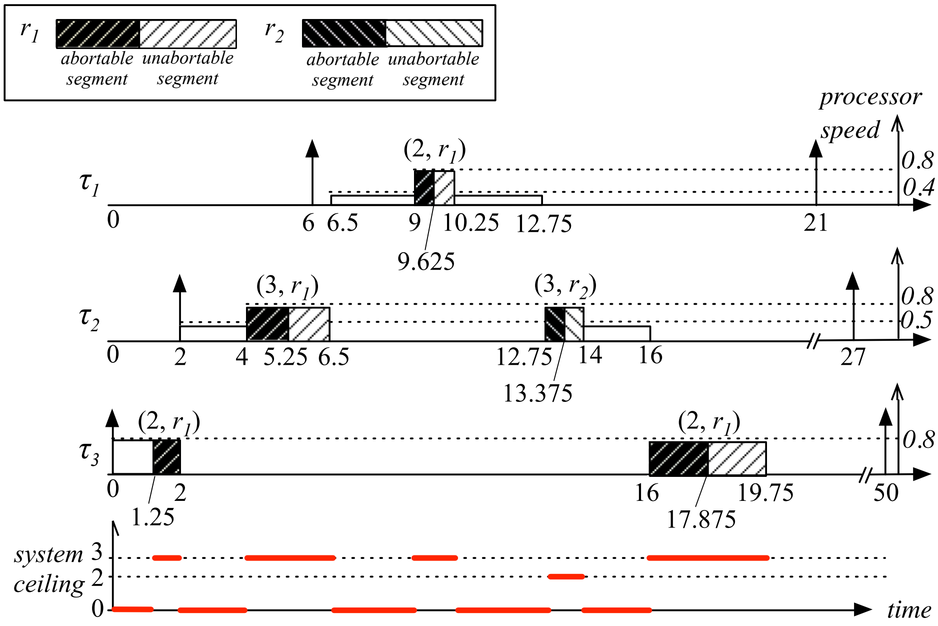

Figure 6. Example CA-SRP schedule (with DSA method)

$$\sum_{k=1}^{n} \frac{C_k + B_k}{D_k} \leq 1.$$

Proof. According to Corollary 1, tasks are schedulable when they are executed at the base processor speed s^b. Note that the utilization of a task instance $\tau_{i,j}$ is no more than $(\frac{C_i+B_i}{s^b})/D_i$ when tasks are executed at s^b.

Once the DSA method is adopted with CA-SRP, a task instance $\tau_{i,j}$ will be executed at a lower processor speed $s^*_{i,j} \leq s^b$ for its non-critical part so that $\tau_{i,j}$'s execution time will be increased. In the worst case (i.e., the task instance is not blocked), its utilization is as follows:

$$\frac{(nC_i/s^*_{i,j}) + (cC_i/s^b)}{D_i} \tag{9}$$

and the maximum value of $s^*_{i,j}$ (in Equations (6), (7), and (8)) is as follows:

$$s^*_{i,j} = \frac{s^b nC_i}{B_i + nC_i}. \tag{10}$$

By combining Equations (9) and (10), we have

$$\left(\frac{nC_i(B_i + nC_i)}{s^b nC_i} + \frac{cC_i}{s^b}\right)/D_i = \left(\frac{C_i + B_i}{s^b}\right)/D_i.$$

Hence, the utilization of any $\tau_{i,j}$ remains the same, and it completes the proof. □

Theorem 5. The CA-SRP prevents deadlocks.

Proof. Under CA-SRP, a task cannot be blocked after it has started. In other words, a task cannot be blocked while holding a resource. It avoids the so called hold-and-wait situation. Therefore, the deadlock is prevented. □

7 PERFORMANCE EVALUATION

The experiments described in this section are aimed to evaluate the capabilities of CA-SRP with DSA method on a non-ideal DVS processor. We have implemented a simulation of a DVS environment for scheduling different task workloads. The performance of the following approaches were evaluated:

Conditional abortable stack resource policy (CA-SRP). This is our proposed approach. When tasks are scheduled by CA-SRP, we also use DSA method to dynamically calculate and assign tasks' execution speeds.

Ceiling-based conditional abortable scheduling (CB-CAS). [29] To the best of our knowledge, CB-CAS is the closest related work of this paper. The design of CB-CAS is similar to that of our proposed CA-SRP. CB-CAS is also an energy-efficient scheduling approach for real-time tasks with abortable critical sections. However, the main difference between CB-CAS and CA-SRP is the method of assigning priorities to tasks. Specifically, CB-CAS and CA-SRP were designed for fixed-priority and dynamic-priority tasks, respectively. Under CB-CAS, tasks are scheduled and synchronized by RM and PCP. Whenever two tasks are conflicting for the same shared resource, CB-CAS allows the higher-priority task to abort the lower-priority task only if it is more energy efficient. CB-CAS uses a dynamic speed adjustment method to calculate and assign proper processor speeds for task executions so that the energy consumption can be reduced. Also note that CB-CAS was designed for single-unit resources. We modified CB-CAS to support multiunit resources according to Chen and Lin's work [6] for which its performance can be compared with our work.

Uniform slowdown with frequency inheritance (USFI). [13] USFI calculates a processor speed for each task's execution. When a task blocks other higher-priority tasks, it inherits the highest processor speed of the blocked tasks. We implemented this approach by using EDF and *dynamic priority ceiling protocol* (DPCP) [5] as its scheduling policy and concurrency control protocol. Note that USFI was not designed for tasks with abortable critical sections. We will not abort any task even if the task is being executed in an abortable segment. Also note that we made necessary modifications to USFI such that multiunit resources are supported.

Maximum speed (MS). The MS is the baseline approach which schedules tasks to be executed at the maximum processor speed under EDF and SRP.

In the rest of this section, we shall present the performance metrics, data sets, and experimental results.

7.1 Performance Metrics and Data Sets

The primary performance metric of interest is the energy consumption of tasks, referred to as $EnergyConsum$, which is the sum of the energy consumption of all task instances executed during the simulation time. In our simulation, the processor speeds and their power consumptions are chosen from Marvell XScale processor [10], as shown in Table 2. The energy consumption $EnergyConsum$ can be calculated by $\int_0^{simTime} PC(s(t))\,\mathrm{d}t$, where $PC()$, $s(t)$, and $simTime$ are the power consumption function, the processor speed at time t, and the simulation time. Note that the value of the power consumption of a specified processor speed can be obtained from Table 2.

Available speed (MHz)	150	400	600	800	1 000
Normalized speed	0.15	0.4	0.6	0.8	1
Power consumption (mW)	80	170	400	900	1 600

Table 2. Available speeds and power consumptions for Marvell XScale [10]

The parameter settings of the evaluated workloads are given in Table 3, which is similar to the work in [32, 13]. Each generated task set consists of 20 to 100 tasks by uniform distribution. The period of a task was selected randomly from one of the three ranges: long period (2 000 $\sim$ 5 000 ms), middle period (500 $\sim$ 2 000 ms), and short period (20 $\sim$ 200 ms). The worst-case computation amounts of the tasks in the three period ranges were selected randomly from (10 $\sim$ 500 ms), (10 $\sim$ 100 ms), and (5 $\sim$ 20 ms). Note that we assume that tasks' relative deadlines are equal to their periods, i.e., $D_i = T_i$, for $1 \leq i \leq n$.

The number of shared multiunit resources in the system was assigned randomly from 5 to 10 and the number of units for each resource was assigned randomly from 1 to 5. The number of shared resources required for a task (i.e. the number of critical sections of a task) was set from 0 to 2 by uniformly distribution. Note that the units of a task's required resource was set from 1 to the maximum units of the resources. The positions and the lengths of the critical sections within a task's execution were selected randomly. As a result, there will be sufficient number of resource conflicts and the performance of evaluated approaches can be better understood.

Parameter	Value
Utilization factor	0.4, 0.6
The number of tasks	$(20 \sim 100)$
Period ranges	long $(2\,000 \sim 5\,000\,\text{ms})$, middle $(500 \sim 2\,000\,\text{ms})$, short $(20 \sim 200\,\text{ms})$
Worst-case computation time	long $(10 \sim 500\,\text{ms})$, middle $(10 \sim 100\,\text{ms})$, short $(5 \sim 20\,\text{ms})$
The number of resources	$(5, 10)$
The number of units for each resource	$(1, 5)$
The number of resources required for a task	$(0, 2)$
Resource usage ratio *rur*	$(0 \sim 0.3)$ step by 0.05
Abortable segment ratio *asr*	$(0 \sim 0.5)$ step by 0.1
Simulation time	1 000 000 ms

Table 3. Parameters of workload

For generating feasible task sets, we set the utilization of a task set as 0.4 and 0.6. After a task set was generated, the worst-case computation amounts of tasks will be scaled so that the utilization of the task set would not exceed the desired value. The sum of the lengths of a task τ_i's critical sections is no larger than $rur \times C_i$, where rur is the *resource usage ratio* and it varied between 0 to 0.3 with an increment of 0.05. Note that tasks do not require any shared resource (i.e., tasks are independent) when $rur = 0$. The range of rur was chosen based on the fact that critical sections are usually short in practical applications. In addition, a larger value of rur might lead to overloading such that the accuracy of the experimental results will be affected. We also set the length of the abortable segment of any critical section $z_{i,j}$ no larger than $asr \times |z_{i,j}|$, where asr is the *abortable segment ratio* and is set from 0 to 0.5 with an increment of 0.1. The simulation time is 1 000 000 ms and over 100 task sets per utilization factor, resource usage ratio, and abortable segment ratio were evaluated in the simulation and their results were averaged.

7.2 Experimental Results

In the first part of our simulation, the effect of the lengths of critical sections on energy consumption is evaluated. We varied the resource usage ratio from 0 to 0.3 (step by 0.05) and fixed the abortable segment ratio to 0.3. Figures 7 a) and b) show the experimental results of evaluated approaches, where the utilization factors are 0.4 and 0.6, respectively. Note that MS was used as the baseline and the energy consumption *EnergyConsum* of the other approaches were normalized with the baseline.

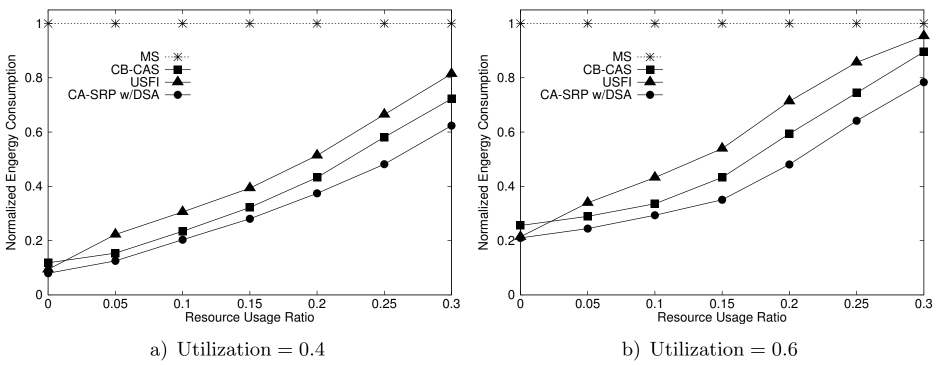

Figure 7. Normalized energy consumption with varying rur when asr is 0.3

The results show that the energy consumption of all evaluated approaches grew with the utilization of tasks, as shown in Figures 7 a) and b). The performance ranking is MS, USFI, CB-CAS, and CASRP (from the worst to the best). MS is the worst one because it always executes tasks at the maximum processor speed s_{max}. Compared with MS, all other approaches consume less energy because their tasks' execution speeds are no higher than the maximum processor speed s_{max}. In particular, the execution speeds of USFI are calculated based on the schedulability analysis so that the energy consumption can be reduced without violating tasks' timing constraints. Similar with USFI, the execution speeds calculated by CB-CAS and CA-SRP are also no higher than s_{max}. However, CB-CAS and CA-SRP outperform USFI due to their dynamic speed adjustment/assignment methods. As a result, the execution speed of a task can be slowdown at run-time so that the energy consumption can be reduced further. Note that CB-CAS and CA-SRP are designed for fixed-priority and dynamic-priority tasks, respectively. In particular, the execution speed of CB-CAS and CA-SRP are calculated based on the schedulability conditions of RM and EDF, respectively. It is obviously that the execution speeds of CB-CAS is higher than that of CA-SRP because the schedulability of RM is less than that of EDF. Furthermore, compared with CB-CAS, CA-SRP's DSA method allows tasks to be executed more times at lower processor speeds. Therefore, CA-SRP outperforms CB-CAS in all the cases.

Because the resource usage ratio is highly related on the lengths of critical sections, the probability of blocking and the blocking time of tasks will be increased when rur become higher. The execution speeds of USFI, CB-CAS and CA-SRP are calculated based on the utilization and the worst-case blocking time of tasks. Since the utilization of evaluated task sets are fixed in Figures 7 a) and b), the execution speeds are dominated by the worst-case blocking time which is varied with the resource usage ratio. Therefore, the energy consumption of tasks grew with the resource usage ratio, as shown in Figure 7. As mentioned above in this section, tasks are independent when $rur = 0$. In this case, the execution speeds of tasks only vary with the utilization of tasks. In Figures 7 a) and b), the energy consumption of CB-CAS is higher than that of USFI and CA-SRP when $rur = 0$ (i.e., tasks are independent) because the execution speeds calculated by CB-CAS are higher than those of USFI and CA-SRP.

In this second part of our simulation, the effect of the lengths of abortable segments on energy consumption is evaluated for our proposed CA-SRP with DSA method. The abortable segment ratio determines the lengths of abortable segments of critical sections. Hence, the probability of aborting and the re-execution time will be increased when asr becomes higher. We fixed the utilization to 0.4 and varied the rur from 0 to 0.3. Figure 8 shows the experimental results of our proposed CA-SRP with DSA method where asr is set as 0 to 0.5 (step by 0.1). As the lengths of abortable segments are increased with asr, the energy consumption is getting higher because the number of abortings is also increased. Note that the re-execution time for aborted tasks are also increased with asr. When $asr = 0$, CA-SRP degrades to have a blocking option only when a resource conflict occurs, thus there is no re-execution cost.

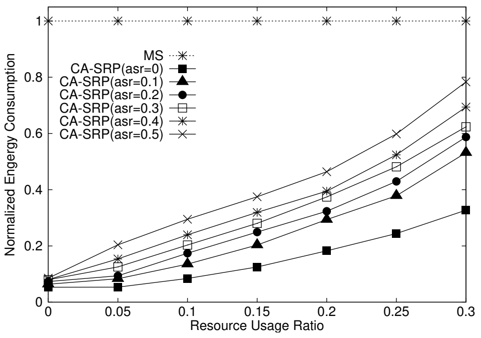

Figure 8. Normalized energy consumption with varying rur with different settings of asr

8 CONCLUSION AND FURTHER WORK

In the recent years, researchers have proposed energy-efficient ceiling-based protocols to manage concurrent accesses to shared resources for real-time systems. However, ceiling-based protocols have a problem of ceiling blocking which imposes a great impact on the performance of real-time systems. In this paper, we are interested in energy-efficient scheduling of dynamic-priority, periodic, preemptible real-time tasks with abortable critical sections on a non-ideal DVS platform. Based on EDF and SRP, we propose the conditional abortable stack resource policy (CA-SRP) to resolve the ceiling-blocking problem by incorporating a conditional abort rule into SRP. We also propose a dynamic speed assignment (DSA) method to dynamically calculate proper processor speeds for task execution so that the energy consumption can be reduced further. The schedulability analysis and the properties of the CA-SRP are given in this paper. The capability of our proposed approaches were evaluated by a series of experiments, for which we have some encouraging results. By using the CA-SRP to schedule tasks on a DVS platform, the performance can be more predictable while the energy consumption can be reduced significantly compared with other existing work.

Acknowledgements

This paper is supported in part by research grants from the Ministry of Science and Technology (MOST) of Taiwan under the grant number MOST-105-2628-E-153-001-MY2.

REFERENCES

[1] Baker, T. P.: A Stack-Based Resource Allocation Policy for Real-Time Processes. Proceedings of the IEEE 11th Real-Time Systems Symposium (RTSS), Lake Buena Vista, Florida, USA, December 4–7, 1990, pp. 191–200.

[2] Buttazzo, G. C.: Rate Monotonic vs. EDF: Judgment Day. Real-Time Systems, Vol. 29, 2005, No. 1, pp. 5–25.

[3] Chen, J.-J.—Kuo, C.-F.: Energy-Efficient Scheduling for Real-Time Systems on Dynamic Voltage Scheduling (DVS) Platforms. Proceedings of the 13th IEEE Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2007, doi: 10.1109/RTCSA.2007.37.

[4] Chen, J.-J.—Kuo, T.-W.: Procrastination Determination for Periodic Real-Time Tasks in Leakage-Aware Dynamic Voltage Scaling Systems. Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD), San Jose, CA, USA, 2007, pp. 284–294.

[5] Chen, M.-I.—Lin, K.-J.: Dynamic Priority Ceilings: A Concurrency Control Protocol for Real-Time Systems. Real Time Systems Journal, Vol. 2, 1990, No. 1, pp. 325–346, doi: 10.1007/BF01995676.

[6] Chen, M.-I.—Lin, K.-J.: A Priority Ceiling Protocol for Multiple-Instance Resources. Proceedings of Twelfth Real-Time Systems Symposium, 1991, pp. 140–149, doi: 10.1109/REAL.1991.160367.

[7] Elewi, A. M.—Awadalla, M. H. A.—Eladawy, M. I.: Energy-Efficient Multi-Speed Algorithm for Scheduling Dependent Real-Time Tasks. Proceedings of the International Conference on Computer Engineering and Systems (ICCES 2008), Cairo, Egypt, November 2008, pp. 237–242, doi: 10.1109/ICCES.2008.4773003.

[8] Elewi, A. M.—Awadalla, M. H. A.—Eladawy, M. I.: Energy Efficient Real-Time Scheduling of Dependent Tasks Sharing Resources. Proceedings of the 2008 High Performance Computing and Simulation Conference (HPCS), Nicosia, Cyprus, June 3–6, 2008, pp. 107–116.

[9] Huang, J.—Stankovic, J. A.—Ramamritham, K.—Towsley, D.: On Using Priority Inheritance in Real-Time Databases. Proceedings of the 12th Real-Time Systems Symposium (RTSS), December 1991, pp. 210–221, doi: 10.1109/REAL.1991.160376.

[10] Intel. Intel XScale Core Developer's Manual, 2004.

[11] Jejurikar, R.—Gupta, R.: Energy Aware Task Scheduling with Task Synchronization for Embedded Real Time Systems. Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES), 2002, pp. 164–169, doi: 10.1145/581630.581655.

[12] Jejurikar, R.—Gupta, R.: Dual Mode Algorithm for Energy Aware Fixed Priority Scheduling with Task Synchronization. Proceedings of the Workshop on Compilers and Operating Systems for Low Power, 2003.

[13] Jejurikar, R.—Gupta, R.: Energy Aware Task Scheduling with Task Synchronization for Embedded Real Time Systems. IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, Vol. 25, 2006, No. 6, pp. 1024–1037, doi: 10.1109/TCAD.2005.855964.

[14] Lam, K.-Y.—Ng, J. K.-Y.: A Conditional Abortable Priority Ceiling Protocol for Scheduling Mixed Real-Time Tasks. Journal of Systems Architecture, Vol. 46, 2000, No. 7, pp. 573–585.

[15] Lee, J.—Koh, K.—Lee, C.-G.: Multi-Speed DVS Algorithms for Periodic Tasks with Non-Preemptible Sections. Proceedings of the 13th IEEE Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2007, pp. 459–468, doi: 10.1109/RTCSA.2007.50.

[16] Liu, C. L.—Layland, J. W.: Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. Journal of the Association for Computing Machinery (JACM), Vol. 20, 1973, No. 1, pp. 46–61.

[17] Mochocki, B.—Hu, X. S.—Quan, G.: Transition-Overhead-Aware Voltage Scheduling for Fixed-Priority Real-Time Systems. ACM Transactions on Design Automation of Electronic Systems, Vol. 12, 2007, No. 11, doi: 10.1145/1230800.1230803.

[18] Mohan, A. L.—Pillai, A. S.: Dynamic Voltage Scaling with Reduced Frequency Switching and Preemptions. International Journal of Electrical and Electronics Engineering, Vol. 1, 2011, No. 1, pp. 10–14.

[19] Mok, A. K.: Fundamental Design Problems for the Hard Real-Time Environment. Ph.D. Thesis, MIT, Cambridge, MA, 1983.

[20] Muhammad, F.—Khurram, B. M.—Muller, F.—Belleudy, C.—Auguin, M.: Precognitive DVFS: Minimizing Switching Points to Further Reduce the Energy Consumption. Proceedings of the Work-in-Progress Session of the Real-Time and Embedded Technology and Applications Symposium (RTAS), 2008, pp. 9–12.

[21] Rajan, D.—Zuck, R.—Poellabauer, C.: Workload-Aware Dual-Speed Dynamic Voltage Scaling. Proceedings of 12th IEEE Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Sydney, Australia, August 16–18, 2006, pp. 251–256.

[22] ZHURAVLEV, S.—SAEZ, J. C.—BLAGODUROV, S.—FEDOROVA, A.—PRIETO, M.: Survey of Energy-Cognizant Scheduling Techniques. IEEE Transactions on Parallel and Distributed Systems, Vol. 24, 2013, No. 7, pp. 1–19, doi: 10.1109/TPDS.2012.20.

[23] SHA, L.—RAJKUMAR, R.—LEHOCZKY, J. P.: Priority Inheritance Protocols: An Approach to Real-Time Synchronization. IEEE Transactions on Computers, Vol. 39, 1990, No. 9, pp. 1175–1185.

[24] SHU, L.-C.—YOUNG, M.: A Mixed Locking/Abort Protocol for Hard Real-Time Systems. Proceedings Real-Time Operating Systems and Software, May 1994, pp. 102–106.

[25] TAKADA, H.—SAKAMURA, K.: Real-Time Synchronization Protocols with Abortable Critical Sections. Proceedings of 1st International Workshop on Real-Time Computing Systems and Application, 1994, pp. 48–52.

[26] TOKUDA, H.—NAKAJIMA, T.: Evaluation of Real-Time Synchronization in Real-Time Mach. Proceedings of the USENIX Mach Symposium, November 1991, pp. 213–221.

[27] WU, J.: Energy Efficient Dual Execution Mode Scheduling for Real-Time Tasks with Shared Resources. International Journal of Computer Systems Science and Engineering (CSSE), Vol. 31, 2016, No. 3, pp. 239–253.

[28] WU, J.: Energy-Efficient Scheduling of Real-Time Tasks with Shared Resources. Future Generation Computer Systems (FGCS), Vol. 56, 2016, pp. 179–191, doi: 10.1016/j.future.2015.05.012.

[29] WU, J.—KE, K.-L.: Energy-Efficient Real-Time Scheduling of Tasks with Abortable Critical Sections. Journal of Information Science and Engineering (JISE), Vol. 30, 2014, No. 3, pp. 765–786.

[30] WU, J.—WU, J.-X.: An SRP-Based Energy-Efficient Scheduling Algorithm for Dependent Real-Time Tasks. International Journal of Embedded Systems (IJES), Vol. 6, 2014, No. 4, pp. 335–350.

[31] YAO, F.—DEMERS, A.—SHENKER, S.: A Scheduling Model for Reduced CPU Energy. Proceedings of 36th IEEE Symposium on Foundations of Computer Science (FOCS), Milwaukee, USA, October 23–25, 1995, pp. 374–382.

[32] ZHANG, F.—CHANSON, S. T.: Processor Voltage Scheduling for Real-Time Tasks with Non-Preemptible Sections. Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS), 2002, pp. 235–245, doi: 10.1109/REAL.2002.1181578.

[33] ZHANG, F.—CHANSON, S. T.: Blocking-Aware Processor Voltage Scheduling for Real-Time Tasks. ACM Transactions on Embedded Computing Systems, Vol. 3, 2004, No. 2, pp. 307–335.

Jun Wu is Associate Professor in the Department of Computer Science and Information Engineering at National Pingtung University, Pingtung, Taiwan. His research interests include real-time embedded systems, energy-efficient design, and virtualization technologies. He received Best Paper Award from the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA) in 2005. He received his B.Sc. degree in computer science and information engineering from I-Shou University, Kaohsiung, Taiwan, in 1996. He received his MBA degree in information management from National Yunlin University of Science and Technology, Yunlin, Taiwan, in 1998. He received his Ph.D. degree in computer science and information engineering from National Chung Cheng University, Chiayi, Taiwan, in 2004. He is a member of the IEEE.