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: qgqy@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...
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 \(\delta\). Obviously,
finding the earliest smooth insertion time, denoted by \(\delta_{\text{earliest}}\),
is significant and challenging.

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

\[
\delta_{\text{earliest}} = \delta_{\text{earliest}} + \text{timestep}.
\] (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 \(\delta_{\text{earliest}}\) 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 \(\delta_{\text{earliest}}\) 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 \(\delta_{\text{earliest}}\)
value in every test.

**Paper Structure:** Section II covers the review of the
descriptions of tasks’ compression. A delaying rule is intro-
duced and an example is provided. Section III presents
Theorem 2 and 3, based on which ESITforSNT is designed for
calculating \(\delta_{\text{earliest}}\). Section IV shows the novel experimental
approach and the simulation results. Section V discusses the
related work and Section VI concludes the paper.
• The computation time of every instance of a task remains unchanged.
• Every old task is released at \( t_{oldb} \) and runs to \( d'_{\text{max}} \) (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.

\[ \forall t \geq t_r, \text{ the processor demand of task } \tau_i(C_i, T_i) \text{ in } [t_r, t] \text{ is labelled with } D_j(t_r, t). \]

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 \textit{Delta check}. \( \Delta(t_r, t) \) is the value of the \textit{Delta check}.

The processor demand of new tasks can be calculated with

\[ D_J(t_r, t) = \sum_{r_j \in J} \left[ \frac{t - r_j}{T_j} \right] C_j. \]  

The processor demand of the compressed task \( \tau_i \) should be computed with

\[ D_j(t_r, t) = \begin{cases} 0, & \text{if } t < t_i + T'_i. \\ c_i(t_r) + \left[ \frac{t - t_i - T'_i}{T'_i} \right] C_i, & \text{if } t \geq t_i + T'_i. \end{cases} \]

From \( t_r \) to \( t_i + T'_i \), the deadline point \( t_i + T'_i \) of \( t_i \) is met only if \( t_i \) 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 \([d'_{\text{min}}, d'_{\text{max}}]\), then the insertion should be delayed and we must have

\[ \delta_{\text{earliest}} \geq r_j + \Delta(t_r, t_x). \]

That is, the delayed insertion becomes possibly smooth only when the release time is delayed by not less than the value of the \textit{Delta 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 \textit{deadline point}. A \textit{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 \textit{new task deadline point}. If some deadlines of different tasks take an identical time value, then this time point is called an \textit{overlapped deadline point}.

**Definition 2 (Delaying Rule):** According to Theorem 1, if the \( \Delta \) check value \( \Delta(t_r, t_x) \) is greater than zero at a deadline point \( t_x \), then \( \delta_{\text{earliest}} \) 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 \textit{Delta 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 re-check until \( \Delta(t_r, t_x) \leq 0 \) is defined as the \textit{delaying rule}. Also, it is declared that \( t_x \) \textit{passes its \( \Delta \) check} as soon as \( \Delta(t_r, t_x) \leq 0 \) becomes true.

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

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 fewer checks are required for \( \delta_{\text{earliest}} \). 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 \( \delta_{\text{earliest}} \). 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 \( t_r = 8 \), \( T_1 \) keeps unchanged, but the period of \( t_0 \) 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 \( d_{\text{min}} = T'_1 = 16 \) and \( d'_{\text{max}} = T_0 = 32 \).

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

\[ \Delta(t_r, 16) = D_{total}(t_r, 16) - (16 - t_r) \]

\[ = D_{total}(8, 16) - 8 \]

\[ = D_0(8, 16) + D_1(8, 16) + D_J(8, 16) - 8 \]

\[ = 0 + c_1(t_r) + \left[ \frac{16 - r_j}{T_j} \right] C_j - 8 = 8 + 2 - 8 = 2. \]

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 \( \Delta \) check again. Unfortunately, \( d'_{\text{min}} \) does not pass its second \( \Delta \) check due to

\[ \Delta(t_r, 16) = 0 + c_1(t_r) + \left[ \frac{16 - 10}{T_j} \right] C_j - 8 \]

\[ = 0 + 8 + 1 - 8 = 1. \]
checks are as follows:

An example using the delaying rule.

![Diagram](https://via.placeholder.com/150)

FIGURE 3. An example using the delaying rule.

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

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

That is to say, five $\Delta$ checks in total have to be done at $d'_{M(1)}$. 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 $\Delta$ check for the first time, thus $r_j$ will not be delayed further. Therefore, nine $\Delta$ checks are needed to get $\delta_{earliest} = 13$ totally. Using the new algorithm $ESITforSNT$ presented in the next section, however, only one $\Delta$ check is enough to reach $\delta_{earliest} = 13$.

III. NEW THEOREMS AND ESITforSNT ALGORITHM

Although the number of the times of the required $\Delta$ checks may be reduced by Theorem 1, a deadline point in $[d'_{M(0)}, d'_{M(1)}]$ 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), $d'_{M(0)} = 16$, $d'_{M(1)} = 32$, and there is only one $M$ task deadline point in $[d'_{M(0)}, d'_{M(1)}]$. 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 $[d'_{M(0)}, d'_{M(1)}]$, 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 $\tau_j$ is labeled by $r_j$. In $[d'_{M(0)}, d'_{M(1)}]$, suppose that the set $M$ has $n$ deadline points: $d'_{M(0)} < d'_{M(1)} < \cdots < d'_{M(n-1)}$. 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 $d'_{M(n)} = d'_{max}$. For $k \in (0, 1, \ldots, n - 1)$, in $(d'_{M(k)}, d'_{M(k+1)})$, assume that the first deadline point (if any) of $\tau_j$ is $d'_{M(k)}$.

Then, if $d'_{M(k)}$ and $d'_{M(k+1)}$ pass their $\Delta$ checks, other deadline points (if any) in $(d'_{M(k)}, d'_{M(k+1)})$ will also pass their $\Delta$ 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 $\Delta$ 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 $[d'_{M(0)}, d'_{M(1)}]$ 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 $\tau_j$ 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 $\tau_j$, then the deadline points $d'_{M(k)}$ and $d'_{M(k+1)}$ pass their $\Delta$ 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

$$L_k = d'_{M(k)} - (r_j + \left\lfloor \frac{d'_{M(k)} - r_j}{T_j} \right\rfloor T_j) + \Delta(t_r, d'_{M(k)})$$

$$\quad + \frac{\Delta(t_r, d'_{M(k)}) - C_j}{C_j} (T_j - C_j).$$  

(6)
Case 2. If \( \Delta(t_r, d'_{M(k)}) \leq 0 \), then
\[
L(k) = \begin{cases} 
0, & \text{if } \Delta(t_r, d'_{M(k)}) \leq 0. \\
\Delta(t_r, d'_{M(k)}), & \text{if } \Delta(t_r, d'_{M(k)}) > 0. 
\end{cases} 
\] (7)

Proof: See Appendix C.

The CASE I of Theorem 3 means that if \( d'_{M(k)} \) does not pass its \( \Delta \) 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 \( d'_{M(k)} \) any more.

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

Now we use Theorem 3 to discuss the example of FIGURE 3. First let \( r_j = t_r = 8 \). Because the \( \Delta \) check value \( \Delta(t_r, d'_{M(0)}) = \Delta(8, 16) > 2 > 0 \), by using (6) we have
\[
L(0) = d'_{M(0)} - (r_j + \frac{d'_{M(0)} - r_j}{T_j})T_j + \Delta(t_r, d'_{M(0)})
\]
\[
= 16 - (8 + \frac{16 - 8}{4})4 + 2 + \frac{2 - 1}{1}(4 - 1) = 5.
\]

Thus we get
\[
\delta_{earliest} = r_j + L(0) = 8 + 5 = 13.
\]

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

In Theorem 2, when multiple \( M \) task deadlines exist in the region \( [d'_{min}, d'_{max}] \), it is assumed that they are sorted from \( d'_{M(0)} \) to \( d'_{M(l-1)} \) 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., \( d'_{M(0)}, d'_{M(1)}, \ldots, d'_{M(l-1)} \), \( \delta_{earliest} \) is not influenced by the priority order of their \( \Delta \) 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 \( t_0 \) are checked: \( t_0 + T'_0, t_0 + 2T'_0, \ldots, t_0 + pT'_0 \). Here, \( p \) is a positive integer and \( t_0 + pT'_0 < d'_{max} \). Consequently, \( \tau_1, \tau_2, \ldots, \) and \( \tau_{m-1} \) will be checked. If the value of any \( \Delta \) 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 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 \( \Delta \) 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. \( \forall t_i(C_i, T_i) \in M, i \in (0, 1, \ldots, m - 1), d'_{M[i]} \) is set to \( t_i + T'_i \). In the initialization, an initial constant \( LSEG \) 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 \( t_{segend} + LSEG - d'_{max} > 0 \), then \( L_{seg} = d'_{max} - t_{segend}; \) otherwise \( L_{seg} = LSEG \). 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 \( d'_{max} \).

Algorithm 1 ESITforSNT

The main function:
Input: \( m, t_r, C_i(t_r), t_t, t_T', C_i, C_j, t_T \).
1: Find out \( d'_{min} \) and \( d'_{max} \)
2: Initialize \( r_j, d'_{M}, L_{seg}, t_{segend}, mark, \) and flag[ ]
3: \( while (t_{segend} - \phi_{seg} < 0) do \)
4: Create a segment \( (L_{seg}, t_{segend}, t_{segstart}) \)
5: mark ++
6: \( for \ (i = 0; \ i < m; \ i++) \ do \)
7: \( while (r_j < d'_{i}[i] < t_{segend}) do \)
8: \( pos_{rel} = d'_{M[i]} - t_{segstart} \)
9: \( if (flag[pos_{rel}] - mark \neq 0) then \)
10: \( r_j = \text{deadlinecheck}(d'_{M[i]}) \)
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 \( r_j \)
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 $\Delta$ 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 $\Delta$ checks are needed: first $d'_{M} [k]$ should be checked and the next is $d'[k][0]$ 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 $\Delta$ checks. Then, how many $\Delta$ 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 worstcase time complexity of this algorithm is equal to $2n$ times of $\Delta$ 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, $t_{old} = 0$.

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: $m$, $t_r$, $c_i(t_r)$, $t_1$, $T_i$, $C_i$, $C_j$, $T_j$.

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 $t_j$ 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 |
|----------|-------|-------|----------------|----------|
| $\tau_0$ | $50 \sim 250$ | $\approx 0.195$ | $0$ | $\tau_j(3, 5)$ |
| $\tau_1$ | $120$ | $0.4$ | $0.75$ | $0$ |
| $\tau_2$ | $180$ | $0.4$ | $0.75$ | $0$ |
| $\tau_3$ | $T_{LCM(0 \sim m-1)} \approx 0.005$ | $0$ | $\tau_j(3, 5)$ |

Definition 3 (Freed Bandwidth Ratio): The Freed bandwidth ratio from a $M$ task $\tau_1(C_i, T_i)$ is defined as $(U_i - U_1(i))/U_1$, 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: $\tau_0$, $\tau_1$, $\tau_2$ and $\tau_3$. 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**, $\tau_0$ and $\tau_3$ 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:

$$U_j = P_{freed(0)}U_0 + P_{freed(1)}U_1 + P_{freed(2)}U_2 + P_{freed(3)}U_3 = 0 \times U_0 + 0.75 \times U_1 + 0.75 \times U_2 + 0 \times U_3 = 0.6$$

The period $T_0$ of $\tau_0$ takes 201 integer values from 50 to 250. Accordingly, $C_0$ has 201 integer values.

For $\tau_1$, 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:

$$C_0 = \lfloor 0.195 \times T_0 \rfloor$$

$$C_3 = (1 - (\frac{C_0}{T_0} + \frac{C_1}{T_1} + \frac{C_2}{T_2})) \times T_3$$

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:
The bandwidth of $\tau_3$ is very small (about 0.005). It is used together with $\tau_0$ 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 ESIT for SNT can be verified under the condition of exactly 100% load. In addition, we set $T_3 = T_{\text{LCM}(0-2)}$ to produce $T_{\text{LCM}(0-3)} = T_3$, 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 $T_{\text{LCM}(0-3)} - 1$) is controllable and not too large.

- $t_1$ and $t_2$ are configured with relatively large freed bandwidth ratios. Both $p_{\text{freed}(1)}$ and $p_{\text{freed}(2)}$ are 0.75. The reason for this is that a transfer with large bandwidth is easy to cause $\delta_{\text{earliest}} > t_r$. This is good for the verification. With an unreasonable configuration, on the contrary, there may be fewer or even no cases with $\delta_{\text{earliest}} > t_r$.

- A relatively short period, $T_j = 5$, is assigned to the new task $t_j$. With a constant $U_j$, the shorter the $T_j$ is, the greater processor demand the process 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.

- $t_0$ 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, $T_1 \leq T_0 \leq T_2$. 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 $\delta_{\text{earliest}} = t_r$, 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:

| index | task $\tau_1$ | task $\tau_2$ | task $\tau_1$ and $\tau_2$ |
|-------|---------------|---------------|---------------------------|
| 1     | $\tau_0(9,50)$ | $\tau_3(36,1800)$ | Before compression: $\tau_1(48,1200)$ $\tau_2(72,1800)$ |
| 2     | $\tau_0(9,31)$ | $\tau_3(144,6120)$ | $\tau_1(90,3120)$ $\tau_2(144,6120)$ |
| 3     | $\tau_0(10,52)$ | $\tau_3(35,4800)$ | $\tau_1(120,3520)$ $\tau_2(144,6120)$ |
| 4     | $\tau_0(10,53)$ | $\tau_3(216,19080)$ | $\tau_1(216,19080)$ $\tau_2(216,19080)$ |
| ...   | ...           | ...           | ...                       |
| 200   | $\tau_0(48,249)$ | $\tau_3(216,29880)$ | $\tau_1(216,29880)$ $\tau_2(216,29880)$ |
| 201   | $\tau_0(48,250)$ | $\tau_3(72,9000)$ | $\tau_1(72,9000)$ $\tau_2(72,9000)$ |

Three examples in region 2:

- $T_0 = 181$, $t_r = 1$, $\delta_{\text{earliest}} = 1$.
- $T_0 = 200$, $t_r = 117$, $\delta_{\text{earliest}} = 126$ (Case 1).
- $T_0 = 200$, $t_r = 1906$, $\delta_{\text{earliest}} = 1907$ (Case 2).

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

Different task configurations from TABLE 1 are also used to test ESIT for SNT. 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 $\text{offset} = \delta_{\text{earliest}} - t_r$.

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 $\tau_i$ 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 $\tau_i$ 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 $\delta = \delta_{\text{earliest}}$.

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

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 $\delta_{\text{earliest}}$ 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.

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 $[d'_{\text{min}}, d'_{\text{max}}]$, which is proved in [8] and [9].
- To get $\delta_{\text{earliest}}$ for inserting a new task with ESITforSNT algorithm, only one $\Delta$ check is needed for each $M$ task deadline point. Totally suppose $M$ tasks have $n$ deadline points in $[d'_{\text{min}}, d'_{\text{max}}]$, then the number of the times of the $\Delta$ checks required for $\delta_{\text{earliest}}$ 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 $\delta_{\text{earliest}}$ 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 $\tau_j$ should be checked. $\tau_j$ may have more than one deadline point in this interval. Suppose $d'_j(k+1)$ is the second deadline point of $\tau_j$. If the $\Delta$ check of $\tau_j$ at the first deadline point $d'_j(k)0$ is passed, then

$$\Delta(t_r, d'_j(k)0) = D_{\text{total}}(t_r, d'_j(k)0) - (d'_j(k)0 - t_r) = D_M(t_r, d'_j(k)0) + D_j(t_r, d'_j(k)0) - (d'_j(k)0 - t_r) \leq 0.$$ 

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

$$\Delta(t_r, d'_j(k+1)) = D_{\text{total}}(t_r, d'_j(k+1)) - (d'_j(k+1) - t_r) = D_M(t_r, d'_j(k+1)) + D_j(t_r, d'_j(k+1)) - T_j - (d'_j(k)0 - t_r) \leq C_j - T_j < 0.$$
 TABLE 3. The Symbols.

| Symbol | Description |
|--------|-------------|
| \( M \) | Set of old tasks |
| \( n \) | Total number of old tasks |
| \( J \) | Set of new tasks |
| \( t_{\text{old}} \) | Start of old mode |
| \( t_{\text{new}} \) | Start of new mode |
| \( t_{\text{rf}} \) | Release time of new tasks |
| \( t_{\text{r}} \) | Release time of single new task \( \tau_j \) |
| \( t_{\text{C},j} \) | New task \( \tau_j \) with computation \( C_j \) and period \( T_j \) (if any) \( \tau_j \in J \) |
| \( U_j \) | Utilization of \( \tau_j \) |
| \( C_i(t_r) \) | Remaining computation of the instance of \( \tau_i \) at \( t_r \) |
| \( r_{\text{C}} \) | Old task \( \tau_i \) with computation \( C_i \) and period \( T_i \), \( t^r \in (0,1, \ldots, m-1), \tau_i \in M \) |
| \( \delta \) | Smooth release time of new tasks |
| \( \delta_{\text{earliest}} \) | Earliest \( \delta \) |
| \( \tau_{\text{M}} \) | \( \tau \) in \( \{ \tau_{\text{M}}, \tau_{\text{max}} \} \) |
| \( \tau_{\text{max}} \) | \( \tau \) in \( \{ \tau_{\text{M}}, \tau_{\text{max}} \} \) |
| \( \tau_{\text{min}} \) | \( \tau \) in \( \{ \tau_{\text{M}}, \tau_{\text{max}} \} \) |
| \( d_j(t_{\text{M}}) \) | First deadline of \( \tau_j \) in \( \{ d_{j,M}(t_{\text{M}}), d_{j,M+1}(t_{\text{M}}) \} \) |
| \( D_j(t_r) \) | Delay of \( \tau_j \) |
| \( L_{\text{MC}}(0-m-1) \) | Least common multiple period of \( M \) before compression |
| \( D_j(x) \) | Processor demand of task \( \tau_j \) in \( [t,x], x \in (t,j) \) |
| \( D_j(x) \) | Sum of processor demands of \( M \) tasks in \( [t,x] \) |
| \( D_j(x) \) | Sum of processor demands of \( M \) tasks in \( [t,x] \) |
| \( D_j(x) \) | Sum of processor demands of \( M \) tasks in \( [t,x] \) |
| \( D_j(x) \) | Sum of processor demands of \( M \) tasks in \( [t,x] \) |
| \( L_j(x) \) | Latest deadline point in \( [d_{j,M}(t), d_{j,M+1}(t)] \) |
| \( r_{\text{free}}(i) \) | Free bandwidth ratio from task \( \tau_i \) |

This indicates that \( d_{j,M}(t) \) also pass its \( \Delta \) check. Similarly, other deadline points of \( \tau_j \) after \( d_{j,M}(t) \) (if any) in the region \( (d_{j,M}(t), d_{j,M+1}(t)) \) will also pass their \( \Delta \) 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 \( \Delta(t_r, d_{j,M}(t_r)) < 0 \), \( r_j \) should be delayed by \( L_{j,M}(t) \). The delaying will of course decrease the processor demand \( D_j(t_r, d_{j,M}(t_r)) \).

If \( \Delta(t_r, d_{j,M}(t_r)) > C_j \), \( L_{j,M}(t) \) must be longer than the length of \( T_j \) to make \( D_j(t_r, d_{j,M}(t_r)) \) reduced enough for \( \Delta(t_r, d_{j,M}(t_r)) \leq 0 \). 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 \( L_{j,M}(t) \) for \( t \) after the first, the second, and the third step, respectively. Correspondingly, their \( \Delta \) check values are \( \Delta(t_r, t) \), \( \Delta(t_r, t_2) \), and \( \Delta(t_r, t_3) \). Naturally

\[
L_{j,M}(t) = L_{j,M}(t_1) + L_{j,M}(t_2) + L_{j,M}(t_3)
\]

For the purpose of convenience, with a discussed time \( t \), \( D_{\text{total}}(t_r, t_1) \), \( D_{\text{total}}(t_r, t_2) \), and \( D_{\text{total}}(t_r, t_3) \) are used to represent the total processor demands in \( (t_r, t) \) after the first, the second, and the third step, respectively. Correspondingly, their \( \Delta \) check values are \( \Delta(t_r, t_1) \), \( \Delta(t_r, t_2) \), and \( \Delta(t_r, t_3) \).

Let \( d_{j,M}(t) \) be the latest deadline point (if any) of \( \tau_j \) in \( (t_r, d_{j,M}(t)) \), that is, \( 0 \leq d_{j,M}(t) < T_j \). \( d_{j,M}(t) \) represents a special deadline point (if any) of \( \tau_j \) before \( d_{j,M}(t) \), as in FIGURE 4.

Before delaying, we have

\[
\Delta(t_r, d_{j,M}(t)) = D_{\text{total}}(t_r, d_{j,M}(t)) - d_{j,M}(t) > 0.
\]

After the first step delay, the release of \( \tau_j \) is delayed by \( L_{j,M}(t_1) \), so that \( d_{j,M}(t_2) > d_{j,M}(t) \). Then

\[
L_{j,M}(t_1) = d_{j,M}(t) - d_{j,M}(t_2) = d_{j,M}(t) - r_j
\]

\[
= d_{j,M}(t) - (r_j + \frac{d_{j,M}(t) - r_j}{T_j}) T_j
\]

Notice that the total demand \( D_{\text{total}}(t_r, d_{j,M}(t)) \) remains unchanged when the first step delay is completed, thus we have

\[
D_{\text{total}}(t_r, d_{j,M}(t)) = D_{\text{total}}(t_r, d_{j,M}(t)) - d_{j,M}(t) > 0.
\]

Through the second step delay, \( r_j \) is increased by \( L_{j,M}(t_2) \), and \( d_{j,M}(t) \) will move to \( d_{j,M}(t_2) \) to produce

\[
L_{j,M}(t_2) = \frac{\Delta(t_r, d_{j,M}(t)) - C_j}{C_j} T_j
\]

The purpose of this step is to make the \( \Delta \) check value be less than or equal to \( C_j \):

\[
\Delta(t_r, d_{j,M}(t)) \leq \Delta(t_r, d_{j,M}(t)) - \frac{L_{j,M}(t_2)}{T_j} C_j
\]

\[
\leq \Delta(t_r, d_{j,M}(t)) - \frac{L_{j,M}(t_2)}{C_j} C_j
\]

\[
= C_j.
\]

FIGURE 4. Using the delaying rule with Case 1.
Now discuss the third step delay. The deadline $d''_j$ will shift to $d''_{j''}$. The $Δ$ check with $d''_{M(k)}$ becomes passed as soon as $L_k |_3 > 0$ since $Δ(t_r, d''_{M(k)}) |_2 ≤ C_j$. However, $L_k |_3 > 0$ may not be enough for $d''_{j''}$ to pass its $Δ$ check. According to the delaying rule, in order to make both $d''_{M(k)}$ and $d''_{j''}$ passed, the minimum value of $L_k |_3$ should be calculated by

$$L_k |_3 = \Delta(t_r, d''_{M(k)}) |_2 = \Delta(t_r, d''_{M(k)}) |_1 - \frac{L_k |_3}{T_j} C_j$$

$$= \Delta(t_r, d''_{M(k)}) - \frac{L_k |_3}{T_j} C_j$$

$$= \Delta(t_r, d''_{M(k)}) - \left[ \frac{\Delta(t_r, d''_{M(k)}) |_1 - C_j}{C_j} \right] C_j.$$ (11)

If $d''_{j''}$ becomes equal to or greater than $d''_{M(k+1)}$ due to the third step delay, then $d''_{j''}$ 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 $d_j$, then we get $L_k |_1 = 0$ from (9) and the first step delay is not needed. Also, if $\Delta(t_r, d''_{M(k)}) |_1 ≤ C_j$, 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, $\Delta(t_r, d''_{M(k)}) |_1 ≤ 0$, thus only the deadline point $d''_{j(k)(0)}$ should be checked in the region $[d''_{M(k)}, d''_{M(k+1)}]$.

If $\Delta(t_r, d''_{j(k)(0)}) |_1 ≤ 0$, 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 that $d''_{j(k)(0)}$ shifts to $d''_{j(k)(0)}$ after the delay and $d''_{j(k)(0)} - d''_{M(k+1)}$.

Note that $D_{total}(t_r, d''_{j(k)(0)})$ is the total demand in $(r_j, d''_{j(k)(0)})$ after the delay. Accordingly, we have the $Δ$ check value $\Delta(t_r, d''_{j(k)(0)})$. Then

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

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

$$= D_{total}(t_r, d''_{j(k)(0)}) - (d''_{j(k)(0)} - t_r) - L_k.$$ (12)

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

$$\Delta(t_r, d''_{j(k)(0)}) = \Delta(t_r, d''_{j(k)(0)}) - L_k.$$ (12)

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

$$L_k = \Delta(t_r, d''_{j(k)(0)}).$$ (12)

If $d''_{j(k)(0)} ≥ d''_{M(k+1)}$ 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 $L_k |_1$ 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 $L_k |_1$. Therefore, $L_k |_1 = \max\{L_0 |_1, L_1 |_1, \cdots, L_{(n−1)} |_1\}$

Obviously, $\delta_{earliest}$ depends on $L_k |_1$. That is to say, the total amount of delay of $r_j$ must be equal to $L_k |_1$ 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. Layland, “Scheduling algorithms for multiprogramming in a hard-real-time environment,” J. ACM, vol. 20, no. 1, pp. 46–61, 1973.

[2] G. C. Buttazzo, G. Lipari, M. Caccamo, and L. Benini, “Elastic scheduling for flexible workload management,” IEEE Trans. Comput., vol. 51, no. 3, pp. 289–302, Mar. 2002.

[3] Q. Gongmuang, “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-Based System mode changes-general and criticality-based,” in Proc. IEEE Real-Time Syst. Symp., Dec. 2014, pp. 3–8.

[8] Q. Gongmuang and S. Shen, “Method to evaluate earliest release time of new real-time tasks,” East China Inst. Comput. Technol. Shangai Comput. Soc. Comput. Eng., vol. 44, no. 12, pp. 62–67, 2018. [Online]. Available: http://www.ecice06.com/CN/10.19678/j.fjiss.1000-3428.0050021

[9] Q. Gongmuang and L. Liwen, “Research on transient process for bandwidth transfer of real-time multitask,” East China Inst. Comput. Technol. Shangai 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. Burauh, 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. Lehoczyk, 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. Prin., 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.