Research Article

A Work-Demand Analysis Compatible with Preemption-Aware Scheduling for Power-Aware Real-Time Tasks

Da-Ren Chen, Mu-Yen Chen, You-Shyang Chen, and Lin-Chih Chen

1 Department of Information Management, National Taichung University of Science and Technology, No. 129, Section 3, Sanmin Road, North District, Taichung City 404, Taiwan
2 Department of Information Management, Hwa Hsia Institute of Technology, No. 111, Gongzhuan Road, Zhonghe District, New Taipei City 235, Taiwan
3 Department of Information Management, National Dong Hwa University, No. 1, Section 2, Da Hsueh Road, Shoufeng, Hualien 97401, Taiwan

Correspondence should be addressed to Da-Ren Chen; danny@nutc.edu.tw

Received 4 January 2013; Accepted 13 April 2013

Academic Editor: Yang Tang

Copyright © 2013 Da-Ren Chen et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Due to the importance of slack time utilization for power-aware scheduling algorithms, we propose a work-demand analysis method called parareclamation algorithm (PRA) to increase slack time utilization of the existing real-time DVS algorithms. PRA is an online scheduling for power-aware real-time tasks under rate-monotonic (RM) policy. It can be implemented and fully compatible with preemption-aware or transition-aware scheduling algorithms without increasing their computational complexities. The key technique of the heuristics method doubles the analytical interval and turns the deferrable workload out the potential slack time. Theoretical proofs show that PRA guarantees the task deadlines in a feasible RM schedule and takes linear time and space complexities. Experimental results indicate that the proposed method combining the preemption-aware methods seamlessly reduces the energy consumption by 14% on average over their original algorithms.

1. Introduction

Power management is increasingly becoming a design factor in portable and hand-held computing/communication systems. Energy minimization is critically important for devices such as laptop computers, smartphones, PDAs, wireless sensor networks (WSNs), and other mobile or embedded computing systems simply because it leads to extended battery lifetime. The power consumption problem has been addressed in the last decade with a multidimensional effort by the introduction of engineering components and devices that consume less power, low power techniques involving the designs of VLSI/IC, computer architecture, algorithm, and compiler developments.

Recently, dynamic power management (DPM) and dynamic voltage scaling (DVS) have been employed as available techniques to reduce the energy consumption of CMOS microprocessor system. DPM changes the power state of cores on chip to lower the energy consumption according to the performance constraints. DVS involves dynamically adjusting the voltage and frequency (hence, the CPU speed). By reducing the frequency at which a component operates, a specific operation will consume less energy but may take longer to complete. Although reducing the frequency alone will reduce the average energy used by a processor over that period of time, it may not always deliver a reduction in energy consumption overall, because the power consumption is linearly dependent on the increased time and quadratically dependent on the increased/decreased voltage. In the context of dynamic voltage scaled processors, DVS in real-time systems is a problem that assigns appropriate clock speeds to a set of periodic tasks and adjusts the voltage accordingly such that no task misses its predefined deadline while the total energy savings in the system is maximized.

Many studies have proposed different real-time scheduling based on different system models [1–9], such as online...
and offline scheduling, handling discrete/continuous voltage levels, assuming average-case execution time (ACET), best-case execution time (BCET), or worst-case execution time (WCET) of each task, allowing intratask/intertask voltage transitions, and assuming fixed/dynamic priority assignment. These approaches have a common objective and encounter the same difficulties. Because reducing the supply voltage decreases the clock speed of processors [10], most DVS algorithms for real-time systems reduce supply voltage dynamically to the lowest possible level while satisfying the soft/hard timing constraints of each task. To satisfy the timing constraints of real-time tasks, DVS technique must utilize available slack time when adjusting voltage/speed levels. Consequently, the energy efficiency of a DVS algorithm markedly depends on the accuracy of computing available slack time.

Work-demand analysis on embedded real-time scheduling has been investigated by previous studies [3, 5–7, 11]. Pillai and Shin [7] proposed a cycle-conserving rate-monotonic (ccRM) scheduling scheme that contains offline and online algorithms. The offline algorithm computes the WCET of each task and derives the maximum speed needed to meet all tasks deadlines. It recomputes the utilization by comparing the actual time for completed tasks with WCET schedule. In other words, when a task completes early, they have to compare the used actual processor cycles to a precomputed worst-case execution time schedule. This WCET schedule is also called canonical schedule [1] whose length could be the least common multiplier of task periods. ccRM is a conservative method, as it only considers possible slack time before the next task arrival (NTA) of a current job. Gruian proposed a DVS method for offline task stretching and online slack distribution [3]. The offline part of this method consists of two separate techniques. One focuses on the intratask stochastic voltage scheduling that employs a task-execution length probability function. The second technique computes stretching factors by using a response time analysis. It is similar to Pillai and Shin’s offline technique, but instead of adopting a stretching factor for all tasks before NTA, Gruian assigns a different stretching factor to the individual task within the longest task period. Kim et al. [6] proposed a greedy online algorithm called the low-power work-demand analysis (lpWDA) that derives slack from low-priority tasks, as opposed to the method in [3, 7] that gains slack time from high-priority tasks. This algorithm also balances the gap in voltage levels between high-priority and low-priority tasks. Its analysis interval limited by the longest of task periods is longer than NTA. Thus, lpWDA gains more energy saving than the previous rate-monotonic (RM) DVS schemes applying NTA. Many slack time analysis methods considered additional assumptions [4, 11, 12]. Kim et al. proposed a preemption-aware DVS algorithm based on lpWDA, which is composed of accelerated-completion (lpWDA-AC) and delayed-preemption (lpWDA-DP) techniques to decrease the preemption times of DVS schedules [11]. lpWDA-AC attempts to avoid preemption by adjusting voltage/clock speed, such that it is higher than the lowest possible values computed using lpWDA. lpWDA-DP postpones preemption points by delaying an activated high-priority task as late as possible while guaranteeing a feasible task schedule. Both techniques reduce energy consumption more than the initial ccRM and lpWDA techniques on the assumption of context-switching overhead. Mochocki et al. in [12] also proposed a transition-aware DVS algorithm for decreasing the number of voltage/speed adjustments, called the low-power limited-demand analysis with transition overhead (lpLDAT) scheme, which accounts for both time and energy transition overhead. Its algorithm computes an efficient speed level based on average-case workload; notably, this speed can be used as a limiter. If the limiter is higher than the speed predicted by lpWDA, lpLDAT knows that lpWDA is being too aggressive and applies the limiter to the present schedule. On the assumption of transition overhead, this technique with slack time analysis also saves considerable energy when compared with that by the previous methods. He and Jia [4] developed a fixed-priority scheduling with threshold (FPPT) scheme that eliminates unnecessary context switches, thereby saving energy. FPPT assigns each task a pair of predefined priority and corresponding preemption threshold. He et al. applied a novel algorithm to compute a static slowdown factors by formulating the problem as a linear optimization problem. In addition, they considered energy consumption of a task set under different preemption threshold assignments.

Recently, experimental results obtained by Kim et al. [6] indicated that recent DVS algorithms for fixed-priority real-time tasks are less efficient than that of dynamic-priority tasks, leading to more improvements for a better DVS method. The main reason for energy inefficiency of RM DVS scheduling is that, in RM schedules, priority-based slack-stealing methods do not work as efficiently as they do in earliest-deadline first (EDF) scheduling [6]. In the EDF schedules, high-priority tasks play an efficient slack distributor of tasks because their slack can be utilized fully by tasks starting before NTA. Therefore, the energy saving achieved by EDF scheduling algorithms, such as that by the ccEDF [7], DRA, and AGR [1] is close to the theoretical lower bound [13].

2. Motivations

So far, there are a large number of studies on DVS-based RM scheduling for energy saving [1–4, 6–8, 11–14]; most existing studies are proposed for computing and predicting the length and occurrence of slack time. The reason is that the more precise estimation on the slack time, the more energy efficiency we obtain. Those methods for computing available slack time either construct a canonical schedule and compare it to current schedule or propose best-effort algorithms under empirical rules and heuristics. Those methods adopting different strategies and assumptions such as task preemption or voltage transition time on the similar models gain considerable energy saving, but few of them can be combined without difficulty to further enhance their performance. Additionally, modern processor with DVS or DPM feature must be equipped with dc-to-dc converter that varies the processor speed in appropriate levels and requires additional switching time and power [15]. It is harmful to
power saving in a system when many fragments of short slack time appear. Many of those methods also propose the notions of postponing and advancing task execution for increasing the length of slack time. Their performances for accumulating a continuous slack time are not impressive due to short analysis interval adopted in the schedules. Therefore, it is necessary to study a transplanted method that can cooperate with different existing methods without modification. This idea originates from the layered architecture used in designing computer software, hardware, and communications in which system or network components are isolated in layers so that changes can be made in one layer without affecting the others. The proposed method according the notion also requires the ability to compute and accumulate the slack time solely. By applying the layered architecture, it can also pass the slack time to lower-layered methods and reveals synergy effect to enhance overall energy saving.

In this paper, we propose an online work-demand analysis called parareclamation algorithm (PRA) for RM scheduling which computes the length of potential slack in an interval which is two times longer than the longest task periods. PRA does not rely on the simulation for stochastic data which usually varies according to different applications, and can be applied to many RM scheduling algorithms with various criteria. Moreover, the proposed algorithm has a time complexity of $O(n)$ where $n$ is the number of tasks. In other words, it does not increase computational complexity of the existing online RM scheduling algorithms. Experimental results indicate that existing RM DVS algorithms combined with the proposed method can reduce energy consumption by 5%-21% compared with that by initial algorithms such as lpWDA and lpLDAT.

The remainder of this paper is organized as follows. Section 3 introduces the preliminaries of power-aware real-time scheduling. Section 4 introduces our technique and algorithm. Section 5 provides theorems to prove the schedulability of PRA as well as lpWDA. We present the performance evaluation in Section 6. Section 7 gives conclusions and the directions for future work.

3. Preliminaries

This paper focuses on how to obtain additional slack for existing RM DVS scheduling methods. Many slack time analysis techniques with different purposes (e.g., transition-aware and preemption-aware schemes) can utilize PRA easily; throughout this paper, these techniques are called host algorithms of PRA. This section also outlines the ideas underlying the lpWDA algorithm. Other techniques, such as the lpLDA, lpWDA-AC, lpWDA-DP [11], and lpLDAT [12] techniques, are abridged.

3.1. System Model. This paper considers preemptive hard real-time systems in which periodic real-time tasks are scheduled under an RM scheduling policy. The DVS processor used in the model operates at a finite set of supply voltage levels $V = \{v_1, ..., v_{\max}\}$, each with an associated speed. Processor speed is normalized by $S_{\max} = 1$ corresponding to $v_{\max} = 1$, yielding a set $S = \{s_1, ..., s_1\}$ of speed levels. A set of $n$ periodic tasks is denoted by $T = \{\tau_1, \tau_2, ..., \tau_n\}$, where the tasks are assumed mutually independent. Each task $\tau_i$ is described by its worst-case execution cycles $w_i$ and average-case execution cycles $a_i$ ($w_i \geq a_i$). Throughout this paper, the execution cycles of each task are called work for short. Additionally, each task $\tau_i$ has a shorter period length $p_i$ (i.e., a higher priority) than that of $\tau_j$ when $i < j$, and $p_i$ is the longest of task periods. The relative deadline $d_i$ of $\tau_j$ is assumed equal to its period length $p_i$. Each task is invoked periodically by a job, and the $k$th job of task $\tau_j$ is $\tau_{jk}$. The first job of each task is assumed activated at time $t = 0$. Each job is described by a release time, $r_{ij}$, deadline, $d_{jk}$, and number of cycles that have been executed $e^k_{ik}$. The utilization $U$ of a task set $T$ is denoted by $\sum_{i \in T} (w_i / p_i)$. During run time, we refer to the earliest job of each task not completed as the current job for that task, and that job is indexed with cur. The deadline of the current job for task $\tau_i$ is $d^*_{ij}$, and $e^k_{ij}$ denotes the number of cycles that the current job of $\tau_i$ has executed.

Without loss of generality, when $\tau_i$ is the first scheduled task after time $r_{nk-1}$, where $i \neq n$, the bottleneck (shortened to bn) is the next release time of $\tau_n$ (i.e., the $r_{nk}$). In the work-demand analysis method, available slack in the interval $[b, r_{nk+1})$ is estimated.

3.2. Low-Power Work-Demand Analysis (lpWDA). This section briefly introduces an online DVS scheme called lpWDA [6]. Notations $e_{\text{Exchange}}, e^\text{right}_{a}$, and $r_{\text{any}}$ belong to PRA algorithm and are presented in Section 4. In line 2 of Algorithm 1, $\epsilon$ is an infinitesimal, and readyQ contains the currently activated tasks, and its subset, $\Gamma_a^\text{ACT}(t)$, containing the active tasks is

$$\Gamma_a^\text{ACT}(t) := \{\tau_k \mid k < a, \tau_k \in \text{readyQ}\}. \quad (1)$$

In the lpWDA, the tasks in readyQ are scheduled according to RM priority policy. When a task is activated (released), its job $\tau_{nk}$ is moved to readyQ, and the remaining WCET of this job is set to $w_{\text{cur}}$, which is $W_{\text{rem}}(t) = w_{\text{cur}}$. When $\tau_n$ is executed at time $t$, load$(t)$ is the amount of work required to be processed in $[t, d_{nk}]$. In Algorithms 1, 2, and 3 and Procedure 1, lpWDA performs in the following steps. First, the system is initialized by setting the initial upcoming deadlines (ud) and remaining worst-case execution ($w_{\text{rem}}$) of each task. When $\tau_a$ is active at time $t$, notation $\text{ud}_k$ of each task $\tau_k$ is defined as follows [6]:

$$\text{ud}_k = \left[ t + \frac{t + \epsilon \gamma_{\text{E}}}{\rho_k} \right] \times \rho_k, \quad (2)$$

where $\epsilon$ is the infinitesimal. The jobs which are active during $[t, \max(\text{ud}_k(t))]$ will be examined for slack estimation. $H_{\text{E}}(t)$ denotes the estimation of higher-priority work that must be executed before $\text{ud}_a$ (lines 1-2). Whenever a job $\tau_a$ is completed or preempted at time $t$, the remaining work $w_{\text{rem}}(t)$, upcoming deadline $\text{ud}_a$, and high-priority work $H_{\text{E}}(t)$ are updated in line 4. In lines 5–8, when a job $\tau_a$ is scheduled for execution at time $t$, Algorithm 2 computes the available slack for $\tau_a$ according to $H_\beta(t)$ and $L_\beta(t)$ (see...
(1) set \( ud_{\alpha} := p_{\alpha} e_{\text{Exchange}} := 0 \) and \( \tau_{\text{asyn}} := 0 \);
(2) Compute \( H_{\alpha}(t) := \sum_{\tau_k \in T_A \cap T(\alpha)} W_{\alpha}^{\text{rem}}(t) + \sum_{\tau_i = 1}^{n-1} \left[ \left( (ud_{\alpha} - \varepsilon) / p_i \right) - \left( (t + \varepsilon) / p_i \right) \right] \cdot w_i \);
(3) When a job \( \tau_{\alpha} \) is activated, set \( u_{\alpha}^{\text{rem}}(t) := u_{\alpha} \);
(4) When a job \( \tau_{\alpha} \) is completed or preempted, \( \text{UpdateLoadInfo}(u_{\alpha}^{\text{rem}}(t), ud_{\alpha}, H_{\alpha}(t)) \);
(5) When a job \( \tau_{\alpha} \) is scheduled for execution

\[
\begin{align*}
&\quad e_{\text{Exchange}} := \text{PRA}(t, T); \quad \text{//get additional slack time} \\
&\quad \text{if } \tau_{\alpha} = \tau_{\text{asyn}} \text{ and } \tau_{\alpha} \in T_A \cap T(\alpha) < e_{\text{Exchange}} \text{ then } \tau_{\alpha} \text{ has the lowest priority in } \text{readyQ}(t) \;
\end{align*}
\]

(6) \( \text{slack}_{\alpha}(t) := \text{CalcSlackTime}(e_{\text{Exchange}}); \quad \text{//get slack time}
(7) set the clock frequency as \( f_{\text{clk}}(t) := u_{\alpha}(t)/(s_{\alpha}(t) + u_{\alpha}^{\text{rem}}(t)) \cdot f_{\text{max}} \);
(8) \( f_{\text{clk}} := \max(f_{\text{clk}}, f_{\text{acl}}) \);
(9) \( \text{Set the voltage accordingly;}

\text{Algorithm 1: IpWDA (PRA with \(*\)). Compute the available execution time and set the voltage/clock speed for } \tau_{\alpha}.

\text{Input: the active task } \tau_{\alpha}, \text{ readyQ and current time } t \\
\text{Output: the slack time } \text{slack}_{\alpha}(t) \text{ for } \tau_{\alpha}

(12) Identify the task \( \tau_{\beta} \) that has the earliest upcoming deadline among tasks whose priorities are not higher than that of \( \tau_{\alpha} \);
(13) \( L_{\beta}(t) := \text{CalcLowerPriorityWork}(\tau_{\beta}, t, e_{\text{Exchange}}) \);
(14) \( \text{load}_{\beta}(t) := u_{\beta}^{\text{rem}}(t) + H_{\beta}(t) + L_{\beta}(t) \);
(15) \( \text{slack}_{\alpha}(t) := \max(0, \text{ud}_{\alpha} - t - \text{load}_{\beta}) \);
(16) \( \text{return} \text{slack}_{\alpha}(t) \);

\text{Algorithm 2: CalcSlackTime(additional slack } e_{\text{Exchange}}\).

lines 13 and 14), where \( ud_{\beta} \) is the earliest upcoming deadline with respect to \( \tau_{\alpha} \). Notably, function \( L_{\beta}(t) \) computing the amount of low-priority work is performed recursively until it finds \( \tau_{\alpha} \) with the longest of task periods and lowest priority with respect to \( \tau_{\alpha} \). As defined in Section 3.1, the length of interval \([0, bn] \) is \( p_{\beta} \). Then, IpWDA computes the length of slack-time stealing from low-priority tasks in the interval \([\tau_{\alpha}, bn] \) and applies the slack to the current job. Therefore, Algorithms 2 and 3 play crucial roles in slack-time analysis and dominate the run time complexity of IpWDA algorithm. Formally, to describe the slack analysis method using IpWDA, the following notations are defined:

\[
\begin{align*}
\text{load}_{\alpha}(t) &:= \text{the amount of work required to be processed in interval } [t, d_{\alpha}); \\
\text{slack}_{\alpha}(t) &:= \text{the available slack for } \tau_{\alpha} \text{ scheduled at time } t \text{ can be computed as follows:}
\end{align*}
\]

\[
\text{slack}_{\alpha} = d_{\alpha} - t - \text{load}_{\alpha}(t).
\] (3)

In (3), \( \text{load}_{\alpha}(t) \) consists of three types of work: (1) \( W_{\alpha}^{\text{rem}}(t) \), (2) \( H_{\alpha}(t) \) from the higher-priority tasks, and (3) \( L_{\alpha}(t) \) from the lower-priority tasks. The work required by higher-priority tasks is derived as follows:

\[
H_{\alpha}(t) = H_{\alpha}^{\text{past}}(t) + H_{\alpha}^{\text{future}},
\] (4)

where \( H_{\alpha}^{\text{past}}(t) \) denotes the work required by uncompleted tasks released before \( t \), and \( H_{\alpha}^{\text{future}}(t) \) denotes the work released during \([t, d_{\alpha}] \). We compute \( H_{\alpha}^{\text{past}}(t) \) and \( H_{\alpha}^{\text{future}}(t) \) as follows:

\[
\begin{align*}
H_{\alpha}^{\text{past}}(t) &= \sum_{\tau_{\epsilon} \in T_{A \cap T}(\alpha)} W_{\epsilon}^{\text{rem}}(t), \\
H_{\alpha}^{\text{future}}(t) &= \sum_{i=1}^{n-1} \left( d_{\alpha} - \varepsilon \right) / p_i - \left( t + \varepsilon \right) / p_i + 1 \right) \cdot w_{\epsilon}, \\
L_{\alpha}(t) &= \left( \text{load}_{\beta}(t) - W_{\alpha}^{\text{rem}}(t) - H_{\alpha}(t) - \left( \text{ud}_{\beta}(t) - d_{\alpha} \right) \right)^{+}, \\
\text{load}_{\alpha}(t) &= H_{\alpha}(t) + L_{\alpha}(t) + W_{\alpha}^{\text{rem}}(t),
\end{align*}
\] (5) (6) (7) (8)

where notation \( x^+ \) stands for \( \max(x, 0) \). Equations (6), (7), and (8) are repeated iteratively until \( \tau_{\alpha} \) is the lowest priority task in \( T \) (i.e., \( L_{\alpha}(t) = 0 \)). Conceptually, IpWDA uses this linear-time heuristics to estimate available slack in an interval up to the upcoming deadline of lower-priority tasks.
Input: a reference task \( r_\beta \) and current time \( t \)

Output: the amount of lower-priority work needed to be done before \( ud_\beta \)

If \( r_j \) is identical to \( r_\beta \), then return 0;

Identify the task \( r_\gamma \) that has the earliest upcoming deadline among the tasks whose priorities are lower than that of \( r_\beta \);

\[ L_\gamma(t) := \text{CalcLowerPriorityWork}(r_\gamma, t, e_{\text{Exchange}}); \]

\[ \text{load}_\gamma(t) := w_{\text{rem}}(t) + H_\gamma(t) + L_\gamma(t); \]

If \( ud_\gamma \) is multiple of \( p_\gamma \), then \( ud_\gamma := ud_\gamma + e_{\text{Exchange}} \);

\[ L_\beta(t) := \max(0, \text{load}_\gamma(t) - w_{\gamma}(t) - H_\beta(t) - ud_\gamma + ud_\beta); \]

return \( L_\beta(t); \)

Algorithm 3: CalcLowerPriorityWork(additional slack \( e_{\text{Exchange}} \).

### Procedure 1: PRA (time \( t \), task set \( T \)).

#### 3.3. Motivational Example. The proposed method is to provide lpWDA-based algorithms (e.g., lpWDA, lpLDAT, lpWDA-DP, and lpWDA-AC) with a subroutine to improve their work-demand analysis. The main advantage is that PRA can be independent of each function-specific slack analysis method. For instance, the main purpose of lpWDA-AC and lpWDA-DP techniques is to decrease context-switch overhead while that of lpLDAT is to reduce transition time and energy overhead. PRA can work together with these lpWDA-based algorithms to enhance their slack computation capability.

**Example 1.** Consider a periodic task set \( T \) in Table 1, which presents the period length, WCET, and ACET of each task. Figure 1(a) presents the execution schedule under the worst-case workload in the first hyperperiod. Figure 1(b) shows the speed schedule using lpWDA algorithm for task set \( T \) and assumes that actual work of each task equals its ACET. Before assigning \( \tau_{1,1} \) at time \( t = 0 \), lpWDA computes available slack time in an interval up to \( d_{3,1} = 6 \) by calling Algorithm 3, recursively. However, interval \([0, 6)\) has no slack-time under the WCET schedule. If the length of the analysis interval is extended to \( 2 \times p_\beta \), one unit of slack time is derived from \( 2 \times p_\beta - \sum_{j=1}^{n} \lfloor 2p_j/p_\beta \rfloor \times w_\gamma \). The slack in \([11, 12)\) can be moved backward to the current scheduling point by a deferred execution of earlier work. For instance, in Figure 1(a), the slack in interval \([11, 12)\) can be exchanged with the work in interval \([7, 8)\), and then slack in interval \([7, 8)\) can be exchanged with the work in interval \([4, 5)\), and it can be exchanged once again with the work in interval \([2, 3)\). Finally, the slack in interval \([2, 3)\) can be exchanged with the work in interval \([1, 2)\). Therefore, \( \tau_{1,1} \) is scheduled with speed \( S_{1,1} = (w_\gamma/(w_\gamma + 1)) \) (Figure 1(c)). Additional slack can be reclaimed without deadline missing from the interval that is, two times longer than the longest task period. Notably, this idea actually neither moves all of the jobs of a schedule to readyQ (e.g., \( t = 0 \)) nor Exchanges the slack with work for using this slack time. However, this primitive idea does not work in some situations. For example, in Figure 1(d), when \( p_2 \) is increased to 6, slack in the interval \([11, 12)\) cannot be transferred before \( t = 6 \). In fact, jobs \( \tau_{1,3}, \tau_{2,2}, \) and \( \tau_{3,2} \) are released simultaneously at time 6. The slack in interval
TABLE I: An example of real-time task set $T$.

<table>
<thead>
<tr>
<th>Task</th>
<th>Period ($p_i$)</th>
<th>WCET ($w_{ci}$)</th>
<th>ACET ($ac_i$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>3</td>
<td>1.0</td>
<td>0.5</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>4</td>
<td>1.0</td>
<td>0.5</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>6</td>
<td>2.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

[11, 12] cannot follow this idea, because a deadline is likely to be missed by one of those three jobs. Our goal is to devise an efficient work-demand analysis method that obtains additional slack while satisfying the tasks’ deadline.

4. Work-Demand Computation

Let $r_{n,k}$ be the bn of $\tau_i$, which is the first scheduled job at time $t$ where $t \geq r_{n,k-1}$. PRA computes the length of additional slack in the interval $[bn, r_{n,k+1})$. As long as the slack time can be reclaimed at a time earlier than bn, lpWDA can utilize it by postponing lower-priority task and improve energy efficiency of schedules. Why PRA focuses on slack computation in the interval $[bn, r_{n,k+1})$ while longer or shorter intervals? Even if all job (except $\tau_n$) periods are within $[bn, r_{n,k+1})$ and cannot make a target slack be available for the task right side to bn, job $\tau_n$ can still postpone its work for moving the slack forward and approaching bn. For example, in Figure 1(a), when the period length of $\tau_1$ is increased from 3 to 4, the slack in interval $[11, 12]$ cannot be reclaimed by postponing the work of $\tau_{1,2}$ or $\tau_{2,3}$ because it is hampered at time 8. Therefore, $\tau_{1,2}$ can defer its work and the slack time in [6, 7] will be available. On the contrary, if one extends the additional analysis interval such that it is longer than or even several times of $p_n$, job $\tau_n$ cannot move the slack after bn + $p_n$ to approach the bn and may be blocked in this interval. For an analytical interval whose length is equal to $p_n$, it has the following advantages. After deriving the amount of slack time which will be available to the tasks nearby bn, those jobs whose period spanning astride the bn can be deferred to reclaim additional slack before bn. That is, the current job can utilize the additional slack by performing a lpWDA-based method. Notably, in an actual scheduling process, PRA does not exchange any work with slack. Instead, it only passes the length of additional slack time for current job to lpWDA and does not affect schedulability of subsequent jobs.

To present the proposed method, we define the following notations:

$$T^b = T - \{\tau_n\},$$ (9)

where $LCM(T^b)$ denotes the least common multiplier of task periods in $T^b$. As shown in Figure 2, the first synchronization point of $T^b$ within the interval $[bn, r_{n,k+1})$ is derived as

$$t(\tau^b, k) = \frac{r_{n,k+1}}{LCM(T^b)} \times \frac{r_n}{LCM(T^b)}.$$ (11)

When $t(\tau^b, k) = t \in (r_{n,k}, r_{n,k+1})$, slack time is likely to be blocked or shrunken at time $t$. In Figure 3, when all tasks except $\tau_n$ are synchronized at time $t$, a slack may not be moved backward from the right to the left side of $t$. In this case, slack can still be moved to the current time by postponing the execution of the work of $\tau_n$. When $n-1$ tasks are synchronized in interval $[bn, r_{n,k+1})$, we can derive $Syn(T^b, k) \geq 0$, and their the earliest synchronization point is derived by $t(T^{n-1}, k)$. The worst-case execution time in interval $[bn, t(T^{n-1}, k)]$ is

$$[t(\tau^{n-1}, k) - bn] \times (1 - U).$$ (12)

Therefore, the available slack for $\tau_{n,k+1}$ in this interval is at least

$$[(t(\tau^{n-1}, k) - bn) \times (1 - U)] - [t(\tau^{n-1}, k)] \times (1 - U) + (U - w_{ci}).$$ (13)

Similarly, when $\{\tau_i, \tau_j\} \subset T^{n-2}$ and $\tau_i \neq \tau_j$, there are $n-2$ tasks that synchronize at time $t(T^{n-2}, k)$. The available slack time for $\tau_{n,k+1}$ in interval $[bn, t(T^{n-2}, k)]$ is at least

$$[\frac{(t(\tau^{n-2}, k) - bn) \times (1 - (1 - U - U))}{(1 - \sum_{\tau_i \in T^{n-2}} w_{ci} / p_i)}].$$ (14)

Therefore, if $f$ of tasks are synchronized in interval $[bn, t(T^{n-1}, k)]$, the minimal available slack time for $\tau_n$ in interval $[bn, t(T^{n-f}, k)]$ is denoted as

$$e_{\text{Exchange}}^{bn} = \left[\left(\frac{t(T^{n-f}, k) - bn}{1 - \sum_{\tau_i \in T^{n-f}} w_{ci} / p_i}\right)\right],$$ (15)

where $e_{\text{Exchange}}^{bn}$ denotes the estimated slack in interval $[bn, t(T^{n}, k)]$. For example, $\tau_j$ does not synchronize with other tasks in $T^b$ (Figure 4). Therefore, one can compute the value of $Syn(T^c = T^b - \{\tau_j\})$ for each $\tau_j$, where $x = n-2$ and $b = n-1$. Suppose $Syn(T^b - \{\tau_j\}, k) > 0$, the earliest synchronization point of tasks in $T^b - \{\tau_j\}$ is derived using (11).

After deriving available slack time within interval $[bn, t(T^{n}, k)]$ where $b \leq n - 1$, we compute the length of the slack time which is available for the task in interval $[bn, t(T^{n}, k)]$. We assume $T^{bn}$ denotes a set of tasks in which task periods go astride the bn. Let $\tau_i \in T^{bn}$; the lengths of left and right parts of $p_i$ split by bn are defined as $e_{\text{left}}^{i, left}$ and $e_{\text{right}}^{i, right}$, respectively, and the longest $e_{\text{left}}^{i, left}$ and $e_{\text{right}}^{i, right}$ are defined as $e_{\text{left}}^{i, max}$ and $e_{\text{right}}^{i, max}$, respectively. Additionally, we define

$$\text{accu}^{bn} = \sum_{\tau_i \in T^{bn}} w_{ci}.$$ (16)
As the total amount of work in $T^{bn}$ as shown in Figure 3, the lengths of $e_{\text{left}}$, $e_{\text{right}}$, and $\text{accu}^{bn}$ limit the maximum length of slack that can be moved in interval $[r_{n,k-1}, bn]$.

Consequently, the restriction on the length of slack time is as follows:

$$e^{bn} = \min \{ e_{\text{left}}^{\text{max}}, e_{\text{right}}^{\text{max}}, \text{accu}^{bn} \}. \quad (17)$$

According to the work demand in a WCET schedule, the slack time in interval $[r_{n,k-1}, r_{n,k+1}]$ is computed as follows:

$$e^{\text{slack}} = 2 \times P_n - \sum_{t \in T} \left[ \frac{2 \times P_n}{P_t} \right] w_{ij}. \quad (18)$$

**Example 2.** Consider the WCET schedule shown in Figure 1(a), $\tau_{1,1}$ is scheduled at time $t = 0$, we set $bn = 6$ and $T^{bn} = \{ \tau_{2,2} \}$ because the period of $\tau_{2,2}$ goes astride $bn$. Procedure PRA computes the length of available slack time from interval $[6, 12]$ as follows.

When task set $T^2 = \{ \tau_1, \tau_2 \}$, Procedure PRA computes $\text{Syn}(T^2, 1) = 0$. Therefore, the bottleneck caused by $\tau_1$ and $\tau_2$ is $t(T^2 - \{ \tau_2, 1 \}) = 9$ and $t(T^2 - \{ \tau_1, 1 \}) = 8$, respectively.

Line 6 derives $\ell = 2$ and $r_{\text{syn}} = \tau_1$. Equations (14)–(17) derive $e_{\text{exchange}}^{\text{max}} = (9 - 6) \times (1 - (11/12 - 1/3)) = 1$. In line 10, the value of $e_{\text{exchange}}^{\text{max}}$, $e_{\text{bn}}$, $e_{\text{slack}}^{\text{max}}$, and $\text{accu}^{bn}$ is $1$, $1$, and $2$, respectively. The value of $e_{\text{min}}^{\text{exchange}}$ is $1$ by line 12. Therefore, Procedure PRA returns $e_{\text{min}}^{\text{exchange}} = 1$ to the lpWDA algorithm and passes additional slack $e_{\text{exchange}}^{\text{max}}$ to $\text{CalcLowerPriorityWork()}$ in Algorithm 3. Notably, the tasks using PRA still execute under RM priority policy except one of the jobs whose periods span astride the bn. At time $t = 0$, when jobs $\tau_{1,1}$, $\tau_{2,1}$, and $\tau_{3,1}$ enter $\text{readyQ}$ at time $t = 0$, $\tau_{1,1}$ has the highest priority and utilizes additional slack $e_{\text{exchange}}^{\text{max}}$ estimated by PRA. Therefore, job $\tau_{1,1}$ obtains one unit of time of slack and changes its voltage level from 1 to 0.5. On the contrary, if primitive lpWDA performs $\tau_{1,1}$ at time $t = 0$, $\tau_{1,1}$ cannot obtain any slack. When lpWDA executes iteratively, the value of $e_{\text{exchange}}^{\text{max}}$ does not change until $\tau_{1,1}$ is completed. Figure 2(c) presents the scheduling result obtained using Procedure PRA. After completing $\tau_{1,1}$, $e_{\text{exchange}}^{\text{max}}$ unit of slack has been run out, primitive lpWDA continuously performs voltage scaling on the subsequent jobs of $\tau_{1,1}$. In the case of $\tau_{2,1}$, it begins after $\tau_{1,1}$ ($t = 1$) and obtains one unit of slack time from primitive lpWDA. Therefore, its WCET...
under voltage $v = 0.5$ is changed to $w_{c_{1,1}} = 2$, and actual execution time is $ac_{1,1} = 1$. At time $t = 4$, job $r_{2,2}$ is released and moved to $readyQ$. Its priority is changed to and lower than the remaining execution time of $r_{3,1}$ by executing line 14 in Procedure PRA. Therefore, job $r_{2,2}$ begins its work after completing the remaining work of $r_{3,1}$. Notably, PRA only changes jobs’ priority in $T_{bn}$ and does not affect the feasibility of lpWDA schedule. The correctness proof is discussed later in the next section. Table 2 shows the values of scheduling parameters. The rightmost job in $readyQ$ is being executed at that time. In Algorithm 1, job $r_{5}$ is a global variable. Whenever job $r_{5}$ executes and $\Delta_{\text{Exchange}} > 0$, Procedure PRA lowers its priority to guarantee the timing constraint of jobs $r_{n}$ and $r_{5}$.

5. Correctness Proof

In this section, we prove the correctness of the schedules produced by lpWDA and PRA based on worst-case response time (WCRT) analysis and assume that the given task sets are feasible under preemptive RM scheduling. For the fixed-priority preemptive scheduling, a critical instant for a task $r_{\alpha}$ is given by a moment in which the release time of $r_{\alpha}$ coincides with all higher-priority tasks. Let $\omega_{\alpha}$ denote the WCRT of $r_{\alpha}$ without loss of generality, the higher-priority tasks have simultaneous release time with the job of $r_{\alpha}$.

Lemma 3. When a task set $T$ contains only one task $r_{\alpha}$, the available slack produced by lpWDA for $r_{\alpha}$ is

$$\text{slack}_{\alpha}(t) = d_{\alpha} - t - w_{\alpha,\text{rem}}(t).$$

Proof. By (8), the amount of work required to be processed in interval $[t, d_{\alpha}]$ is

$$\text{load}_{\alpha}(t) = w_{\alpha,\text{rem}}(t).$$

According to (3), the available slack is derived as

$$\text{slack}_{\alpha}(t) = d_{\alpha} - t - \text{load}_{\alpha}(t)
= d_{\alpha} - t - w_{\alpha,\text{rem}}(t),$$

which completes the proof.

In lpWDA, the slack derived from lower-priority task is given to the highest priority job in the $readyQ$. In the WCET case, after applying the slack to the highest priority job, the execution cycles of lower-priority jobs are postponed and their WCRTs will be increased in the length equal to that slack.

Lemma 4. When a task set $T$ contains $n$ tasks where $n \geq 2$, the amount of work required to be processed in $[t, d_{\alpha}]$ ($\alpha \leq n$) for the highest priority job $r_{\alpha}$ is

$$\text{load}_{\alpha}(t) = w_{\alpha,\text{rem}}(t) + H_{\alpha}(t) - (u_{d_{\alpha}} - d_{\alpha}).$$

Proof. Assuming that $T$ contains $n$ tasks where $n \geq 2$, we prove that this lemma form the lowest priority task (i.e., $r_{n}$) to the highest priority task using mathematical induction. The case of $r_{n}$ is proved separately because the third term of $\text{load}_{\alpha}(t)$ is different from those with $\alpha < n$ in (22). From (8), when $r_{n}$ is the lowest priority task (i.e., $\alpha = n$), the workload of the tasks whose priorities lower than $r_{n}$ are zero (i.e., $L_{\alpha}(t) = 0$). Therefore, the amount of work required to be processed at time $t$ is

$$\text{load}_{n}(t) = w_{n,\text{rem}}(t) + H_{n}(t).$$

Because $r_{n} = u_{d_{n}}$ before completing $r_{n}$, we add $(u_{d_{n}} - r_{n})$ to (23) and derive

$$\text{load}_{n}(t) = w_{n,\text{rem}}(t) + H_{n}(t) - (u_{d_{n}} - d_{n}),$$

and this completes the proof of $r_{n}$.

Basic Step. First, we discuss the value of $\text{load}_{n-1}(t)$ which can be estimated under two cases: $u_{d_{n}}(t) > d_{n-1}$ (Case 1) and $u_{d_{n}}(t) \leq d_{n-1}$ (Case 2).

Case 1 ($u_{d_{n}}(t) > d_{n-1}$). If $u_{d_{n}}(t)$ is later than $d_{n-1}$, $\text{load}_{n}(t)$ is greater than $w_{n-1,\text{rem}}(t) + H_{n-1}(t)$, that is, the work in $w_{n-1,\text{rem}}(t) + H_{n-1}(t)$ is the subset of the work in $\text{load}_{n}(t)$. Let $\Delta = \text{load}_{n}(t) - w_{n-1,\text{rem}}(t) + H_{n-1}(t)$, if $\Delta \leq u_{d_{n}}(t) - d_{n-1}$, $\Delta$ amount of work can be processed in $[d_{n-1}, u_{d_{n}}(t)]$, and only $w_{n-1,\text{rem}}(t) + H_{n-1}(t)$ is required to be processed before $d_{n-1}$. Otherwise, $\Delta - (u_{d_{n}}(t) - d_{n-1})$ amount of work should be processed before $d_{n-1}$. That is,

$$L_{n-1}(t) = (u_{d_{n}}(t) - w_{n-1,\text{rem}}(t) - H_{n-1}(t)) - (u_{d_{n}}(t) - d_{n-1}),$$

where the notation $x^+$ stands for $\max(x, 0)$. Substituting (23) in the above equation, we get

$$L_{n-1}(t) = ((w_{n-1,\text{rem}}(t) + H_{n}(t)) - w_{n-1,\text{rem}}(t) - H_{n-1}(t)) - (u_{d_{n}}(t) - d_{n-1}).$$

Figure 4: The slack reclamation in lpWDA algorithm.
Substituting this result in $load_{n-1}(t)$ defined in (8), we obtain
\[
load_{n-1}(t) = w_{n-1}^{rem}(t) - H_{n-1}(t) + (ud_n(t) - d_{n-1})^+.\tag{27}
\]

Case 2 ($ud_n(t) \leq d_{n-1}$). When $ud_n(t)$ is earlier than $d_{n-1}$, $load_n(t)$ must be processed before $d_{n-1}$. However, the value of $load_n(t)$ does not change, and the value of $load_{n-1}(t)$ is obtained as follows:
\[
L_{n-1}(t) = w_{n-1}^{rem}(t) + H_{n}(t). \tag{28}
\]

**Inductive Step.** When task $\tau_3$ is considered, the value of $load_2(t)$ can be obtained as follows:
\[
load_2(t) = (w_2^{rem}(t) + H_n(t) - (ud_n(t) - d_2))^+.\tag{29}
\]

When we consider the task with the highest priority (i.e., $\tau_1$), the amount of work of its lower-priority task is
\[
L_1(t) = (load_2(t) - w_1^{rem}(t) - H_1(t) - (ud_2(t) - d_1))^+. \tag{30}
\]

Substituting $load_2(t)$ in (30), we get
\[
L_1(t) = [(w_2^{rem} + H_n(t) - (ud_n(t) - d_2)] - w_1^{rem}(t) - H_1(t) - (ud_2(t) - d_1))^+. \tag{31}
\]

When all tasks release at time $t$, we have $ud_2(t) = d_2$. Therefore, we get
\[
L_1(t) = (w_1^{rem} + H_1(t) - w_1^{rem}(t) - H_1(t) - (ud_n(t) - d_1))^+. \tag{32}
\]

By (8), substituting (32) in $load_1(t)$, we get
\[
load_1(t) = w_1^{rem}(t) + H_1(t) + L_1(t)
+ w_1^{rem}(t) + H_1(t)
+ [w_1^{rem} + H_1(t) - w_1^{rem}(t) - H_1(t)
- (ud_n(t) - d_1)^+]
= w_1^{rem} + H_n(t) - (ud_n(t) - d_1). \tag{33}
\]

The proof of Case 2 is similar to that of Case 1 and this completes the proof. \hfill \Box

**Lemma 5.** The length of slack, that is, provided by $lp\text{WDA}$ for the highest priority task in $readyQ$ is at most
\[
slack_\alpha(t) = ud_\alpha(t) - t - w_\alpha^{rem}(t) - H_n(t). \tag{34}
\]

**Proof.** Assuming that $\tau_\alpha$ has the highest priority in $readyQ$, this proof can be derived directly from (3), (19), and (22). \hfill \Box

The following theorem proves the schedulability of $lp\text{WDA}$ by using worst-case response time analysis. We consider each active job $\tau_i$ in $readyQ$ has a simultaneous release with all higher priority tasks.

**Theorem 6.** Given a set $T$ of tasks is feasible in RM schedule, the maximum response time of task $\tau_i$ under $lp\text{WDA}$ is less than or equal to its deadline.

**Proof.** Assuming job $\tau_\alpha$ has the highest priority in $readyQ$. By (8), we get $WR_\alpha = load_\alpha$ at time $t = 0$. Due to (3) proposed by $lp\text{WDA}$, the deadline of $\tau_\alpha$ can be guaranteed by
\[
slack_\alpha(t) = d_\alpha - load_\alpha, \tag{35}
\]

when $t = 0$. Therefore, we get
\[
WR_\alpha + slack_\alpha \leq d_\alpha. \tag{36}
\]
When $\tau_\alpha$ runs out slack, all of subsequent jobs of $\tau_\alpha$ have to postpone their response times separately at most slack unit of times comparing to those in their WCET RM schedule. Assuming $\tau_\alpha$ has lower priority than that of $\tau_\alpha$, we prove that the length of new WCRT of $\tau_\alpha$ including slack is less than $d_\gamma$. In a feasible RM schedule, the WCRT of $\tau_\gamma$ is

$$WR_\gamma \leq wc_\gamma + \sum_{i < \gamma} \left[ \frac{p_i}{p_\gamma} \right] \cdot wc_i.$$  

(37)

The new WCRT of $\tau_\gamma$ considering the length of slack is denoted as

$$WR_{\gamma}^{new} = wc_\gamma + \sum_{i < \gamma} \left[ \frac{p_i}{p_\gamma} \right] \cdot wc_i + slack_\alpha$$

(38)

Based on (22) in Lemma 4 and set $t = 0$, we derive

$$WR_{\gamma}^{new} = wc_\gamma + \sum_{i < \gamma} \left[ \frac{p_i}{p_\gamma} \right] \cdot wc_i + d_\alpha$$

$$- \left[ wc_\gamma + H_\gamma(0) - ud_\gamma + d_\alpha \right]$$

$$= \sum_{i < \gamma} \left[ \frac{p_i}{p_\gamma} \right] \cdot wc_i - H_\gamma(0) + ud_\gamma.$$  

(39)

From the definitions of function $H_\gamma(t)$ in (4) and (5), we derive

$$WR_{\gamma}^{new} = \sum_{i < \gamma} \left[ \frac{p_i}{p_\gamma} \right] \cdot wc_i - \left[ H_\gamma^{past} + H_\gamma^{future} \right] + d_\gamma$$

$$= \sum_{i < \gamma} \left[ \frac{p_i}{p_\gamma} \right] \cdot wc_i$$

$$- \left( \sum_{i < \gamma} wc_i + \sum_{i < \gamma} \left( \frac{d_\gamma - \varepsilon}{p_i} \right) + 1 \cdot wc_i \right) + d_\gamma$$

$$= \sum_{i < \gamma} \left[ \frac{p_i}{p_\gamma} \right] \cdot wc_i - \left( \sum_{i < \gamma} \left( \frac{d_\gamma - \varepsilon}{p_i} \right) + 1 \cdot wc_i \right) + d_\gamma.$$  

(40)

and $\varepsilon$ is the infinitesimal. Because

$$WR_{\gamma}^{new} = \sum_{i < \gamma} \left[ \frac{p_i}{p_\gamma} \right] \cdot wc_i \leq \sum_{i < \gamma} \left( \frac{d_\gamma - \varepsilon}{p_i} \right) + 1 \cdot wc_i.$$  

(41)

we derive

$$WR_{\gamma}^{new} \leq d_\gamma.$$  

(42)

From (41) in Theorem 6, the difference between $WR_{\gamma}^{new}$ and $d_\gamma$ is derived from the following corollary.

**Corollary 7.** For some tasks $\tau_\gamma \in T^{bn}$, $i < \gamma$ and $p_\gamma$ is not the multiple of these $p_\gamma$s; the difference between $WR_{\gamma}^{new}$ and $d_\gamma$ is formulated as

$$d_\gamma - WR_{\gamma}^{new} = \sum_{k < \gamma} \left( \frac{d_\gamma - \varepsilon}{p_k} \right) + 1 \cdot wc_k$$

$$- \sum_{k < \gamma} \left[ \frac{p_k}{p_\gamma} \right] \cdot wc_k.$$  

(43)

Notably, $WR_{\gamma}^{new}$ presents the length of WCRT proposed by lpWDA, and therefore the slack between $WR_{\gamma}^{new}$ and $d_\gamma$ could be utilized by PRA.

Consider the example shown in Figure 4(a). The value of load, is set to the sum of $wc_\gamma$ and $H_\gamma(0)$ which is shown in the gray box of Figure 4(b). There are 6 time units that are required to be processed before $ud_\gamma = 8$. In order to guarantee the feasible schedule of higher-priority jobs whose periods span $astrideud_\gamma$ (i.e., $\tau_1$ and $\tau_2$), lpWDA estimates how much time should be reserved for the higher-priority jobs. In this case, $wc_{1,2} + wc_{2,2} = 2$ is derived from (43). We investigate the difference between $d_\gamma$ and $WR_{\gamma}^{new}$ to keep the deadlines of PRA jobs.

**Lemma 8.** Algorithm lpWDA selects an effective feasible speed for the active job in the analysis scope generated by upcoming deadlines.

**Proof.** Lemma 8 is derived directly from Theorem 6 and Corollary 7.

**Lemma 9.** Let $\gamma > \delta$, $d_\gamma = bn$ and $\tau_\delta \in T^{bn}$. When job $\tau_\gamma$ is feasible under PRA, $\tau_\delta$ also keeps its deadline.

**Proof.** According to Theorem 6, we have

$$WR_{\gamma}^{new} \leq d_\gamma.$$  

(44)

The WCRT of job $\tau_\gamma$ under PRA is changed to

$$WR_{\gamma}^{PRA} = WR_{\gamma}^{new} - wc_\gamma + \epsilon_{\text{exchange}}.$$  

(45)

By (45), the deadline of job $\tau_\gamma$ is still guaranteed by PRA. Additionally, the priority of job $\tau_\delta$ is changed temporarily lower than $\tau_\gamma$, and $\tau_\delta$ executes immediately after $wc_\gamma$. Therefore, the WCRT of job $\tau_\delta$ is denoted as

$$WR_{\delta}^{PRA} = WR_{\delta}^{PRA} + wc_\gamma$$

$$= WR_{\delta}^{PRA} + \epsilon_{\text{exchange}}.$$  

(46)

$$\leq d_\gamma + \epsilon_{\text{exchange}}.$$
Due to the definition of \( \epsilon^\text{right}_\delta \), we get
\[
d_\gamma + \epsilon^\text{Exchange} \leq d_\gamma + \epsilon^\text{right}_\delta \quad (47)
\]
and complete the proof. \( \square \)

Lemma 9 proves that additional slack produced by PRA is shorter than the right part of \( p_\delta \) split by bn. Therefore, the deadline of job \( r_\delta \) is kept after changing the priority of \( r_\delta \) to the lowest priority in the \( \text{ready}Q \). After completing \( r_\delta \), the schedule is performed continuously under \( \text{lpWDA} \). The schedulability proof in interval \([bn, r_{n,k+1}]\) is similar to Theorem 6 except additional work \( \epsilon^\text{Exchange} \) in the WCET schedule.

**Lemma 10.** An \( \text{lpWDA} \) schedule remains feasible when \( \epsilon^\text{Exchange} \) unit of work is postponed to interval \([bn, r_{n,k+1}]\) using PRA.

**Proof.** In the interval \([bn, r_{n,k+1}]\), the critical instant appears in a situation that job \( r_\delta \) released before \( bn \) may remain incomplete at \( bn \), while other tasks are released at \( bn \). In accordance with (47), the maximum uncompleted work for job \( r_\delta \) at \( bn \) is \( \epsilon^\text{Exchange} \). Let \( WR'_\gamma \) denote the WCRT of job \( r_{\gamma,k} \) in the interval \([bn, r_{n,k+1}]\) where \( \gamma > \delta > \alpha \). According to (40) in Theorem 6, we have
\[
WR'_\gamma = \sum_{i < \gamma} \left[ \frac{\frac{d_i}{p_i}}{} \right] \cdot wc_i - \left( \sum_{i < \gamma} \left( \frac{d_i - e_i}{p_i} \right) + 1 \right) \cdot wc_i + d_\gamma - wc_\delta + \epsilon^\text{Exchange}.
\]

According to line 13 in Algorithm PRA, the length of available slack is \( \epsilon^\text{Exchange} \leq wc_\delta \). From (48), the following statement still holds:
\[
\sum_{i < \gamma} \left[ \frac{\frac{d_i}{p_i}}{} \right] \cdot wc_i 
\leq \left( \sum_{i < \gamma} \left( \frac{d_i - e_i}{p_i} \right) + 1 \right) \cdot wc_i - wc_\delta + \epsilon^\text{Exchange}.
\]

Therefore, we derive
\[
WR'_\gamma \leq d_{\gamma,k} = r_{\gamma,k+1} \quad (50)
\]
which completes the proof. \( \square \)

In fact, Procedure PRA focuses on providing the potential and available slack time to the \( \text{lpWDA} \)-based algorithms.

**Theorem 11.** Procedure PRA provides additional slack that guarantees all task deadlines in the \( \text{lpWDA} \).

**Proof.** In the interval \([r_{n,k-1}, bn]\), suppose job \( r_\alpha \) is being executed in \([r_{n,k-1}, bn]\). By executing line 9 in Algorithm 1 while passing the additional slack \( \epsilon^\text{Exchange} \) to the function \( \text{CalcSlackTime()} \) in the \( \text{lpWDA} \) algorithm, Algorithm 2 computes the length of the slack, which \( r_\alpha \) can use by calling the function \( \text{CalcLowerPriorityWork()} \) recursively. When job \( r_\alpha \) uses complete \( \epsilon^\text{Exchange} \) and all of its subsequent jobs execute in their WCET, job \( r_{n,k-1} \) is likely to miss its deadline. However, line 14 in Procedure PRA solves this problem by changing the priority of job \( r_\delta \) to the lowest priority job in \( \text{ready}Q \). Because \( \epsilon^\text{right}_\delta \geq \epsilon^\text{Exchange} \), according to Lemma 9, the deadline of jobs \( r_\delta \) and \( r_n \) is guaranteed in the interval \([r_{n,k-1}, bn]\).

In the interval \([bn, r_{n,k+1}]\), Procedure PRA computes the length of additional slack in \([bn, r_{n,k+1}]\). Given that \( r_\delta \) has been assigned to the lowest priority in \( \text{ready}Q \) and all deadlines before the completion time of \( r_\delta \) are met. Based on the completion time of \( r_\delta \), we divide it into two cases.

**Case 1.** Job \( r_\delta \) completes at time \( t^\delta \) and \( t^\delta \leq bn \). In this case, the slack computed by Procedure PRA is not to be used by the jobs started before \( t^\delta \). Because \( \epsilon^\text{Exchange} \) produced by Procedure PRA does not shift to the time before \( bn \), it has no influence on the job execution cycles after \( bn \). Therefore, the initial \( \text{lpWDA} \) algorithm guarantees the deadline of jobs that started after the \( bn \).

**Case 2.** Job \( r_\delta \) completes at time \( t^\delta \) and \( t^\delta > bn \). When a scheduling point is at \( t^\gamma \), the analysis scope defined in (2) is extended up to \( r_{n,k+1} \) by calling the Procedure \( \text{CalcLowerPriorityWork()} \) in Algorithm 3 recursively. In a feasible schedule, when slack \( \epsilon^\text{Exchange} \) has been exchanged on the left side of the \( bn \) and jobs before \( t^\gamma \) have been performed according to their WCET, the length of delayed work is not longer than \( \epsilon^\text{Exchange} \). Therefore, the length of the additional work moved in interval \([bn, r_{n,k+1}]\) is at most \( \epsilon^\text{Exchange} \). By Lemma 10, the additional work does not affect the feasibility of \( \text{lpWDA} \) schedule and completes the proof. \( \square \)

## 6. Performance Evaluation

In this section, we evaluate the time and energy efficiency of PRA scheme for randomly generated task sets and compare them with those of the ccRM, \( \text{lpWDA} \) and \( \text{lpLDAT} \) schemes. Both the ccRM and \( \text{lpWDA} \) are modified to account for transition time overhead. In simulations, \( \text{lpWDA} \) and \( \text{lpLDAT} \) are called the host algorithms of PRA and cooperate with PRA to compare its performance with those of initial ccRM, \( \text{lpWDA} \), and \( \text{lpLDAT} \) methods.

### 6.1. Complexity and Execution Time of Algorithms

**Theorem 12.** The PRA algorithm has a computational complexity of \( O(n) \) per scheduling point, where \( n \) denotes the number of tasks in the systems.

**Proof.** Lines 4 and 5 are completed in constant time for each iterative step according to (19) and (23). In line 8, the value of \( \epsilon^\text{Exchange} \) is derived from (24)–(27), where the value of \( \epsilon^\text{Exchange} \) (\( i \)) in (27) needs \( O(n) \) time to compute the length of slack time. In line 10, the computation of \( \epsilon^\text{bn} \) and \( \epsilon^\text{slack} \) needs \( O(n) \) time. Therefore, the overall time complexity is \( O(n) \). \( \square \)
In the simulation results, for any given pair of total tasks, $U$ and $bc/uc$ ratio in $T$, 10000 task sets are generated randomly. The experiment result is the average value over the 10000-task sets. In a task set, every task period $p_i$ (as well as deadline $d_i$) is uniformly distributed in the range $[1, 100]$ ms. The length of each schedule is at least ten times of its tasks hyperperiod except the fourth experiment in Section 6.2. Execution time $w_c$ of each task is assigned a real number in the range of $[1, \min(p_i-1, 90)]$ ms. In a task set, after assigning values to the execution time of all tasks, we give a utilization $U$ to the task set and rescale the $p_i$ of each task, such that the summation of task weights (i.e., $w_c/p_i$) equals the given $U$.

Due to the limited execution speed of embedded processors, an LCM($T_b$) function is implemented by offline non-recursive programs. This function is composed of $lcm(a, b)$ and $gcd(a, b)$ which compute the least common multiplier (LCM) and greatest common divisor (GCD), respectively. In these programs, each integer is represented by a 32-bit word. Each experiment (schedule) has a maximum of 20 tasks, and each task has an integer variable for storing the period lengths of accumulated values of an LCM. Additionally, two integers are needed to record GCD and LCM of all task periods. Therefore, each schedule requires at most 100 bytes for storing data, which include local variables.

In Figure 5, we examine the execution time required by each online algorithm, including the following algorithms:

- ccRM: the ccRM algorithm from [7] is modified to account for transition overhead;
- lpWDA: the lpWDA algorithm from [6] is modified to account for transition overhead;
- lpLDAT: the algorithm from [12];
- lpWDA-PRA: the lpWDA is the host algorithm of PRA and cooperates with Procedure PRA;
- lpLDAT-PRA: the lpLDAT is the host algorithm of PRA and cooperates with Procedure PRA;
- lpWDA-DP-PRA: the lpWDA-DP is a host algorithm and cooperates with Procedure PRA;
- lpWDA-AC-PRA: the lpWDA-AC is a host algorithm and cooperates with Procedure PRA.

Figure 5 presents maximum execution time of each algorithm on a processor versus the number of tasks in the system. Notably, the simulation results produced by lpWDA-DP-PRA and lpWDA-AC-PRA are very close to those of lpWDA-PRA and lpLDAT-PRA, for a clear presentation of this figure, they are abridged. All algorithms executing on the simulated processor are based on ARM8 core with the highest speed (100 MHz) and voltage level. The measurement results were generated by inserting a system timer function and executing each algorithm individually. Obviously, ccRM has a significant advantage in terms of execution time when compared with that of the other online algorithms. Because the algorithms are invoked upon each release and completion, one must increase execution time of each task by two times the maximum execution time of the algorithm to account for scheduling overhead. To present the maximum execution times of these algorithms, the simulation process is as follows. First, a set of experiments performed Procedure PRA with its host algorithms (lpWDA and lpLDAT) and other initial algorithms. The functions of the system timer are used to record the duration of each algorithm, choose their longest execution times in each schedule, and accumulate execution times separately with respect to different methods. At experiment end, these accumulated execution times are divided by the number of schedules generated. PRA is an efficient online algorithm that increases in average additional execution time by less than 12% of those of their host algorithms.

6.2. Simulation Results. The following four parameters are varied in simulations: (1) number of tasks $total\ tasks$ in $T$ is varied at 2–18 in two task increments; (2) utilization $U$ for task set is varied at 0.1–0.9; (3) the $bc/uc$ ratio of BCET to WCET is varied at 0.1–0.9; and (4) the analytical interval in bound being the multiples of $p_i$ is denoted as $M_{p_i}^{\text{analysis}}$. Before performing these experiments, 10000 task sets have been generated randomly including the number of tasks in each set, task period lengths, and their worst-case execution requirements in accordance with a uniformed distribution function. Early completion time of each job in simulation (1), (2), and (4) was randomly drawn from a Gaussian distribution in the range of $[\text{BCET}, \text{WCET}]$, where BCET/WCET = 0.1. In simulation (3), each experiment was performed by varying BCET at 10%–90% of WCET.

For all experiments, we assume that 10 frequency levels are available in the range of 10–100 MHz, with corresponding voltage levels of 1–3.3 Volts. The energy consumption caused by memory access and cache misses are ignored, and all experimental results are normalized against the same processor running at maximum speed without a DVS technique (non-DVS for short). Table 3 presents the power specification of the ARM8 processor [15].
Table 3: Power specification.

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Voltage</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sleep (idle)</td>
<td>0.8 V</td>
<td>&lt;500 µW</td>
</tr>
<tr>
<td>10 MHz</td>
<td>1.1 V</td>
<td>18 mW</td>
</tr>
<tr>
<td>20 MHz</td>
<td>1.2 V</td>
<td>32.3 mW</td>
</tr>
<tr>
<td>30 MHz</td>
<td>1.5 V</td>
<td>40.4 mW</td>
</tr>
<tr>
<td>40 MHz</td>
<td>1.8 V</td>
<td>56.5 mW</td>
</tr>
<tr>
<td>50 MHz</td>
<td>2.0 V</td>
<td>68.7 mW</td>
</tr>
<tr>
<td>60 MHz</td>
<td>2.3 V</td>
<td>86.9 mW</td>
</tr>
<tr>
<td>70 MHz</td>
<td>2.5 V</td>
<td>110 mW</td>
</tr>
<tr>
<td>80 MHz</td>
<td>2.7 V</td>
<td>137.5 mW</td>
</tr>
<tr>
<td>90 MHz</td>
<td>2.95 V</td>
<td>178 mW</td>
</tr>
<tr>
<td>100 MHz</td>
<td>3.3 V</td>
<td>220 mW</td>
</tr>
</tbody>
</table>

The overhead considered in simulations is as follows.

(1) **Algorithm Execution Time and Energy.** The execution time overhead is referred to the simulation results in Figure 5. Energy overhead is obtained under the assumption of maximum speed $S_{\text{max}}$.

(2) **Voltage Transition Time and Energy.** The assumption of voltage scaling overhead is the same with that in [17]. For the voltage scaling from $V_{dd1}$ to $V_{dd2}$, the transition time is

$$\Delta t \approx 2 \cdot C \cdot \frac{V_{dd2} - V_{dd1}}{I_{\text{max}}},$$

where $C$ and $I_{\text{max}}$ the charge to the capacitor and maximum output current of the converter, respectively. Transition time is at most 70 µs between maximum transition [15]. The energy consumed during each transition is

$$\Delta E = n (1 - \lambda) \cdot C \cdot |V_{dd2} - V_{dd1}|,$$

where $\lambda$ denotes the efficiency of a DC-DC converter.

(3) **Context-Switch Time and Energy.** Context-switch time is assumed 50 µs at the highest speed $S_{\text{max}}$ as in [18].

Figures 6, 7, 8, and 9 list the energy consumption of each method. Energy consumption includes both execution duration of PRA and its host algorithms (i.e., the lpWDA and lpLDAT), and the context-switch time required to switch to and from other real-time tasks. Since the range of task periods has been shortened to a scale of [1, 100] ms, the difference between task periods and context-switch times or transition times is smaller than those assumed in [6, 12]. Additionally, these energy overheads arose from PRA, and its host algorithms are also taken into account, such that the experimental results are close to actual situations. In these simulations, the host algorithms cooperating with PRA are better than its initial algorithms, respectively.

Figures 6, 7, 8, and 9 also present the results for a clairvoyant algorithm, named bound, which knows the actual execution cycle of each task beforehand and adopts the optimal speed accordingly. The length of analytical interval utilized by algorithm bound is set at least four times the length of $p_n$ except the fourth experiment in Figure 9. This setting ensures that bound's analytical length is longer than those utilized by other methods and equal to the length of the schedules. Every scheduling point in the entire schedule is examined when looking for the best start and finish times. The context-switch and transition overhead assumed in the bound are the same as those with other methods while its execution time is assumed zero. In fact, bound is not a practical algorithm because it is extremely time consuming when finding the suitable start, preemption, and completion times, and no algorithm can predict the exact amount of job execution cycles beforehand. Thus, bound functions a
Lower layer tries to schedule tasks with higher speed and leaves additional slack that PRA is unaware of. Therefore, this slack time becomes a fragment of slack time that is harmful to power saving. The following series of experiments almost reveal the phenomenon that the performance of lpWDA-AC-PRA is worse than those of lpWDA-PRA and lpWDA-DP-PRA. The value of $U$ of each task set is assigned randomly at 10%–70% by uniform probability distribution function. The value of $U$ and the $bc/uc$ ratio for each task set are 0.8 and 0.5, respectively. With a large $U$, PRA outperforms its host algorithms in a small total task, consuming up to 25% and 24% less energy than lpWDA and lpLDAT, respectively. Although execution overhead of PRA is included, for a large total task, PRA still outperforms its host algorithms. The reason is likely that when the number of tasks increases, the number of task synchronization points appeared in the analytical interval likely decreases and benefits slack time computation.

Experimental results in Figure 7 indicate that lpWDA-PRA and lpLDAT-PRA saved up to 25% and 4% more energy than their host algorithms. Among lpWDA-based methods, although lpWDA-DP-PRA has the best performance, it slightly outperforms lpWDA-PRA. In other words, when PRA is applied in lpWDA, the delayed-preemption (DP) technique contributes little energy saving to the host methods. In the experiment, the total task and the $bc/uc$ ratio of each task set are 10 and 0.5, respectively. Increasing the value of $U$ in Figure 7 increases the energy consumption of PRA and its host algorithms. With a small $U$, the gain from PRA is modest, with 1% and 4% saving compared to that of the initial lpLDAT and lpWDA algorithms, respectively. Additionally, with these methods, $U$ is an important factor when computing the slack for deciding processor speeds. With a moderate $U$ value, lpWDA-PRA and lpLDAT-PRA consume at most 16% and 4% less energy than that of the initial lpWDA and lpLDAT algorithm, respectively. Therefore, PRA utilizes not only the advantages of its host algorithms but also slack belonging to the value of $1−U$ as possible and shifts the slack to the current job.

In Figure 8, the set of experiments varies the $bc/uc$ ratio at 0.1–0.9, and the value of $U$ and total task is 0.8 and 10, respectively. The energy consumed by PRA is positively correlated with the $bc/uc$ ratio, while their hosts are not sensitive to the $bc/uc$ ratio. In this experiment, lpWDA-DP-PRA still outperforms other lpWDA-based methods but not obviously. With a low $bc/uc$ ratio, PRA is the best, consuming up to 26% and 16% less energy than the lpWDA and lpLDAT, respectively. With a $bc/uc$ ratio of 0.9, PRA collaborated with lpLDAT which consumes slightly more energy than the initial lpLDAT algorithm. The reason is likely that the additional saving gained by the PRA algorithm is compromised by its execution overhead.

In Figure 9, the analytical interval in bound is exactly $M_{\text{analysis}}$ times the length of $p_n$. The values of $M_{\text{analysis}}$ are controlled by the simulation from 2 to 18 and the total task of each task set is assigned randomly at 2 to 20 tasks. For simplicity, the length of each schedule is also controlled in $M_{\text{analysis}} \times p_n$. When the value of $M_{\text{analysis}}$ increases,
the energy saving of bound is not obvious, and the energy consumption required by the schemes with PRA are not sensitive to $M_{\text{analysis}}$. Notably, when $M_{\text{analysis}} = 2$, PRA and bound have equal length of analytical interval; bound gains at most 33% energy savings less than the proposed schemes. Therefore, extending the additional analysis interval, such that it is several times longer than $p_n$, does not increase an already substantial energy saving but rather increases computing overhead during slack time analysis. There exist some limits and difficulties whenever extending analysis interval that is longer than two times of $p_n$. Firstly, the length of the potential slack time is affected by the length of analytical interval, the values of $\text{Syn}(T^f, k)$ and $t(T^f, k)$, and by the locations of actual execution cycles of job. In Figure 3, the locations of the job workload affect the available length of slack time. Additionally, utilization of a task set influences the length of available slack time. For example, for a task set with high utilization, actual workload of many tasks may appear after $t(T^f, k)$ where $b = n$ or $b = n - 1$ and cannot be exchanged with the slack. Therefore, the longer the analytical interval is, the harder the prediction of the length of additional slack becomes. For a longer interval than $p_n$, we may need considerably more time and memory space than those required by PRA. Additionally, the length of a hyperperiod can vary from one to many times of $p_n$ as long as one task period is changed. When an analytical interval extends up to a hyperperiod, execution time required by algorithms may change severely and adversely affect the predictability of a real-time schedule.

7. Conclusions

In this paper, we proposed the parareclamation algorithm (PRA) based on the concept of work-demand computation. This method can serve many existing RM scheduling methods as a guest algorithm. PRA cooperating with the host algorithms such as lpWDA, lpWDA-DP, lpWDA-AC, lpLDA, and lpLDAT can further decrease energy consumption without increasing time complexities. It is fully compatible not only with transition-aware (i.e., lpLDAT,) methods but also preemption-aware (i.e., lpWDA-AC and lpWDA-DP) methods. Experimental results indicate that PRA can utilize the additional slack produced by lpWDA-AC and lpWDA-DP and reduce average energy consumption by 14% when compared with that of the initial schemes.

Acknowledgment

The authors would like to thank the National Science Council of the Republic of China, Taiwan, for financially supporting this paper under NSC 102-2221-E-025-003.

References


on low Power Electronics and Design (ISLPED ’00), pp. 9–14, Redondo Beach, Calif, USA, July 2000.

Submit your manuscripts at http://www.hindawi.com