Received November 7, 2020, accepted November 20, 2020, date of publication December 8, 2020,
date of current version December 18, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3043260

The Earliest Smooth Release Time for a New Task Based on EDF Algorithm

GUANGMING QIAN
College of Information Science and Engineering, Hunan Normal University, Changsha 410081, China
e-mail: qqyy@hunnu.edu.cn

ABSTRACT Although EDF (Earliest Deadline First) algorithm has received extensive study during the past more than 40 years, only a few researchers have published their efforts on the bandwidth transfer between tasks. If current running tasks are compressed to free part of their occupied bandwidth to accommodate new requirements, such as a new task's insertion, then a basic requirement of this operation is smoothness, that is, no deadline should be missed. Suppose current tasks are immediately compressed at the request time of the new task, in order to guarantee the smoothness, the new task may have to be released later than the request time. An interesting and challenging problem is to find the earliest smooth release time. In this paper, an algorithm to evaluate the earliest release time for single task's insertion is presented and formally proved. To finish the algorithm, only the deadlines during the transition should be checked, and each of them needs to be checked at most once. A novel experimental approach is adopted and more than 4549320 different tests are implemented to verify the theorems in simulation.

INDEX TERMS Bandwidth transfer, deadline points, earliest smooth insertion time, transition, Δ check.

I. INTRODUCTION

Bandwidth transfer and reallocation are often unavoidable in bandwidth limited applications. Consider an Internet network channel. If the bandwidth of the channel is shared by a few users, the Internet may seem very fast. When new consumers request access to the network through the same channel, one or more current users have to free part of their bandwidth.

In embedded devices with limited energy, a low operating frequency is usually selected under a light load condition. The frequency may be raised when the load becomes heavier to meet the time constraints of the system tasks. If a new and urgent task requests to come into a system that is already 100% loaded at the highest frequency, then transferring a certain percentage of the bandwidth from less important current (or old) tasks to the new one is a reasonable decision.

In fact, bandwidth transfer is also worth considering even if the operating frequency is not the highest and the load is not 100%. Suppose the system runs at a frequency f_1 and it is sure that there will be a deadline loss due to the insertion of a new task. One possible choice to avoid the loss is to increase the frequency from f_1. Another option is to reduce the bandwidth occupied by old tasks and the operating frequency remains unchanged.

In this paper, the bandwidth transfer based on the EDF (Earliest Deadline First) algorithm in real-time applications is discussed [1]. One or more running tasks are compressed to meet the new bandwidth requirements that come from the acceleration of other current tasks and/or the insertion of new tasks [2]. The compression means a task′s period is prolonged while its computation time remains unchanged, thus its occupied bandwidth (or utilization) is decreased. On the contrary, to accelerate a task is to shorten its period. It is proved that the acceleration of a current task can be treated as the insertion of an equivalent new task [3]. Therefore, as for as new requirements, we need to discuss the insertion only.

New tasks′ insertion can be categorized into the mode-change problem [4]–[7]. It has three stages as shown in FIGURE 1: the old mode starting from t_{oldb}, the transition process from t_r and the new mode from t_{newb}. The request of the insertion occurs at t_r and certain current tasks start to be compressed.

If new tasks are inserted at t_r immediately, it is known that deadline missing may occur even though the sum of the utilizations of all the tasks, called the total utilization or total bandwidth, does not exceed one [2], [3]. In [8] and [9], it is proved that deadline missing is only possible during the time

The associate editor coordinating the review of this manuscript and approving it for publication was Vivek Kumar Sehgal.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

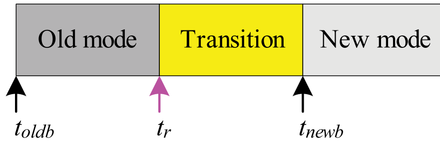


FIGURE 1. Three stages of a mode-change.

interval $[d'_{min}, d'_{max})$ that is part of the transition, where d'_{min} and d'_{max} represent the earliest and the latest deadline of the current instances of all the old tasks, respectively.

$\forall t \geq t_r$, if new tasks are released at t without causing any deadline missing afterwards, then t is called a *smooth insertion time* (or *smooth release time*), denoted by δ. Obviously, finding the *earliest smooth insertion time*, denoted by $\delta_{earliest}$, is significant and challenging.

A concise formula for calculating $\delta_{earliest}$ is given in [3], but it is not guaranteed to be $\delta_{earliest}$. One obvious way to get $\delta_{earliest}$ is to do multiple rounds of deadline checks. First we assume $\delta_{earliest} = t_r$ and check every deadline from d'_{min} to d'_{max}. If deadline missing is impossible, then we conclude that $\delta_{earliest}$ is really equal to t_r and no more check is required; otherwise we need next round of deadline checking with the assumption

$$\delta_{earliest} = \delta_{earliest} + timestep. \tag{1}$$

The *timestep* is the increment of the release time of the new task from the current round to the next. The *Smart way* presented in [8] shows that the *timestep* can be greater than one time unit in some cases so that the real $\delta_{earliest}$ can be reached quickly. However, each deadline point in $[d'_{min}, d'_{max})$ may need to be checked multiple times, even if there is only one new task.

Paper Contributions: (i).To get the real $\delta_{earliest}$ for the insertion of a new task, an advanced algorithm, denoted as *ESITforSNT* (*Earliest Smooth Insertion Time for Single New Task*), is presented and proved. With *ESITforSNT*, the deadlines of the tasks in the region $[d'_{min}, d'_{max})$ need to be checked at most once. This is much smaller than that from the *Smart way* in many situations. (ii).To verify the correctness of the new algorithm, a novel approach is shown in simulation. Firstly, every experimental task set with its total bandwidth exactly equal to 100% is carefully chosen so that every logical branch in *ESITforSNT* can be tested, and these task sets may be referenced for other researchers in the future. Secondly, an *offline* iteration algorithm that is obviously correct, though time-consuming, is used for comparison. It shows that the *offline* algorithm and *ESITforSNT* produce the same $\delta_{earliest}$ value in every test.

Paper Structure: Section II covers the review of the descriptions of tasks′ compression. A delaying rule is introduced and an example is provided. Section III presents Theorem 2 and 3, based on which *ESITforSNT* is designed for calculating $\delta_{earliest}$. Section IV shows the novel experimental approach and the simulation results. Section V discusses the related work and Section VI concludes the paper.

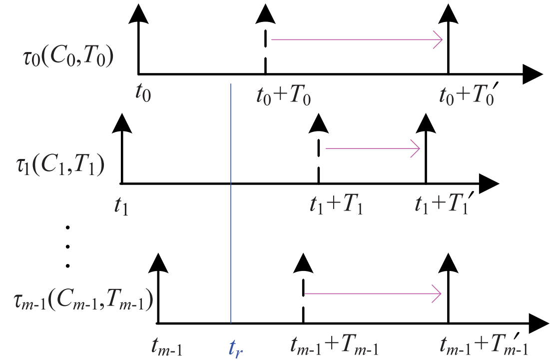


FIGURE 2. The compression of task set *M*.

II. SYSTEM MODULE AND AN EXAMPLE

Multiple tasks′ compression and the calculation of the processor demands of system tasks after compression are recalled in this section [3], [8], [9]. A delaying rule is proposed as the basic means for approaching $\delta_{earliest}$. An example is provided to help in understanding relevant theories. The main symbols used in this paper are summarized in Appendix A.

A. MULTIPLE TASKS′ COMPRESSION

As shown in FIGURE 2, the system has m current tasks that form a task set M. $\forall \tau_i(C_i, T_i) \in M$, $i \in (0, 1, \cdots, m-1)$, it has its computation time C_i, period T_i, and utilization $U_i = C_i/T_i$. The starting point of its current period is t_i. At t_r, new tasks are requested to be inserted and thus τ_i is compressed. Its period increases to T'_i from T_i and utilization decreases to $U'_i = C_i/T'_i$. Its remaining computation of the current instance is $c_i(t_r)$. The total freed bandwidth from the compression of the tasks in M is equal to $\sum_0^{m-1}(U_i - U'_i)$.

Suppose new tasks constitute a subset J and they are inserted into the system at the same time. The sum of the utilizations of the tasks in J is denoted as U_J, and r_J is introduced to represent the release time of their first instances, $r_J \geq t_r$. In order to keep the system schedulable in the new mode, it is assumed that

$$U_J \leq \sum_0^{m-1}(U_i - U'_i). \tag{2}$$

For the convenience of the associated descriptions, d'_{min} and d'_{max} are introduced to represent the earliest deadline and the latest deadline of all the current instances of the tasks in M after compression, respectively, that is,

$$\begin{aligned} d'_{min} &= min\{t_0 + T'_0, t_1 + T'_1, \cdots, t_{m-1} + T'_{m-1}\}, and \\ d'_{max} &= max\{t_0 + T'_0, t_1 + T'_1, \cdots, t_{m-1} + T'_{m-1}\}. \end{aligned}$$

As described above, deadline missing is only possible during $[d'_{min}, d'_{max})$, which is proved in [8] and [9]. [8] is in English while [9] is in Chinese.

For the convenience of literature indexing, it is better to list several important assumptions of the system model studied in this paper:

- Every task is periodic and scheduled in one processor. Every instance has an implicit deadline.

- The computation time of every instance of a task remains unchanged.
- Every old task is released at t_{oldb} and runs to d'_{max} (or after this time point) without pausing.
- The total bandwidth (or total utilization) of the tasks in the system is exactly 100%, both before and after compression.
- The bandwidth transferred from the compression equals that required by new tasks.

B. PROCESSOR DEMANDS AFTER COMPRESSION

The proof of subsequent theorems relies on the processor demand criterion [10], [11]. With the problem of compression, we can evaluate the processor demand of each task from t_r on.

$\forall t \geq t_r$, the processor demand of task $\tau_x(C_x, T_x)$ in $[t_r, t]$ is labelled with $D_x(t_r, t)$. The sum of the processor demands of all the tasks in J and M are indicated with $D_J(t_r, t)$ and $D_M(t_r, t)$, respectively. The sum of $D_J(t_r, t)$ and $D_M(t_r, t)$ is called the total processor demand, denoted as $D_{total}(t_r, t)$.

Then, $\Delta(t_r, t) = D_{total}(t_r, t) - (t - t_r)$ is introduced. According to the processor demand criterion, the deadlines at t are met if and only if $\Delta(t_r, t)$ is less than or equal to zero. Checking whether $\Delta(t_r, t)$ is greater than zero or not is called a Δ *check*. $\Delta(t_r, t)$ is the value of the Δ check.

The processor demand of new tasks can be calculated with

$$D_J(t_r, t) = \sum_{\tau_j \in J} \lfloor \frac{t - r_J}{T_j} \rfloor C_j. \tag{3}$$

The processor demand of the compressed task τ_i should be computed with

$$D_i(t_r, t) = \begin{cases} 0, & if\ t < t_i + T'_i. \\ c_i(t_r) + \lfloor \frac{t - t_i - T'_i}{T'_i} \rfloor C_i, & if\ t \geq t_i + T'_i. \end{cases} \tag{4}$$

From t_r to $t_i + T'_i$, the deadline point $t_i + T'_i$ of τ_i is met only if τ_i is assigned the processor time equal to $c_i(t_r)$, thus its processor demand equals $c_i(t_r)$ in this interval.

C. A DELAYING RULE

With (3) and (4), the important Theorem 1 is presented and proved in [8].

Theorem 1: With compressing the task set M, suppose that new tasks are released from r_J and deadline missing occurs at a time t_x in $[d'_{min}, d'_{max})$, then the insertion should be delayed and we must have

$$\delta_{earliest} \geq r_J + \Delta(t_r, t_x). \tag{5}$$

That is, the delayed insertion becomes possibly smooth only when the release time is delayed by not less than the value of the Δ check at t_x.

In the following descriptions, a task in the task set M shown as in FIGURE 2 is called a M task. Now we give the definitions of deadline points and an important delaying rule based on Theorem 1.

*Definition 1 (Deadline Points): A time point when at least one deadline of task instances occurs is called **a deadline point**. A M **task deadline point** refers to a time when there is at least one deadline of M task instances. Comparatively, if a deadline of an instance of any new task occurs, then we have **a new task deadline point**. If some deadlines of different tasks take an identical time value, then this time point is called **an overlapped deadline point**.*

*Definition 2 (Delaying Rule): According to Theorem 1, if the Δ check value $\Delta(t_r, t_x)$ is greater than zero at a deadline point t_x,then $\delta_{earliest}$ will not be earlier than $r_J + \Delta(t_r, t_x)$. Therefore, r_J should be delayed to $r_J + \Delta(t_r, t_x)$. If t_x is a M task deadline point, then it is necessary to do the Δ check at t_x again with the delayed r_J. If t_x is a new task deadline point that moves with the delaying of r_J, then $t_x = t_x + \Delta(t_r, t_x)$ should be checked. This type of check with t_x may take many times to get $\Delta(t_r, t_x) \leq 0$. This approach of check and delay and recheck until $\Delta(t_r, t_x) \leq 0$ is defined as **the delaying rule**. Also, it is declared that t_x **passes its Δ check** as soon as $\Delta(t_r, t_x) \leq 0$ becomes true.*

To get $\delta_{earliest}$ using the delaying rule, we start the Δ check with $r_J = t_r$ and $t_x = d'_{min}$. Once every deadline point in the region $[d'_{min}, d'_{max})$ passes its Δ check, then the newest r_J is the real $\delta_{earliest}$.

If $\Delta(t_r, t_x) > 0$, it can be seen from the delaying rule that the greater the value of $\Delta(t_r, t_x)$ is, the more delay the r_J will have, then the fewer checks are required for $\delta_{earliest}$. This is the major contribution of Theorem 1.

D. AN EXAMPLE

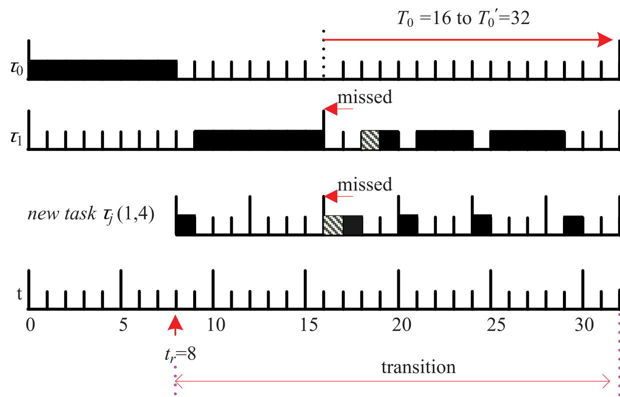
An example in FIGURE 3 is demonstrated to show the use of the delaying rule to evaluate $\delta_{earliest}$. If there is only one new task $\tau_j(C_j, T_j)$, r_j is used to indicate its release time. In this figure, before $t_r = 8$, the system has two tasks: $\tau_0(8, 16)$ and $\tau_1(8, 16)$. The total utilization equals one. After t_r, T_1 keeps unchanged, but the period of τ_0 is prolonged from 16 to 32 thus a bandwidth of 1/4 is transferred to the new task $\tau_j(1, 4)$. The total utilization remains 100%. It is easy to see $d'_{min} = T'_1 = 16$ and $d'_{max} = T'_0 = 32$.

If the new task is released at $r_j = t_r = 8$ as shown in FIGURE 3(a), the deadline point $d'_{min} = 16$ can not pass its Δ check for the first time because the value of this Δ check is

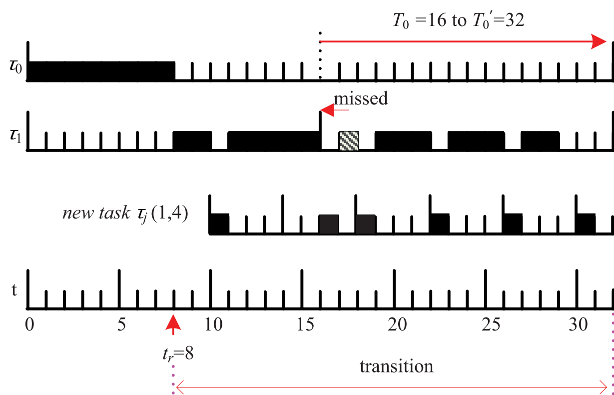
$$\begin{aligned} \Delta(t_r, 16) &= D_{total}(t_r, 16) - (16 - t_r) \\ &= D_{total}(8, 16) - 8 \\ &= D_0(8, 16) + D_1(8, 16) + D_j(8, 16) - 8 \\ &= 0 + c_1(t_r) + \lfloor \frac{16 - r_j}{T_j} \rfloor C_j - 8 = 8 + 2 - 8 = 2. \end{aligned}$$

According to the delaying rule, $r_j = r_j + \Delta(t_r, 16) = 10$ is implemented as shown in FIGURE 3(b) and we do the Δ check again. Unfortunately, d'_{min} does not pass its second Δ check due to

$$\begin{aligned} \Delta(t_r, 16) &= 0 + c_1(t_r) + \lfloor \frac{16 - 10}{T_j} \rfloor C_j - 8 \\ &= 0 + 8 + 1 - 8 = 1. \end{aligned}$$



(a) If the new task is released at $r_j = t_r = 8$.



(b) If the new task is delayed to $r_j = 10$.

FIGURE 3. An example using the delaying rule.

In this way, further delay is needed. The consecutive checks are as follows:

- The third Δ check: $r_j = 11$, $\Delta(t_r, 16) = 1$.
- The fourth Δ check: $r_j = 12$, $\Delta(t_r, 16) = 1$.
- The fifth Δ check: $r_j = 13$, $\Delta(t_r, 16) = 0$.

That is to say, five Δ checks in total have to be done at d'_{min}. Then four deadlines of the new task have to be checked to get the real $\delta_{earliest}$ by *Smart way*: $t = 17, t = 21, t = 25$ and $t = 29$. It is easy to see that each of the four deadlines will pass its Δ check for the first time, thus r_j will not be delayed further. Therefore, nine Δ checks are needed to get $\delta_{earliest} = 13$ totally. Using the new algorithm *ESITforSNT* presented in the next section, however, only one Δ check is enough to reach $\delta_{earliest} = 13$.

III. NEW THEOREMS AND ESITforSNT ALGORITHM

Although the number of the times of the required Δ checks may be reduced by Theorem 1, a deadline point in $[d'_{min}, d'_{max})$ may need to be checked multiple times even if there is only one new task. In this section, new theorems are presented and proved for single new task's insertion, with which every M task deadline point needs to be checked only once to get $\delta_{earliest}$. Based on these theorems, an advanced algorithm, called *ESITforSNT*, is provided.

A. NEW THEOREMS

In Figure 3(a), $d'_{min} = 16$, $d'_{max} = 32$, and there is only one M task deadline point in $[d'_{min}, d'_{max})$. But the new task has four deadline points: $t = 16$, $t = 20$, $t = 24$ and $t = 28$. Note that these points of the new task will also shift with the delaying of r_j.

Based on Theorem 1, we should not only check the M task deadline points in $[d'_{min}, d'_{max})$, but also check the new task deadline points. Fortunately, this can be simplified with Theorem 2 and 3.

Theorem 2 (Criterion 1): With compressing the task set M for the insertion of single new task $\tau_j(C_j, T_j)$ under EDF, the release time of τ_j is labeled by r_j. In $[d'_{min}, d'_{max})$, suppose that the set M has n deadline points: $d'_{M(0)}, d'_{M(1)}, \cdots,$ and $d'_{M(n-1)}$. They are arranged in ascending order, that is,

$$d'_{M(0)} < d'_{M(1)} < \cdots < d'_{M(n-1)} < d'_{max}.$$

Apparently, $d'_{M(0)} = d'_{min}$. Let $d'_{M(n)} = d'_{max}$. $\forall k \in (0, 1, \cdots, n-1)$, in $(d'_{M(k)}, d'_{M(k+1)})$, assume that the first deadline point (if any) of τ_j is $d_{j(k)(0)}$.

Then, if $d'_{M(k)}$ and $d_{j(k)(0)}$ pass their Δ checks, other deadline points (if any) in $(d'_{M(k)}, d'_{M(k+1)})$ will also pass their Δ checks.

Proof: See Appendix B.

Multiple new task deadline points may exist in the region $(d'_{M(k)}, d'_{M(k+1)})$. Theorem 2 declares that if the first one passes its Δ check, then the rest of them will. As a result, every deadline point of M tasks and some deadline points of the new task in $[d'_{min}, d'_{max})$ should be checked to obtain $\delta_{earliest}$. Theorem 3 will indicate that these points should be checked at most once.

Take the FIGURE 3(a) as an example. It can be seen that $n = 1$. We need to check the only deadline point of the M tasks at $t = 16$ and the first deadline point of τ_j at $t = 20$ in the region [16,32). It is not necessary to check any other deadline points. Note that the point at $t = 16$ is an overlapped one since both a M task and the new task have a deadline at this point.

Theorem 3 (Criterion 2): If there is only one new task τ_j, then the deadline points $d'_{M(k)}$ and $d_{j(k)(0)}$ pass their Δ checks as soon as the current r_j is delayed by $L_{(k)}$ (in number of time units) that is calculated according to the following Case 1 or Case 2:

Case 1. If $\Delta(t_r, d'_{M(k)}) > 0$, then

$$L_{(k)} = d'_{M(k)} - (r_j + \lfloor \frac{d'_{M(k)} - r_j}{T_j} \rfloor T_j) + \Delta(t_r, d'_{M(k)}) + \lceil \frac{\Delta(t_r, d'_{M(k)}) - C_j}{C_j} \rceil (T_j - C_j). \quad (6)$$

Case 2. If $\Delta(t_r, d'_{M(k)}) \leq 0$, then

$$L_{(k)} = \begin{cases} 0, & \text{if } \Delta(t_r, d_{j(k)(0)}) \leq 0. \\ \Delta(t_r, d_{j(k)(0)}), & \text{if } \Delta(t_r, d_{j(k)(0)}) > 0. \end{cases} \quad (7)$$

Proof: See Appendix C.

The CASE 1 of Theorem 3 means that if $d'_{M(k)}$ does not pass its Δ check, then the release time r_j of the new task should be delayed by a time amount $L_{(k)}$ given by (6) and there is no need to check $d_{j(k)(0)}$ any more.

If $d'_{M(k)}$ pass its Δ check, then we come to the CASE 2 of Theorem 3. The Δ check of $d_{j(k)(0)}$ must be done and r_j is delayed by (7) according to the Δ check value.

Now we use Theorem 3 to discuss the example of FIGURE 3. First let $r_j = t_r = 8$. Because the Δ check value $\Delta(t_r, d'_{M(0)}) = \Delta(8, 16) = 2 > 0$, by using (6) we have

$$\begin{aligned} L_{(0)} &= d'_{M(0)} - (r_j + \lfloor \frac{d'_{M(0)} - r_j}{T_j} \rfloor T_j) + \Delta(t_r, d'_{M(0)}) \\ &\quad + \lceil \frac{\Delta(t_r, d'_{M(0)}) - C_j}{C_j} \rceil (T_j - C_j) \\ &= 16 - (8 + \lfloor \frac{16-8}{4} \rfloor 4) + 2 + \lceil \frac{2-1}{1} \rceil (4-1) = 5. \end{aligned}$$

Thus we get

$$\delta_{earliest} = r_j + L_{(0)} = 8 + 5 = 13.$$

This shows that only *one* Δ *check* is needed to get $\delta_{earliest}$. Remember that *nine* Δ *checks* are required with the *Smart way*, as described before.

In Theorem 2, when multiple M task deadline points exist in the region $[d'_{min}, d'_{max})$, it is assumed that they are sorted from $d'_{M(0)}$ to $d'_{M(n-1)}$ according to the times they appear. Although this assumption is helpful for the description of the theorem, sorting causes some overhead. Fortunately, Lemma 1 declares that $\delta_{earliest}$ can be calculated without sorting.

Lemma 1: As for all the M task deadline points in $[d'_{min}, d'_{max})$, i.e., $d'_{M(0)}, d'_{M(1)}, \cdots$, and $d'_{M(n-1)}$, $\delta_{earliest}$ is not influenced by the priority order of their Δ checks.

Proof: See Appendix D.

B. ESITforSNT ALGORITHM

Now we start to design a new algorithm, named *ESITforSNT*, to calculate $\delta_{earliest}$ based on Theorem 2 and 3. In $[d'_{min}, d'_{max})$, the checking process can be implemented task by task since queuing deadline points is not necessary from Lemma 1. First the deadline points of task τ_0 are checked: $t_0 + T'_0, t_0 + 2T'_0, \cdots$, and $t_0 + pT'_0$. Here, p is a positive integer and $t_0 + pT'_0 < d'_{max}$. Consequently, $\tau_1, \tau_2, \ldots$, and τ_{m-1} will be checked. If the value of any Δ check is greater than zero, then r_j is updated according to Theorem 3. $\delta_{earliest}$ takes the value of r_j after all the deadline points are checked.

Overlapped deadline points should be checked only once. Therefore, it is necessary to introduce an array *flag*[ ] to mark whether a deadline point is an overlapped one. A problem is

Algorithm 1 ESITforSNT

The main function:

Input: $m, t_r, c_i(t_r), t_i, T_i, T'_i, C_i, C_j, T_j$.

1: Find out d'_{min} and d'_{max}
2: Initialize r_j, $d'_M[\,]$, L_{seg}, t_{segend}, $mark$, and $flag[\,]$
3: **while** $(t_{segend} - d'_{max} < 0)$ **do**
4:     Create a segment($L_{seg}, t_{segend}, t_{segstart}$)
5:     $mark++$
6:     **for** $(i = 0; i < m; i++)$ **do**
7:         **while** $(r_j < d'_M[i] < t_{segend})$ **do**
8:             $pos_{rel} = d'_M[i] - t_{segstart}$
9:             **if** $(flag[pos_{rel}] - mark \neq 0)$ **then**
10:                 $r_j = deadlinecheck(d'_M[i])$
11:                 $flag[pos_{rel}] = mark$
12:             **end if**
13:             $d'_M[i] = d'_M[i] + T'_i$
14:         **end while**
15:     **end for**
16: **end while**
17: $\delta_{earliest} = r_j$

The sub function *deadlinecheck*():

Input: $d'_{M(k)} = d'_M[i]$.

1: calculate $D_{total}(t_r, d'_{M(k)})$ with (3) and (4)
2: $\Delta(t_r, d'_{M(k)}) = D_{total}(t_r, d'_{M(k)}) - (d'_{M(k)} - t_r)$
3: **if** $(\Delta(t_r, d'_{M(k)}) > 0)$ **then**
4:     calculate $L_{(k)}$ with (6)
5: **else**
6:     calculate $L_{(k)}$ with (7)
7: **end if**
8: $r_j = r_j + L_{(k)}$
9: **return** r_j

that if the region $[d'_{min}, d'_{max})$ is lengthy, then the capacity of *flag*[ ] will be too large. To solve this, the region $[d'_{min}, d'_{max})$ is divided into segments in the algorithm. We do Δ checking segment by segment and *flag*[ ] needs to deal with one segment only. The starting time point, ending point, and length of a segment are denoted as $t_{segstart}$, t_{segend} and L_{seg}, respectively. At the beginning, an initial constant is assigned to L_{seg}. Note that the length of the last segment ending with d'_{max} may be less than this constant.

In the main function of this algorithm, first d'_{min} and d'_{max} should be calculated. Then the initialization is implemented (line 2). r_j is initialized with t_r. An array $d'_M[\,]$ is introduced to denote M task deadlines. $\forall \tau_i(C_i, T_i) \in M, i \in (0, 1, \cdots, m-1)$, $d'_M[\,]$ is set to $t_i + T'_i$. In the initialization, an initial constant L_{SEG} is assigned to L_{seg}. $mark$ and *flag*[ ] are cleared, $t_{segstart} = d'_{min}$. Here, $mark$ is prepared to set the value of the element in *flag*[ ].

A segment is built in line 4. If $t_{segend} + L_{SEG} - d_{max} > 0$, then $L_{seg} = d'_{max} - t_{segend}$; otherwise $L_{seg} = L_{SEG}$. Next, t_{segend} will be added by L_{seg} and $t_{segstart} = t_{segend} - L_{seg}$. In this way, the first segment starts from d'_{min} and the last one ends at d'_{max}.

All the M tasks are checked within a selected segment from line 6 to 15. From line 7, a deadline point $d'_M[i]$ is selected and its position pos_{rel} relative to the start of the segment is calculated, which corresponds to $flag[pos_{rel}]$. If $d'_M[i]$ is not an overlapped deadline point, then a sub function *deadlinecheck*() is called for Δ check and $flag[pos_{rel}]$ is set with the current value of *mark* that indicates a marked point to later checks. Theorem 3 is implemented in the sub function.

The Temporal Complexity: In this algorithm, the most time-consuming operation is the execution of the sub function *deadlinecheck*(). In other words, the time required to fulfill the algorithm mainly depends on this sub function. When it is called, (3) and (4) are required to evaluate the processor demands of the tasks and one or two Δ checks are needed: first $d'_M[k]$ should be checked and the next is $d_{j(k)(0)}$ if $\Delta(t_r, d'_M[k]) \leq 0$. Therefore, it is reasonable to measure the temporal complexity of the algorithm with the number of the times of Δ checks. Then, how many Δ checks will be required to get $\delta_{earliest}$? In Theorem 2, it is assumed that the M task set has n deadline points in $[d'_{min}, d'_{max})$. Therefore, the worst-case time complexity of this algorithm is equal to $2n$ *times of* Δ *checks*.

IV. EXPERIMENTS

The purpose of the following experiments is to verify Theorem 2, 3 and the *ESITforSNT* algorithm. In these experiments, the release time of all the M tasks is assumed to be zero, that is, $t_{oldb} = 0$.

The problem is how to implement the verification. A very simple *offline* and *standard* algorithm that is obviously correct, though time consuming, is used. In every experiment, the $\delta_{earliest}$ values from *ESITforSNT* and from the *offline* algorithm are compared. The pseudo code of the *standard* algorithm is provided in Algorithm 2.

To start Algorithm 2, first $r_j = t_r$ is assumed. All the tasks are scheduled and the system simply runs from t_r to d'_{max}. If deadline missing occurs after t_r, then $r_j = r_j + 1$ is done and the system runs from t_r to d'_{max} again. If no deadline is missed till d'_{max}, then the algorithm ends with $\delta_{earliest} = r_j$. The correctness of this process is obvious, thus this algorithm can be used as a *standard* algorithm. Remember the time sequence

$$0 \leq t_r \leq r_j < d'_{max}.$$

Let $T_{LCM(0\sim m-1)}$ be the least common multiple of the periods of the M tasks before compression. If $t_r = 0$ and $t_r = T_{LCM(0\sim m-1)}$, obviously, the new task can be smoothly inserted immediately at t_r. Therefore, for a set of M tasks, we only do the experiments with the cases from $t_r = 1$ to $t_r = T_{LCM(0\sim m-1)} - 1$. With each case, we compare the $\delta_{earliest}$ value obtained from *ESITforSNT* with that from the *standard* algorithm.

For the convenience of describing a bandwidth transfer process, Definition 3 is provided first.

Algorithm 2 *Offline* or *Standard* Algorithm

The main function:

Input: m, t_r, $c_i(t_r)$, t_i, T_i, T'_i, C_i, C_j, T_j.

1: Find out d'_{max}
2: Schedule M tasks from $t = 0$ to t_r
3: Compress M tasks and set $r_j = t_r$
4: **loop**:
5: Schedule M tasks from $t = t_r$ to r_j
6: Schedule M tasks and τ_j from $t = r_j$ to d'_{max}
7: **if**(deadline missing occurs) **then**
8: $r_j + +$
9: **go to loop**
10: **end if**
11: $\delta_{earliest} = r_j$

TABLE 1. A configuration of tasks in experiments.

M task	T_i	U_i	$p_{freed(i)}$	New task
τ_0	$50 \sim 250$	≈ 0.195	0	$\tau_j(3, 5)$
τ_1	120	0.4	0.75	
τ_2	180	0.4	0.75	
τ_3	$T_{LCM(0\sim 2)}$	≈ 0.005	0	

Definition 3 (Freed Bandwidth Ratio): The Freed bandwidth ratio from a M task $\tau_i(C_i, T_i)$ is defined as $(U_i - U'_i)/U_i$, denoted as $p_{freed(i)}$.

In the experiments, tasks are configured mainly depending on $p_{freed(i)}$. TABLE 1 is an initial configuration of task parameters. The new task is $\tau_j(3, 5)$ with a bandwidth $U_j = 0.6$. There are four tasks in the M task set in total: τ_0, τ_1, τ_2 and τ_3. Before compression, we have $U_1 = 0.4$ and $U_2 = 0.4$. After compression, each of the two tasks frees its bandwidth according to a ratio of 0.75, that is, $p_{freed(1)} = p_{freed(2)} = 0.75$.

In TABLE 1, τ_0 and τ_3 does not transfer any bandwidth, that is, $p_{freed(0)} = p_{freed(3)} = 0$.

Then, we get the bandwidth U_j of the new task with an expression, called *the bandwidth allocation expression*:

$$\begin{aligned} U_j &= p_{freed(0)}U_0 + p_{freed(1)}U_1 \\ &\quad + p_{freed(2)}U_2 + p_{freed(3)}U_3 \\ &= 0 \times U_0 + 0.75 \times U_1 + 0.75 \times U_2 + 0 \times U_3 = 0.6. \end{aligned}$$

The period T_0 of τ_0 takes 201 integer values from 50 to 250. Accordingly, C_0 has 201 integer values.

For τ_3, its period T_3 takes the value of the least common multiple of T_0, T_1 and T_2, denoted as $T_{LCM(0\sim 2)}$. The values of C_3 and C_0 are calculated as follows:

$$\begin{aligned} C_0 &= \lfloor 0.195 \times T_0 \rfloor, \textit{ and} \\ C_3 &= (1 - (\frac{C_0}{T_0} + \frac{C_1}{T_1} + \frac{C_2}{T_2})) \times T_3. \end{aligned}$$

In this way, $U_0 + U_3 = 0.2$ is guaranteed.

The total number of the M task sets generated from TABLE 1 is 201. These task sets are listed in TABLE 2.

Now we give the reasons for the above configuration:

TABLE 2. The 201 M task sets from TABLE 1.

index	task τ_0	task τ_3	task τ_1 and τ_2
1	$\tau_0(9,50)$	$\tau_3(36,1800)$	Before compression:
2	$\tau_0(9,51)$	$\tau_3(144,6120)$	$\tau_1(48,120)$
3	$\tau_0(10,52)$	$\tau_3(36,4680)$	$\tau_2(72,180)$
4	$\tau_0(10,53)$	$\tau_3(216,19080)$	After compression:
...	...	...	$\tau_1(48,480)$
200	$\tau_0(48,249)$	$\tau_3(216,29880)$	$\tau_2(72,720)$
201	$\tau_0(48,250)$	$\tau_3(72,9000)$	

- The bandwidth of τ_3 is very small (about 0.005). It is used together with τ_0 to produce $U_0 + U_3 = 0.2$ so that both $\sum U$ and $\sum U'$(the total bandwidths before and after compression) are exactly equal to 1. Thus *ESITforSNT* can be verified under the condition of exactly 100% load. In addition, we set $T_3 = T_{LCM(0\sim2)}$ to produce $T_{LCM(0\sim3)} = T_3$, that is to say, the least common multiple of the periods of all the M tasks is T_3. Thus, the number of the possible values of t_r (from 1 to $T_{LCM(0\sim3)} - 1$) is controllable and not too large.
- τ_1 and τ_2 are configured with relatively large freed bandwidth ratios. Both $p_{freed(1)}$ and $p_{freed(2)}$ are 0.75. The reason for this is that a transfer with large bandwidth is easy to cause $\delta_{earliest} > t_r$. This is good for the verification. With an unreasonable configuration, on the contrary, there may be fewer or even no cases with $\delta_{earliest} > t_r$.
- A relatively short period, $T_j = 5$, is assigned to the new task τ_j. With a constant U_j, the shorter the T_j is, the greater processor demand the τ_j has in a given time region, and the more likely the deadline will be lost. Therefore, a new task with a short period is used in simulation.
- τ_0 does not transfer any bandwidth and its period is changeable. Let us compare T_0 with T_1 and T_2. When T_0 takes a value in [50,120) (region 1), it has the shortest value, $T_0 < T_1 < T_2$. When T_0 takes a value in [120,180) (region 2), it is in the middle, $T_1 \leq T_0 < T_2$. If T_0 is in [180,250) (region 3), it is the longest period, $T_1 < T_2 \leq T_0$. No matter which region it is, many task sets are available to generate three different types of delay: zero delay with $\delta_{earliest} = t_r$, the delay corresponding to the Case 1 of Theorem 3, and the delay with the Case 2. This is favorable for verification. Here are several examples:
  Three examples in region 1:
  $$T_0 = 81, t_r = 1, \delta_{earliest} = 1.$$
  $$T_0 = 90, t_r = 328, \delta_{earliest} = 333(Case\ 1).$$
  $$T_0 = 90, t_r = 321, \delta_{earliest} = 322(Case\ 2).$$
  Three examples in region 2:
  $$T_0 = 121, t_r = 1, \delta_{earliest} = 1.$$
  $$T_0 = 125, t_r = 3575, \delta_{earliest} = 3581(Case\ 1).$$
  $$T_0 = 125, t_r = 3581, \delta_{earliest} = 3582(Case\ 2).$$
  Three examples in region 3:
  $$T_0 = 181, t_r = 1, \delta_{earliest} = 1.$$
  $$T_0 = 200, t_r = 117, \delta_{earliest} = 126(Case\ 1).$$
  $$T_0 = 200, t_r = 1906, \delta_{earliest} = 1907(Case\ 2).$$

The Experimental Results: With the 201 different sets of M tasks, there are 4549320 tests for 4549320 $\delta_{earliest}$ values. For a specific M task set, every test is done with a different t_r value. In each test, an identical $\delta_{earliest}$ value is obtained from *ESITforSNT* and from the *standard* algorithm.

Different task configurations from TABLE 1 are also used to test *ESITforSNT*. All the experimental results show that the above theorems are correct.

V. RELATED WORK

There are many papers related with mode-change problems [4]–[6]. A mode-change is initiated whenever a significant change in the internal state or an event from the environment is detected. The reasons for changing the operational modes are well listed in [12]. There are four basic requirements: schedulability, promptness, periodicity, and consistency [13]. To meet the four basic requirements, two points have been emphasized by researchers: the protocol and the *offset*.

Two major types of protocols are synchronous and asynchronous [12], [14], [15]. With synchronous protocols, new-mode tasks (allowed to be released after t_r) can not be released until all the old-mode tasks (only released in the old mode) have completed their last activations, while with asynchronous protocols, new-mode and old-mode tasks can be executed at the same time during the transition process. A comparison is made between these two [13]. It is pointed out that synchronous protocols are generally simple and require no specific schedulability analysis. They do not give good promptness. Asynchronous protocols, however, often provide a faster response to mode-change requests, and some of them provide periodicity. But they need a specific schedulability analysis.

The *offset* is the time delay a protocol may impose to the first release of a new-mode task after t_r. In this paper, we take *offset*$= \delta_{earliest} - t_r$.

Sometimes the feasibility of a mode-change highly relies on finding the *offset* [3], [16]. Some researchers use the *offset* in their models or point out that it is important, others try to find a way to calculate it [5], [13]. In [17], for example, an asynchronous protocol that uses *offsets* was provided by Pedro and Burns, but no way to calculate such *offsets* is given. In [13], it is declared that how to calculate the *offset* is an open problem and an iterative method based on fixed priorities is presented. The *offsets* of new-mode tasks are chosen from possible maximum values to minimum. If a transition is not feasible with the selected values, then shorter values are tried until the minimum values for consistency are exactly the same as those from the previous iteration.

In [18], dynamic voltage scaling with RM (Rate Monotonic) and EDF are studied. From the simulations, Pillai *et al.*

note an interesting phenomenon, that is, the dynamic addition of a task to a task set may cause transient missed deadlines unless one is very careful. However, the temporal complexity of such an insertion is not analyzed.

The mode-change based on EDF is also discussed in the case studies with video streams [16]. An iterative method is provided to calculate the *offset*. Given a length of time, for example 400ms, schedulability is first checked with an assumed $offset = 0$. If the system is not schedulable, then the analysis is performed with different sizes of assumed values which are chosen by binary search. Analysis stops when the smallest value is found that makes the system schedulable. This iterative method depends on the processor demand criterion for EDF. It is not well suitable for use at run-time due to the logarithmic complexity. Additionally, further studies are needed to define the length of the transition process and to find the time region in which deadline missing is possible.

A protocol to handle the admission control is provided in [19]. The framework can deal with overlapping scheduling transients and sporadic tasks. But the model of [19] is quite different from that of this paper in which bandwidth transfer between tasks is discussed.

Determining task shares on processors are discussed in [20]. A task has an initial weight (bandwidth). This weight may be increased or decreased, which means a task can be accelerated or decelerated actually. However, the earliest time to start this operation without deadline missing is not discussed in [20].

Andersson claims that it is unfortunate that the research literature offers no mode-change protocol and corresponding schedulability analysis for a processor scheduled by EDF, and then he presents an analysis for this problem based on a rule that task τ_i switches from its old mode to a new mode *at the next release time* if the beginning of the current instance is earlier than t_r [12]. This rule is different from the compression shown in FIGURE 2 of this paper, where τ_i increases its period immediately at t_r, which is better for promptness.

The most relevant works to the model of this paper are [2], [3], [8], [9]. Buttazzo *et al.* present an elastic scheduling model for the task set based on EDF, in which the compression, the acceleration, and the insertion are discussed [2], [21]. An insertion time $\delta = (t_i + T_i) - c_i(t_r)/U_i$ is provided. A deeper research is made by Qian and an earlier smooth time $\delta = (t_i + T_i) - c_i(t_r)/(U_i - U_i')$ is proved in [3], but there is no guarantee for $\delta = \delta_{earliest}$.

The problem of multiple tasks′ compression is studied in [8] and [9]. It is proved that deadline missing is only possible in $[d'_{min}, d'_{max})$. Theorem 1 is also presented in [8]. This theorem indicates that the time step from the current Δ check to the next may be greater than one so that $\delta_{earliest}$ can be reached quickly.

VI. CONCLUSION

In summary, with the bandwidth transfer from multiple periodic tasks scheduled with EDF, the following important points are shown in this paper:

- Some important conclusions are recalled. For example, deadline missing is only possible in $[d'_{min}, d'_{max})$, which is proved in [8] and [9].
- To get $\delta_{earliest}$ for inserting a new task with *ESITforSNT* algorithm, only one Δ check is needed for each M task deadline point. Totally suppose M tasks have n deadline points in $[d'_{min}, d'_{max})$, then the number of the times of the Δ checks required for $\delta_{earliest}$ will not be greater than $2n$.
- The experiments of the bandwidth transfer are specially designed. Firstly, a *standard* algorithm is utilized. Although it is time consuming, its correctness is obvious. Every experimental $\delta_{earliest}$ obtained from *ESITforSNT* is compared with the one from the *standard*. Secondly, effective task sets used in experiments are configured. In the experiments, configuring task sets randomly is not a good way for the type of bandwidth transfer. Therefore, typical, accurate and well-selected configurations are adopted and more than 201 task sets are tested. These configurations are not only effective in the verification of the current theorems, but also convenient for the comparisons in future studies.

APPENDIX A
SYMBOLS

TABLE 3 lists the main symbols used in this paper.

APPENDIX B
PROOF OF THEOREM 2

Proof: In $[d'_{M(k)}, d'_{M(k+1)})$, $d'_{M(k)}$ and the deadline points of τ_j should be checked. τ_j may have more than one deadline point in this interval. Suppose $d_{j(k)(1)}$ is the second deadline point of τ_j. If the Δ check of τ_j at the first deadline point $d_{j(k)(0)}$ is passed, then

$$\begin{aligned}
\Delta(t_r, d_{j(k)(0)}) &= D_{total}(t_r, d_{j(k)(0)}) - (d_{j(k)(0)} - t_r) \\
&= D_M(t_r, d_{j(k)(0)}) + D_j(t_r, d_{j(k)(0)}) - (d_{j(k)(0)} - t_r) \\
&\le 0.
\end{aligned}$$

From $d_{j(k)(0)}$ to $d_{j(k)(1)}$, the new task increases its processor demand while a M task does not. Thus we have

$$\begin{aligned}
&\Delta(t_r, d_{j(k)(1)}) \\
&\quad = D_{total}(t_r, d_{j(k)(1)}) - (d_{j(k)(1)} - t_r) \\
&\quad = D_{total}(t_r, d_{j(k)(1)}) - (d_{j(k)(1)} - d_{j(k)(0)}) \\
&\qquad - (d_{j(k)(0)} - t_r) \\
&\quad = D_{total}(t_r, d_{j(k)(1)}) - T_j - (d_{j(k)(0)} - t_r) \\
&\quad = D_M(t_r, d_{j(k)(1)}) + D_j(t_r, d_{j(k)(1)}) - T_j \\
&\qquad - (d_{j(k)(0)} - t_r) \\
&\quad = D_M(t_r, d_{j(k)(0)}) + D_j(t_r, d_{j(k)(0)}) - (d_{j(k)(0)} - t_r) \\
&\qquad + C_j - T_j \\
&\quad \le C_j - T_j < 0.
\end{aligned}$$

TABLE 3. The Symbols.

M	Set of old tasks
m	Total number of old tasks
n	Total number of M task deadline points in $[d'_{min},d'_{max})$
J	Set of new tasks
t_{oldb}	Start of old mode
t_{newb}	Start of new mode
r_J	Release time of new tasks
r_j	Release time of single new task τ_j
$\tau_j(C_j, T_j)$	New task τ_j with computation C_j and period T_j,$\tau_j \in J$
U_j	Utilization of τ_j
$\tau_i(C_i, T_i)$	Old task τ_i with computation C_i and period T_i,$i \in (0, 1, \cdots, m-1)$,$\tau_i \in M$
U_i	Utilization of τ_i
$c_i(t_r)$	Remaining computation of the instance of τ_i at t_r
T'_i	Period of τ_i after compression
U'_i	Utilization of τ_i after compression
t_i	Start of the current period of τ_i, $t_i \leq t_r < t_i + T_i$
t_r	Request time of new tasks
δ	Smooth release time of new tasks
$\delta_{earliest}$	Earliest δ
d'_{min}	$min\{t_0 + T'_0, t_1 + T'_1, \cdots, t_{m-1} + T'_{m-1}\}$
d'_{max}	$max\{t_0 + T'_0, t_1 + T'_1, \cdots, t_{m-1} + T'_{m-1}\}$
$d'_{M(k)}$	A M task deadline point in $[d'_{min},d'_{max})$, $k \in (0, 1, \cdots, n-1)$
$d_{j(k)(0)}$	First deadline of τ_j in $(d'_{M(k)},d'_{M(k+1)})$
$L_{(k)}$	Delay of r_j
$T_{LCM(0\sim m-1)}$	Least common multiple period of M tasks before compression
$D_x(t_r, t)$	Processor demand of task τ_x in $[t_r, t]$, $x \in (i, j)$
$D_M(t_r, t)$	Sum of processor demands of M tasks in $[t_r, t]$
$D_J(t_r, t)$	Sum of processor demands of new tasks in $[t_r, t]$
$D_{total}(t_r, t)$	$D_M(t_r, t)$+$D_J(t_r, t)$
$\Delta(t_r, t)$	$D_M(t_r, t)$+$D_J(t_r, t) - (t - t_r)$
$P_{freed(i)}$	Freed bandwidth ratio from task τ_i

This indicates that $d_{j(k)(1)}$ also pass its Δ check. Similarly, other deadline points of τ_j after $d_{j(k)(1)}$ (if any) in the region $(d'_{M(k)},d'_{M(k+1)})$ will also pass their Δ checks.

APPENDIX C
PROOF OF THEOREM 3

Proof: There are two cases: Case 1 and Case 2.

Proof of Case 1: In this case, due to $\Delta(t_r, d'_{M(k)}) > 0$, r_j should be delayed by $L_{(k)}$. The delaying will of course decrease the processor demand $D_j(t_r, d'_{M(k)})$.

If $\Delta(t_r, d'_{M(k)}) > C_j$, $L_{(k)}$ must be longer than the length of T_j to make $D_j(t_r, d'_{M(k)})$ reduced enough for $\Delta(t_r, d'_{M(k)}) \leq 0$. As shown in FIGURE 4, the delaying is implemented by three steps: *the first delay*, *the second delay* and *the third delay*. The amount of delay by each step is denoted by $L_{(k)}|_1$, $L_{(k)}|_2$, and $L_{(k)}|_3$, respectively. Naturally

$$L_{(k)} = L_{(k)}|_1 + L_{(k)}|_2 + L_{(k)}|_3 \tag{8}$$

For the purpose of convenience, with a discussed time t, $D_{total}(t_r, t)|_1$, $D_{total}(t_r, t)|_2$ and $D_{total}(t_r, t)|_3$ are used to represent the total processor demands in $(t_r, t]$ after the first, the second, and the third step, respectively. Correspondingly, Their Δ check values are $\Delta(t_r, t)|_1$,$\Delta(t_r, t)|_2$ and $\Delta(t_r, t)|_3$.

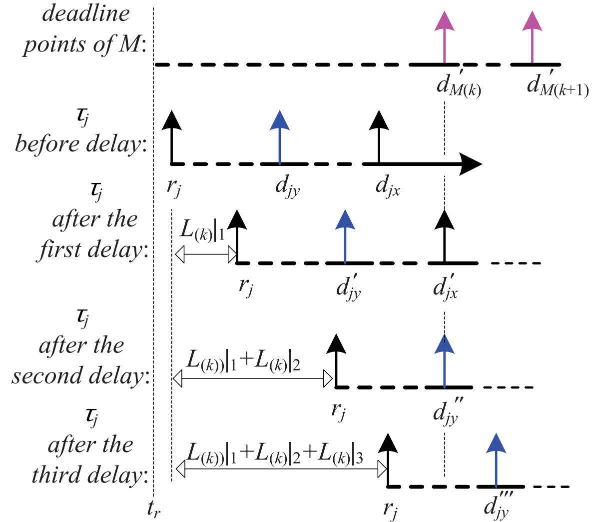


FIGURE 4. Using the delaying rule with Case 1.

Let d_{jx} be the latest deadline point (if any) of τ_j in $(t_r, d'_{M(k)}]$, that is, $0 \leq d'_{M(k)} - d_{jx} < T_j$. d_{jy} represents a special deadline point(if any) of τ_j before d_{jx}, as in FIGURE 4.

Before delaying, we have

$$\Delta(t_r, d'_{M(k)}) = D_{total}(t_r, d'_{M(k)}) - (d'_{M(k)} - t_r) > 0.$$

After the first step delay, the release of τ_j is delayed by $L_{(k)}|_1$ so that d_{jy} shifts to d'_{jy}, and d_{jx} to $d'_{jx} = d'_{M(k)}$. Then

$$\begin{aligned} L_{(k)}|_1 &= d'_{M(k)} - d_{jx} \\ &= d'_{M(k)} - (r_j + \lfloor \frac{d'_{M(k)} - r_j}{T_j} \rfloor T_j) \end{aligned} \tag{9}$$

Notice that the total demand $D_{total}(t_r, d'_{M(k)})$ remains unchanged when the first step delay is completed, thus we have

$$\begin{aligned} D_{total}(t_r, d'_{M(k)})|_1 &= D_{total}(t_r, d'_{M(k)}), \textit{and} \\ \Delta(t_r, d'_{M(k)})|_1 &= \Delta(t_r, d'_{M(k)}) > 0. \end{aligned}$$

Through the second step delay, r_j is increased by $L_{(k)}|_2$, and d'_{jy} will move to d''_{jy} to produce

$$L_{(k)}|_2 = \lceil \frac{\Delta(t_r, d'_{M(k)})|_1 - C_j}{C_j} \rceil T_j \tag{10}$$

The purpose of this step is to make the Δ check value become less than or equal to C_j:

$$\begin{aligned} D_{total}(t_r, d'_{M(k)})|_2 &= D_{total}(t_r, d'_{M(k)})|_1 - \frac{L_{(k)}|_2}{T_j} C_j, \textit{and} \\ \Delta(t_r, d'_{M(k)})|_2 &= \Delta(t_r, d'_{M(k)})|_1 - \frac{L_{(k)}|_2}{T_j} C_j \\ &\leq \Delta(t_r, d'_{M(k)})|_1 \\ &\quad - \frac{\Delta(t_r, d'_{M(k)})|_1 - C_j}{C_j} C_j = C_j. \end{aligned}$$

Now discuss the third step delay. The deadline d''_{jy} will shift to d'''_{jy}. The Δ check with $d'_{M(k)}$ becomes passed as soon as $L_{(k)}|_3 > 0$ since $\Delta(t_r, d'_{M(k)})|_2 \le C_j$. However, $L_{(k)}|_3 > 0$ may not be enough for d'''_{jy} to pass its Δ check. According to the delaying rule, in order to make both $d'_{M(k)}$ and d'''_{jy} passed, the minimum value of $L_{(k)}|_3$ should be calculated by

$$L_{(k)}|3 = \Delta(t_r, d'_{M(k)})|_2 = \Delta(t_r, d'_{M(k)})|_1 - \frac{L_{(k)}|_2}{T_j}C_j$$
$$= \Delta(t_r, d'_{M(k)}) - \frac{L_{(k)}|_2}{T_j}C_j$$
$$= \Delta(t_r, d'_{M(k)}) - \lceil \frac{\Delta(t_r, d'_{M(k)})|_1 - C_j}{C_j} \rceil C_j. \quad (11)$$

If d'''_{jy} becomes equal to or greater than $d'_{M(k+1)}$ due to the third step delay, then d'''_{jy} will be checked when we do the Δ check with the next time interval $[d'_{M(k+1)}, d'_{M(k+2)})$, or some interval after that. Even this happens, $L_{(k)}|_3$ must not be less than the value given by (11) according to Theorem 1. Therefore, (11) is correct in any case when we do the Δ check with $d'_{M(k)}$.

In addition, if $d'_{M(k)}$ equals d_{jx}, then we get $L_{(k)}|_1 = 0$ from (9) and the first step delay is not needed. Also, if $\Delta(t_r, d'_{M(k)}) \le C_j$, then we have $L_{(k)}|_2 = 0$ from (10) and the second step delay is omitted.

Adding (9), (10), and (11) in (8), we get (6) in Theorem 3.

Proof of Case 2: In this case, since $\Delta(t_r, d'_{M(k)}) \le 0$, thus only the deadline point $d_{j(k)(0)}$ should be checked in the region $[d'_{M(k)}, d'_{M(k+1)})$.

If $\Delta(t_r, d_{j(k)(0)}) \le 0$, then the delay of r_j is not required and $L_{(k)} = 0$.

If $\Delta(t_r, d_{j(k)(0)}) > 0$, r_j should be delayed. Suppose $d_{j(k)(0)}$ shifts to $d'_{j(k)(0)}$ after the delay and $d'_{j(k)(0)} \le d'_{M(k+1)}$. Note that $D_{total}(t_r, d'_{j(k)(0)})$ is the total demand in $(t_r, d'_{j(k)(0)}]$ after the delay. Accordingly, we have the Δ check value $\Delta(t_r, d'_{j(k)(0)})$. Then

$$\Delta(t_r, d'_{j(k)(0)}) = D_{total}(t_r, d'_{j(k)(0)}) - (d'_{j(k)(0)} - t_r)$$
$$= D_{total}(t_r, d'_{j(k)(0)}) - (d_{j(k)(0)} - t_r)$$
$$- (d'_{j(k)(0)} - d_{j(k)(0)})$$
$$= D_{total}(t_r, d'_{j(k)(0)}) - (d_{j(k)(0)} - t_r) - L_{(k)}.$$

If $d'_{j(k)(0)} < d'_{M(k+1)}$, then the total demand of the system does not increase after the delay, that is,

$$D_{total}(t_r, d'_{j(k)(0)}) = D_{total}(t_r, d_{j(k)(0)}).$$

Thus we have

$$\Delta(t_r, d'_{j(k)(0)}) = \Delta(t_r, d_{j(k)(0)}) - L_{(k)}.$$

Since we need $\Delta(t_r, d'_{j(k)(0)}) = 0$ to make the Δ check at $d'_{j(k)(0)}$ *just* passed, therefore,

$$L_{(k)} = \Delta(t_r, d_{j(k)(0)}). \quad (12)$$

If $d'_{j(k)(0)} \ge d'_{M(k+1)}$ after the delay, then $d'_{j(k)(0)}$ will be checked when we do the Δ check with $d'_{M(k+1)}$, or deadline points after $d'_{M(k+1)}$. Even this happens, $L_{(k)}$ must not be less than the value given by (12) according to Theorem 1. Therefore, (12) is correct anyhow when we do the Δ check with $d_{j(k)(0)}$.

APPENDIX D
PROOF OF LEMMA 1

Proof: Start with $r_j = t_r$. let $L_{(k)first}$ be the delay of r_j to make the deadline point $d'_{M(k)}$ to pass its Δ check if this point is checked before all other deadline points. The maximum value of all these delays is denoted as $L_{(k)first_max}$, that is,

$$L_{(k)first_max} = max\{L_{(0)first}, L_{(1)first}, \cdots, L_{(n-1)first}\}.$$

Obviously, $\delta_{earliest}$ depends on $L_{(k)first_max}$ only. That is to say, the total amount of delay of r_j must be equal to $L_{(k)first_max}$ to make the Δ checks of all the points passed, no matter which point is checked first.

REFERENCES

[1] C. L. Liu and J. W. Laylan, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 40–61, 1973.
[2] G. C. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni, "Elastic scheduling for flexible workload management," *IEEE Trans. Comput.*, vol. 51, no. 3, pp. 289–302, Mar. 2002.
[3] Q. Guangming, "An earlier time for inserting and/or accelerating tasks," *Real-Time Syst.*, vol. 41, no. 3, pp. 181–194, Apr. 2009.
[4] N. Fisher and M. Ahmed, "Tractable real-time schedulability analysis for mode changes under temporal isolation," in *Proc. 9th IEEE Symp. Embedded Syst. Real-Time Multimedia*, Oct. 2011, pp. 130–139.
[5] C.-S. Shih and C.-M. Yang, "Schedulability analysis of mode change for imprecise computation on multi-core platforms," in *Proc. Int. Conf. Res. Adapt. Convergent Syst.*, Sep. 2017, pp. 261–268.
[6] M. Zimmerling, L. Mottola, P. Kumar, F. Ferrari, and L. Thiele, "Adaptive real-time communication for wireless cyber-physical systems," *ACM Trans. Cyber-Phys. Syst.*, vol. 1, no. 2, pp. 1–29, Feb. 2017.
[7] A. Burns, "System Mode Changes - General and Criticality–BasedSystem mode changes-general and criticality-based," in *Proc. IEEE Real-Time Syst. Symp.*, Dec. 2014, pp. 3–8.
[8] Q. Guangming and S. Shen, "Method to evaluate earliest release time of new real-time tasks," *East China Inst. Comput. Technology Shanghai Comput. Soc. Comput. Eng.*, vol. 44, no. 12, pp. 62–67, 2018. [Online]. Available: http://www.ecice06.com/CN/10.19678/j.issn.1000-3428.0050021
[9] Q. Guangming and L. Liwen, "Research on transient process for bandwidth transfer of real-time multitask," *East China Inst. Comput. Technology Shanghai Comput. Soc. Comput. Eng.*, vol. 43, no. 12, pp. 292–302,2017. [Online]. Available: http://www.ecice06.com/CN/10.3969/j.issn.1000-3428.2017.12.052
[10] S. K. Baruah, L. E. Rosier, and R. R. Howell, "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor," *Real-Time Syst.*, vol. 2, no. 4, pp. 301–324, Nov. 1990.
[11] K. Jeffay and D. L. Stone, "Accounting for interrupt handling costs in dynamic priority task systems," in *Proc. 14th Real-Time Syst. Symp.*, Raleigh Durham, NC, USA, Dec. 1993, pp. 212–221.
[12] B. Andersson, "Uniprocessor EDF scheduling with mode change," in *Proc. 12th Int. Conf. Princ. Distrib. Syst.*, Berlin, Germany: Springer, Dec. 2008, pp. 572–577.
[13] J. Real and A. Crespo, "Mode change protocols for real-time systems: A survey and a new proposal," *Real-Time Syst.*, vol. 26, no. 2, pp. 161–197, 2004.
[14] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham, "Mode change protocols for priority-driven preemptive scheduling," *Real-Time Syst.*, vol. 1, no. 3, pp. 243–264, Dec. 1989.
[15] L. T. X. Phan, I. Lee, and O. Sokolsky, "A semantic framework for mode change protocols," in *Proc. 17th IEEE Real-Time Embedded Technol. Appl. Symp.*, Apr. 2011, pp. 91–100.
[16] N. Stoimenov, S. Perathoner, and L. Thiele, "Reliable mode changes in real-time systems with fixed priority or EDF scheduling," in *Proc. Design, Autom. Test Eur. Conf. Exhib.*, Nice, France, Apr. 2009, pp. 99–104.

[17] P. Pedro and A. Burns, ''Schedulability analysis for mode changes in flexible real-time systems,'' in *Proc. 10th EUROMICRO Workshop Real-Time Syst.*, Jun. 1998, pp. 172–179.

[18] P. Pillai and K. G. Shin, ''Real-time dynamic voltage scaling for low-power embedded operating systems,'' in *Proc. 18th ACM Symp. Operating Syst. Princ.*, Banff, AB, Canada, Aug. 2001, pp. 89–102.

[19] D. Casini, A. Biondi, and G. Buttazzo, ''Handling transients of dynamic real-time workload under EDF scheduling,'' *IEEE Trans. Comput.*, vol. 68, no. 6, pp. 820–835, Jun. 2019.

[20] A. Block and J. H. Anderson, ''Accuracy versus migration overhead in real-time multiprocessor reweighting algorithms,'' in *Proc. 12th Int. Conf. Parallel Distrib. Syst. (ICPADS)*, 2006, pp. 355–364.

[21] G. C. Buttazzo, G. Lipari, and L. Abeni, ''Elastic task model for adaptive rate control,'' in *Proc. 19th IEEE Real-Time Syst. Symp.*, Madrid, Spain, Dec. 1998, pp. 286–295.

GUANGMING QIAN was born in 1963. He is currently a Professor with the College of Information Science and Engineering, Hunan Normal University, Changsha, China. His research interests include real time systems, embedded technology, and wireless networks.

• • •