A Feasibility Decision Algorithm for Rate Monotonic Scheduling of Periodic Real-Time Tasks

Yoshifumi Manabe Shigemi Aoyagi

NTT Basic Research Laboratories
3-1 Morinosato-Wakamiya, Atsugi-shi, Kanagawa 243-01 Japan

Abstract

The rate monotonic scheduling algorithm is a commonly used task scheduling algorithm for periodic real-time task systems. This paper discusses feasibility decision for a given real-time task system by the rate monotonic scheduling algorithm. It presents a new necessary and sufficient condition for a given task system to be feasible, and a new feasibility decision algorithm based on that condition. The time complexity of this algorithm depends solely on the number of tasks. This algorithm can be applied to the inverse-deadline scheduling algorithm, which is an extension of the rate monotonic scheduling algorithm.

1 Introduction

In real-time systems, there is a time constraint on computation, which is just as important as the correctness of computation. In an attempt to satisfy this constraint, many scheduling algorithms have been discussed [3]. The rate monotonic scheduling algorithm [10] is one of commonly used scheduling algorithms for periodic real-time task systems because it is optimal among fixed-priority preemptive scheduling algorithms. Furthermore, various extensions have been discussed, for example, scheduling aperiodic tasks while still meeting the deadlines of periodic tasks [11], scheduling when a task is added or deleted or a task period is modified [4][5][12], and scheduling when some tasks share resources [1][13].

A necessary and sufficient condition for a given periodic real-time task system to be feasible by the rate monotonic scheduling algorithm has been shown [10]. Two feasibility decision algorithms, the scheduling point test algorithm [6] and the completion-time test algorithm [14], have been shown. The time complexities of these two algorithms depend on both the number of tasks and the task periods.

This paper presents a new necessary and sufficient condition for feasibility along with a new feasibility decision algorithm, a reduced scheduling point test algorithm. The time complexity of this algorithm depends solely on the number of tasks. Thus it is a constant if the number of tasks is a constant. This algorithm can also be applied to determine the feasibility by the inverse-deadline scheduling algorithm [9][7], which is an extension of the rate monotonic scheduling algorithm.

In this paper, section 2 defines real-time task scheduling. Section 3 summarizes previous results for feasibility decision algorithms. Section 4 gives the necessary and sufficient condition to be feasible, and shows the feasibility decision algorithm based on this condition. Section 5 discusses an extension for the inverse-deadline scheduling algorithm. Section 6 summarizes the paper.

2 Scheduling periodic real-time tasks

This section gives the formulation of the real-time scheduling.

Definition 1 (Periodic real-time task system) [10]

- *Periodic real-time task system* $X = \{\tau_1, \tau_2, \ldots, \tau_n\}$ *is a set of tasks. These tasks are independent in that job requests for a certain task do not depend on the completion of requests for other tasks.*
- *Each task* τ_i *is periodic, with a constant interval* p_i *between job requests.*
- *Each task* τ_i *has an initialization time* $I_i (\geq 0)$. *Assume that* $min_{1 \leq i \leq n} I_i = 0$. *The jobs for task* τ_i *are initiated at* $I_i + kp_i (k = 0, 1, \ldots)$.

1080-1812/95 $04.00 © 1995 IEEE

- *Each task τ_i has a deadline $d_i(\leq p_i)$. A job initiated at $I_i + kp_i(k = 0, 1, \ldots)$ must be completed before $I_i + kp_i + d_i(k = 0, 1, \ldots)$.*

- *The computation requirement in each job for task τ_i is a constant $c_i(\leq d_i)$.*

- *Jobs can be preempted at any time and the overhead for job swapping can be ignored.* ■

Unless otherwise stated, we assume that $d_i = p_i(1 \leq i \leq n)$, that is, each task must be completed before the next request for it. Section 5 will discuss the case in which $d_i < p_i$.

When $I_i = I_j$ for all $1 \leq i, j \leq n$, the task system is said to be synchronous. Otherwise, it is asynchronous. This paper addresses only synchronous systems. Note that, as easily derived from Property 1, a synchronous system is less feasible than an asynchronous system with the same intervals, deadlines, and computation requirements. Thus, when the initialization times are unknown, the worst case can be obtained by testing the feasibility of a synchronous system. The feasibility of asynchronous systems has been discussed in some papers [2][8].

A scheduling algorithm is a set of rules that determine the jobs to be executed at a particular time. This paper is concerned only with fixed-priority preemptive scheduling algorithms, which work as follows. A distinct and fixed priority is assigned to each task. When a job request arises with a priority higher than the one currently being executed, the current job is immediately interrupted and the new job is started. Dynamic-priority scheduling algorithms, in which the priority for each request varies during execution, have also been discussed [2][8]. Although these can be more effective than fixed-priority scheduling algorithms, they are more difficult to implement. This paper thus considers only fixed-priority scheduling algorithms.

Definition 2 (Feasibility) [10] *For a task system X and a scheduling algorithm S, X is feasible by S if and only if all deadlines of the tasks in X are met by S.*

X is feasible if and only if it is feasible by some scheduling algorithm. ■

Thus, the optimal scheduling algorithm S is considered to be the one which satisfies the following: if X is feasible, X is feasible by S.

3 Previous results on the rate monotonic scheduling algorithm

This section summarizes previous results on feasibility by the rate monotonic scheduling algorithm.

Definition 3 (Rate monotonic scheduling) [10] *The rate monotonic scheduling algorithm is a fixed-priority preemptive scheduling algorithm which assigns a higher priority to a task with a shorter period.* ■

The rate monotonic scheduling algorithm is optimal among fixed-priority preemptive scheduling algorithms [10].

Unless otherwise stated, we label the tasks so that $p_1 \leq p_2 \leq \cdots \leq p_n$.

Definition 4 (Critical instant) [10] *The response time for a job request of a certain task is defined as the time span between the request and the end of the response to that request. A critical instant for a task is defined as an instant at which a job request for that task will have the longest response time.* ■

Property 1 [10] *When $d_i = p_i$ for all i, a critical instant for any task occurs whenever the task is requested simultaneously with requests for all higher priority tasks.* ■

Since $I_i = 0(1 \leq i \leq n)$, the following property can be immediately derived from this property.

Property 2 [10] *Task system X is feasible by the rate monotonic scheduling algorithm if the first job of each task meets its deadline.* ■

Let us define the utilization factor U_i of task τ_i as $U_i = c_i/p_i$. The following relations between feasibility and the utilization factor have been shown. The latter is trivial.

Property 3 [10] *Task system X is feasible by the rate monotonic scheduling algorithm if $\sum_{i=1}^{n} U_i \leq n(2^{1/n} - 1)$.* ■

Property 4 *Task system X is not feasible by any scheduling algorithm if $\sum_{i=1}^{n} U_i > 1$.* ■

The following necessary and sufficient condition to be feasible has been proved.

Property 5 [6] *Task system X is feasible by the rate monotonic scheduling algorithm if and only if $L = max_{1 \le i \le n} L_i \le 1$, where*
$L_i = min_{t \in S_i} L_i(t)$, $\quad L_i(t) = \sum_{j=1}^{i} c_j \cdot \lceil t/p_j \rceil / t$,
$S_i = \{k \cdot p_j \mid j = 1, 2, \ldots, i;\ k = 1, 2, \ldots, \lfloor p_i/p_j \rfloor\}$. ■

The scheduling point set S_i for task τ_i is defined as τ_i's first deadline p_i and the deadlines of higher priority tasks prior to p_i. This means that the task system is feasible if and only if, at some scheduling point, all currently existing jobs can be executed.

(Example 1) $n = 3$, $p_1 = 100$, $c_1 = 40$, $p_2 = 150$, $c_2 = 40$, $p_3 = 350$, and $c_3 = 100$.
When the scheduling point test algorithm is applied to this task system, the following equations are checked:
$i = 1$: $S_1 = \{100\}$.
  $t = 100$: $L_1(100) = 40/100 < 1$. Thus, $L_1 < 1$.
$i = 2$: $S_2 = \{100, 150\}$.
  $t = 100$: $L_2(100) = 80/100 < 1$. Thus, $L_2 < 1$.
$i = 3$: $S_3 = \{100, 150, 200, 300, 350\}$.
  $t = 100$: $L_3(100) = 180/100 > 1$.
  $t = 150$: $L_3(150) = 220/150 > 1$.
  $t = 200$: $L_3(200) = 260/200 > 1$.
  $t = 300$: $L_3(300) = 300/300 = 1$.
Thus, $L_3 \le 1$, so the task system is feasible. ■

(Example 2) $n = 3$, $p_1 = 100$, $c_1 = 60$, $p_2 = 150$, $c_2 = 50$, $p_3 = 350$, and $c_3 = 20$.
$i = 1$: $S_1 = \{100\}$.
  $t = 100$: $L_1(100) = 60/100 < 1$. Thus, $L_1 < 1$.
$i = 2$: $S_2 = \{100, 150\}$.
  $t = 100$: $L_2(100) = 110/100 > 1$.
  $t = 150$: $L_2(150) = 170/150 > 1$.
Thus, $L_2 > 1$, $\{\tau_1, \tau_2\}$ is not feasible, so the whole task system is not feasible. ■

Note that $L_i \le 1$ does not imply that $L_j \le 1 (j < i)$. In the above example, $L_3(300) = 1$ and thus $L_3 \le 1$. Calculation of L_i for all $i (1 \le i \le n)$ is therefore necessary.

Though this algorithm presents all instant when jobs might be preempted, the time complexity of this algorithm depends on the number of tasks and the maximum period when the task periods are integers. There are cases that the time complexity of this algorithm is not so small even if the number of tasks is a constant.

Lemma 1 *For any integer k, there is a task system X which satisfies $|X| = 2$ and the time complexity of the scheduling point test algorithm for X is at least $O(k)$.* ■

(Proof) Consider the following task system:
$n = 2$, $p_1 = T$, $c_1 = (k-2)/(k-1) \cdot T$, $p_2 = (k^2 - 2)/k \cdot T$, and $c_2 = (k^2 - k + 1)/(k^2 - k) \cdot T$.

T is arbitrary and all values of p_i and c_i can be integers if T is selected appropriately. This task system is not feasible.

Now, apply the scheduling point test algorithm. Since $(k-1)p_1 < p_2 < kp_1$, $S_2 = \{T, 2T, \ldots, (k-1)T, (k^2-2)/k \cdot T\}$. In order to detect that this task system is not feasible, it is necessary to calculate $L_2(t)$ for all $t \in S_2$. Therefore, the time complexity of the scheduling point test algorithm is at least $O(k)$. ■

Sha et al. [14] have shown another algorithm called the completion-time test algorithm. This algorithm is shown in Figure 1 [1]. The completion-time test algorithm first determines the computation requirement necessary for initially existing jobs. The value is obtained from $t := \sum_{j=1}^{i} c_j$. Some new jobs might arise until t. Thus, by calculating $t' := \sum_{j=1}^{i} c_j \cdot \lceil t/p_j \rceil$, the computation necessary for all jobs until time t is obtained. If $t' = t$, all current jobs can be executed at time t. Thus, $\{\tau_1 \ldots, \tau_i\}$ is feasible. If $t' > t$, the computation necessary for all jobs until t' is calculated again using the above expression. If t' satisfies $t' > p_i$, the task system is not feasible.

(Example 3) Apply the completion-time test to Example 1.
$i = 1$: It is trivial that τ_1 is feasible.
$i = 2$: Initially, $t = c_1 + c_2 = 80$.
In the *repeat-until* loop, $t' = c_1 \cdot \lceil 80/p_1 \rceil + c_2 \cdot \lceil 80/p_2 \rceil = 80$. Since $t' = t$ is satisfied, $\{\tau_1, \tau_2\}$ is feasible.
$i = 3$: Initially, $t = c_1 + c_2 + c_3 = 180$.
In the *repeat-until* loop, $t' = c_1 \cdot \lceil 180/p_1 \rceil + c_2 \cdot \lceil 180/p_2 \rceil + c_3 \cdot \lceil 180/p_3 \rceil = 260$.
Since $t \ne t'$ and $t' \le p_3$, the *repeat-until* loop is executed again and $t = 260$, $t' = c_1 \cdot \lceil 260/p_1 \rceil + c_2 \cdot \lceil 260/p_2 \rceil + c_3 \cdot \lceil 260/p_3 \rceil = 300$.
Since $t \ne t'$ and $t' \le p_3$, the loop is executed again and $t = 300$, $t' = c_1 \cdot \lceil 300/p_1 \rceil + c_2 \cdot \lceil 300/p_2 \rceil + c_3 \cdot \lceil 300/p_3 \rceil = 300$. Since $t' = t$, the task system is feasible. ■

(Example 4) Apply the completion-time test to Example 2.
$i = 1$: It is trivial that τ_1 is feasible.
$i = 2$: Initially, $t = c_1 + c_2 = 110$.
In the *repeat-until* loop, $t' = c_1 \cdot \lceil 110/p_1 \rceil + c_2 \cdot \lceil 110/p_2 \rceil = 170$.
Since $t' > p_2$ is satisfied, $\{\tau_1, \tau_2\}$ is not feasible, and thus the whole task system is not feasible. ■

[1] The original algorithm does not answer "not feasible" when the task set is not feasible. The algorithm in Figure 1 is therefore modified to answer "not feasible".

Though this algorithm gives the minimum time when all existing jobs are completed, the complexity depends on both the number of tasks and the maximum period when the periods are integers. There are some cases that the complexity of the algorithm is not so small even if the number of tasks is a constant.

Lemma 2 *For any integer k, there is a task system X which satisfies $|X| = 2$ and the time complexity of the completion-time test algorithm for X is at least $O(k)$.* ■

(Proof) Consider the example in the proof of Lemma 1. Note that $jp_1 < jc_1 + c_2 < (j+1)p_1$ $(j = 1, \ldots, k-1)$ holds.

For $i = 2$, initially $t = c_1 + c_2$. In the *repeat-until* loop, $t' = c_1 \cdot \lceil (c_1 + c_2)/p_1 \rceil + c_2 \cdot \lceil (c_1 + c_2)/p_2 \rceil = 2c_1 + c_2$, since $p_1 < c_1 + c_2 < 2p_1$. In the next iteration of the *repeat-until* loop, $t' = 3c_1 + c_2$. Similarly, t' is calculated as $4c_1 + c_2, 5c_1 + c_2, \ldots, (k-1)c_1 + c_2$ in each iteration of the loop. Since there are k iterations, the time complexity is at least $O(k)$. ■

4 A new feasibility decision algorithm

This section shows the new necessary and sufficient condition for a given task system to be feasible by the rate monotonic scheduling algorithm. Now, we define the reduced scheduling point set, R_i, as follows:

Definition 5 (Reduced scheduling point set)
$R_i = \bigcup_{j=1}^{i} Q_j^i$, *where* $Q_i^i = \{p_i\}$,
$Q_j^i = \{\lfloor t/p_j \rfloor \cdot p_j |\ t \in Q_k^i (j+1 \le k \le i)\}$ $(1 \le j < i)$. ■

This set satisfies $R_i \subseteq S_i$, where S_i is the scheduling point set. R_i consists of the following elements: p_i, the last deadline of task τ_{i-1} before p_i, the last deadlines of τ_{i-2} before each of these two deadlines, the last deadlines of τ_{i-3} before each of these four deadlines, and so on. Since some of the elements might be identical, $|R_i| \le 2^{i-1}$.

(Example 5) For the task system in Examples 1 and 2, $R_1 = \{100\}$, $R_2 = \{100, 150\}$, and $R_3 = \{300, 350\}$.
For the task system in the proof of Lemma 1, $R_1 = \{T\}$ and $R_2 = \{(k-1)T, (k^2-2)/k \cdot T\}$. ■

Consider the task system in the proof of Lemma 1. The scheduling point test algorithm does not have to test $L_2(t)$ at $t = T$. The reason is as follows. Suppose that $\{\tau_1\}$ is feasible. The total length of time when the jobs of τ_1 is not executed is longer in the time interval $[0, (k-1)T]$ than in $[0, T]$. Thus, the possibility to finish the execution of the job for τ_2 is greater at $(k-1)T$ than at T, because the number of τ_2's jobs in $[0, T]$ and $[0, (k-1)T]$ are the same. Thus, if $L_2((k-1)T) > 1$, $L_2(T) > 1$ and if $L_2(T) \le 1$, $L_2((k-1)T) \le 1$. Therefore, it is unnecessary to test $L_2(t)$ at $t = T$ if $L_2(t)$ is tested at $t = (k-1)T$. By a similar argument, it is enough to test $L_i(t)$ only at the "last" deadlines. The formal proof that the feasibility can be determined by examining $L_i(t)$ only at R_i is as follows.

Theorem 1 *Task system X is feasible by the rate monotonic scheduling algorithm if and only if*
$L' = max_{1 \le i \le n} L_i' \le 1$, *where*
$L_i' = min_{t \in R_i} L_i(t)$, $\quad L_i(t) = \sum_{j=1}^{i} c_j \cdot \lceil t/p_j \rceil / t$. ■

(Proof) Since $R_i \subseteq S_i$, it is trivial that, if $L' \le 1$, the task system is feasible.

Now assume that the task system is feasible. From Property 5, for all i, $L_i(t) \le 1$ for some $t (t \in S_i)$. Let T be the maximum $t (t \in S_i)$ which satisfies $L_i(t) \le 1$. If $T \in R_i$, the theorem is proved. Thus, suppose that $T \notin R_i$ and show a contradiction. Since $p_i \in R_i$, $T < p_i$ holds. From the definition of S_i, T is a deadline of one or more tasks. Among these tasks, let τ_j be the one whose priority is lowest. j satisfies $j < i$.

Now, let T^* be a deadline of a task $\tau_k (j \le k < i)$, which satisfies the following conditions:

$T \le T^*$, $T^* + p_k < p_i$, and
there is no deadline of any task $\tau_h (k < h \le i)$ within the time interval $(T, T^* + p_k)$.

First, prove that there exists T^* which satisfies these conditions. To do so, consider the following procedure.

```
$t := T$;    $x := j$;
repeat
    $t'$ := the first time after $t$ which is a deadline
        of some task $\tau_y (y > x)$;
    if $t + p_x \le t'$ then return($t$); /* $T^* = t$. */
    $t := t'$;
    $x := y$ (if $t'$ is a deadline of several tasks,
        let $y$ be the lowest priority task's index);
until $t \ge p_i$;
Error('$T^*$ is not found.');
```

This procedure returns the value T^*. If an error occurs, the value sequence of t during the iteration of the procedure, $T_{j_1}^* (= T), T_{j_2}^*, \ldots, T_{j_m}^* (= p_i)$ satisfies the following condition: (Note that let τ_{j_h} be the task whose priority is the lowest among the tasks which have a deadline at $T_{j_h}^*$.)

$j_1(=j) < j_2 < \ldots < j_m(=i)$.
$T^*_{j_1} < T^*_{j_2} < \ldots < T^*_{j_m}$.
$T^*_{j_h} + p_{j_h} > T^*_{j_{h+1}}(1 \le h \le m-1)$.
There is no deadline of any task whose priority is lower than τ_{j_h} within the time interval $(T, T^*_{j_{h+1}})$.

$T^*_{j_m}(=p_i) \in Q^i_{j_m}(=Q^i_i)$ holds. Since $T^*_{j_{m-1}} + p_{j_{m-1}} > T^*_{j_m}$, $T^*_{j_{m-1}}$ is the last deadline of $\tau_{j_{m-1}}$ before $T^*_{j_m}(\in Q^i_{j_m})$. Thus, $T^*_{j_{m-1}} \in Q^i_{j_{m-1}}$ is satisfied. Similarly, $T^*_{j_h}(h = m-2, m-3, \ldots, 1)$ is the last deadline of τ_{j_h} before $T^*_{j_{h+1}}(\in Q^i_{j_{h+1}})$, thus $T^*_{j_h} \in Q^i_{j_h}(h = m-2, m-3, \ldots, 1)$. Therefore, $T(=T^*_{j_1}) \in Q^i_{j_1}$ and $T \in R_i$. It is contradictory that $T \notin R_i$. Thus, there is a T^* which satisfies the above conditions.

Now consider the interval $[T, T^* + p_k)$ and suppose there are no jobs requested before T. Since there is no deadline of any task whose priority is lower than τ_k in $[T, T^* + p_k)$, there are no new jobs for such a task during that interval. This situation can be regarded as the case that the task system is $\{\tau_1, \ldots, \tau_k\}$ and each task is initiated at some time after T, that is, $I_i \ge T(i = 1, \ldots, k)$ and $I_k = T^*$. Let $f_i(t)$ be the number of τ_i's jobs requested during the time interval $[T, t)$. Let t_0 be the time when the τ_k's job requested at T^* is finished. Since $\{\tau_1, \ldots, \tau_k\}$ is feasible, t_0 satisfies $T^* < t_0 \le T^* + p_k$. $\sum_{h=1}^{k} c_h \cdot f_h(t_0) \le t_0 - T$ is satisfied because all current jobs are finished at t_0.

Since $L_i(T) \le 1$, $\sum_{h=1}^{i} c_h \cdot \lceil T/p_h \rceil \le T$. Now calculete $L_i(t_0)$. Since there are no new jobs for $\{\tau_{k+1} \ldots \tau_i\}$ in $[T, t_0)$, $\sum_{h=1}^{i} c_h \cdot \lceil t_0/p_h \rceil = \sum_{h=1}^{i} c_h \cdot \lceil T/p_h \rceil + \sum_{h=1}^{k} c_h \cdot f_h(t_0) \le T + (t_0 - T) = t_0$.

Therefore, at $t_0(T \le T^* < t_0 \le T^* + p_k \le p_i)$, $L_i(t_0) \le 1$. Let T' be the first deadline after t_0. T' satisfies $L_i(T') \le 1$ and $T' \in S_i$. It is contradictory that T is the maximum $t(t \in S_i)$ which satisfies $L_i(t) \le 1$. Therefore, $T \in R_i$. ■

The new feasibility decision algorithm based on this theorem is shown in Figure 2. At the beginning of the loop by j, *rlist* contains the elements of $\bigcup_{k=j+1}^{i} Q^i_k$. In the *repeat-until* loop, each element $t \in Q^i_j$ is calculated. If $t \notin$ *rlist*, $L_i(t) \le 1$ is tested for feasibility and t is appended to *rlist*. If $L_i(t) \le 1$ for some $t \in$ *rlist*, $L_i \le 1$.

Now estimate the time complexity of the algorithm. $|R_i| \le 2^{i-1}$ and for each element $t \in R_i$, $O(i)$ multiplications and divisions are necessary to test whether $L_i(t) \le 1$. Since this procedure is executed for each $i = 1, \ldots, n$, the number of multiplications and divisions are at most $O(\sum_{i=1}^{n} i \cdot 2^{i-1})$, that is, $O(n \cdot 2^n)$.

Note that the time complexity of this algorithm is independent of the task periods. It is bounded by a function of the number of tasks. Thus, if n is a constant, the time complexity is also a constant.

5 Feasibility of the inverse-deadline scheduling algorithm

This section considers the case in which $d_i \ne p_i$ for some i. This paper assumes that $d_i \le p_i$. For the case in which $d_i > p_i$, Shih et al. have presented a feasibility condition [15].

The inverse-deadline scheduling algorithm [9] is a fixed-priority preemptive scheduling algorithm which is an extension of the rate monotonic scheduling algorithm[2]. It assigns higher priorities to tasks with shorter deadlines and it is the optimal fixed-priority preemptive scheduling algorithm when $d_i \le p_i$ [9].

In this section we label the tasks so that $d_1 \le d_2 \le \ldots \le d_n$. For the inverse-deadline scheduling algorithm, the following properties have been proved.

Property 6 [9] *A critical instant for any task occurs whenever the task is requested simultaneously with requests for all higher priority tasks.* ■

Property 7 [9] *Task system X is feasible by the inverse-deadline scheduling algorithm if the first job of each task meets its deadline.* ■

The following lemma can be shown by a proof similar to that in Ref. [6].

Lemma 3 *Task system X is feasible by the inverse-deadline scheduling algorithm if and only if*
$\tilde{L} = max_{1 \le i \le n} \tilde{L}_i \le 1$, *where*
$\tilde{L}_i = min_{t \in \tilde{S}_i} L_i(t)$, $\quad L_i(t) = \sum_{j=1}^{i} c_j \cdot \lceil t/p_j \rceil / t$,
$\tilde{S}_i = \{d_i\} \cup \{k \cdot p_j \mid \; j = 1, 2, \ldots, i-1;$
$k = 1, 2, \ldots, \lfloor d_i/p_j \rfloor\}$. ■

The reduced scheduling point set $\tilde{R}_i$ for the inverse-deadline scheduling algorithm is defined as follows.

Definition 6 $\tilde{R}_i = \bigcup_{j=1}^{i} \tilde{Q}^i_j$, *where*
$\tilde{Q}^i_i = \{d_i\}$,
$\tilde{Q}^i_j = \{\lfloor t/p_j \rfloor \cdot p_j \mid \; t \in \tilde{Q}^i_k(j+1 \le k \le i)$ *and* $t < \lfloor t/p_j \rfloor \cdot p_j + d_j\}(1 \le j < i)$. ■

[2]The inverse-deadline scheduling algorithm is different from the deadline scheduling algorithm [10] which is a dynamic scheduling algorithm.

By employing a discussion similar to the one in the previous section, the following theorem is proved.

Theorem 2 *Task system X is feasible by the inverse-deadline scheduling algorithm if and only if* $\tilde{L}' = max_{1 \le i \le n} \tilde{L}'_i \le 1$, *where* $\tilde{L}'_i = min_{t \in \tilde{R}_i} L_i(t)$, $\quad L_i(t) = \sum_{j=1}^{i} c_j \cdot \lceil t/p_j \rceil / t$. ■

(Sketch of proof) Assume that $L_i(t) \le 1$. Let T be the maximum $t (t \in \tilde{S}_i)$, which satisfies $L_i(t) \le 1$. Suppose that $T \notin \tilde{R}_i$. From the definition of $\tilde{R}_i$, there exists T^* which satisfies the following:

> T^* is a job request time of a task $\tau_k (k < i)$ (that is, $T^* = m \cdot p_k$ for some integer m),
> $T \le T^*$, $T^* + d_k < p_i$, and
> there is no job request time for any task $\tau_h (k < h < i)$ within the time interval $(T, T^* + d_k)$.

If no such T^* exists, there is a sequence $T^*_{j_1} (= T), T^*_{j_2}, \ldots, T^*_{j_m} (= d_i)$ which satisfies the following:

> Let $\tau_{j_h} (h = 1, \ldots, m-1)$ be the task whose priority is the lowest among the tasks which have a job request time at $T^*_{j_h}$.
> $j_1 (= j) < j_2 < \ldots < j_{m-1} < j_m (= i)$.
> $T^*_{j_1} < T^*_{j_2} < \ldots < T^*_{j_m}$.
> $T^*_{j_h} + d_{j_h} > T^*_{j_{h+1}} (1 \le h \le m-1)$.
> There is no job request time for any task whose priority is lower than τ_{j_h} within the time interval $(T^*_{j_h}, T^*_{j_{h+1}})$.

Since $T^*_{j_h} \in \tilde{Q}^i_{j_h} (1 \le h \le m)$, $T \in \tilde{R}_i$ holds. It is contradictory that $T \notin \tilde{R}_i$. Therefore, there is a T^* which satisfies the above conditions. From the conditions, there is a T' within the interval $(T^*, T^* + d_k]$ which satisfies $L_i(T') \le 1$. It is contradictory that T is the maximum. ■

6 Conclusion

This paper has shown a new necessary and sufficient condition for a periodic real-time task system to be feasible by the rate monotonic scheduling algorithm. We presented a feasibility decision algorithm based on this condition and showed that its time complexity depends solely on the number of tasks. This algorithm can be extended for the feasibility decision of the inverse-deadline scheduling algorithm.

In this algorithm, the definition of the reduced scheduling point set R_i does not require the computation requirement c_i for each task. A better scheduling point set might be obtained if the values of c_i are used effectively.

Acknowledgements

The authors would like to thank Ms. Naoko Kosugi of Keio University. This work was inspired by discussions with her. Thanks are also due to Dr. Rikio Onai of NTT for his encouragement and suggestions, and to Mr. Koji Sato of NTT for his valuable comments.

References

[1] Baker, T. P.: "Stack-Based Scheduling of Real-time Processes," J. of Real-Time Systems, Vol. 3, pp. 67–99 (1991).

[2] Baruah, S. K., Rosier, L. E., and Howell, R. R.: "Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor," J. of Real-Time Systems, Vol. 3, pp. 301–324 (1990).

[3] Cheng, S.-C.: "Scheduling Algorithms for Hard Real-Time Systems," J. A. Stankovic and K. Ramamritham Eds. Hard Real-Time Systems, IEEE Computer Society Press, pp. 150–173 (1987).

[4] Kosugi, N., Takashio, K., and Tokoro, M.: "Modification and Adjustment of Real-Time Tasks with Rate Monotonic Scheduling Algorithm," Proc. of 2nd Workshop on Parallel and Distributed Real-Time Systems (Apr. 1994).

[5] Kuo, T.-W. and Mok, A. K.: "Load Adjustment in Adaptive Real-Time Systems," Proc. of Real-Time Systems Symp. pp. 160–170 (Dec. 1991)

[6] Lehoczky, J., Sha, L., and Ding, Y.: "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," Proc. of Real-Time Systems Symp. pp. 166–171 (Dec. 1989).

[7] Lehoczky, J. P.: "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines," Proc. of Real-Time Systems Symp. pp. 201–209 (Dec. 1990).

[8] Leung, J. Y.-T. and Merril, M. L.: "A Note on Preemptive Scheduling of Periodic, Real-Time Tasks," Information Processing Letters, Vol. 11, No. 3, pp. 115–118 (Nov. 1980).

[9] Leung, J. Y.-T. and Whitehead, J.: "On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks," Performance Evaluation, Vol. 2, pp. 237–250 (1982).

[10] Liu, C. L. and Layland, J. W.: "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," J. of the ACM, Vol. 20, No. 1, pp. 46–61 (Jan. 1973).

[11] Ramos-Thuel, S. and Lehoczky, J. P.: "On-Line Scheduling of Hard Deadline Aperiodic Tasks in Fixed-Priority Systems," Proc. of Real-Time Systems Symp. pp. 160–171 (Dec. 1993).

[12] Sha, L., Rajkumar, R., Lehoczky, J., and Ramamritham, K.: "Mode Change Protocols for Priority-Driven Preemptive Scheduling," J. of Real-Time Systems, Vol. 1, pp. 243–264 (1989).

[13] Sha, L., Rajkumar, R., and Lehoczky, J. P.: "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," IEEE Trans. Computers, Vol. 39, No. 9, pp. 175–185 (Sep. 1990).

[14] Sha, L. and Sathaye, S. S.: "A Systematic Approach to Designing Distributed Real-Time Systems," IEEE Computer, Vol. 26, No. 9, pp. 68–78 (Sep. 1993).

[15] Shih, W. K., Liu, J. W. S., and Liu, C. L.: "Scheduling Periodic Jobs with Deferred Deadlines," Tech. Report UIUCDS-R-90-1593, Univ. of Illinois (1990).

```
function CompletionTimeTest
  ( $n$: integer; $c[n], p[n]$: real);
    /* $n$ : number of tasks.
       $c[n]$ : computation requirement.
       $p[n]$ : interval. */
begin
  for $i := 1$ to $n$ do begin
    /* testing feasibility of $\{\tau_1, \ldots, \tau_i\}$. */
    $t' := \sum_{j=1}^{i} c[j]$;
    repeat
      $t := t'$;
      $t' := \sum_{j=1}^{i} c[j] \cdot \lceil t/p[j] \rceil$;
      if $t' > p[i]$ then return('not feasible');
    until $t = t'$;
  end; /* end of loop by $i$. */
  return('Feasible')
end;
```

Figure 1. Completion-time test algorithm.

```
function ReducedSchedulingPointTest
        ($n$: integer; $c[n], p[n]$: real);
var $u[n]$ : real; /* utilization factor. */
begin
  $u[0] := 0$;
  for $i := 1$ to $n$ do $u[i] := u[i-1] + c[i]/p[i]$;
    /* $u[i]$ : utilization of $\{\tau_1, \ldots, \tau_i\}$. */
  if $u[n] \le n(2^{1/n} - 1)$ then return('Feasible');
  if $u[n] > 1$ then return('not feasible');
  for $i := 1$ to $N$ do begin
    if L_Test($i$)='not feasible' then
        return('not feasible')
  end;
  return('Feasible')
end; /* end of main routine */

function L_Test($i$: integer);
      /* testing feasibility of $\{\tau_1, \ldots, \tau_i\}$. */
var $rlist, rlist2$: list; /* lists of scheduling points. */
begin
  if $u[i] \le i(2^{1/i} - 1)$ then return('Feasible');
  if $p[i] \ge \sum_{m=1}^{i} c[m] \cdot \lceil p[i]/p[m] \rceil$ then
     return('Feasible');
  $rlist := \{p[i]\}$; /* $rlist = Q_i^i$ now. */
  for $j := i-1$ to 1 step $-1$ do begin
    copy $rlist$ to $rlist2$; /* $rlist = \bigcup_{k=j+1}^{i} Q_k^i$ now. */
    $pointer := first(rlist)$;
    repeat /* derive an element in $Q_j^i$. */
      $t$ :=the first element in $rlist2$;
      delete $t$ from $rlist2$;
      $t_0 := \lfloor t/p[j] \rfloor \cdot p[j]$;
      while $element(pointer) < t_0$ do
          $pointer := next(pointer)$;
              /* skip in $rlist$ until $element \ge t_0$. */
      if $element(pointer) \ne t_0$ then
        begin /* $t_0$ is a new element. */
          if $t_0 \ge \sum_{m=1}^{i} c[m] \cdot \lceil t_0/p[m] \rceil$ then
              return('Feasible');
          insert $t_0$ just before $pointer$ in $rlist$;
          $pointer := previous(pointer)$
              /* $element(pointer) = t_0$ now. */
        end
    until $rlist2$ =empty
  end; /* end of loop by $j$. */
  return('not feasible')
end;
```

Figure 2. Reduced scheduling point test algorithm.