Two Methods to Release a New Real-time Task

Guangming Qian[1], Xianghua Chen [2]
College of Mathematics and Computer Science
Hunan Normal University
Changsha, 410081, China
qqyy@hunnu.edu.cn

Gang Yao [3]
Siebel Center for Computer Science
201 N. Goodwin Avenue
Urbana, IL 61801, USA.
yaog99@gmail.com

Abstract

In some real-time systems, sometimes there is a need to insert a new task at run time. If the system is fully loaded, the scheduler has to compress one or more current tasks to free some bandwidth. Naturally, finding the earliest time to start the insertion without deadline missing is interesting, which is still under investigation. Instead of this, this paper discusses how to do the compression and insertion to make the insertion as early as possible, based on the conclusions in the published literature with the earliest deadline first algorithm(EDF). By introducing the instance remaining bandwidth(RB), an algorithm called maximum remaining bandwidth compression(MRBC) is provided to choose a proper task as the compressed one. An associated theorem and its lemma are also provided, which declares that the safety adjustment can be implemented immediately upon request if the increase of the bandwidth due to insertion does not exceed the lowest utilization of individual current tasks in the system and if every task is allowed to be compressed. Another algorithm named part utilization first(PUF) is also proposed, according to which a new task is inserted with a lower speed first, then to its full utilization after a certain time.

Keywords: remaining bandwidth; task insertion; real-time system; deadline.

I. INTRODUCTION

The utilization of a real-time task is often referred to as its bandwidth in periodic real-time systems. With a fully loaded system scheduled with EDF, the sum of the utilizations of the tasks in the system is equal to 1 [1]. In this case, if a new task requests to be inserted, the system will become unschedulable, i.e. deadline missing will occur. Compressing one or more current tasks (decreasing their utilizations) to free some bandwidth is usually a way to tackle this [2, 3].

In [2], Buttazzo presented an elastic scheduling model for the task set based on EDF, the problem of new tasks' insertion was discussed in detail [2, 4]. An expression was given to calculate **the time** the insertion can be done without deadline missing. A deeper research was made and a modified expression was proved to evaluate **an earlier time** in [3], which is the conclusion to date with the problem of insertion with EDF.

Finding the earliest time for starting the insertion without deadline missing is interesting, which is still under investigation. **The purpose of this paper** is **not** on this, **but** on the choice of a proper task for compressing and the way of the release of the new task to make the insertion as early as possible. By introducing the remaining bandwidth of an instance, an algorithm is provided to choose a proper task as the compressed one. A useful theorem and its lemma are also provided, which declares that the safety adjustment can be implemented immediately upon request if the increase of the bandwidth due to insertion does not exceed the lowest utilization of individual current tasks in the system and every task is allowed to be compressed. The part utilization first algorithm is also proposed, according to which a new task is inserted with a lower speed first, then to its full utilization after a certain time.

In the next section, two requirements with task insertion are reviewed. The concept of the remaining bandwidth is defined in Section III, based on which the algorithm MRBC is described and the theorem for judging immediate insertion is provided. The PUF algorithm is also proposed in this section. The paper concludes in section IV.

This work has been supported by Hunan Provincial Natural Science Foundation of China under grant agreement No. 09JJ5040.

II. TWO REQUIREMENTS

In Figure 1, suppose there are two tasks in a real-time system, $\tau_0(8,16)$ and $\tau_1(12,24)$. The task model is $\tau_i(C_i,T_i)$, where C_i is its computation , T_i is its period, and utilization $U_i = C_i / T_i$. In this example, we have $U_0=C_0/T_0=8/16=1/2$, and $U_1=C_1/T_1 =12/24=1/2$. The total utilization of the task set is $\Sigma U = U_0 + U_1=1$, which indicates the system is schedulable based on EDF and fully loaded. Figure 1 shows its part of execution.

If at some time t_r, a new task $\tau_2(1,4)$ requests to run. Because $\Sigma U = U_0 + U_1 + U_2=1+1/4>1$ makes the new task set unschedulable, so certain bandwidth has to be freed from current tasks. Let us compress $\tau_0(8,16)$ to $\tau_0(8,32)$ so that $U_0'=1/4$ and $\Sigma U = U_0' + U_1 + U_2=1$.

However, both [2] and [3] pointed out that **even if** $\Sigma U \leq 1$ after insertion, deadline missing may occur if the insertion of τ_2 is too early. The operation of "compression→insertion" is anyway a type of interference in the running system.

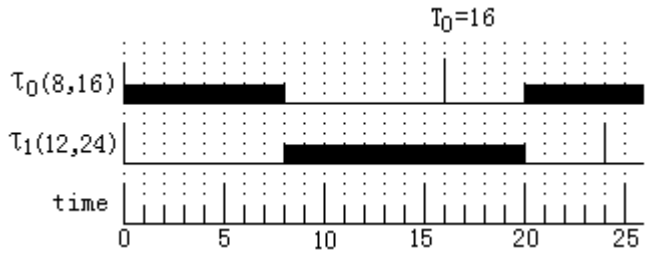

Figure 1. The scheduling of a simple task set

It is not welcome that any deadline is missed for any task, even lower priority tasks. An accurate scheduling algorithm should behave as what it declares. For example, if an Internet Service Provider declares that the bandwidth allocated to a user is decreased, then the decreased bandwidth should be guaranteed.

Therefore, generally, **the first requirement** for the insertion of a new task is **smooth**, i.e. no deadline will be missed after insertion.

A time point from which on a smooth insertion can be implemented is called **a smooth insertion time**.

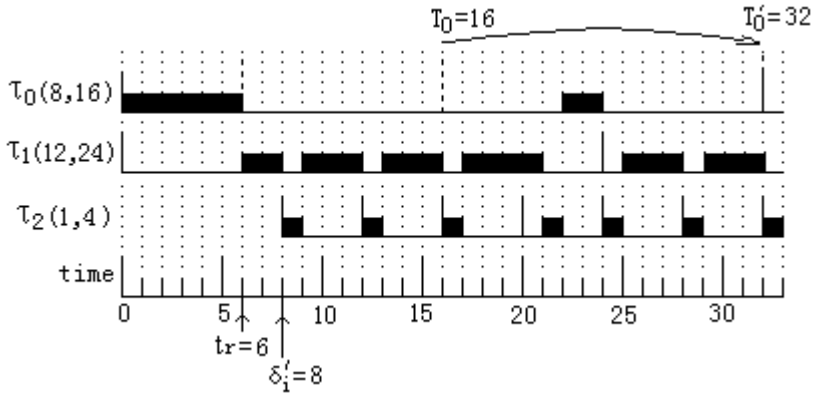

Figure 2. An example of smooth insertion

Again take the example in Figure 1. Suppose t_r =6 and the period of τ_0 is prolonged from 16 to 32. The remaining computation time of the current instance of τ_0 is equal to $C_0(t_r)= C_0(6)=2$. According to [2], a smooth insertion time $\delta_i = \delta_0=12$ can be calculated. Comparatively, an earlier smooth insertion point from [3] is $\delta_i' =\delta_0' =8$. Figure 2 shows the smooth insertion based on [3].

Although an earlier smooth insertion time may be evaluated from [3] than from [2], the accurate algorithm for the earliest smooth time is still under investigation.

Let us list the expression of δ_i' in [3] as Equation (1) for convenience.

$$\delta_i' = d_i - \frac{C_i(t_r)}{U_i - U_i'}. \tag{1}$$

Where d_i represents the deadline of the current instance of the compressed task τ_i. $C_i(t_r)$ is its remaining computation time up to d_i. U_i and U_i' are its utilizations before and after compression respectively. In our discussion in this paper, we assume $d_i=T_i$. In the example of Figure 2, we have d_i=16, U_i =1/2, U_i' =1/4, and $C_i(t_r)$ =2. In [3], it is also pointed out that if δ_i' from Equation (1) is less than or equal to t_r, then we take $\delta_i' = t_r$, which means an immediate smooth insertion can be done at t_r.

An obvious observation from Equation (1) is that an insertion at the end point of the current instance of τ_i (as d_i =16 in Figure 2) is definitely smooth. However, if a larger time interval between t_r and d_i exists, the new task has to wait impatiently. Reasonably we should **insert the new task as early as possible**. This is our **second requirement**.

As in Figure 2, inserting τ_2 at $t = \delta_i'$ = 8 is normally welcome than at $t = d_i$ =16, especially in urgent cases.

III. THE MRBC AND PUF ALGORITHMS

3.1 The maximum remaining bandwidth compressing(MRBC) algorithm

In Figure 2, if τ_1 is selected to be compressed instead of τ_0, i.e. from τ_1(12,24) to τ_1(12,48), then we have δ_i' = 6, which implies an immediate smooth insertion. There is no "earlier" than "immediate".

To find this reason, we convert (1) to Equation (2).

$$U_i - U_i' = \frac{C_i(t_r)}{d_i - \delta_i'}. \tag{2}$$

The left side indicates the freed bandwidth from the compressed task τ_i. The right is the utilization of the remaining computation time $C_i(t_r)$ between δ_i' and d_i.

Definition 1(the remaining bandwidth of the current instance) *If compressing τ_i starts at t_r , $C_i(t_r)$ is the remaining computation time of the current instance, d_i is its deadline, then $C_i(t_r)/(d_i - t_r)$ is called* ***the remaining bandwidth of the current instance*** *of τ_i at t_r ,or* ***the instance remaining bandwidth,*** *is represented by* $\boldsymbol{RB_i(t_r)}$.

If $\delta_i' = t_r$, then the right side of Equation (2) is the remaining bandwidth of the current instance of τ_i at t_r. Through careful observation, it is found that the greater the remaining bandwidth at t_r is, the earlier the smooth insertion point δ_i' can be gotten. The earliest possible time is of course t_r.

Definition 2 *At t_r , for every task in the current task set, we calculate the value of the remaining bandwidth of every current instance. The maximum one is defined as* ***the maximum instance remaining bandwidth****, represented with* $\boldsymbol{RB_{max}(t_r)}$. *Similarly, the minimum one is called* ***the minimum instance remaining bandwidth*** *and named* $\boldsymbol{RB_{min}(t_r)}$.

Again take the example in Figure 2. The utilization of the new task is U_2=1/4. The remaining bandwidth of τ_0 at t_r=6 is $RB_0(6)$=2/(16–6)=1/5, while τ_1 has $RB_1(6)$=12/(24–6)=2/3. Obviously, selecting τ_1 as the compressed task other than τ_0 possibly produces an earlier smooth insertion time. Here $RB_{max}(t_r)=RB_{max}(6)=RB_1(6)$=2/3, and $RB_{min}(t_r)=RB_{min}(6) =RB_0(6)$ =1/5.

Now we formally give out the algorithm in Figure 3 for selecting a current task to compress based on the remaining bandwidth.

The Algorithm MRBC: maximum remaining bandwidth compression.

Step 1. With t_r, find out the current task τ_i to make $RB_i(t_r) \geq U_i - U_i'$ hold and go to Step 4. If no such a task is found, then choose the one that has the maximum instance remaining bandwidth as τ_i and go to Step 2.

Step 2. Evaluate the smooth insertion time δ_i' with $U_i - U_i' = C_i(t_r)/(d_i - \delta_i')$ and go to Step 3.

Step 3. Compress τ_i, insert τ_j from $t=\delta_i'$ on, and **finish the algorithm**.

Step 4. Compress τ_i, insert τ_j from $t= t_r$ on, and **finish the algorithm**.

Figure 3. The description for the MRBC Algorithm.

Suppose there are $n(n\geq 1)$ current tasks in the system. In order to find τ_i that makes $RB_i(t_r) \geq U_i - U_i'$ hold or has the maximum instance remaining bandwidth, the step 1 of the algorithm may have to go through n instances. The remaining computation time of every instance must be evaluated, and thus n counters have to be maintained at run time. This is a type of expenditure. The temporal complexity of the algorithm is apparently (n).

This algorithm shows that the maximum instance remaining bandwidth $RB_{max}(t_r)$ is important to the task insertion. Theorem 1 discovers a useful rule for judging this maximum bandwidth.

Theorem 1 *Suppose arbitrary preemption is allowed in a periodic real-time system at run time based on EDF, and task τ_{min} has the lowest utilization U_{min} among all the tasks. Then, no matter which task is selected to be compressed and when this compression starts, $RB_{max}(t_r) \geq U_{min}$ is always true.*

Proof Considering a current task $\tau_i(C_i,T_i)$ at run time. At the beginning of an instance, the instance remaining bandwidth is U_i. At any other time, according to the preemption before that time, three possible cases exist.

Case1. It has been preempted more by other tasks so that its instance remaining bandwidth is greater than U_i.

Case2. It has preempted other tasks so that the instance remaining bandwidth is less than U_i.

Case3. The results from "has been preempted" and "has preempted" are the same up to now, and thus it has the instance remaining bandwidth equal to U_i.

For case 1 at t_r, we have $RB_{max}(t_r) > U_i \geq U_{min}$.

For case 2 at t_r, suppose there is a task $\tau_j(C_j, T_j)$ that has been preempted more by τ_i, and the utilization of τ_j is U_j, which implies that the remaining bandwidth of the current instance of τ_j is greater than U_j, and thus $RB_{max}(t_r) > U_j \geq U_{min}$.

For case 3 at t_r, $RB_{max}(t_r)$ is at least not less than U_i, and greater than or equal to U_{min}.

Therefore, in any case, $RB_{max}(t_r) \geq U_{min}$ always holds.

This proves Theorem 1.

Lemma 1 *If the utilization of a new task is not greater than the lowest utilization U_{min} of the current tasks, and any of them is permitted to be compressed, then this new task can be immediately inserted at any time.*

Proof It is obvious from Theorem 1.

Now with Theorem 1 and Lemma 1, for the problem of the task insertion, we find out the lowest value among all the single task's utilizations of the current tasks, if the bandwidth of the new task in a period is less than or equal to this value and if every task is allowed to be compressed, then an immediate insertion is smooth whenever it requests to join in the system. With a fixed utilization of every task, we can even predict this before running the system.

Let us come back to the example in Figure 1. $U_{min}=U_0= U_1=1/2$, and the task set (τ_0, τ_1) has a hyperperiod (least common multiple period) $T_{LCM}=48$. According to different expressions, we can list $RB_{max}(t)$ during the first hyperperiod as follows.

At time t =0,	$RB_{max}(0)=U_{min}=\mathbf{1/2}$.
$\forall\ t,\ 0 < t < 16$,	$RB_{max}(t)= RB_1(t) > 1/2$.
At time t =16,	$RB_{max}(16)=U_{min}=\mathbf{1/2}$.
$\forall\ t,\ 16 < t < 24$,	$RB_{max}(t)= RB_0(t) > 1/2$.
At time t =24,	$RB_{max}(24)=U_{min}=\mathbf{1/2}$.
$\forall\ t,\ 24 < t < 32$,	$RB_{max}(t)= RB_1(t) > 1/2$.
At time t =32,	$RB_{max}(32)=U_{min}=\mathbf{1/2}$.
$\forall\ t,\ 32 < t < 48$,	$RB_{max}(t)= RB_0(t) > 1/2$.

Totally, there are 4 points at which $RB_{max}(t)=U_{min}=1/2$, at any other time $RB_{max}(t) > U_{min}$ is always satisfied.

Actually, in many systems $RB_{max}(t) > U_{min}$ holds at any time. Figure 4 illustrates such an example. There are three tasks in the set. The hyperperiod T_{LCM} is equal to 16. $U_{min}=U_0=U_2=1/4$. Similarly, $RB_{max}(t)$ is listed as follows.

$$\forall\ t,\ 0 \leq t < (17-\sqrt{97})/2, \qquad RB_{max}(t)= RB_1(t) > U_{min}.$$

$$\forall\, t,\ (17-\sqrt{97})/2 \le t < 8, \qquad RB_{max}(t) = RB_0(t) > U_{min}.$$

$$\forall\, t,\ 8 \le t < 14, \qquad RB_{max}(t) = RB_1(t) > U_{min}.$$

$$\forall\, t,\ 14 \le t < 16, \qquad RB_{max}(t) = RB_2(t) > U_{min}.$$

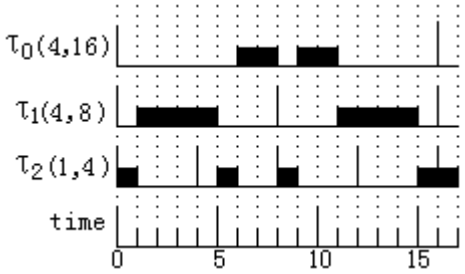

Figure 4. An example of $RB_{max}(t_r) > U_{min}$ at any time.

3.2 **The part utilization first(PUF) algorithm**

Possibly, the remaining bandwidths with n tasks have to be compared based on MRBC. If the new task can not start at t_r, then the insertion has to be delayed.

Actually, sometimes a real-time task has a flexible period, that is, it can vary from a minimum period T_{min} to a maximum period T_{max}. Suppose the period T_j of a new task $\tau_j(C_j,T_j)$ ranges from T_{jmin} to T_{jmax}. The selected compressed task is still represented by $\tau_i(C_i,T_i)$. In order to make the system schedulable, we assume

$$U_i - U_i^{'} = \frac{C_j}{T_{j\min}}. \tag{3}$$

This implies that after the end of the current instance of τ_i, the new task is allowed to run with T_{jmin} in full speed, but before that, it may be executed with T_{jmax} and thus part of its full utilization to get a possible earlier release of its first instance. This is the idea of the PUF algorithm, which is formally described as in Figure 5.

The Algorithm PUF: part utilization first.

Step 1. **If** C_j/T_{jmin} is less than or equal to the lowest utilization U_{min} in the current task set and if every task is permitted to be compressed, **then** compress any of the current tasks, insert the new task τ_j with T_{jmin} from $t = t_r$ on, and **finish the algorithm**.

Step 2. With t_r, find out the current task τ_i to make $RB_i(t_r) \ge C_j/T_{jmin}$ hold and go to Step 5. If no such a task is found, then choose the one that has the maximum instance remaining bandwidth as τ_i and go to Step 3.

Step 3. Calculate δ_i' with $C_j/T_{jmax} = C_i(t_r)/(d_i - \delta_i')$. If $\delta_i' \le t_r$, then take $\delta_i' = t_r$. Go to Step 4.

Step 4. **If** $\delta_i' < d_i$, **then** compress τ_i, release every instance of τ_j starting before d_i with T_{jmax}, run its instances starting after d_i with T_{jmin}, and **finish the algorithm**. **If** $\delta_i' = d_i$, **then** compress τ_i, insert τ_j with T_{jmin} from $t = d_i$ on and **finish the algorithm**.

Step 5. Compress τ_i, insert τ_j with T_{jmin} from $t = t_r$ on and **finish the algorithm**.

Figure 5. The description for the PUF algorithm

There are three points that should be noticed.

- The conclusion from Theorem 1 and Lemma 1 is included in Step 1 of PUF. If the full utilization C_j/T_{jmin} is less than or equal to the lowest utilization U_{min} in the system, then Step 2 to Step 5 are not needed.
- If a task τ_i is found with $RB_i(t_r) \ge C_j/T_{jmin}$, then the new task τ_j can be inserted immediately from t_r on. Otherwise, the smooth time δ_i' is evaluated with its part utilization C_j/T_{jmax}.

- $t = d_i$ is used as a critical time. If the release of the first instance of τ_j is with T_{jmax}, then other instances of this task starting before d_i, if any, will also be with T_{jmax}. The instances after d_i can definitely be released with T_{jmin} because Equation (1) declares $\delta_i' \leq d_i$.

Figure 6 is an example of PUF. Suppose $\tau_0(6,16)$ is the only task allowed to be compressed. The freed utilization is $U_0 - U_0' = 6/16 - 6/48 = 1/4$. The request time is $t_r = 4$. The remaining bandwidth of the current instance of τ_0 is $(6-4)/(16-4)=1/6$, which is less than the full utilization 1/4 of the new task $\tau_2(1,4)$. The period of τ_2 ranges from $T_{jmin} = T_{2min} = 4$ to $T_{jmax} = T_{2max} = 6$. If the first instance of τ_2 starts at $t=4$ with $T_{2min} = 4$, then the deadline at $t=16$ will be missed since up to $t=16$ the sum of the processor demands by τ_2 and τ_1 exceeds the length of the interval, that is, 3+10=13>12, which makes the system unschedulable [5, 6].

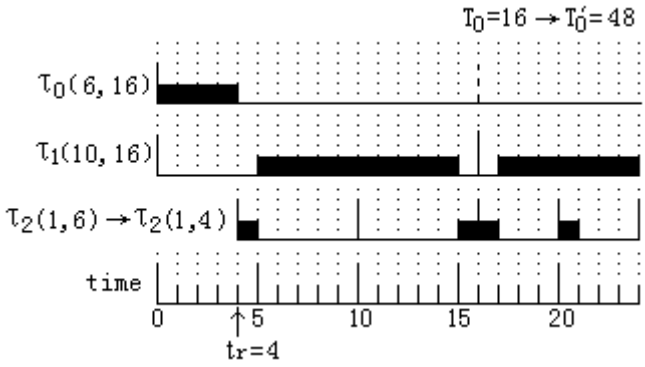

Figure 6. An example of the PUF algorithm.

Therefore, in order to start the first instance early at $t=4$ and make every deadline satisfied, we insert the new task τ_2 in two steps. The first two instances are with $T_{2max} = 6$, while other instances starting after $t=16$ are released with $T_{2min} = 4$, which is shown in Figure 6.

IV. CONCLUSIONS

How to insert a new real-time task as early as possible? Two approaches are provided in this paper. The first approach relates mainly with compression. Theorem 1 is very simple and useful since it says the new task can be inserted immediately at any time if the utilization of the new task does not exceed the lowest utilization of individual current tasks in the system and if every task is allowed to be compressed. The MRBC algorithm chooses a task to compress that has the maximum remaining bandwidth, based on which the earliest insertion can be implemented, but it has the temporal complexity (n). The second approach relates with insertion. With an accepted variation of its period to some extent, the new task can be inserted in two steps, first with a longest period T_{jmax} then a shortest one T_{jmin}. In this way, not only the schedulablity is guaranteed, but also the insertion can be done as quickly as possible.

ACKNOWLEDGMENT

Guangming Qian thanks Hunan Provincial Natural Science Foundation of China under grant agreement No. **09JJ5040** for financial support to this research.

REFERENCES

[1] C.L.Liu, 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.Computers*, vol. 51, no.3, pp.289-302, 2002.

[3] Q.Guangming, "An Earlier Time for Inserting and/or Accelerating Tasks," *Springer. Real Time Systems*, vol.41, no.3, pp.181-194, 2009.

[4] G.C.Buttazzo, G.Lipari, and L.Abeni, "Elastic Task Model for Adaptive Rate Control," In *Proc.19th IEEE Real-Time Systems Symp*., pp.286-295, Madrid, Spain, Dec.1998.

[5] S.Kbarauh, R.R.Howell, and L.E.Rosier,"Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic Real-Time Tasks on One Processor," *Springer. Real Time Systems*, vol.2, pp. 301-324,1990.

[6] K.Jeffay, and D.L.Stone,"Accounting for Interrupt Handling Costs in Dynamic Priority Task Systems," In *Proc.14th IEEE Real-Time Systems Symp*., pp.212-221, Raleigh Durham, NC, Dec.1993.

AUTHORS PROFILE

Guangming Qian is a professor at College of Mathematics and Computer Science, Hunan Normal University, China(P.R.C). His current research interests include real-time scheduling and network switching.
Xianghua Chen is a postgraduate student of Hunan Normal University.
Gang Yao is a Postdoctoral Research Collaborator at the University of Illinois at Urbana Champaign. He received a Ph.D. in Computer Engineering from the Scuola Superiore Sant'Anna of Pisa, Italy, in 2010, the BE and ME degrees from Tsinghua University, Beijing, China. His main interests include real-time scheduling algorithms, safety-critical systems and shared resource protocols.