

Method to Evaluate Earliest Release Time of New Real-time Tasks

QIAN Guangming, SU Shen

(College of Information Science and Engineering, Hunan Normal University, Changsha 410081, China)

[Abstract] In a periodic real-time system scheduled with the Earliest Deadline First (EDF) algorithm, it is necessary to compress some current tasks to avoid overloading if new task requests to run. Compressing a task means that its period is prolonged while its computation time keeps unchanged. An interesting problem is to find the earliest time to release new tasks without any deadline missing, that is, the earliest smooth insertion time. In this paper, a general frame to calculate the earliest time with multiple rounds of deadline checking is given, which shows that the checking can be done from the request time of the new tasks. A smart way is provided and proved, which takes the value of the Δ checking of the current round as the time step to the next. These techniques potentially reduce the amount of the calculation and the number of the rounds of the checking to get the earliest time. Simulation results are also given to support the conclusion.

[Key words] total utilization; new tasks insertion; simple way; smart way; Δ checking

Reference Format: QIAN Guangming, SU Shen. Method to evaluate earliest release time of new real-time tasks[J]. Computer Engineering, 2018, 44(12): 62-67.

DOI: 10.19678/j.issn.1000-3428.0050021

0 Summary

Many real-time systems often schedule tasks adaptively with the load^[1-2]. In this paper, the load refers to the sum of the utilizations (or bandwidths) of all the tasks in the system, that is, the total utilization.

A light load system may operate at low frequency to save energy, or even sleep^[3-4]. As the load becomes heavier, the operating frequency should probably be increased to meet the timing constraints of the tasks in the system. Consider a heavy load system scheduled with the Earliest Deadline First (EDF) algorithm^[5]. If some new bandwidth requirements arrive, then the total utilization may be greater than one and the system will become overloaded and unschedulable even if the working frequency is increased. A possible way to avoid overloading is to compress some current tasks, that is, to decrease their utilizations^[1].

The new bandwidth requirements may be some current tasks' acceleration and/or new tasks' insertion into the current task set^[6]. Some application scenarios are described in References [1, 7], such as the scheduling for robot target approaching. A task's acceleration means that its period is decreased while

the computation time keeps unchanged. It is proved that the acceleration of a current task can be treated as the insertion of an equivalent new task^[6]. Therefore, as for as new bandwidth requirements, new tasks' insertion (or insertion for short) is discussed only.

The problem of the insertion belongs to a kind of mode-change problem^[8-10]. It has three stages: the old mode starting from t_{oldb} , the transition process from t_r and the new mode from t_{newb} . t_r is the time point at which the request of the insertion occurs and some current tasks may be compressed.

If new tasks are inserted at t_r immediately, it is known that deadline missing may occur even though the total utilization does not exceed one^[1, 11]. In Reference [12], the transition process is reasonably defined, and it is proved that deadline missing is only possible during the transition process $[t_r, t_{newb})$ if the total utilization is not greater than one.

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

A simple way to evaluate $\delta_{earliest}$ is to do multiple

Fund Project: Changsha Municipal Science and Technology Foundation (K15ZD053-43).

Biography: QIAN Guangming (1963—), male, professor, main research direction is application in real-time and embedded systems; SU Shen, master.

Receipt Date: 2018-01-08

Revised Date: 2018-04-16

E-mail: qqyy@hunnu.edu.cn

rounds of deadline checking, named “Simple way” in the following sections. In this way, first this paper assumes $\delta_{\text{earliest}} = t_r$ and checks every deadline from δ_{earliest} to t_{newb} . If deadline missing is not found then it concludes that δ_{earliest} is really equal to t_r and no more checking is required, otherwise, the next round of checking is needed with

$$\delta_{\text{earliest}} = \delta_{\text{earliest}} + t_{\text{step}} \quad (1)$$

The t_{step} in Equation (1) takes one time unit. Similarly, the third round of checking is needed if a missed deadline is asserted from the second round. And so on. Finally, the number of the required rounds to find the real δ_{earliest} depends on the length of the transition process and the number of the deadlines during the transition.

The major contribution of this paper is that a more efficient way to calculate δ_{earliest} is provided and proved, named “Smart way” or “Smart try”. Instead of one time unit in the Simple way, the t_{step} in the Smart way takes the difference of the total processor demand minus the length of the discussed time interval, which is greater than or equal to one time unit.

1 Processor demands after compression

In this section, first the model of tasks' compression provided in References [12-13] is recalled and improved, then several expressions for evaluating the associated processor demands after compression are presented, which are used to judge whether the deadlines in the transition process are satisfied.

1.1 System model

As in Reference [12], the task model $\tau_x(C_x, T_x)$ is used in this paper, where T_x is the period and C_x represents the computation time in each period.

Suppose that a periodic real-time task set M has m tasks, $m \geq 1$. As shown in Figure 1, $\forall \tau_i(C_i, T_i) \in M, i \in (0, 1, \dots, (m-1))$, the utilization of τ_i is $U_i = C_i/T_i$ and the starting point of its current period is t_i . At t_r , new tasks are required to be inserted and thus τ_i is compressed, its period increases to T'_i from T_i and utilization decreases to $U'_i = C_i/T'_i$. Apparently, its computation time C_i of the current instance is divided into two parts by t_r : the first part $C_i - c_i(t_r)$ has already been completed with the old deadline $t_i + T_i$, and the second part $c_i(t_r)$ (the remaining computation time) is to be executed with the new deadline $t_i + T'_i$.

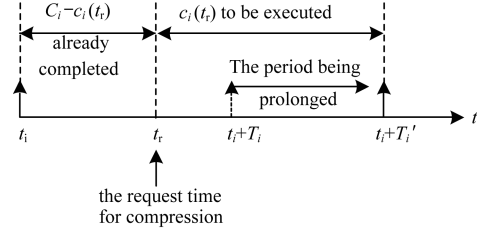


Fig. 1 Compression of the task τ_i

Note that $T'_i \geq T_i$. This model is also reasonable even if there are some tasks not to be compressed. If the period of τ_i is not prolonged then $T'_i = T_i$ and $U_i - U'_i = 0$.

In order to simplify the associated descriptions below, $d'_{\min}$ 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,

$$d'_{\min} = \min \{ (t_0 + T'_0), (t_1 + T'_1), \dots, (t_{m-1} + T'_{m-1}) \}$$

$$d'_{\max} = \max \{ (t_0 + T'_0), (t_1 + T'_1), \dots, (t_{m-1} + T'_{m-1}) \}$$

The task set M is compressed for new tasks' insertion. In the following descriptions, it is supposed that all the new tasks form a subset J and they are inserted into the system at the same time, the sum of the utilizations of these tasks is U_J and r_J is introduced to represent the release time of their first instances. Obviously, $r_J \geq t_r$. In order to keep the system schedulable in the new mode, U_J is not greater than the freed utilization from M , that is,

$$U_J \leq \sum_{i=0}^{m-1} (U_i - U'_i) \quad (2)$$

Up to now, it is clear that our system has the task set M before compression. It contains M (compressed) and the subset J after compression. The running of the system in $[t_r, d'_{\max})$ is defined as the transition process^[12].

The release point r_J of the first instances of the new tasks may be any time from t_r . The earliest r_J that does not cause any deadline missing is δ_{earliest} . The insertion is called an immediate smooth insertion if $\delta_{\text{earliest}} = t_r$.

1.2 Processor demand calculation

The proof of the theorems below requires the processor demand criterion^[14-15]. Firstly, some associated symbols and notations are given.

In $[t_1, t_2]$, the processor demand of the task $\tau_x(C_x, T_x)$ is labeled with $D_x(t_1, t_2)$. The sum of the processor demands of all the tasks in J and M are indicated with $D_J(t_1, t_2)$ and $D_M(t_1, t_2)$, respectively. The sum of $D_J(t_1, t_2)$ and $D_M(t_1, t_2)$ is called the total processor demand $D_{\text{total}}(t_1, t_2)$.

Then, $\Delta(t_1, t_2) = D_{\text{total}}(t_1, t_2) - (t_2 - t_1)$ is introduced. According to the criterion in Reference [14], the deadlines at t_2 are met if and only if $\Delta(t_1, t_2)$ is less than or equal to zero. Checking whether $\Delta(t_1, t_2)$ is greater than zero is called “ Δ checking” in the following descriptions.

Again see Figure 1. The processor demand of task τ_i in $[t_r, t_i + T'_i]$ is its remaining computation time $c_i(t_r)$. In other words, its deadline $t_i + T'_i$ is met if τ_i is assigned the processor time equal to the amount of $c_i(t_r)$ in this interval. Furthermore, the bandwidth requirement of τ_i from the system is $U'_i = C_i/T'_i$ after $t_i + T'_i$. Therefore, its processor demand from t_r to a later time t can be expressed as

$$D_i(t_r, t) = c_i(t_r), \text{ if } t = t_i + T'_i \quad (3)$$

$$D_i(t_r, t) = c_i(t_r) + \left\lfloor \frac{t - (t_i + T'_i)}{T'_i} \right\rfloor C_i, \text{ if } t > t_i + T'_i \quad (4)$$

The new tasks in the subset J are released at r_j . $\forall \tau_j (C_j, T_j) \in J$, from r_j to any time t , it gets $D_j(t_r, t) = \lfloor (t - r_j)/T_j \rfloor C_j$. Therefore,

$$D_j(t_r, t) = \sum_{j \in J} \left\lfloor \frac{t - r_j}{T_j} \right\rfloor C_j \quad (5)$$

Based on these expressions, Theorem 1 is given to check whether the deadlines are satisfied (thus calculating the earliest smooth insertion time).

Theorem 1 With compressing the task set M , suppose d_x is a deadline during the transition process. Then deadline missing will not occur at $t = d_x$ if and only if

$$\sum_{j \in J} \left\lfloor \frac{d_x - r_j}{T_j} \right\rfloor C_j + \sum_{i=0}^{m-1} D_i(t_r, d_x) \leq (d_x - t_r) \quad (6)$$

$$D_i(t_r, d_x) = \begin{cases} 0, & \text{if } d_x < t_i + T'_i \\ c_i(t_r), & \text{if } d_x = t_i + T'_i \\ c_i(t_r) + \left\lfloor \frac{d_x - t_i - T'_i}{T'_i} \right\rfloor C_i, & \text{if } d_x > t_i + T'_i \end{cases} \quad (7)$$

Proof: In $[t_r, d_x]$ it gets

$$D_{\text{total}}(t_r, d_x) = D_J(t_r, d_x) + D_M(t_r, d_x) = \sum_{j \in J} \left\lfloor \frac{d_x - r_j}{T_j} \right\rfloor C_j + \sum_{i=0}^{m-1} D_i(t_r, d_x)$$

Then Δ checking is calculated

$$\Delta(t_r, d_x) = D_{\text{total}}(t_r, d_x) - (d_x - t_r)$$

The deadline at $t = d_x$ is satisfied if and only if $\Delta(t_r, d_x) \leq 0$. Replacing t with d_x in Equation (3) ~ Equation (5), it is easy to see that Equation (6) and Equation (7) are correct.

Theorem 1 is proved.

2 The Smart way

In order to clearly demonstrate the advantages of the Smart way, it is necessary to pay special attention to the time sequence:

$$t_r \leq d'_{\min} < d'_{\max} \quad (8)$$

To get the earliest smooth insertion time δ_{earliest} with the Simple way, it is assumed that the release time of the new tasks is $r_j = t_r$ and start the first round of checking. Δ checking should be done with all the deadlines before $d'_{\max}$. If no deadline is missed then $\delta_{\text{earliest}} = t_r$ and no more checking is needed. If a missed deadline is found then the first round of checking is ended and the next round with $r_j = r_j + t_{\text{step}}$ is started.

Two features with the Simple way should be noticed: 1) Δ checking should be done with every deadline and 2) the t_{step} is equal to one.

Comparatively, based on the following Theorem 2 and Theorem 3, there are two advantages with the Smart way: Δ checking is only needed with the deadlines in $[d'_{\min}, d'_{\max})$. Also, since its t_{step} is greater than or equal to one time unit, then probably only some of (not all) the deadlines in this region need to be checked.

Theorem 2 With compressing the task set M , the system must be schedulable in $[t_r, d'_{\min})$, that is, all the deadlines in this region are satisfied.

Proof: Consider a deadline d_x in $[t_r, d'_{\min})$. From t_r to d_x , based on Theorem 1, it gets

$$D_M(t_r, d_x) = \sum_{i=0}^{m-1} D_i(t_r, d_x) = 0$$

$$D_J(t_r, d_x) \leq \sum_{j \in J} \left\lfloor \frac{d_x - t_r}{T_j} \right\rfloor C_j \leq (d_x - t_r) U_J \leq (d_x - t_r)$$

$$D_{\text{total}}(t_r, d_x) = D_M(t_r, d_x) + D_J(t_r, d_x) \leq (d_x - t_r)$$

Therefore,

$$\Delta(t_r, d_x) = D_{\text{total}}(t_r, d_x) - (d_x - t_r) \leq 0$$

This means that d_x must be met.

Theorem 2 is proved.

Theorem 2 shows that the deadlines before $d'_{\min}$ will never be missed whenever the new tasks are inserted from t_r . Therefore, the first round of Δ checking is started with the deadlines only from $d'_{\min}$.

Theorem 3 With compressing the task set M , suppose that the new tasks are released from r_j and deadline missing occurs at some time t_x in $[d'_{\min}, d'_{\max})$, then the insertion should be delayed and it must have

$$\delta_{\text{earliest}} \geq r_j + \Delta(t_r, t_x) \quad (9)$$

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

Proof: When Δ checking is done at t_x , A_1 , B_1 , and C_1 are calculated as

$$A_1 = D_{\text{total}}(t_r, t_x)$$

$$B_1 = D_M(t_r, t_x)$$

$$C_1 = D_J(r_J, t_x)$$

Obviously, $A_1 = B_1 + C_1$ and $\Delta(t_r, t_x) = A_1 - (t_x - t_r)$.

If the value of the Δ checking with this round is $\Delta(t_r, t_x) > 0$, then the release time of the new tasks should be delayed to $r_J + t_{\text{step}}$ and the next round of checking is started. It is guessed that this t_{step} should be greater than or equal to $\Delta(t_r, t_x)$.

This paper continues the discussion with Case 1 and Case 2.

Case 1 If there is at least one deadline from the new tasks at t_x , as shown in Figure 2.

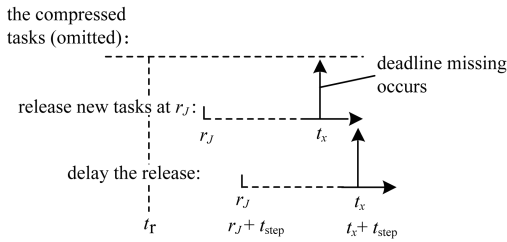


Fig. 2 New tasks have at least one deadline at t_x

Then, this deadline will shift to $t_x + t_{\text{step}}$ with the delaying. Notice that after the delaying, not only the deadlines (if any) at t_x should be met, but also the deadlines at $t_x + t_{\text{step}}$ must be satisfied.

Let us introduce A_2 , B_2 , and C_2 for this new round of checking after the delaying:

$$A_2 = D_{\text{total}}(t_r, t_x + t_{\text{step}})$$

$$B_2 = D_M(t_r, t_x + t_{\text{step}})$$

$$C_2 = D_J(r_J + t_{\text{step}}, t_x + t_{\text{step}})$$

$$A_2 = B_2 + C_2$$

High attention should be paid to the new tasks that have $C_2 = C_1$, that is, the processor demand of every new task does not increase till $t_x + t_{\text{step}}$. However, the old tasks have $B_2 \geq B_1$, since their processor demands may increase. Thus, the total processor demand of the system is

$$D_{\text{total}}(t_r, t_x + t_{\text{step}}) = A_2 = B_2 + C_2 \geq B_1 + C_1 = A_1$$

With this new round of checking, the deadlines at $t_x + t_{\text{step}}$ are satisfied if and only if the Δ checking value is not greater than zero, that is,

$$D_{\text{total}}(t_r, t_x + t_{\text{step}}) - (t_x + t_{\text{step}} - t_r) = \Delta(t_r, t_x + t_{\text{step}}) \leq 0$$

Then,

$$t_{\text{step}} \geq D_{\text{total}}(t_r, t_x + t_{\text{step}}) - (t_x + t_{\text{step}} - t_r) \geq A_1 - (t_x - t_r)$$

Here, it should be noticed that $A_1 - (t_x - t_r)$ is the Δ checking value of the previous round. This shows that the minimal value of this t_{step} is equal to the Δ checking value of the previous round, hence Equation (9) is true.

Case 2 If no deadline from the new tasks exists at t_x , as shown in Figure 3.

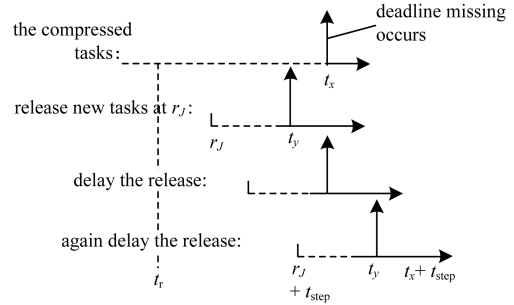


Fig. 3 No deadline from new tasks exists at t_x

That is to say, the missed deadline at t_x must be from some compressed tasks. Suppose that the deadline from the new tasks closest to the left of t_x is at t_y . The release of the new tasks is delayed if the Δ checking value is $\Delta(t_r, t_x) > 0$. The first step delay taken is so that the deadline from the new tasks at t_y moves to t_x . However, because the processor demands of the compressed tasks as well as the new tasks remain unchanged in $[t_r, t_x]$, $\Delta(t_r, t_x)$ keeps the same value as before the delay, and hence further delay has to be done.

Note that it is an exact Case 1 after the first step delay. This implies that with Case 2, the minimal value of its t_{step} is greater than the Δ checking value of the previous round, thus Equation (9) is correct.

Taking Case 1 and Case 2 together, Equation (9) is got.

Theorem 3 is proved.

Theorem 3 indicates that if the release of the new tasks based on this round is unsuccessful, then there must be a Δ checking value greater than zero (at least one time unit). This value can be taken as the t_{step} to the next round, and $t_{\text{step}} \geq 1$.

The Smart way has the advantage that the t_{step} may be greater than one. Let us call it “advantage of step”. This paper discusses how many times of the Δ checking can be saved by this advantage in two cases. 1) Suppose that the unsatisfied deadline d_x being checked is from the compressed tasks. Using the Simple way, d_x will be rechecked after the delay of r_J

by one time unit, that is, $r_J = r_J + 1$. However, the Smart way rechecks d_x after $r_J = r_J + t_{\text{step}}$ so that the times of the checks that can be omitted is “ $t_{\text{step}} - 1$ ”.
 2) Suppose that the unsatisfied d_x is from the new tasks. In this case, the next deadlines to be checked after delay with the Simple way and the Smart way are $d_x + 1$ and $d_x + t_{\text{step}}$, respectively. Then, the times of the checks saved by the Smart way is also “ $t_{\text{step}} - 1$ ”.

Another advantage of the Smart way is that only the deadlines in $[d'_{\min}, d'_{\max})$ need to be checked. Let us call it “advantage of region”. Apparently, this advantage depends on the number of the deadlines of the new tasks in $[t_r, d'_{\min})$. Because a deadline from a new task will be shifted with the delay of r_J , it is recommended to check all the deadlines of the compressed tasks before the check of J , that is, M first. Note that one time of the Δ checking is enough for every deadline in the region $[t_r, d'_{\min})$ due to the fact that it must be met based on the Theorem 2. Therefore, the number of the times of the checks saved by “advantage of region” is equal to “the number of the deadlines from the subset J before $d'_{\min}$ after the inspection of M ”.

3 Simulations

The Theorems described above are verified by simulation. An illustrative example is shown in Figure 4. The compressed task set M is composed of $\tau_0(8, 16)$ and $\tau_1(8, 16)$, while the subset J of the new tasks contains only one task $\tau_2(2, 8)$.

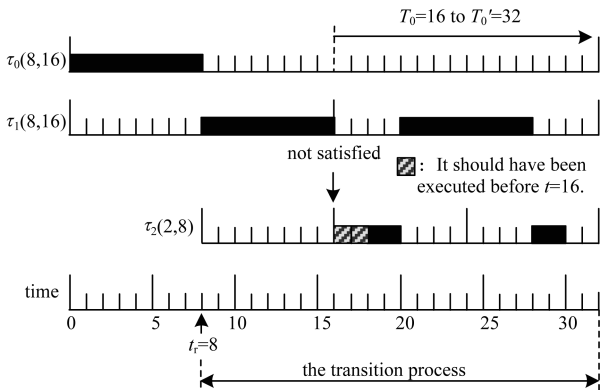


Fig. 4 Example of deadline missing after insertion

First let $t_r = 8$, then different values of t_r are discussed. Here, τ_0 is compressed while the period of τ_1 remains unchanged, and

$$t_0 + T'_0 = 0 + 32 = 32 = d'_{\max}$$

$$t_1 + T'_1 = 16 = d'_{\min}$$

The remaining computation $c_0(t_r) = 0, c_1(t_r) = 8$.

The transition process is in $[t_r, d'_{\max}) = [8, 32)$.

The total processor demand of the system before and after compression is $U_0 + U_1 = 8/16 + 8/16 = 1$ and $U'_0 + U_1 + U_2 = 8/32 + 8/16 + 2/8 = 1$, respectively. The system is schedulable both in the old mode and the new mode based on the EDF algorithm.

Now the Smart way to evaluate the earliest smooth time is used. Note that only the deadlines in $[d'_{\min}, d'_{\max}) = [16, 32)$ should be checked.

Round 1:

Let $r_J = t_r = 8$. The deadlines of the compressed tasks in M is checked. As shown in Figure 4, τ_0 has no deadline in $[16, 32)$, thus the deadline of τ_1 at $t = 16$ is checked first.

$$\begin{aligned} \Delta(8, 16) &= D_{\text{total}}(8, 16) - (16 - 8) = \\ &= D_0(8, 16) + D_1(8, 16) + \\ &= D_2(8, 16) - 8 = 0 + 8 + 2 - 8 = 2 > 0 \end{aligned}$$

This positive value of the Δ checking means that there is deadline missing at $t = 16$. One deadline of τ_2 is not satisfied as in Figure 4.

Round 2:

Taking the positive value from Round 1 as the t_{step} , the release time of the new task is increased to $r_J = r_J + t_{\text{step}} = 8 + 2 = 10$. Again the deadline of τ_1 is checked at $t = 16$:

$$\begin{aligned} \Delta(8, 16) &= D_{\text{total}}(8, 16) - (16 - 8) = \\ &= D_0(8, 16) + D_1(8, 16) + D_2(8, 16) - 8 = \\ &= 0 + 8 + 0 - 8 = 0 \end{aligned}$$

This indicates that no deadline will be missed at $t = 16$. Now the inspection of M is completed and the check of J is started. τ_2 has two deadlines at $t = 18$ and $t = 26$, the Δ checking values of them are 0 and -6 , respectively. Both are satisfied.

Therefore, the second round of checking draws the conclusion that the system is schedulable. Then the earliest smooth insertion time is

$$\delta_{\text{earliest}} = 10$$

In this way, δ_{earliest} is obtained through 4 times of Δ checking with 2 rounds. Comparatively, if the Simple way ($t_{\text{step}} = 1$) is used, it is not difficult to find that 3 rounds and 5 times of Δ checking are necessary: The first round is done with $r_J = 8$, the second round with $r_J = 9$, and the third with $r_J = 10$.

Let NO_{smart} and NO_{simple} represent the required times of Δ checking with the Smart way and the Simple way, respectively. Also, the “percentage of reduction”, labeled with P is introduced that

$$P = (NO_{\text{simple}} - NO_{\text{smart}}) / NO_{\text{simple}}$$

The greater the P , the greater the advantage of the Smart way has. With the above example,

$$P = (5 - 4) / 5 = 20\%$$

In Figure 4, if $t_r = 9$ is assumed instead of $t_r = 8$, the new task can not be inserted immediately either, and the Smart way or the Simple way is required to get δ_{earliest} . The value of the corresponding P is

$$t_r = 9, P = (4 - 4) / 4 = 0\%$$

The Smart way shows a certain advantage when $t_r = 8$. This is because of a larger Δ checking value of 2. Compared with the Simple way, one time ($2 - 1 = 1$) of the Δ checking can be omitted by this “advantage of step”. In addition, the “advantage of region” in this example is zero, as there is no deadline from J before $d'_{\min}$. These advantages of the Smart way may vary with different parameters of the tasks or different values of t_r . For instance, if the tasks in Figure 4 take these parameters $\tau_0(64, 128)$, $\tau_1(64, 128)$, $\tau_2(16, 64)$, $T'_0 = 256$, and $t_r = 64$, then it gets a larger Δ checking value of 16 so that 15 checks can be saved. By further calculation, it gets $P = (19 - 4) / 19 = 78.9\%$. This value is quite significant.

4 Conclusions

This paper discusses the problem of when new tasks can be inserted smoothly after compressing the task set M , that is, to evaluate the earliest smooth insertion time δ_{earliest} . The main features of the proposed Smart way are as follows:

1) Rather than from t_{oldb} , Theorem 1 gives the expressions to calculate the processor demands of the tasks from t_r that is the start of the transition process and is usually later than t_{oldb} . This helps to reduce the amount of calculation.

2) Only the deadlines in $[d'_{\min}, d'_{\max})$ need to be checked to get δ_{earliest} .

3) In some cases, δ_{earliest} can be calculated by Δ checking with the t_{step} greater than one, thus the number of the rounds and the times of the Δ checking are reduced to reach δ_{earliest} .

However, if in a certain round of checking, a missed deadline is asserted by a Δ checking with a value of 1, then the t_{step} of the Smart way is the same as that of the Simple way. In addition, if too many deadlines exist in $[d'_{\min}, d'_{\max})$, then the trying for δ_{earliest} is still time consuming. We will try to study such problems in the future.

Reference

- [1] BUTTAZZO G C, LIPARI G, CACCAMO M, et al. Elastic scheduling for flexible workload management [J]. IEEE Transactions on Computers, 2002, 51(3): 289-302.
- [2] BECCARI G, CASELLI S, ZANICHELLI F. A technique for adaptive scheduling of soft real-time tasks [J]. Real Time Systems, 2005, 30(3): 187-215.
- [3] BAMBAGINI M, MARINOIN M, AYDIN H, et al. Energy-aware scheduling for real-time systems: a survey [J]. ACM Transactions on Embedded Computing Systems, 2016, 15(1): 1-7, 34.
- [4] ALBERS S. Energy-efficient algorithms [J]. Communications of the ACM, 2010, 53(5): 86-96.
- [5] LIU C L, LAYLAN J W. Scheduling algorithms for multiprogramming in a hard real-time environment [J]. Journal of the ACM, 1973, 20(1): 40-61.
- [6] QIAN G. An earlier time for inserting and/or accelerating tasks [J]. Real Time Systems, 2009, 41(3): 181-194.
- [7] BUTTAZZO G C, LIPARI G, ABENI L. Elastic task model for adaptive rate control [C] // Proceedings of IEEE Real-time Systems Symposium. Washington D. C., USA: IEEE Press, 1998: 286-295.
- [8] REAL J, CRESPO A. Mode change protocols for real-time systems: a survey and a new proposal [J]. Real Time Systems, 2004, 26(2): 161-197.
- [9] AHMED M, FISHER N. Tractable schedulability analysis and resource allocation for real-time multi-modal systems [J]. ACM Transactions on Embedded Computing Systems, 2014, 13(2s): 1-39.
- [10] PEDRO P, BURNS A. Schedulability analysis for mode change [C] // Proceedings of 10th EUROMICRO Workshop on Real-Time Systems Symposium. Berlin, Germany: [s. n.], 1998: 172-179.
- [11] PILLAI P, SHIN K G. Real-time dynamic voltage scaling for low-power embedded operating systems [C] // Proceedings of 18th ACM Symposium Operating Systems Principles. New York, USA: ACM Press, 2001: 89-102.
- [12] QIAN G. Research on transition process based on earliest deadline first algorithm [J]. Computer Engineering, 2014, 40(9): 55-58.
- [13] QIAN G, LIANG L. The research on the transition of the bandwidth transfer from multiple real-time tasks [J]. Computer Engineering, 2017, 43(12): 292-302.
- [14] JEFFAY K, STONE D L. Accounting for interrupt handling costs in dynamic priority task systems [C] // Proceedings of IEEE Real-Time Systems Symposium. Washington D. C., USA: IEEE Press, 1993: 212-221.
- [15] QIAN G, ZHOU Y. Research on bandwidth transfer of task pair based on earliest deadline first algorithm [J]. Computer Engineering, 2016, 42(4): 65-69.