Scheduling Periodic Tasks In a Hard Real-Time Environment

Yingfeng Oh

Department of Computer Science
University of Virginia
Charlottesville, VA 22903
USA

Consider the traditional problem of scheduling a set of periodic tasks on a mulitprocessor system, where task deadlines must be guaranteed. We first derive a general schedulability condition for Rate-Monotonic, which reduces the uniprocessor schedulability condition obtained by Liu and Layland and by Serlin, and the multiprocessor schedulability condition recently derived by Burchard, Liebeherr, Oh, and Son to its two specific cases. Then a tight upper bound is obtained for the number of processors required in an optimal schedule for any given set of tasks with a fixed number of tasks and a fixed utilization. Finally, similar conditions are derived for the Earliest Deadline First scheduling. These conditions shed new light on the periodic task scheduling problem.

I. Introduction

In this paper, we consider the problem of scheduling a set of periodic tasks. The task set to be considered is defined as follows:

(1) The requests of each task are periodic, with constant interval between requests.

(2) The deadline constraints specify that each request must be completed before the next request of the same task occurs.

(3) The tasks are independent in that the requests of a task do not depend on the initiation or the completion of the requests of other tasks.

(4) The worst-case run-time (or computation time) for the request of a task is constant for the task. Run-time here refers to the time a processor takes to execute the request without interruption.

The question we are interested in is whether any given set of *n* tasks can be scheduled on *m* processors. In general, this decision problem is NP-complete [5]. Practical solutions to this problem often rely on some kinds of heuristic algorithms, which can deliver approximate solutions in

a short period of time. Heuristic solutions often trade computational time complexity for accuracy of solutions. The approach we take in this paper is to find a schedulability condition for any given set of tasks such that as long as the total utilization or load of the task set is under certain threshold number, the task set can be feasibly scheduled on a certain number of processors. The derivation of this bound subsumes the previous results on schedulability conditions derived by Liu and Layland [9], and by Serlin [12] for Rate-Monotonic scheduling on a single processor system, and by Burchard, Liebeherr, Oh, and Son [2] for Rate-Monotonic scheduling on a multiprocessor system. This tight bound can serve as the basis for constructing more effective heuristic algorithms and for proving tighter worst-case performance guarantee. For more details on how to use schedulability conditions in the design and analysis of heuristics algorithms, reader may refer to [11].

From the description of the task set, it follows that a task is completely defined by two numbers, the run-time of the requests and the request period. We shall denote a task τ_i by the ordered pair (C_i, T_i), where C_i is the computation time and T_i is the period of the requests of the task τ_i. The ratio C_i/T_i is called the utilization (or load) of the task τ_i, and the total utilization (or load) of a set of *n* tasks is given by $U = \sum_{i=1}^{n} C_i/T_i$. All the processors are identical in the sense that the run-time of a task remains the same across all processors.

Tasks can be scheduled for execution on a processor by using a priority-driven algorithm, in which each task is assigned a priority and the task with the highest priority is always the one to be executed. By assigning different priorities to tasks, we therefore determine the schedule of the execution of tasks. A priority assignment algorithm is fixed if the priority of a task remains fixed once it is assigned. Otherwise, it is a dynamic priority assignment algorithm. Here we concern ourselves with priority-driven algorithms only.

If a set of tasks can be scheduled such that all task deadlines can be met by some algorithms, then we say that the task set is *feasible*. If a set of periodic tasks can be feasibly scheduled on a single processor, then the *Rate-Monotonic* (or *RM*) [9] or *Intelligent Fixed Priority* algorithm [12] is optimal for fixed priority assignment, in the sense that no other fixed priority assignment algorithm can schedule a task set which cannot be scheduled by the RM algorithm. The RM algorithm assigns priorities to tasks according to their periods, where the priority of a task is in inverse relationship to its period. In other words, a task with a shorter period is assigned a higher priority. The execution of a low-priority task will be preempted if a high-priority task arrives. Liu and Layland proved that a set of *n* periodic tasks can be feasibly scheduled by the Rate-Monotonic algorithm if

the total utilization of the tasks is no more than a threshold number, which is given by $n\left(2^{1/n}-1\right)$.

One of the important properties of Rate-Monotonic scheduling is that for a single processor system where Rate-Monotonic algorithm is employed to schedule tasks, as long as the CPU utilization of the tasks lies below a certain bound, they will meet their deadlines without the programmer knowing exactly when any given request of a task is running. Even if a transient overload occurs, a fixed subset of the most frequently arrived tasks will still meet their deadlines as long as their total CPU utilization lies below a certain bound. This property puts the real-time software development on a sound analytical footing.

For dynamic priority assignment, the Earliest Deadline First (or EDF) algorithm [9] is optimal in the sense that no other dynamic priority assignment algorithm can schedule a task set which cannot be scheduled by the EDF algorithm. The request of a task is assigned the highest priority if its deadline is the closest. Furthermore, a set of periodic tasks can be feasibly scheduled on a single processor system by the EDF algorithm if and only if its total utilization is no more than one.

Although the schedulability condition, i.e., $\sum_{i=1}^{n} C_i/T_i \le n\,(2^{1/n}-1)$, given by Liu and Layland is simple and elegant, they are pessimistic in nature since the condition is derived under the worst case conditions. Several more efficient conditions were later derived [4, 2, 10, 11]. All these conditions are sufficient but not necessary. The necessary and sufficient condition to schedule a set of periodic tasks using fixed-priority algorithms was given by Joseph and Pandya [6], and by Lehoczky, Sha, and Ding [7].

The results about Rate-Monotonic scheduling are presented in Section II, while similiar results for Earliest Deadline First scheduling are given in Section III. We conclude this paper in Section IV by discussing some remaining issues.

II. Fundamental Conditions for Rate-Monotonic Scheduling

In Theorem 1, we present a general result about rate-monotonic scheduling. It puts a tight upper bound on the number of processors that are required to schedule a set of n tasks such that each task is guaranteed its deadline by the Rate-Monotonic algorithm.

Theorem 1: *For a set of n tasks, if any $m+1$ of the n tasks cannot be feasibly scheduled on a processor by the Rate-Monotonic algorithm, then the total utilization of the n tasks must be greater than $n/\sum_{i=0}^{m} 2^{i/n}$, where $n \ge m+1$, $n \ge 1$ and $m \ge 0$.*

If $m = 0$, then $n/\sum_{i=0}^{m} 2^{i/n} = n$. This is equivalent to saying that if any task cannot be

scheduled on a single processor by the Rate-Monotonic algorithm, then the total utilization of the n tasks must be greater than n. This is trivial true.

If $m = 1$, then $n/\sum_{i=0}^{m} 2^{i/n} = n / \sum_{i=0}^{1} 2^{i/n} = n / \left(2^{1/n} + 1\right)$. This is equivalent to saying that if any two tasks cannot be scheduled on a processor by the Rate-Monotonic algorithm, then the total utilization of the n tasks must be greater than $n / \left(2^{1/n} + 1\right)$. This reduces to the result obtained by Burchard, Liebeherr, Oh, and Son in [2].

If $n = m + 1$, then $n/\sum_{i=0}^{m} 2^{i/n} = n/\sum_{i=0}^{n-1} 2^{i/n} = n\left(2^{1/n} - 1\right)$. This is equivalent to saying that if any n tasks cannot be scheduled on a processor, then the total utilization of the n tasks must be greater than $n\left(2^{1/n} - 1\right)$. This reduces to the result originally obtained by Liu and Layland [9] and Serlin [12].

When n is large, i.e., $n \to \infty$, $n/\sum_{i=0}^{m} 2^{i/n} \to n/(m+1)$. This implies that compared with the utilization bound for the Earliest Deadline First scheduling, the bound given by Theorem 1 for the Rate-Monotonic scheduling is very favorable, i.e., very close to the “best” possible.

In order to prove Theorem 1, we need the following lemma that was proven in [2].

Lemma 1: *If a set of tasks* $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$ *cannot be scheduled on N processors, then the task set* $\Sigma = \{\tau_i' = (C_i', T_i') \mid i = 1, 2, \ldots, n\}$ *given by* $C_i' = T_i' * C_i/T_i$, $T_i' = 2^{V_i}$, *and* $V_i = \log_2 T_i - \lfloor \log_2 T_i \rfloor$ *cannot be scheduled on the N processor either.*

Proof of Theorem 1: Let the set of n tasks be $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$, where C_i and T_i are the computation time and the period of the task τ_i. Note that the theorem is true when either $n = 1$ or $m = 0$. Hence we need only to consider the case where $n \ge 2$ and $m \ge 1$.

According to Lemma 1, we can assume, without loss of generality, that

$$T_1 \le T_2 \le \ldots \le T_n < 2T_1 \tag{1}$$

Since no $m + 1$ of the n tasks can be scheduled together on a processor, the following conditions must hold according to the necessary and sufficient condition [6, 7]:

$$\begin{cases} C_{i_1} + C_{i_2} + \ldots + C_{i_m} + C_{i_{m+1}} > T_{i_1} \\ 2C_{i_1} + C_{i_2} + \ldots + C_{i_m} + C_{i_{m+1}} > T_{i_2} \\ \ldots\ldots\ldots \\ 2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_m} + C_{i_{m+1}} > T_{i_{m+1}} \end{cases} \tag{2}$$

where $1 \le i_1 < \ldots < i_m < i_{m+1} \le n$.

We want to find the minimum of $U = \sum_{i=1}^{n} C_i/T_i$ subject to the constraints of (1), (2), and

(3).

$$0 < C_i \le T_i \qquad i = 1, \ldots, n \tag{3}$$

In order to ensure that the minimum is obtained at some point, we replace “>” by “≥”. This replacement will not affect the minimum.

We proceed in three steps to obtain the minimum of U:

(1) Fix the values $\bar{C} = (C_1, C_2, \ldots, C_n)$ and express $\bar{T} = (T_1, T_2, \ldots, T_n)$ in terms of $\bar{C}$ in the minimization problem.

(2) Reduce the minimization problem to a convex optimization problem by proving that $C_1 \le C_2 \le \ldots \le C_n \le 2C_1$ if the minimum of *U* is achieved.

(3) Solve the optimization problem using standard methods.

Let us define

$S_i = \{2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_m} | i_1 < \ldots < i_m, i_x < i, x \in [1\ldots m]\} \cup$
$\{2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_{m-1}} + C_{i_m} | i_1 < \ldots < i_{m-1}, i_x < i, i_m > i, x \in [1\ldots(m-1)]\} \cup$
$\ldots \cup \{C_{i_1} + C_{i_2} + \ldots + C_{i_m} | i_1 < \ldots < i_m, i_x > i, x \in [1\ldots m]\}$,

where $i = 1, 2, \ldots, n$, $C_i > 0$, $n \ge m + 1$, and $1 \le i_x \le n$. The cardinality of each set S_i is given by $|S_i| = \binom{n-1}{m}$. In other words, there are $\binom{n-1}{m}$ inequalities associated with each T_i term that must be satisfied if any $(m+1)$ tasks cannot be scheduled on a single processor.

Let $\alpha_i = \min(S_i)$, i.e., α_i is the minimum member in value of the set S_i. If we view each member of the set S_i as a summation of *m* terms from $(C_1, C_2, \ldots, C_n)$, then α_i is the minimum summation among the $\binom{n-1}{m}$ ones.

Let us further define that for any *i* and *j* such that $i \in [1\ldots n]$, $j \in [1\ldots n]$, and $i \ne j$, if the term C_j appears in the summation of α_i, then we say that $C_j \in \alpha_i$ (note that α_i is not a set!). Otherwise, $C_j \notin \alpha_i$.

First, let us assume that $\bar{C} = (C_1, C_2, \ldots, C_n)$ is known.

Since

$$\frac{\partial U}{\partial T_i} = -\frac{C_i}{T_i^2} \tag{4}$$

U decreases as T_i increases. But the increase of T_i cannot exceed the limit that is imposed by the constraints in (2). In other words, *U* is minimized when

$T_i = C_i + \min(\{2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_m} | i_1 < \ldots < i_m, i_x < i, x \in [1\ldots m]\} \cup$

$\{2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_{m-1}} + C_{i_m} | i_1 < \ldots < i_{m-1}, i_x < i, i_m > i, x \in [1\ldots(m-1)]\} \cup$
$\ldots \cup \{C_{i_1} + C_{i_2} + \ldots + C_{i_m} | i_1 < \ldots < i_m, i_x > i, x \in [1\ldots m]\}$),
for $i = 1, 2, \ldots, n$.

According to the definition of α_i, we rewrite T_i as $T_i = C_i + \alpha_i$. The minimization problem then becomes

$$U(\bar{C}, \bar{T}) = \sum_{i=1}^{n} C_i / T_i = \sum_{i=1}^{n} C_i / (C_i + m_i) . \quad (5)$$

Next we show that the minimum of *U* is achieved at $C_1 \le C_2 \le \ldots \le C_n \le 2C_1$. This is accomplished by proving the following three claims.

Claim 1: For every $j \in [1\ldots n]$, there exists at least one index *i* such that

$$C_j \in \alpha_i \text{ or } 2C_j \in \alpha_i . \quad (6)$$

Suppose that when the minimum of $U(\bar{C}, \bar{T})$ is achieved and (6) is not satisfied, i.e., for some index *j* there does not exist an index $i \ne j$ such that $C_j \in \alpha_i$ if $i < j$ or $2C_j \in \alpha_i$ if $i > j$. Then $U(\bar{C}, \bar{T})$ can be phrased exclusively in terms of $\bar{C}$. Since

$$\frac{\partial}{\partial C_j} U(\bar{C}, \bar{T}) = \frac{\alpha_j}{(C_j + \alpha_j)^2} > 0 ,$$

meaning that *U* increases as C_j increases, we can lower the value of *U* by lowering the value of C_j. Thus, condition (6) is satisfied for any index *j*.

Claim 2: For every C_i with $i \in [1\ldots n]$, there are at most *m* C_ks such that $C_i > C_k$ for $k > i$ i or $C_i > 2C_k$ for $k < i$.

Suppose that the contrary is true, i.e., there exists an index $i \in [1\ldots n]$ such that there are $l \ge m + 1$ C_ks such that $C_i > C_k$ for $k > i$ or $C_i > 2C_k$ for $k < i$. Then for any $k > i$, $C_i \notin \alpha_k$ because there are *l* terms that are smaller than C_i. Similarly, for any $k < i$, $2C_i \notin \alpha_k$. This is a contradiction to Claim 1. Hence Claim 2 must be true. A corollary of this claim is that if $C_i > C_{i+1}$, then $C_{i+1} \in \alpha_i$.

Claim 3: The minimum of *U* is achieved at $C_1 \le C_2 \le \ldots \le C_n \le 2C_1$.

Suppose that there exists a task set Σ' such that the minimum of *U* is achieved when, for some index $i \in [1\ldots n]$, $C_i > C_{i+1}$ for $i < n$ or $C_i > 2C_1$ for $i = n$. We will only present the proof for the case of $i < n$ since the proof for the case of $i = n$ is symmetric.

Claim 3.1: For such a task set, there must exist an index $k \ne i + 1$ such that either (1) $k \le i$, $C_i = C_k$, and $2C_k \in \alpha_{i+1}$; or (2) $k > i + 1$, $2C_i = C_k$, and $C_k \in \alpha_{i+1}$.

The core of the above claim is that the term $2C_i$ must be included in the summation of α_{i+1} if the value of $2C_i$ is unique. The bulk of the claim covers the case where there might be other C_k that is equal to C_i or $2C_i$ in value. Hence it is apparent that we need only to prove that $2C_i \in \alpha_{i+1}$ assuming that $2C_i$ is unique.

If $2C_i \notin \alpha_{i+1}$, then there are at least m C_ks such that $C_k < 2C_i$ for $k > i + 1$ or $C_k < C_i$ for $k < i$. Since $C_i > C_{i+1}$, there are at least $(m + 1)$ such C_ks that are smaller than C_i. This is a contradiction to Claim 2. Therefore, the claim must be true.

Now we can construct a new task set Σ' from the task set Σ as follows: the computation times of the tasks are given by

$$C'_1 = C_1$$

$$C'_2 = C_2$$

$$\ldots\ldots$$

$$C'_{i-1} = C_{i-1}$$

$$C'_i = C_{i+1}$$

$$C'_{i+1} = C_i$$

$$C'_{i+2} = C_{i+2}$$

$$\ldots\ldots$$

$$C'_n = C_n$$

and the task periods are given by

$$T'_1 = T_1$$

$$T'_2 = T_2$$

$$\ldots\ldots$$

$$T'_{i-1} = T_{i-1}$$

$$T'_i = \alpha_i + C_i$$

$$T'_{i+1} = \alpha_{i+1} + C_{i+1} - C_i$$

$$T'_{i+2} = T_{i+2}$$

$$\ldots\ldots$$

$$T'_n = T_n .$$

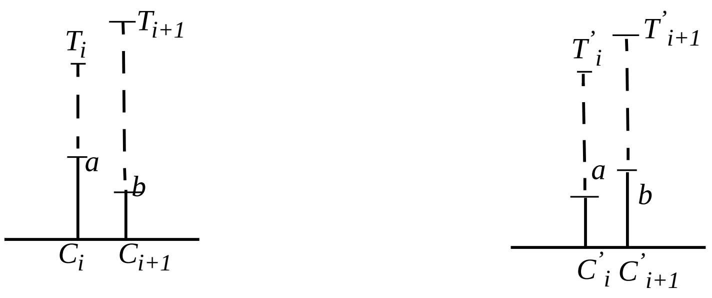

Figure 1: Relationship between two task sets

We want to prove that any $m + 1$ tasks within the newly constructed task set cannot be scheduled on a single processor and $U > U'$, where $U' = \sum_{i=1}^{n} C'_i / T'_i$. Note that this newly constructed task set may not satisfy Claim 1 (but it does not affect the validity of our argument).

First we want to assert that the tasks in Σ' are in the order of non-increasing task periods. From the construction, we need only to consider the order among the tasks T'_{i-1}, T'_i, T'_{i+1}, and T'_{i+2}, since the order among the rest of the tasks does not change.

From the definition, it is immediate that $T'_{i-1} \le T'_i$, since $T'_i = T_i$ and $T'_{i+1} < T_{i+1} \le T'_{i+2}$. Since $C_{i+1} \in \alpha_i$ and $2C_i \in \alpha_{i+1}$, the difference between α_{i+1} and α_i is given by $2C_i - C_{i+1}$, i.e., $\alpha_{i+1} - \alpha_i \ge 2C_i - C_{i+1}$. Hence $T'_{i+1} - T'_i = \alpha_{i+1} - \alpha_i + C_{i+1} - 2C_i \ge 0$. Therefore, $T'_{i-1} \le T'_i \le T'_{i+1} \le T'_{i+2}$, and the whole task set is in non-decreasing order of task periods.

Then let us prove that any $m + 1$ tasks from the task set Σ' cannot be scheduled on a single processor.

Obviously, for any l such that $l < i$ or $l > i + 1$, the inequalities for the new task set related to T'_l hold, since the exchange of the values of C_{i+1} and C_i does not affect the original inequalities. More specifically, for each T'_l, there are $\binom{n-1}{m}$ inequalities to verify. According to the definition of α_l, if $\alpha'_l + C'_l \ge T_l'$, then the rest of the $\binom{n-1}{m} - 1$ inequalities holds. Since the exchange of the values of C_{i+1} and C_i does not affect the equalities $\alpha'_l = \alpha_l$ for $l < i$ or $l > i + 1$, we have $\alpha'_l + C'_l \ge T_l'$.

Now we shall verify that $\alpha'_i + C'_i \ge T'_i$ and $\alpha'_{i+1} + C'_{i+1} \ge T'_{i+1}$.

Case (i): We shall prove the claim that $\alpha'_i \ge \alpha_i - C_{i+1} + C_i$.

Since $C_i > C_{i+1}$, then $C_{i+1} \in \alpha_i$ according to Claim 2. Then $C_i \in \alpha_i'$ must be true, i.e., the term C_i (or a term C_x with $C_i = C_x$) must be included. Otherwise, for the original C_i in the old task set, there are $(m+1)$ C_ks such that $C_i > C_k$ for $k > i$ or $C_i > 2C_k$ for $k < i$ and the minimum of U is achieved.

Therefore the difference between α_i' and α_i is given by $C_{i+1} - C_i$, i.e., $\alpha_i' \geq \alpha_i - C_{i+1} + C_i$.

$$\alpha_i' + C_i' \geq \alpha_i - C_{i+1} + C_i + C_{i+1} = \alpha_i + C_i = T_i = T_i' .$$

Case (i+1): We shall prove that $\alpha_{i+1}' \geq \alpha_{i+1} - 2C_i + C_{i+1}$.

Since $C_i > C_{i+1}$, then $C_{i+1} \in \alpha_i$ according to Claim 2. Furthermore, there are at most m C_ks such that $C_i > C_k$ for $k > i$ or $C_i > 2C_k$ for $k < i$. Then there are at most $(m-1)$ C_ks such that $C_{i+1} > C_k$ for $k > i+1$ or $C_{i+1} > 2C_k$ for $k < i$

Since $C_i \in \alpha_{i+1}$, it follows that $2C_i' \in \alpha_{i+1}'$. The difference between α_{i+1}' and α_{i+1} is given by $2C_i' - 2C_i$, i.e., $\alpha_{i+1}' \geq \alpha_{i+1} - 2C_i + 2C_{i+1} \geq \alpha_{i+1} - 2C_i + C_{i+1}$.

$$\alpha_{i+1}' + C_{i+1}' \geq \alpha_{i+1} - 2C_i + C_{i+1} + C_i = \alpha_{i+1} + C_{i+1} - C_i = T_{i+1}' .$$

Therefore, any $(m+1)$ tasks in the new task set cannot be scheduled on a single processor by the rate-monotonic algorithm.

Finally, let us prove that $U > U'$.

$$U - U' = \left(\frac{C_i}{T_i} + \frac{C_{i+1}}{T_{i+1}}\right) - \left(\frac{C_i'}{T_i'} + \frac{C_{i+1}'}{T_{i+1}'}\right) = \left(\frac{C_i}{T_i} + \frac{C_{i+1}}{T_{i+1}}\right) - \left(\frac{C_{i+1}}{T_i} + \frac{C_i}{T_{i+1}'}\right)$$

Since $T_i = T_i' < T_{i+1}' < T_{i+1}$, $C_i > C_{i+1}$, and $T_i \geq C_i$, we have $U > U'$.

Therefore, the minimum of $U(x)$ is achieved when

$$C_1 \leq C_2 \leq \ldots \leq C_n \leq 2C_1 .$$

According to the definition of α_i, we have

$$\alpha_i = C_{i+1} + \ldots + C_{i+m} \text{ for } i = 1, 2, \ldots, n-m \text{, and}$$

$$\alpha_i = C_{i+1} + \ldots + C_n + 2C_1 + \ldots + 2C_{i-(n-m)} \text{ for } i = n-m+1, \ldots, n .$$

In other words, the minimum of $U(x)$ is achieved when the task periods satisfy

$$T_i = C_i + C_{i+1} + \ldots + C_{i+m} \text{ for } i = 1, 2, \ldots, n-m \text{, and}$$

$$T_i = C_i + C_{i+1} + \ldots + C_n + 2C_1 + \ldots + 2C_{i-(n-m)} \text{ for } i = n-m+1, \ldots, n .$$

The minimization problem of $U = \sum_{i=1}^{n} C_i / T_i$ now becomes a convex optimization problem.

Finally, we solve the problem by using one of the standard method.

$$\sum_{i=1}^{n} C_i / T_i = \sum_{i=1}^{n-m} \frac{C_i}{\sum_{j=0}^{m} C_{i+j}} + \sum_{i=n-m+1}^{n} \frac{C_i}{\sum_{j=i}^{n} C_j + \sum_{j=1}^{i-(n-m)} 2C_j} \tag{7}$$

Let us define

$$x_i = \log \frac{C_{i+1}}{C_i} \tag{8}$$

for $i = 1, 2, \ldots, n-1$, and

$$x_n = \log \frac{2C_1}{C_n} . \tag{9}$$

Then $\sum_{i=1}^{n} x_i = 1$.

We want to minimize

$$U = \sum_{i=1}^{n-m+1} \frac{1}{1 + \sum_{j=0}^{m-1} 2^{\sum_{k=0}^{j} x_{i+k}}} +$$

$$\sum_{i=n-m+2}^{n} \frac{1}{1 + \sum_{j=0}^{n-i} 2^{\sum_{k=0}^{j} x_{i+k}} + \sum_{j=1}^{i-(n-m+1)} 2^{\sum_{k=i}^{n} x_k + \sum_{k=1}^{j} x_k}} \tag{10}$$

subject to

$$x_i > 0, i = 1, 2, \ldots, n \tag{11}$$

$$\sum_{i=1}^{n} x_i = 1. \tag{12}$$

Since the function and its conditions are symmetric under permutation of indices, the minimum is achieved at $x_i = 1/n$.

Therefore $U = n / \sum_{i=0}^{m} 2^{i/n}$.

Next we show that there indeed exists some task sets such that the above bound is achieved. In other words, the given bound is tight.

Let ε be an arbitrarily small positive number and *a* be a positive number. Then for a task set given by

$$\tau_i = (C_i, T_i) = \left(a2^{i/n} + \varepsilon, a2^{i/n}\left(\sum_{j=0}^{m} 2^{j/n}\right)\right),$$

for $i = 1, 2, \ldots, n$, any $(m + 1)$ of the *n* tasks cannot be scheduled on a single processor. ■

Theorem 2: *For any given set of n tasks* $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, 2, \ldots, n\}$, *no more than* $\min(n, \lceil 1/\{\log[1 + n(2^{1/n} - 1)/U] - 1/n\}\rceil)$ *processors are required in an optimal schedule, such that the tasks can be feasibly scheduled by the Rate-Monotonic algorithm, where* $U = \sum_{i=1}^{n} C_i/T_i$.

Proof: For any given set of *n* tasks with a utilization of $U = \sum_{i=1}^{n} C_i/T_i$, it is apparent that at most *n* processors are needed for the feasible scheduling of the tasks by the Rate-Monotonic algorithm, as long as $C_i/T_i \le 1$ for $i = 1, 2, \ldots, n$.

What we want to find out is, with regard to *n* and *U* (i.e., the set of the task sets each with *n* tasks and a utilization of *U*), the maximum numbers of processors that are necessary to feasibly schedule the task sets in all the optimal schedules. In other words, we are trying to find the maximum of processors used in an optimal schedule in the worst cases. The worst cases are thus defined as such that for some task sets with *n* tasks each and a utilization of *U*, the number of processors required by these task sets is no smaller than those required by any set of *n* tasks with the same utilization of *U*.

Before we can proceed further, we need to define a procedure for finding the minimum number of processors for any given set Σ of *n* tasks with a utilization of *U*, i.e., the number of processors in an optimal schedule. In other words, we will design the optimal algorithm for scheduling a set of periodic tasks, the one which always returns the minimum number of processors for any given set of tasks. By describing such algorithm, we actually define a canonical form for the optimal schedule, to be used in the proof that follows. Note that the Rate-Monotonic algorithm is employed by each processor as its scheduling algorithm.

(1) $m = 1$; $S \leftarrow \Sigma$;

(2) Find the largest subset $S_m \subseteq S$ of tasks such that

(i) any $(m + 1)$ tasks in S_m cannot be scheduled on a single processor;

(ii) any *i* tasks in S_m cannot be scheduled with any $(m - i + 1)$ tasks in $S - S_m$ for $i = 1, 2, \ldots, m$;

(iii) for each task in S_m, there exist some groups of $(m - 1)$ tasks in S_m that these *m*

tasks can be feasibly scheduled on a single processor.

Then $S \leftarrow S - S_m$.

We give the following procedure that can compute such S_m:

(a) $S_m \leftarrow \{\}$;

(b) Rename the task set S such that the $|S|$ tasks are indexed from 1 to $|S|$.

For all $j = 1, 2, \ldots, m$, $\tau_{i_j} \in S$, $i_j = 1, \ldots, |S|$, and $i_k \neq i_l$ with $k, l \in [1 \ldots m]$, repeat the following until either $S = \{\}$ or the conditions (I) and (II) are not true.

If

(I) these m tasks $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ can be scheduled on a processor; and

(II) any i tasks among the m tasks $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ cannot be scheduled with any $(m - i + 1)$ tasks in $S - S_m$ on a processor for $i = 1, 2, \ldots, m$

then $S_m \leftarrow S_m + \{\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}\}$ and $S \leftarrow S - \{\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}\}$

(3) If $S \neq \{\}$, then $m \leftarrow m + 1$ and goto (2).

From the algorithm, it is apparent that in the final schedule, if l is the maximum number of tasks assigned on a processor, then any $(l + 1)$ tasks among the n tasks cannot be scheduled on a processor.

If $U > \frac{n}{1 + 2^{1/n}}$, then in the worst cases, any two of the n tasks cannot be scheduled in a single processor. Hence, the number of processors required for the scheduling of this set is n. Since

$$\left\lceil \frac{1}{\log\left[1 + n\left(2^{1/n} - 1\right)/U\right] - 1/n} \right\rceil > n$$

for $U \geq \frac{n}{1 + 2^{1/n}}$, the theorem holds.

If $U \leq \frac{n}{1 + 2^{1/n}}$, we claim that in the worst cases, the maximum numbers of processors in the optimal schedules, that are required for the feasible scheduling of the task sets, say P, is given by $\left\lceil \frac{n}{m} \right\rceil$, where m is determined by finding the smallest m such that

$$U \geq \frac{n}{\sum_{i=0}^{m} 2^{i/n}}. \tag{13}$$

Let us note that such a number of m does exist, since $U \leq n / \sum_{i=0}^{m} 2^{i/n}$ for $m = 1$, and the function $f(m) = n / \sum_{i=0}^{m} 2^{i/n}$ is a monotonically decreasing function with regard to m. Furthermore, by solving inequality (13), we obtain

$$m \geq n \log\left[1 + \frac{n\left(2^{1/n} - 1\right)}{U}\right] - 1. \tag{14}$$

Hence,

$$P = \left\lceil \frac{n}{m} \right\rceil \le \left\lceil \frac{1}{\log\,[1 + n\,(2^{1/n} - 1)\,/U] \; - 1/n} \right\rceil. \tag{15}$$

Now suppose that the claim is not true, i.e., there exists a worst case where the number of processors required is Q such that $Q > P$. Then let n_i be the number of processors on each of which i tasks are assigned and k and l are the minimum and the maximum number of tasks assigned to a processor, respectively. Then $Q = \sum_{i=k}^{l} n_i$ and $n = \sum_{i=k}^{l} i n_i$.

If $l < m$, i.e., each processor is assigned less than m tasks, then since any $(l + 1)$ of the n tasks cannot be scheduled on a processor, the total utilization of the task set must be greater than $n/\sum_{i=0}^{l} 2^{i/n}$. Since m is the smallest number such that $U \ge n/\sum_{i=0}^{m} 2^{i/n}$, therefore $U < n/\sum_{i=0}^{l} 2^{i/n}$. This indicates that the task set has a greater utilization and thus a contradiction is introduced.

If $l > m$, i.e., some processors are assigned more than m tasks, then since any $(l + 1)$ of the n tasks cannot be scheduled on a processor, the total utilization of the task set must be greater than $\frac{n}{\sum_{i=0}^{l} 2^{i/n}}$. Since m is the smallest number such that $U \ge \frac{n}{\sum_{i=0}^{m} 2^{i/n}}$,

$$U \ge \frac{n}{\sum_{i=0}^{m} 2^{i/n}} > \frac{n}{\sum_{i=0}^{l} 2^{i/n}}.$$

This results in a contradiction.

Therefore, the theorem must be true. ■

III. Earliest Deadline First Scheduling of Periodic Tasks

Before establishing the first result for the Earliest Deadline First algorithm, we restate a result that was first proven in [9] and will serve as a basis for our proof.

Lemma 2: *A set of n tasks* $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$ *can be feasibly scheduled by the Earliest Deadline First algorithm if and only if* $\sum_{i=1}^{n} C_i/T_i \le 1$.

Theorem 3: *For a set of n tasks, if any* $m + 1$ *of the n tasks cannot be feasibly scheduled on a processor by the Earliest Deadline First algorithm, then the total utilization of the n tasks must be greater than* $n/(m + 1)$, *where* $n \ge m + 1$, $n \ge 1$ *and* $m \ge 0$.

Proof: Let the set of n tasks be $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$, where C_i and T_i are the computation time and the period of the task τ_i. Note that the theorem is true when either $n = 1$

or $m = 0$. Hence we need only to consider the case where $n \geq 2$ and $m \geq 1$.

Since any $(m+1)$ of n tasks cannot be feasibly scheduled on a processor by the Earliest Deadline First algorithm, then

$$\sum_{j=1}^{m+1} u_{i_j} > 1 \tag{16}$$

for all $j = 1, 2, \ldots, m+1$, $i_j \in [1 \ldots n]$, $i_k \neq i_l$, and $k, l \in [1 \ldots (m+1)]$, where $u_i = C_i / T_i$. Note that there are a total of $\binom{n}{m+1}$ inequalities in (16).

Summing up the inequalities in (16) yields

$$\binom{n-1}{m} \sum_{i=1}^{n} u_i > \binom{n}{m+1}. \tag{17}$$

Hence, $\sum_{i=1}^{n} u_i > n / (m+1)$ ■

Now we are ready to state the second result for the Earliest Deadline First algorithm.

Theorem 4: *For any given set of n tasks* $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$, *no more than* $\min(n, \lceil U + U^2 / (n - U) \rceil)$ *processors are required in an optimal schedule, such that the task set can be feasibly with the Earliest Deadline First algorithm, where* $U = \sum_{i=1}^{n} C_i / T_i$.

Proof: For any given set of n tasks with a utilization of $U = \sum_{i=1}^{n} C_i / T_i$, it is apparent that at most n processors are needed for the feasible scheduling of the tasks by the Earliest Deadline First algorithm, as long as $C_i / T_i \leq 1$ for $i = 1, 2, \ldots, n$.

What we want to find out is, with regard to *n* and *U* (i.e., the set of the task sets each with *n* tasks and a utilization of *U*), the maximum numbers of processors that are necessary to feasibly schedule the task sets in all the optimal schedules. In other words, we are trying to find the maximum of processors used in an optimal schedule in the worst cases. The worst cases are thus defined as such that for some task sets with *n* tasks each and a utilization of *U*, the number of processors required by these task sets is no smaller than those required by any set of *n* tasks with the same utilization of *U*.

Before we can proceed further, we need to define a procedure for finding the minimum number of processors for any given set Σ of *n* tasks with a utilization of *U*. In other words, we will design the optimal algorithm for scheduling a set of periodic tasks, the one which always returns the minimum number of processors for any given set of tasks. By describing such algorithm, we actually define a canonical form for the optimal schedule, to be used in the proof that follows. Note that the Earliest Deadline First algorithm is employed by each processor as its scheduler.

An optimal algorithm is given as follows:

```
(1) $m = 1$; $S \leftarrow \Sigma$;
(2) Find the largest subset $S_m \subseteq S$ of tasks such that
    (i)   any $(m + 1)$ tasks in $S_m$ cannot be scheduled on a single processor;
    (ii)  any $i$ tasks in $S_m$ cannot be scheduled with any $(m - i + 1)$ tasks in $S - S_m$ for
          $i = 1, 2, \ldots, m$;
    (iii) for each task in $S_m$, there exist some groups of $(m - 1)$ tasks in $S_m$ that these $m$
          tasks can be feasibly scheduled on a single processor.
    Then $S \leftarrow S - S_m$.
    We give the following procedure that can compute such $S_m$:
    (a) $S_m \leftarrow \{\}$;
    (b) Rename the task set $S$ such that the $|S|$ tasks are indexed from 1 to $|S|$.
        For all $j = 1, 2, \ldots, m$, $\tau_{i_j} \in S$, $i_j = 1, \ldots, |S|$, and $i_k \neq i_l$ with $k, l \in [1 \ldots m]$,
        repeat the following until either $S = \{\}$ or the conditions (I) and (II) are not true.
            If
            (I)  these $m$ tasks $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ can be scheduled on a processor; and
            (II) any $i$ tasks among the $m$ tasks $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ cannot be scheduled with any
                 $(m - i + 1)$ tasks in $S - S_m$ on a processor for $i = 1, 2, \ldots, m$
            then $S_m \leftarrow S_m + \{\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}\}$ and $S \leftarrow S - \{\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}\}$
(3) If $S \neq \{\}$, then $m \leftarrow m + 1$ and goto (2).
```

From the algorithm, it is apparent that in the final schedule, if l is the maximum number of tasks assigned on a processor, then any $(l + 1)$ tasks among the n tasks cannot be scheduled on a processor.

If $U > n/2$, then in the worst cases, any two of the n tasks cannot be scheduled in a single processor. Hence, the number of processors required for the scheduling of this set is n. Since

$$\lceil U + U^2/(n - U) \rceil > n$$

for $U \geq n/2$, the theorem holds.

If $U \leq n/2$, we claim that in the worst cases, the maximum numbers of processors in the optimal schedules, that are required for the feasible scheduling of the task sets, say P, is given by $\left\lceil \frac{n}{m} \right\rceil$, where m is determined by finding the smallest m such that

$$U \geq \frac{n}{m+1}. \tag{18}$$

Let us note that such a number of *m* does exist, since $U \leq n/(m+1)$ for *m* = 1, and the function $f(m) = n/(m+1)$ is a monotonically decreasing function with regard to *m*. Furthermore, by solving inequality (13), we obtain

$$m \geq \frac{n}{U} - 1. \tag{19}$$

Hence,

$$P = \left\lceil \frac{n}{m} \right\rceil \leq \left\lceil U + \frac{U^2}{n-U} \right\rceil. \tag{20}$$

Now suppose that the claim is not true, i.e., there exists a worst case where the number of processors required is *Q* such that *Q* > *P*. Then let n_i be the number of processors on each of which *i* tasks are assigned and *k* and *l* are the minimum and the maximum number of tasks assigned to a processor, respectively. Then $Q = \sum_{i=k}^{l} n_i$ and $n = \sum_{i=k}^{l} i n_i$.

If $l < m$, i.e., each processor is assigned less than *m* tasks, then since any (*l* + 1) of the *n* tasks cannot be scheduled on a processor, the total utilization of the task set must be greater than $n/(l+1)$. Since *m* is the smallest number such that $U \geq n/(m+1)$, therefore $U < n/(l+1)$. This indicates that the task set now has a greater utilization and thus a contradiction is introduced.

If $l > m$, i.e., some processors are assigned more than *m* tasks, then since any (*l* + 1) of the *n* tasks cannot be scheduled on a processor, the total utilization of the task set must be greater than $\frac{n}{l+1}$. Since *m* is the smallest number such that $U \geq \frac{n}{m+1}$,

$$U \geq \frac{n}{m+1} > \frac{n}{l+1}.$$

This results in a contradiction.

Therefore, the theorem must be true. ■

IV. Concluding Remarks

The discovery of the general schedulability condition as given in Theorem 1 for Rate-Monotonic scheduling was the direct result of our attempt to design and analyze effective heuristic algorithms for scheduling periodic tasks on a multiprocessor system. In analyzing various heuristic algorithms for their worst-case performance, we realize that for any given set of tasks, some upper

bounds on the number of processors must be established. However, we also learn from our proof that it is very difficult to establish such bounds, because it usually involves the solutions to non-convex optimization problems.

Further questions remain as how to derive meaningful conditions for periodic tasks which share resources in a multiprocessor environment. Also of interest to our research is the derivation of similar schedulability conditions for scheduling periodic tasks whose deadlines are shorter than their periods. A number of researchers have addressed this problem to some extents, see [1] for example.

Acknowledgments: I would like to thank Dr. Sang H. Son for his support.

References

[1] N.C. AUDSLEY, A. BURNS, M.F. RICHARDSON, K.W. TINDELL, AND A.J. WELLINGS. “Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling,” *Software Engineering Journal* **8(5)**: 284-292 (1993).

[2] A. BURCHARD, J. LIEBEHERR, Y. OH, AND S.H. SON. “Assigning Real-Time Tasks to Homogeneous Multiprocessor Systems,” *IEEE Transactions on Computer* (to appear).

[3] S. DAVARI AND S.K. DHALL. “An On Line Algorithm for Real-Time Tasks Allocation,” *IEEE Real-Time Systems Symposium*, 194-200 (1986).

[4] S.K. DHALL AND C.L. LIU. “On a Real-Time Scheduling Problem,” *Operations Research* **26**: 127-140 (1978).

[5] M.R. GAREY AND D.S. JOHNSON. *Computers and Intractability: A Guide to the Theory of NP-completeness*, W.H. Freeman and Company, NY, 1978.

[6] M. JOSEPH AND P. PANDYA. “Finding Response Times in a Real-Time System,” *The Computer Journal* **29(5)**: 390-395 (1986).

[7] J. LEHOCZKY, L. SHA, AND Y. DING. “The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior,” *IEEE Real-Time Symposium*, 166-171 (1989).

[8] J.Y.T. LEUNG AND J. WHITEHEAD. “On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks,” *Performance Evaluation* **2**: 237-250 (1982).

[9] C.L. LIU AND J. LAYLAND. “Scheduling Algorithms for Multiprogramming in a Hard

Real-Time Environment,” *J. Assoc. Comput. Machinery* **10(1)**: 174-189 (1973).

[10] Y. OH AND S.H. SON. “Allocating Fixed-priority Periodic Tasks on Multiprocessor Systems,” *Journal of Real-Time Systems* (to appear).

[11] Y. OH. The Design and Analysis of Scheduling Algorithms for Real-Time and Fault-Tolerant Computer Systems. Ph.D. Dissertation, Dept. of Computer Science, University of Virginia, May 1994.

[12] P. SERLIN, “Scheduling of Time Critical Processes,” *Proceedings of the Spring Joint Computers Conference* **40**: 925-932 (1972).

[13] L. SHA, R. RAJKUMAR, AND J.P. LEHOCZKY. “Priority Inheritance Protocols: An Approach to Real-Time Synchronization,” *IEEE Transactions on Computers* **39(9)**: 1175-1185 (1990).

Scheduling Periodic Tasks In a Hard Real-Time Environment

Yingfeng Oh

Department of Computer Science
University of Virginia
Charlottesville, VA 22903
USA

Consider the traditional problem of scheduling a set of periodic tasks on a mulitprocessor system, where task deadlines must be guaranteed. We first derive a general schedulability condition for Rate-Monotonic, which reduces the uniprocessor schedulability condition obtained by Liu and Layland and by Serlin, and the multiprocessor schedulability condition recently derived by Burchard, Liebeherr, Oh, and Son to its two specific cases. Then a tight upper bound is obtained for the number of processors required in an optimal schedule for any given set of tasks with a fixed number of tasks and a fixed utilization. Finally, similar conditions are derived for the Earliest Deadline First scheduling. These conditions shed new light on the periodic task scheduling problem.

I. Introduction

In this paper, we consider the problem of scheduling a set of periodic tasks. The task set to be considered is defined as follows:

(1) The requests of each task are periodic, with constant interval between requests.

(2) The deadline constraints specify that each request must be completed before the next request of the same task occurs.

(3) The tasks are independent in that the requests of a task do not depend on the initiation or the completion of the requests of other tasks.

(4) The worst-case run-time (or computation time) for the request of a task is constant for the task. Run-time here refers to the time a processor takes to execute the request without interruption.

The question we are interested in is whether any given set of *n* tasks can be scheduled on *m* processors. In general, this decision problem is NP-complete [5]. Practical solutions to this problem often rely on some kinds of heuristic algorithms, which can deliver approximate solutions in

a short period of time. Heuristic solutions often trade computational time complexity for accuracy of solutions. The approach we take in this paper is to find a schedulability condition for any given set of tasks such that as long as the total utilization or load of the task set is under certain threshold number, the task set can be feasibly scheduled on a certain number of processors. The derivation of this bound subsumes the previous results on schedulability conditions derived by Liu and Layland [9], and by Serlin [12] for Rate-Monotonic scheduling on a single processor system, and by Burchard, Liebeherr, Oh, and Son [2] for Rate-Monotonic scheduling on a multiprocessor system. This tight bound can serve as the basis for constructing more effective heuristic algorithms and for proving tighter worst-case performance guarantee. For more details on how to use schedulability conditions in the design and analysis of heuristics algorithms, reader may refer to [11].

From the description of the task set, it follows that a task is completely defined by two numbers, the run-time of the requests and the request period. We shall denote a task τ_i by the ordered pair (C_i, T_i), where C_i is the computation time and T_i is the period of the requests of the task τ_i. The ratio C_i/T_i is called the utilization (or load) of the task τ_i, and the total utilization (or load) of a set of *n* tasks is given by $U = \sum_{i=1}^{n} C_i/T_i$. All the processors are identical in the sense that the run-time of a task remains the same across all processors.

Tasks can be scheduled for execution on a processor by using a priority-driven algorithm, in which each task is assigned a priority and the task with the highest priority is always the one to be executed. By assigning different priorities to tasks, we therefore determine the schedule of the execution of tasks. A priority assignment algorithm is fixed if the priority of a task remains fixed once it is assigned. Otherwise, it is a dynamic priority assignment algorithm. Here we concern ourselves with priority-driven algorithms only.

If a set of tasks can be scheduled such that all task deadlines can be met by some algorithms, then we say that the task set is *feasible*. If a set of periodic tasks can be feasibly scheduled on a single processor, then the *Rate-Monotonic* (or *RM*) [9] or *Intelligent Fixed Priority* algorithm [12] is optimal for fixed priority assignment, in the sense that no other fixed priority assignment algorithm can schedule a task set which cannot be scheduled by the RM algorithm. The RM algorithm assigns priorities to tasks according to their periods, where the priority of a task is in inverse relationship to its period. In other words, a task with a shorter period is assigned a higher priority. The execution of a low-priority task will be preempted if a high-priority task arrives. Liu and Layland proved that a set of *n* periodic tasks can be feasibly scheduled by the Rate-Monotonic algorithm if

the total utilization of the tasks is no more than a threshold number, which is given by $n\left(2^{1/n}-1\right)$.

One of the important properties of Rate-Monotonic scheduling is that for a single processor system where Rate-Monotonic algorithm is employed to schedule tasks, as long as the CPU utilization of the tasks lies below a certain bound, they will meet their deadlines without the programmer knowing exactly when any given request of a task is running. Even if a transient overload occurs, a fixed subset of the most frequently arrived tasks will still meet their deadlines as long as their total CPU utilization lies below a certain bound. This property puts the real-time software development on a sound analytical footing.

For dynamic priority assignment, the Earliest Deadline First (or EDF) algorithm [9] is optimal in the sense that no other dynamic priority assignment algorithm can schedule a task set which cannot be scheduled by the EDF algorithm. The request of a task is assigned the highest priority if its deadline is the closest. Furthermore, a set of periodic tasks can be feasibly scheduled on a single processor system by the EDF algorithm if and only if its total utilization is no more than one.

Although the schedulability condition, i.e., $\sum_{i=1}^{n} C_i/T_i \le n\,(2^{1/n}-1)$, given by Liu and Layland is simple and elegant, they are pessimistic in nature since the condition is derived under the worst case conditions. Several more efficient conditions were later derived [4, 2, 10, 11]. All these conditions are sufficient but not necessary. The necessary and sufficient condition to schedule a set of periodic tasks using fixed-priority algorithms was given by Joseph and Pandya [6], and by Lehoczky, Sha, and Ding [7].

The results about Rate-Monotonic scheduling are presented in Section II, while similiar results for Earliest Deadline First scheduling are given in Section III. We conclude this paper in Section IV by discussing some remaining issues.

II. Fundamental Conditions for Rate-Monotonic Scheduling

In Theorem 1, we present a general result about rate-monotonic scheduling. It puts a tight upper bound on the number of processors that are required to schedule a set of n tasks such that each task is guaranteed its deadline by the Rate-Monotonic algorithm.

Theorem 1: *For a set of n tasks, if any $m+1$ of the n tasks cannot be feasibly scheduled on a processor by the Rate-Monotonic algorithm, then the total utilization of the n tasks must be greater than $n/\sum_{i=0}^{m} 2^{i/n}$, where $n \ge m+1$, $n \ge 1$ and $m \ge 0$.*

If $m = 0$, then $n/\sum_{i=0}^{m} 2^{i/n} = n$. This is equivalent to saying that if any task cannot be

scheduled on a single processor by the Rate-Monotonic algorithm, then the total utilization of the n tasks must be greater than n. This is trivial true.

If $m = 1$, then $n/\sum_{i=0}^{m} 2^{i/n} = n / \sum_{i=0}^{1} 2^{i/n} = n / \left(2^{1/n} + 1\right)$. This is equivalent to saying that if any two tasks cannot be scheduled on a processor by the Rate-Monotonic algorithm, then the total utilization of the n tasks must be greater than $n / \left(2^{1/n} + 1\right)$. This reduces to the result obtained by Burchard, Liebeherr, Oh, and Son in [2].

If $n = m + 1$, then $n/\sum_{i=0}^{m} 2^{i/n} = n/\sum_{i=0}^{n-1} 2^{i/n} = n\left(2^{1/n} - 1\right)$. This is equivalent to saying that if any n tasks cannot be scheduled on a processor, then the total utilization of the n tasks must be greater than $n\left(2^{1/n} - 1\right)$. This reduces to the result originally obtained by Liu and Layland [9] and Serlin [12].

When n is large, i.e., $n \to \infty$, $n/\sum_{i=0}^{m} 2^{i/n} \to n/(m+1)$. This implies that compared with the utilization bound for the Earliest Deadline First scheduling, the bound given by Theorem 1 for the Rate-Monotonic scheduling is very favorable, i.e., very close to the “best” possible.

In order to prove Theorem 1, we need the following lemma that was proven in [2].

Lemma 1: *If a set of tasks $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$ cannot be scheduled on N processors, then the task set $\Sigma = \{\tau'_i = (C'_i, T'_i) \mid i = 1, 2, \ldots, n\}$ given by $C'_i = T'_i * C_i / T_i$, $T'_i = 2^{V_i}$, and $V_i = \log_2 T_i - \lfloor \log_2 T_i \rfloor$ cannot be scheduled on the N processor either.*

Proof of Theorem 1: Let the set of n tasks be $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$, where C_i and T_i are the computation time and the period of the task τ_i. Note that the theorem is true when either $n = 1$ or $m = 0$. Hence we need only to consider the case where $n \geq 2$ and $m \geq 1$.

According to Lemma 1, we can assume, without loss of generality, that

$$T_1 \leq T_2 \leq \ldots \leq T_n < 2T_1 \tag{1}$$

Since no $m + 1$ of the n tasks can be scheduled together on a processor, the following conditions must hold according to the necessary and sufficient condition [6, 7]:

$$\begin{cases} C_{i_1} + C_{i_2} + \ldots + C_{i_m} + C_{i_{m+1}} > T_{i_1} \\ 2C_{i_1} + C_{i_2} + \ldots + C_{i_m} + C_{i_{m+1}} > T_{i_2} \\ \quad\cdots\cdots\cdots \\ 2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_m} + C_{i_{m+1}} > T_{i_{m+1}} \end{cases} \tag{2}$$

where $1 \leq i_1 < \ldots < i_m < i_{m+1} \leq n$.

We want to find the minimum of $U = \sum_{i=1}^{n} C_i / T_i$ subject to the constraints of (1), (2), and

(3).

$$0 < C_i \le T_i \qquad i = 1, \ldots, n \tag{3}$$

In order to ensure that the minimum is obtained at some point, we replace “>” by “≥”. This replacement will not affect the minimum.

We proceed in three steps to obtain the minimum of U:

(1) Fix the values $\bar{C} = (C_1, C_2, \ldots, C_n)$ and express $\bar{T} = (T_1, T_2, \ldots, T_n)$ in terms of $\bar{C}$ in the minimization problem.

(2) Reduce the minimization problem to a convex optimization problem by proving that $C_1 \le C_2 \le \ldots \le C_n \le 2C_1$ if the minimum of U is achieved.

(3) Solve the optimization problem using standard methods.

Let us define

$S_i = \{2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_m} | i_1 < \ldots < i_m, i_x < i, x \in [1\ldots m]\} \cup$
$\{2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_{m-1}} + C_{i_m} | i_1 < \ldots < i_{m-1}, i_x < i, i_m > i, x \in [1\ldots(m-1)]\} \cup$
$\ldots \cup \{C_{i_1} + C_{i_2} + \ldots + C_{i_m} | i_1 < \ldots < i_m, i_x > i, x \in [1\ldots m]\}$,

where $i = 1, 2, \ldots, n$, $C_i > 0$, $n \ge m + 1$, and $1 \le i_x \le n$. The cardinality of each set S_i is given by $|S_i| = \binom{n-1}{m}$. In other words, there are $\binom{n-1}{m}$ inequalities associated with each T_i term that must be satisfied if any $(m + 1)$ tasks cannot be scheduled on a single processor.

Let $\alpha_i = \min(S_i)$, i.e., α_i is the minimum member in value of the set S_i. If we view each member of the set S_i as a summation of *m* terms from $(C_1, C_2, \ldots, C_n)$, then α_i is the minimum summation among the $\binom{n-1}{m}$ ones.

Let us further define that for any *i* and *j* such that $i \in [1\ldots n]$, $j \in [1\ldots n]$, and $i \ne j$, if the term C_j appears in the summation of α_i, then we say that $C_j \in \alpha_i$ (note that α_i is not a set!). Otherwise, $C_j \notin \alpha_i$.

First, let us assume that $\bar{C} = (C_1, C_2, \ldots, C_n)$ is known.

Since

$$\frac{\partial U}{\partial T_i} = -\frac{C_i}{T_i^2} \tag{4}$$

U decreases as T_i increases. But the increase of T_i cannot exceed the limit that is imposed by the constraints in (2). In other words, *U* is minimized when

$T_i = C_i + \min(\{2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_m} | i_1 < \ldots < i_m, i_x < i, x \in [1\ldots m]\} \cup$

$\{2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_{m-1}} + C_{i_m} | i_1 < \ldots < i_{m-1}, i_x < i, i_m > i, x \in [1\ldots(m-1)]\} \cup$
$\ldots \cup \{C_{i_1} + C_{i_2} + \ldots + C_{i_m} | i_1 < \ldots < i_m, i_x > i, x \in [1\ldots m]\}$),
for $i = 1, 2, \ldots, n$.

According to the definition of α_i, we rewrite T_i as $T_i = C_i + \alpha_i$. The minimization problem then becomes

$$U(\bar{C}, \bar{T}) = \sum_{i=1}^{n} C_i / T_i = \sum_{i=1}^{n} C_i / (C_i + m_i) . \tag{5}$$

Next we show that the minimum of U is achieved at $C_1 \le C_2 \le \ldots \le C_n \le 2C_1$. This is accomplished by proving the following three claims.

Claim 1: For every $j \in [1\ldots n]$, there exists at least one index i such that

$$C_j \in \alpha_i \text{ or } 2C_j \in \alpha_i. \tag{6}$$

Suppose that when the minimum of $U(\bar{C}, \bar{T})$ is achieved and (6) is not satisfied, i.e., for some index j there does not exist an index $i \ne j$ such that $C_j \in \alpha_i$ if $i < j$ or $2C_j \in \alpha_i$ if $i > j$. Then $U(\bar{C}, \bar{T})$ can be phrased exclusively in terms of $\bar{C}$. Since

$$\frac{\partial}{\partial C_j} U(\bar{C}, \bar{T}) = \frac{\alpha_j}{(C_j + \alpha_j)^2} > 0 ,$$

meaning that U increases as C_j increases, we can lower the value of U by lowering the value of C_j. Thus, condition (6) is satisfied for any index j.

Claim 2: For every C_i with $i \in [1\ldots n]$, there are at most m C_ks such that $C_i > C_k$ for $k > i$ i or $C_i > 2C_k$ for $k < i$.

Suppose that the contrary is true, i.e., there exists an index $i \in [1\ldots n]$ such that there are $l \ge m + 1$ C_ks such that $C_i > C_k$ for $k > i$ or $C_i > 2C_k$ for $k < i$. Then for any $k > i$, $C_i \notin \alpha_k$ because there are l terms that are smaller than C_i. Similarly, for any $k < i$, $2C_i \notin \alpha_k$. This is a contradiction to Claim 1. Hence Claim 2 must be true. A corollary of this claim is that if $C_i > C_{i+1}$, then $C_{i+1} \in \alpha_i$.

Claim 3: The minimum of U is achieved at $C_1 \le C_2 \le \ldots \le C_n \le 2C_1$.

Suppose that there exists a task set Σ' such that the minimum of U is achieved when, for some index $i \in [1\ldots n]$, $C_i > C_{i+1}$ for $i < n$ or $C_i > 2C_1$ for $i = n$. We will only present the proof for the case of $i < n$ since the proof for the case of $i = n$ is symmetric.

Claim 3.1: For such a task set, there must exist an index $k \ne i + 1$ such that either (1) $k \le i$, $C_i = C_k$, and $2C_k \in \alpha_{i+1}$; or (2) $k > i + 1$, $2C_i = C_k$, and $C_k \in \alpha_{i+1}$.

The core of the above claim is that the term $2C_i$ must be included in the summation of α_{i+1} if the value of $2C_i$ is unique. The bulk of the claim covers the case where there might be other C_k that is equal to C_i or $2C_i$ in value. Hence it is apparent that we need only to prove that $2C_i \in \alpha_{i+1}$ assuming that $2C_i$ is unique.

If $2C_i \notin \alpha_{i+1}$, then there are at least m C_ks such that $C_k < 2C_i$ for $k > i + 1$ or $C_k < C_i$ for $k < i$. Since $C_i > C_{i+1}$, there are at least $(m + 1)$ such C_ks that are smaller than C_i. This is a contradiction to Claim 2. Therefore, the claim must be true.

Now we can construct a new task set Σ' from the task set Σ as follows: the computation times of the tasks are given by

$$C_1' = C_1$$

$$C_2' = C_2$$

$$\ldots\ldots$$

$$C_{i-1}' = C_{i-1}$$

$$C_i' = C_{i+1}$$

$$C_{i+1}' = C_i$$

$$C_{i+2}' = C_{i+2}$$

$$\ldots\ldots$$

$$C_n' = C_n$$

and the task periods are given by

$$T_1' = T_1$$

$$T_2' = T_2$$

$$\ldots\ldots$$

$$T_{i-1}' = T_{i-1}$$

$$T_i' = \alpha_i + C_i$$

$$T_{i+1}' = \alpha_{i+1} + C_{i+1} - C_i$$

$$T_{i+2}' = T_{i+2}$$

$$\ldots\ldots$$

$$T'_n = T_n.$$

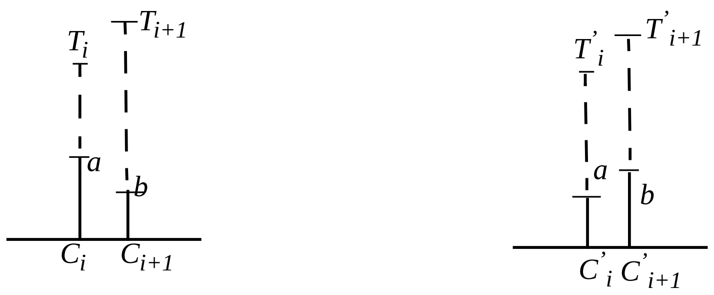

Figure 1: Relationship between two task sets

We want to prove that any $m+1$ tasks within the newly constructed task set cannot be scheduled on a single processor and $U > U'$, where $U' = \sum_{i=1}^{n} C'_i / T'_i$. Note that this newly constructed task set may not satisfy Claim 1 (but it does not affect the validity of our argument).

First we want to assert that the tasks in Σ' are in the order of non-increasing task periods. From the construction, we need only to consider the order among the tasks T'_{i-1}, T'_i, T'_{i+1}, and T'_{i+2}, since the order among the rest of the tasks does not change.

From the definition, it is immediate that $T'_{i-1} \le T'_i$, since $T'_i = T_i$ and $T'_{i+1} < T_{i+1} \le T'_{i+2}$. Since $C_{i+1} \in \alpha_i$ and $2C_i \in \alpha_{i+1}$, the difference between α_{i+1} and α_i is given by $2C_i - C_{i+1}$, i.e., $\alpha_{i+1} - \alpha_i \ge 2C_i - C_{i+1}$. Hence $T'_{i+1} - T'_i = \alpha_{i+1} - \alpha_i + C_{i+1} - 2C_i \ge 0$. Therefore, $T'_{i-1} \le T'_i \le T'_{i+1} \le T'_{i+2}$, and the whole task set is in non-decreasing order of task periods.

Then let us prove that any $m+1$ tasks from the task set Σ' cannot be scheduled on a single processor.

Obviously, for any l such that $l < i$ or $l > i+1$, the inequalities for the new task set related to T'_l hold, since the exchange of the values of C_{i+1} and C_i does not affect the original inequalities. More specifically, for each T'_l, there are $\binom{n-1}{m}$ inequalities to verify. According to the definition of α_l, if $\alpha'_l + C'_l \ge T_l{}'$, then the rest of the $\binom{n-1}{m} - 1$ inequalities holds. Since the exchange of the values of C_{i+1} and C_i does not affect the equalities $\alpha'_l = \alpha_l$ for $l < i$ or $l > i+1$, we have $\alpha'_l + C'_l \ge T_l{}'$.

Now we shall verify that $\alpha'_i + C'_i \ge T'_i$ and $\alpha'_{i+1} + C'_{i+1} \ge T'_{i+1}$.

Case (i): We shall prove the claim that $\alpha'_i \ge \alpha_i - C_{i+1} + C_i$.

Since $C_i > C_{i+1}$, then $C_{i+1} \in \alpha_i$ according to Claim 2. Then $C_i \in \alpha_i'$ must be true, i.e., the term C_i (or a term C_x with $C_i = C_x$) must be included. Otherwise, for the original C_i in the old task set, there are $(m+1)$ C_ks such that $C_i > C_k$ for $k > i$ or $C_i > 2C_k$ for $k < i$ and the minimum of U is achieved.

Therefore the difference between α_i' and α_i is given by $C_{i+1} - C_i$, i.e., $\alpha_i' \geq \alpha_i - C_{i+1} + C_i$.

$$\alpha_i' + C_i' \geq \alpha_i - C_{i+1} + C_i + C_{i+1} = \alpha_i + C_i = T_i = T_i' .$$

Case (i+1): We shall prove that $\alpha_{i+1}' \geq \alpha_{i+1} - 2C_i + C_{i+1}$.

Since $C_i > C_{i+1}$, then $C_{i+1} \in \alpha_i$ according to Claim 2. Furthermore, there are at most m C_ks such that $C_i > C_k$ for $k > i$ or $C_i > 2C_k$ for $k < i$. Then there are at most $(m-1)$ C_ks such that $C_{i+1} > C_k$ for $k > i+1$ or $C_{i+1} > 2C_k$ for $k < i$

Since $C_i \in \alpha_{i+1}$, it follows that $2C_i' \in \alpha_{i+1}'$. The difference between α_{i+1}' and α_{i+1} is given by $2C_i' - 2C_i$, i.e., $\alpha_{i+1}' \geq \alpha_{i+1} - 2C_i + 2C_{i+1} \geq \alpha_{i+1} - 2C_i + C_{i+1}$.

$$\alpha_{i+1}' + C_{i+1}' \geq \alpha_{i+1} - 2C_i + C_{i+1} + C_i = \alpha_{i+1} + C_{i+1} - C_i = T_{i+1}' .$$

Therefore, any $(m+1)$ tasks in the new task set cannot be scheduled on a single processor by the rate-monotonic algorithm.

Finally, let us prove that $U > U'$.

$$U - U' = \left(\frac{C_i}{T_i} + \frac{C_{i+1}}{T_{i+1}}\right) - \left(\frac{C_i'}{T_i'} + \frac{C_{i+1}'}{T_{i+1}'}\right) = \left(\frac{C_i}{T_i} + \frac{C_{i+1}}{T_{i+1}}\right) - \left(\frac{C_{i+1}}{T_i} + \frac{C_i}{T_{i+1}'}\right)$$

Since $T_i = T_i' < T_{i+1}' < T_{i+1}$, $C_i > C_{i+1}$, and $T_i \geq C_i$, we have $U > U'$.

Therefore, the minimum of $U(x)$ is achieved when

$$C_1 \leq C_2 \leq \ldots \leq C_n \leq 2C_1 .$$

According to the definition of α_i, we have

$$\alpha_i = C_{i+1} + \ldots + C_{i+m} \text{ for } i = 1, 2, \ldots, n-m \text{, and}$$

$$\alpha_i = C_{i+1} + \ldots + C_n + 2C_1 + \ldots + 2C_{i-(n-m)} \text{ for } i = n-m+1, \ldots, n .$$

In other words, the minimum of $U(x)$ is achieved when the task periods satisfy

$$T_i = C_i + C_{i+1} + \ldots + C_{i+m} \text{ for } i = 1, 2, \ldots, n-m \text{, and}$$

$$T_i = C_i + C_{i+1} + \ldots + C_n + 2C_1 + \ldots + 2C_{i-(n-m)} \text{ for } i = n-m+1, \ldots, n .$$

The minimization problem of $U = \sum_{i=1}^{n} C_i/T_i$ now becomes a convex optimization problem.

Finally, we solve the problem by using one of the standard method.

$$\sum_{i=1}^{n} C_i/T_i = \sum_{i=1}^{n-m} \frac{C_i}{\sum\limits_{j=0}^{m} C_{i+j}} + \sum_{i=n-m+1}^{n} \frac{C_i}{\sum_{j=i}^{n} C_j + \sum_{j=1}^{i-(n-m)} 2C_j} \tag{7}$$

Let us define

$$x_i = \log \frac{C_{i+1}}{C_i} \tag{8}$$

for $i = 1, 2, \ldots, n-1$, and

$$x_n = \log \frac{2C_1}{C_n} . \tag{9}$$

Then $\sum_{i=1}^{n} x_i = 1$.

We want to minimize

$$U = \sum_{i=1}^{n-m+1} \frac{1}{1 + \sum_{j=0}^{m-1} 2^{\sum_{k=0}^{j} x_{i+k}}} +$$

$$\sum_{i=n-m+2}^{n} \frac{1}{1 + \sum_{j=0}^{n-i} 2^{\sum_{k=0}^{j} x_{i+k}} + \sum_{j=1}^{i-(n-m+1)} 2^{\sum_{k=i}^{n} x_k + \sum_{k=1}^{j} x_k}} \tag{10}$$

subject to

$$x_i > 0, i = 1, 2, \ldots, n \tag{11}$$

$$\sum_{i=1}^{n} x_i = 1. \tag{12}$$

Since the function and its conditions are symmetric under permutation of indices, the minimum is achieved at $x_i = 1/n$.

Therefore $U = n/\sum_{i=0}^{m} 2^{i/n}$.

Next we show that there indeed exists some task sets such that the above bound is achieved. In other words, the given bound is tight.

Let ε be an arbitrarily small positive number and *a* be a positive number. Then for a task set given by

$$\tau_i = (C_i, T_i) = \left(a2^{i/n} + \varepsilon, a2^{i/n}\left(\sum_{j=0}^{m} 2^{j/n}\right)\right),$$

for $i = 1, 2, \ldots, n$, any $(m + 1)$ of the *n* tasks cannot be scheduled on a single processor. ■

Theorem 2: *For any given set of n tasks* $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, 2, \ldots, n\}$, *no more than* $\min(n, \lceil 1/\{\log[1 + n(2^{1/n} - 1)/U] - 1/n\}\rceil)$ *processors are required in an optimal schedule, such that the tasks can be feasibly scheduled by the Rate-Monotonic algorithm, where* $U = \sum_{i=1}^{n} C_i/T_i$.

Proof: For any given set of *n* tasks with a utilization of $U = \sum_{i=1}^{n} C_i/T_i$, it is apparent that at most *n* processors are needed for the feasible scheduling of the tasks by the Rate-Monotonic algorithm, as long as $C_i/T_i \le 1$ for $i = 1, 2, \ldots, n$.

What we want to find out is, with regard to *n* and *U* (i.e., the set of the task sets each with *n* tasks and a utilization of *U*), the maximum numbers of processors that are necessary to feasibly schedule the task sets in all the optimal schedules. In other words, we are trying to find the maximum of processors used in an optimal schedule in the worst cases. The worst cases are thus defined as such that for some task sets with *n* tasks each and a utilization of *U*, the number of processors required by these task sets is no smaller than those required by any set of *n* tasks with the same utilization of *U*.

Before we can proceed further, we need to define a procedure for finding the minimum number of processors for any given set Σ of *n* tasks with a utilization of *U*, i.e., the number of processors in an optimal schedule. In other words, we will design the optimal algorithm for scheduling a set of periodic tasks, the one which always returns the minimum number of processors for any given set of tasks. By describing such algorithm, we actually define a canonical form for the optimal schedule, to be used in the proof that follows. Note that the Rate-Monotonic algorithm is employed by each processor as its scheduling algorithm.

(1) $m = 1$; $S \leftarrow \Sigma$;

(2) Find the largest subset $S_m \subseteq S$ of tasks such that

(i) any $(m + 1)$ tasks in S_m cannot be scheduled on a single processor;

(ii) any *i* tasks in S_m cannot be scheduled with any $(m - i + 1)$ tasks in $S - S_m$ for $i = 1, 2, \ldots, m$;

(iii) for each task in S_m, there exist some groups of $(m - 1)$ tasks in S_m that these *m*

tasks can be feasibly scheduled on a single processor.

Then $S \leftarrow S - S_m$.

We give the following procedure that can compute such S_m:

(a) $S_m \leftarrow \{\}$;

(b) Rename the task set S such that the $|S|$ tasks are indexed from 1 to $|S|$.

For all $j = 1, 2, \ldots, m$, $\tau_{i_j} \in S$, $i_j = 1, \ldots, |S|$, and $i_k \neq i_l$ with $k, l \in [1 \ldots m]$, repeat the following until either $S = \{\}$ or the conditions (I) and (II) are not true.

If

(I) these m tasks $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ can be scheduled on a processor; and

(II) any i tasks among the m tasks $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ cannot be scheduled with any $(m - i + 1)$ tasks in $S - S_m$ on a processor for $i = 1, 2, \ldots, m$

then $S_m \leftarrow S_m + \{\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}\}$ and $S \leftarrow S - \{\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}\}$

(3) If $S \neq \{\}$, then $m \leftarrow m + 1$ and goto (2).

From the algorithm, it is apparent that in the final schedule, if l is the maximum number of tasks assigned on a processor, then any $(l + 1)$ tasks among the n tasks cannot be scheduled on a processor.

If $U > \frac{n}{1 + 2^{1/n}}$, then in the worst cases, any two of the n tasks cannot be scheduled in a single processor. Hence, the number of processors required for the scheduling of this set is n. Since

$$\left\lceil \frac{1}{\log [1 + n(2^{1/n} - 1)/U] - 1/n} \right\rceil > n$$

for $U \geq \frac{n}{1 + 2^{1/n}}$, the theorem holds.

If $U \leq \frac{n}{1 + 2^{1/n}}$, we claim that in the worst cases, the maximum numbers of processors in the optimal schedules, that are required for the feasible scheduling of the task sets, say P, is given by $\left\lceil \frac{n}{m} \right\rceil$, where m is determined by finding the smallest m such that

$$U \geq \frac{n}{\sum_{i=0}^{m} 2^{i/n}}. \tag{13}$$

Let us note that such a number of m does exist, since $U \leq n / \sum_{i=0}^{m} 2^{i/n}$ for $m = 1$, and the function $f(m) = n / \sum_{i=0}^{m} 2^{i/n}$ is a monotonically decreasing function with regard to m. Furthermore, by solving inequality (13), we obtain

$$m \geq n \log \left[1 + \frac{n(2^{1/n} - 1)}{U} \right] - 1. \tag{14}$$

Hence,

$$P = \left\lceil \frac{n}{m} \right\rceil \le \left\lceil \frac{1}{\log\,[1 + n\,(2^{1/n} - 1)\,/\,U] - 1/n} \right\rceil. \tag{15}$$

Now suppose that the claim is not true, i.e., there exists a worst case where the number of processors required is Q such that $Q > P$. Then let n_i be the number of processors on each of which i tasks are assigned and k and l are the minimum and the maximum number of tasks assigned to a processor, respectively. Then $Q = \sum_{i=k}^{l} n_i$ and $n = \sum_{i=k}^{l} i n_i$.

If $l < m$, i.e., each processor is assigned less than m tasks, then since any $(l + 1)$ of the n tasks cannot be scheduled on a processor, the total utilization of the task set must be greater than $n/\sum_{i=0}^{l} 2^{i/n}$. Since m is the smallest number such that $U \ge n/\sum_{i=0}^{m} 2^{i/n}$, therefore $U < n/\sum_{i=0}^{l} 2^{i/n}$. This indicates that the task set has a greater utilization and thus a contradiction is introduced.

If $l > m$, i.e., some processors are assigned more than m tasks, then since any $(l + 1)$ of the n tasks cannot be scheduled on a processor, the total utilization of the task set must be greater than $\frac{n}{\sum_{i=0}^{l} 2^{i/n}}$. Since m is the smallest number such that $U \ge \frac{n}{\sum_{i=0}^{m} 2^{i/n}}$,

$$U \ge \frac{n}{\sum_{i=0}^{m} 2^{i/n}} > \frac{n}{\sum_{i=0}^{l} 2^{i/n}}.$$

This results in a contradiction.

Therefore, the theorem must be true. ∎

III. Earliest Deadline First Scheduling of Periodic Tasks

Before establishing the first result for the Earliest Deadline First algorithm, we restate a result that was first proven in [9] and will serve as a basis for our proof.

Lemma 2: *A set of n tasks* $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$ *can be feasibly scheduled by the Earliest Deadline First algorithm if and only if* $\sum_{i=1}^{n} C_i/T_i \le 1$.

Theorem 3: *For a set of n tasks, if any* $m + 1$ *of the n tasks cannot be feasibly scheduled on a processor by the Earliest Deadline First algorithm, then the total utilization of the n tasks must be greater than* $n/(m + 1)$, *where* $n \ge m + 1$, $n \ge 1$ *and* $m \ge 0$.

Proof: Let the set of n tasks be $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$, where C_i and T_i are the computation time and the period of the task τ_i. Note that the theorem is true when either $n = 1$

or $m = 0$. Hence we need only to consider the case where $n \geq 2$ and $m \geq 1$.

Since any $(m+1)$ of n tasks cannot be feasibly scheduled on a processor by the Earliest Deadline First algorithm, then

$$\sum_{j=1}^{m+1} u_{i_j} > 1 \tag{16}$$

for all $j = 1, 2, \ldots, m+1$, $i_j \in [1 \ldots n]$, $i_k \neq i_l$, and $k, l \in [1 \ldots (m+1)]$, where $u_i = C_i / T_i$. Note that there are a total of $\binom{n}{m+1}$ inequalities in (16).

Summing up the inequalities in (16) yields

$$\binom{n-1}{m} \sum_{i=1}^{n} u_i > \binom{n}{m+1}. \tag{17}$$

Hence, $\sum_{i=1}^{n} u_i > n / (m+1)$ ■

Now we are ready to state the second result for the Earliest Deadline First algorithm.

Theorem 4: *For any given set of n tasks* $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$, *no more than* $\min(n, \lceil U + U^2/(n-U) \rceil)$ *processors are required in an optimal schedule, such that the task set can be feasibly with the Earliest Deadline First algorithm, where* $U = \sum_{i=1}^{n} C_i / T_i$.

Proof: For any given set of n tasks with a utilization of $U = \sum_{i=1}^{n} C_i / T_i$, it is apparent that at most n processors are needed for the feasible scheduling of the tasks by the Earliest Deadline First algorithm, as long as $C_i / T_i \leq 1$ for $i = 1, 2, \ldots, n$.

What we want to find out is, with regard to *n* and *U* (i.e., the set of the task sets each with *n* tasks and a utilization of *U*), the maximum numbers of processors that are necessary to feasibly schedule the task sets in all the optimal schedules. In other words, we are trying to find the maximum of processors used in an optimal schedule in the worst cases. The worst cases are thus defined as such that for some task sets with *n* tasks each and a utilization of *U*, the number of processors required by these task sets is no smaller than those required by any set of *n* tasks with the same utilization of *U*.

Before we can proceed further, we need to define a procedure for finding the minimum number of processors for any given set Σ of *n* tasks with a utilization of *U*. In other words, we will design the optimal algorithm for scheduling a set of periodic tasks, the one which always returns the minimum number of processors for any given set of tasks. By describing such algorithm, we actually define a canonical form for the optimal schedule, to be used in the proof that follows. Note that the Earliest Deadline First algorithm is employed by each processor as its scheduler.

An optimal algorithm is given as follows:

```
(1) $m = 1$; $S \leftarrow \Sigma$;
(2) Find the largest subset $S_m \subseteq S$ of tasks such that
    (i)   any $(m + 1)$ tasks in $S_m$ cannot be scheduled on a single processor;
    (ii)  any $i$ tasks in $S_m$ cannot be scheduled with any $(m - i + 1)$ tasks in $S - S_m$ for
          $i = 1, 2, \ldots, m$;
    (iii) for each task in $S_m$, there exist some groups of $(m - 1)$ tasks in $S_m$ that these $m$
          tasks can be feasibly scheduled on a single processor.
    Then $S \leftarrow S - S_m$.
    We give the following procedure that can compute such $S_m$:
    (a) $S_m \leftarrow \{\}$;
    (b) Rename the task set $S$ such that the $|S|$ tasks are indexed from 1 to $|S|$.
        For all $j = 1, 2, \ldots, m$, $\tau_{i_j} \in S$, $i_j = 1, \ldots, |S|$, and $i_k \neq i_l$ with $k, l \in [1 \ldots m]$,
        repeat the following until either $S = \{\}$ or the conditions (I) and (II) are not true.
            If
            (I)  these $m$ tasks $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ can be scheduled on a processor; and
            (II) any $i$ tasks among the $m$ tasks $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ cannot be scheduled with any
                 $(m - i + 1)$ tasks in $S - S_m$ on a processor for $i = 1, 2, \ldots, m$
            then $S_m \leftarrow S_m + \{\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}\}$ and $S \leftarrow S - \{\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}\}$
(3) If $S \neq \{\}$, then $m \leftarrow m + 1$ and goto (2).
```

From the algorithm, it is apparent that in the final schedule, if l is the maximum number of tasks assigned on a processor, then any $(l + 1)$ tasks among the n tasks cannot be scheduled on a processor.

If $U > n/2$, then in the worst cases, any two of the n tasks cannot be scheduled in a single processor. Hence, the number of processors required for the scheduling of this set is n. Since

$$\lceil U + U^2/(n - U) \rceil > n$$

for $U \geq n/2$, the theorem holds.

If $U \leq n/2$, we claim that in the worst cases, the maximum numbers of processors in the optimal schedules, that are required for the feasible scheduling of the task sets, say P, is given by $\left\lceil \frac{n}{m} \right\rceil$, where m is determined by finding the smallest m such that

$$U \geq \frac{n}{m+1}. \tag{18}$$

Let us note that such a number of *m* does exist, since $U \leq n/(m+1)$ for *m* = 1, and the function $f(m) = n/(m+1)$ is a monotonically decreasing function with regard to *m*. Furthermore, by solving inequality (13), we obtain

$$m \geq \frac{n}{U} - 1. \tag{19}$$

Hence,

$$P = \left\lceil \frac{n}{m} \right\rceil \leq \left\lceil U + \frac{U^2}{n-U} \right\rceil. \tag{20}$$

Now suppose that the claim is not true, i.e., there exists a worst case where the number of processors required is *Q* such that *Q* > *P*. Then let n_i be the number of processors on each of which *i* tasks are assigned and *k* and *l* are the minimum and the maximum number of tasks assigned to a processor, respectively. Then $Q = \sum_{i=k}^{l} n_i$ and $n = \sum_{i=k}^{l} i n_i$.

If $l < m$, i.e., each processor is assigned less than *m* tasks, then since any (*l* + 1) of the *n* tasks cannot be scheduled on a processor, the total utilization of the task set must be greater than $n/(l+1)$. Since *m* is the smallest number such that $U \geq n/(m+1)$, therefore $U < n/(l+1)$. This indicates that the task set now has a greater utilization and thus a contradiction is introduced.

If $l > m$, i.e., some processors are assigned more than *m* tasks, then since any (*l* + 1) of the *n* tasks cannot be scheduled on a processor, the total utilization of the task set must be greater than $\frac{n}{l+1}$. Since *m* is the smallest number such that $U \geq \frac{n}{m+1}$,

$$U \geq \frac{n}{m+1} > \frac{n}{l+1}.$$

This results in a contradiction.

Therefore, the theorem must be true. ∎

IV. Concluding Remarks

The discovery of the general schedulability condition as given in Theorem 1 for Rate-Monotonic scheduling was the direct result of our attempt to design and analyze effective heuristic algorithms for scheduling periodic tasks on a multiprocessor system. In analyzing various heuristic algorithms for their worst-case performance, we realize that for any given set of tasks, some upper

bounds on the number of processors must be established. However, we also learn from our proof that it is very difficult to establish such bounds, because it usually involves the solutions to non-convex optimization problems.

Further questions remain as how to derive meaningful conditions for periodic tasks which share resources in a multiprocessor environment. Also of interest to our research is the derivation of similar schedulability conditions for scheduling periodic tasks whose deadlines are shorter than their periods. A number of researchers have addressed this problem to some extents, see [1] for example.

Acknowledgments: I would like to thank Dr. Sang H. Son for his support.

References

[1] N.C. AUDSLEY, A. BURNS, M.F. RICHARDSON, K.W. TINDELL, AND A.J. WELLINGS. "Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling," *Software Engineering Journal* **8(5)**: 284-292 (1993).

[2] A. BURCHARD, J. LIEBEHERR, Y. OH, AND S.H. SON. "Assigning Real-Time Tasks to Homogeneous Multiprocessor Systems," *IEEE Transactions on Computer* (to appear).

[3] S. DAVARI AND S.K. DHALL. "An On Line Algorithm for Real-Time Tasks Allocation," *IEEE Real-Time Systems Symposium*, 194-200 (1986).

[4] S.K. DHALL AND C.L. LIU. "On a Real-Time Scheduling Problem," *Operations Research* **26**: 127-140 (1978).

[5] M.R. GAREY AND D.S. JOHNSON. *Computers and Intractability: A Guide to the Theory of NP-completeness*, W.H. Freeman and Company, NY, 1978.

[6] M. JOSEPH AND P. PANDYA. "Finding Response Times in a Real-Time System," *The Computer Journal* **29(5)**: 390-395 (1986).

[7] J. LEHOCZKY, L. SHA, AND Y. DING. "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," *IEEE Real-Time Symposium*, 166-171 (1989).

[8] J.Y.T. LEUNG AND J. WHITEHEAD. "On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks," *Performance Evaluation* **2**: 237-250 (1982).

[9] C.L. LIU AND J. LAYLAND. "Scheduling Algorithms for Multiprogramming in a Hard

Real-Time Environment," *J. Assoc. Comput. Machinery* **10(1)**: 174-189 (1973).

[10] Y. OH AND S.H. SON. "Allocating Fixed-priority Periodic Tasks on Multiprocessor Systems," *Journal of Real-Time Systems* (to appear).

[11] Y. OH. The Design and Analysis of Scheduling Algorithms for Real-Time and Fault-Tolerant Computer Systems. Ph.D. Dissertation, Dept. of Computer Science, University of Virginia, May 1994.

[12] P. SERLIN, "Scheduling of Time Critical Processes," *Proceedings of the Spring Joint Computers Conference* **40**: 925-932 (1972).

[13] L. SHA, R. RAJKUMAR, AND J.P. LEHOCZKY. "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Transactions on Computers* **39(9)**: 1175-1185 (1990).

Scheduling Periodic Tasks In a Hard Real-Time Environment

Yingfeng Oh

Department of Computer Science
University of Virginia
Charlottesville, VA 22903
USA

Consider the traditional problem of scheduling a set of periodic tasks on a mulitprocessor system, where task deadlines must be guaranteed. We first derive a general schedulability condition for Rate-Monotonic, which reduces the uniprocessor schedulability condition obtained by Liu and Layland and by Serlin, and the multiprocessor schedulability condition recently derived by Burchard, Liebeherr, Oh, and Son to its two specific cases. Then a tight upper bound is obtained for the number of processors required in an optimal schedule for any given set of tasks with a fixed number of tasks and a fixed utilization. Finally, similar conditions are derived for the Earliest Deadline First scheduling. These conditions shed new light on the periodic task scheduling problem.

I. Introduction

In this paper, we consider the problem of scheduling a set of periodic tasks. The task set to be considered is defined as follows:

(1) The requests of each task are periodic, with constant interval between requests.

(2) The deadline constraints specify that each request must be completed before the next request of the same task occurs.

(3) The tasks are independent in that the requests of a task do not depend on the initiation or the completion of the requests of other tasks.

(4) The worst-case run-time (or computation time) for the request of a task is constant for the task. Run-time here refers to the time a processor takes to execute the request without interruption.

The question we are interested in is whether any given set of *n* tasks can be scheduled on *m* processors. In general, this decision problem is NP-complete [5]. Practical solutions to this problem often rely on some kinds of heuristic algorithms, which can deliver approximate solutions in

a short period of time. Heuristic solutions often trade computational time complexity for accuracy of solutions. The approach we take in this paper is to find a schedulability condition for any given set of tasks such that as long as the total utilization or load of the task set is under certain threshold number, the task set can be feasibly scheduled on a certain number of processors. The derivation of this bound subsumes the previous results on schedulability conditions derived by Liu and Layland [9], and by Serlin [12] for Rate-Monotonic scheduling on a single processor system, and by Burchard, Liebeherr, Oh, and Son [2] for Rate-Monotonic scheduling on a multiprocessor system. This tight bound can serve as the basis for constructing more effective heuristic algorithms and for proving tighter worst-case performance guarantee. For more details on how to use schedulability conditions in the design and analysis of heuristics algorithms, reader may refer to [11].

From the description of the task set, it follows that a task is completely defined by two numbers, the run-time of the requests and the request period. We shall denote a task τ_i by the ordered pair (C_i, T_i), where C_i is the computation time and T_i is the period of the requests of the task τ_i. The ratio C_i/T_i is called the utilization (or load) of the task τ_i, and the total utilization (or load) of a set of *n* tasks is given by $U = \sum_{i=1}^{n} C_i/T_i$. All the processors are identical in the sense that the run-time of a task remains the same across all processors.

Tasks can be scheduled for execution on a processor by using a priority-driven algorithm, in which each task is assigned a priority and the task with the highest priority is always the one to be executed. By assigning different priorities to tasks, we therefore determine the schedule of the execution of tasks. A priority assignment algorithm is fixed if the priority of a task remains fixed once it is assigned. Otherwise, it is a dynamic priority assignment algorithm. Here we concern ourselves with priority-driven algorithms only.

If a set of tasks can be scheduled such that all task deadlines can be met by some algorithms, then we say that the task set is *feasible*. If a set of periodic tasks can be feasibly scheduled on a single processor, then the *Rate-Monotonic* (or *RM*) [9] or *Intelligent Fixed Priority* algorithm [12] is optimal for fixed priority assignment, in the sense that no other fixed priority assignment algorithm can schedule a task set which cannot be scheduled by the RM algorithm. The RM algorithm assigns priorities to tasks according to their periods, where the priority of a task is in inverse relationship to its period. In other words, a task with a shorter period is assigned a higher priority. The execution of a low-priority task will be preempted if a high-priority task arrives. Liu and Layland proved that a set of *n* periodic tasks can be feasibly scheduled by the Rate-Monotonic algorithm if

the total utilization of the tasks is no more than a threshold number, which is given by $n\left(2^{1/n}-1\right)$.

One of the important properties of Rate-Monotonic scheduling is that for a single processor system where Rate-Monotonic algorithm is employed to schedule tasks, as long as the CPU utilization of the tasks lies below a certain bound, they will meet their deadlines without the programmer knowing exactly when any given request of a task is running. Even if a transient overload occurs, a fixed subset of the most frequently arrived tasks will still meet their deadlines as long as their total CPU utilization lies below a certain bound. This property puts the real-time software development on a sound analytical footing.

For dynamic priority assignment, the Earliest Deadline First (or EDF) algorithm [9] is optimal in the sense that no other dynamic priority assignment algorithm can schedule a task set which cannot be scheduled by the EDF algorithm. The request of a task is assigned the highest priority if its deadline is the closest. Furthermore, a set of periodic tasks can be feasibly scheduled on a single processor system by the EDF algorithm if and only if its total utilization is no more than one.

Although the schedulability condition, i.e., $\sum_{i=1}^{n} C_i/T_i \le n\,(2^{1/n}-1)$, given by Liu and Layland is simple and elegant, they are pessimistic in nature since the condition is derived under the worst case conditions. Several more efficient conditions were later derived [4, 2, 10, 11]. All these conditions are sufficient but not necessary. The necessary and sufficient condition to schedule a set of periodic tasks using fixed-priority algorithms was given by Joseph and Pandya [6], and by Lehoczky, Sha, and Ding [7].

The results about Rate-Monotonic scheduling are presented in Section II, while similiar results for Earliest Deadline First scheduling are given in Section III. We conclude this paper in Section IV by discussing some remaining issues.

II. Fundamental Conditions for Rate-Monotonic Scheduling

In Theorem 1, we present a general result about rate-monotonic scheduling. It puts a tight upper bound on the number of processors that are required to schedule a set of n tasks such that each task is guaranteed its deadline by the Rate-Monotonic algorithm.

Theorem 1: *For a set of n tasks, if any $m+1$ of the n tasks cannot be feasibly scheduled on a processor by the Rate-Monotonic algorithm, then the total utilization of the n tasks must be greater than $n/\sum_{i=0}^{m} 2^{i/n}$, where $n \ge m+1$, $n \ge 1$ and $m \ge 0$.*

If $m = 0$, then $n/\sum_{i=0}^{m} 2^{i/n} = n$. This is equivalent to saying that if any task cannot be

scheduled on a single processor by the Rate-Monotonic algorithm, then the total utilization of the n tasks must be greater than n. This is trivial true.

If $m = 1$, then $n/\sum_{i=0}^{m} 2^{i/n} = n / \sum_{i=0}^{1} 2^{i/n} = n / \left(2^{1/n} + 1\right)$. This is equivalent to saying that if any two tasks cannot be scheduled on a processor by the Rate-Monotonic algorithm, then the total utilization of the n tasks must be greater than $n / \left(2^{1/n} + 1\right)$. This reduces to the result obtained by Burchard, Liebeherr, Oh, and Son in [2].

If $n = m + 1$, then $n/\sum_{i=0}^{m} 2^{i/n} = n/\sum_{i=0}^{n-1} 2^{i/n} = n\left(2^{1/n} - 1\right)$. This is equivalent to saying that if any n tasks cannot be scheduled on a processor, then the total utilization of the n tasks must be greater than $n\left(2^{1/n} - 1\right)$. This reduces to the result originally obtained by Liu and Layland [9] and Serlin [12].

When n is large, i.e., $n \to \infty$, $n/\sum_{i=0}^{m} 2^{i/n} \to n/(m+1)$. This implies that compared with the utilization bound for the Earliest Deadline First scheduling, the bound given by Theorem 1 for the Rate-Monotonic scheduling is very favorable, i.e., very close to the “best” possible.

In order to prove Theorem 1, we need the following lemma that was proven in [2].

Lemma 1: *If a set of tasks* $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$ *cannot be scheduled on N processors, then the task set* $\Sigma = \{\tau_i' = (C_i', T_i') \mid i = 1, 2, \ldots, n\}$ *given by* $C_i' = T_i' * C_i / T_i$, $T_i' = 2^{V_i}$, *and* $V_i = \log_2 T_i - \lfloor \log_2 T_i \rfloor$ *cannot be scheduled on the N processor either.*

Proof of Theorem 1: Let the set of n tasks be $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$, where C_i and T_i are the computation time and the period of the task τ_i. Note that the theorem is true when either $n = 1$ or $m = 0$. Hence we need only to consider the case where $n \geq 2$ and $m \geq 1$.

According to Lemma 1, we can assume, without loss of generality, that

$$T_1 \leq T_2 \leq \ldots \leq T_n < 2T_1 \tag{1}$$

Since no $m + 1$ of the n tasks can be scheduled together on a processor, the following conditions must hold according to the necessary and sufficient condition [6, 7]:

$$\begin{cases} C_{i_1} + C_{i_2} + \ldots + C_{i_m} + C_{i_{m+1}} > T_{i_1} \\ 2C_{i_1} + C_{i_2} + \ldots + C_{i_m} + C_{i_{m+1}} > T_{i_2} \\ \ldots\ldots\ldots \\ 2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_m} + C_{i_{m+1}} > T_{i_{m+1}} \end{cases} \tag{2}$$

where $1 \leq i_1 < \ldots < i_m < i_{m+1} \leq n$.

We want to find the minimum of $U = \sum_{i=1}^{n} C_i / T_i$ subject to the constraints of (1), (2), and

(3).

$$0 < C_i \le T_i \qquad i = 1, \ldots, n \tag{3}$$

In order to ensure that the minimum is obtained at some point, we replace “>” by “≥”. This replacement will not affect the minimum.

We proceed in three steps to obtain the minimum of U:

(1) Fix the values $\bar{C} = (C_1, C_2, \ldots, C_n)$ and express $\bar{T} = (T_1, T_2, \ldots, T_n)$ in terms of $\bar{C}$ in the minimization problem.

(2) Reduce the minimization problem to a convex optimization problem by proving that $C_1 \le C_2 \le \ldots \le C_n \le 2C_1$ if the minimum of *U* is achieved.

(3) Solve the optimization problem using standard methods.

Let us define

$S_i = \{2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_m} | i_1 < \ldots < i_m, i_x < i, x \in [1 \ldots m]\} \cup$
$\{2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_{m-1}} + C_{i_m} | i_1 < \ldots < i_{m-1}, i_x < i, i_m > i, x \in [1 \ldots (m-1)]\} \cup$
$\ldots \cup \{C_{i_1} + C_{i_2} + \ldots + C_{i_m} | i_1 < \ldots < i_m, i_x > i, x \in [1 \ldots m]\}$,

where $i = 1, 2, \ldots, n$, $C_i > 0$, $n \ge m + 1$, and $1 \le i_x \le n$. The cardinality of each set S_i is given by $|S_i| = \binom{n-1}{m}$. In other words, there are $\binom{n-1}{m}$ inequalities associated with each T_i term that must be satisfied if any $(m + 1)$ tasks cannot be scheduled on a single processor.

Let $\alpha_i = \min(S_i)$, i.e., α_i is the minimum member in value of the set S_i. If we view each member of the set S_i as a summation of *m* terms from $(C_1, C_2, \ldots, C_n)$, then α_i is the minimum summation among the $\binom{n-1}{m}$ ones.

Let us further define that for any *i* and *j* such that $i \in [1 \ldots n]$, $j \in [1 \ldots n]$, and $i \ne j$, if the term C_j appears in the summation of α_i, then we say that $C_j \in \alpha_i$ (note that α_i is not a set!). Otherwise, $C_j \notin \alpha_i$.

First, let us assume that $\bar{C} = (C_1, C_2, \ldots, C_n)$ is known.

Since

$$\frac{\partial U}{\partial T_i} = -\frac{C_i}{T_i^2} \tag{4}$$

U decreases as T_i increases. But the increase of T_i cannot exceed the limit that is imposed by the constraints in (2). In other words, *U* is minimized when

$T_i = C_i + \min(\{2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_m} | i_1 < \ldots < i_m, i_x < i, x \in [1 \ldots m]\} \cup$

$\{2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_{m-1}} + C_{i_m} | i_1 < \ldots < i_{m-1}, i_x < i, i_m > i, x \in [1\ldots(m-1)]\} \cup$
$\ldots \cup \{C_{i_1} + C_{i_2} + \ldots + C_{i_m} | i_1 < \ldots < i_m, i_x > i, x \in [1\ldots m]\}$),
for $i = 1, 2, \ldots, n$.

According to the definition of α_i, we rewrite T_i as $T_i = C_i + \alpha_i$. The minimization problem then becomes

$$U(\bar{C}, \bar{T}) = \sum_{i=1}^{n} C_i / T_i = \sum_{i=1}^{n} C_i / (C_i + m_i) . \tag{5}$$

Next we show that the minimum of *U* is achieved at $C_1 \le C_2 \le \ldots \le C_n \le 2C_1$. This is accomplished by proving the following three claims.

Claim 1: For every $j \in [1\ldots n]$, there exists at least one index *i* such that

$$C_j \in \alpha_i \text{ or } 2C_j \in \alpha_i. \tag{6}$$

Suppose that when the minimum of $U(\bar{C}, \bar{T})$ is achieved and (6) is not satisfied, i.e., for some index *j* there does not exist an index $i \ne j$ such that $C_j \in \alpha_i$ if $i < j$ or $2C_j \in \alpha_i$ if $i > j$. Then $U(\bar{C}, \bar{T})$ can be phrased exclusively in terms of $\bar{C}$. Since

$$\frac{\partial}{\partial C_j} U(\bar{C}, \bar{T}) = \frac{\alpha_j}{(C_j + \alpha_j)^2} > 0 ,$$

meaning that *U* increases as C_j increases, we can lower the value of *U* by lowering the value of C_j. Thus, condition (6) is satisfied for any index *j*.

Claim 2: For every C_i with $i \in [1\ldots n]$, there are at most *m* C_ks such that $C_i > C_k$ for $k > i$ i or $C_i > 2C_k$ for $k < i$.

Suppose that the contrary is true, i.e., there exists an index $i \in [1\ldots n]$ such that there are $l \ge m+1$ C_ks such that $C_i > C_k$ for $k > i$ or $C_i > 2C_k$ for $k < i$. Then for any $k > i$, $C_i \notin \alpha_k$ because there are *l* terms that are smaller than C_i. Similarly, for any $k < i$, $2C_i \notin \alpha_k$. This is a contradiction to Claim 1. Hence Claim 2 must be true. A corollary of this claim is that if $C_i > C_{i+1}$, then $C_{i+1} \in \alpha_i$.

Claim 3: The minimum of *U* is achieved at $C_1 \le C_2 \le \ldots \le C_n \le 2C_1$.

Suppose that there exists a task set Σ' such that the minimum of *U* is achieved when, for some index $i \in [1\ldots n]$, $C_i > C_{i+1}$ for $i < n$ or $C_i > 2C_1$ for $i = n$. We will only present the proof for the case of $i < n$ since the proof for the case of $i = n$ is symmetric.

Claim 3.1: For such a task set, there must exist an index $k \ne i+1$ such that either (1) $k \le i$, $C_i = C_k$, and $2C_k \in \alpha_{i+1}$; or (2) $k > i+1$, $2C_i = C_k$, and $C_k \in \alpha_{i+1}$.

The core of the above claim is that the term $2C_i$ must be included in the summation of α_{i+1} if the value of $2C_i$ is unique. The bulk of the claim covers the case where there might be other C_k that is equal to C_i or $2C_i$ in value. Hence it is apparent that we need only to prove that $2C_i \in \alpha_{i+1}$ assuming that $2C_i$ is unique.

If $2C_i \notin \alpha_{i+1}$, then there are at least m C_ks such that $C_k < 2C_i$ for $k > i + 1$ or $C_k < C_i$ for $k < i$. Since $C_i > C_{i+1}$, there are at least (m + 1) such C_ks that are smaller than C_i. This is a contradiction to Claim 2. Therefore, the claim must be true.

Now we can construct a new task set Σ' from the task set Σ as follows: the computation times of the tasks are given by

$$C_1' = C_1$$

$$C_2' = C_2$$

$$\ldots\ldots$$

$$C_{i-1}' = C_{i-1}$$

$$C_i' = C_{i+1}$$

$$C_{i+1}' = C_i$$

$$C_{i+2}' = C_{i+2}$$

$$\ldots\ldots$$

$$C_n' = C_n$$

and the task periods are given by

$$T_1' = T_1$$

$$T_2' = T_2$$

$$\ldots\ldots$$

$$T_{i-1}' = T_{i-1}$$

$$T_i' = \alpha_i + C_i$$

$$T_{i+1}' = \alpha_{i+1} + C_{i+1} - C_i$$

$$T_{i+2}' = T_{i+2}$$

$$\ldots\ldots$$

$$T'_n = T_n .$$

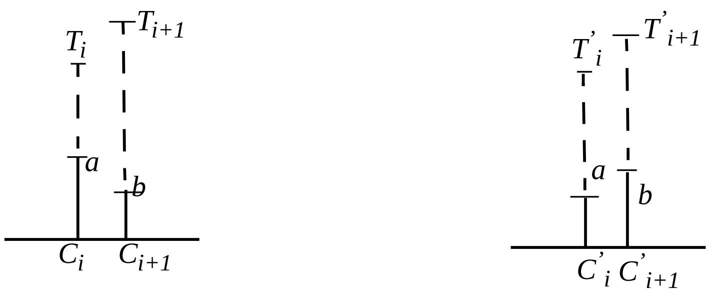

Figure 1: Relationship between two task sets

We want to prove that any $m+1$ tasks within the newly constructed task set cannot be scheduled on a single processor and $U > U'$, where $U' = \sum_{i=1}^{n} C'_i / T'_i$. Note that this newly constructed task set may not satisfy Claim 1 (but it does not affect the validity of our argument).

First we want to assert that the tasks in Σ' are in the order of non-increasing task periods. From the construction, we need only to consider the order among the tasks T'_{i-1}, T'_i, T'_{i+1}, and T'_{i+2}, since the order among the rest of the tasks does not change.

From the definition, it is immediate that $T'_{i-1} \le T'_i$, since $T'_i = T_i$ and $T'_{i+1} < T_{i+1} \le T'_{i+2}$. Since $C_{i+1} \in \alpha_i$ and $2C_i \in \alpha_{i+1}$, the difference between α_{i+1} and α_i is given by $2C_i - C_{i+1}$, i.e., $\alpha_{i+1} - \alpha_i \ge 2C_i - C_{i+1}$. Hence $T'_{i+1} - T'_i = \alpha_{i+1} - \alpha_i + C_{i+1} - 2C_i \ge 0$. Therefore, $T'_{i-1} \le T'_i \le T'_{i+1} \le T'_{i+2}$, and the whole task set is in non-decreasing order of task periods.

Then let us prove that any $m+1$ tasks from the task set Σ' cannot be scheduled on a single processor.

Obviously, for any l such that $l < i$ or $l > i+1$, the inequalities for the new task set related to T'_l hold, since the exchange of the values of C_{i+1} and C_i does not affect the original inequalities. More specifically, for each T'_l, there are $\binom{n-1}{m}$ inequalities to verify. According to the definition of α_l, if $\alpha'_l + C'_l \ge T_l'$, then the rest of the $\binom{n-1}{m} - 1$ inequalities holds. Since the exchange of the values of C_{i+1} and C_i does not affect the equalities $\alpha'_l = \alpha_l$ for $l < i$ or $l > i+1$, we have $\alpha'_l + C'_l \ge T_l'$.

Now we shall verify that $\alpha'_i + C'_i \ge T'_i$ and $\alpha'_{i+1} + C'_{i+1} \ge T'_{i+1}$.

Case (i): We shall prove the claim that $\alpha'_i \ge \alpha_i - C_{i+1} + C_i$.

Since $C_i > C_{i+1}$, then $C_{i+1} \in \alpha_i$ according to Claim 2. Then $C_i \in \alpha'_i$ must be true, i.e., the term C_i (or a term C_x with $C_i = C_x$) must be included. Otherwise, for the original C_i in the old task set, there are $(m+1)$ C_ks such that $C_i > C_k$ for $k > i$ or $C_i > 2C_k$ for $k < i$ and the minimum of U is achieved.

Therefore the difference between α'_i and α_i is given by $C_{i+1} - C_i$, i.e., $\alpha'_i \geq \alpha_i - C_{i+1} + C_i$.

$$\alpha'_i + C'_i \geq \alpha_i - C_{i+1} + C_i + C_{i+1} = \alpha_i + C_i = T_i = T'_i \ .$$

Case (i+1): We shall prove that $\alpha'_{i+1} \geq \alpha_{i+1} - 2C_i + C_{i+1}$.

Since $C_i > C_{i+1}$, then $C_{i+1} \in \alpha_i$ according to Claim 2. Furthermore, there are at most m C_ks such that $C_i > C_k$ for $k > i$ or $C_i > 2C_k$ for $k < i$. Then there are at most $(m-1)$ C_ks such that $C_{i+1} > C_k$ for $k > i+1$ or $C_{i+1} > 2C_k$ for $k < i$

Since $C_i \in \alpha_{i+1}$, it follows that $2C'_i \in \alpha'_{i+1}$. The difference between α'_{i+1} and α_{i+1} is given by $2C'_i - 2C_i$, i.e., $\alpha'_{i+1} \geq \alpha_{i+1} - 2C_i + 2C_{i+1} \geq \alpha_{i+1} - 2C_i + C_{i+1}$.

$$\alpha'_{i+1} + C'_{i+1} \geq \alpha_{i+1} - 2C_i + C_{i+1} + C_i = \alpha_{i+1} + C_{i+1} - C_i = T'_{i+1} \ .$$

Therefore, any $(m+1)$ tasks in the new task set cannot be scheduled on a single processor by the rate-monotonic algorithm.

Finally, let us prove that $U > U'$.

$$U - U' = \left(\frac{C_i}{T_i} + \frac{C_{i+1}}{T_{i+1}}\right) - \left(\frac{C'_i}{T'_i} + \frac{C'_{i+1}}{T'_{i+1}}\right) = \left(\frac{C_i}{T_i} + \frac{C_{i+1}}{T_{i+1}}\right) - \left(\frac{C_{i+1}}{T_i} + \frac{C_i}{T'_{i+1}}\right)$$

Since $T_i = T'_i < T'_{i+1} < T_{i+1}$, $C_i > C_{i+1}$, and $T_i \geq C_i$, we have $U > U'$.

Therefore, the minimum of $U(x)$ is achieved when

$$C_1 \leq C_2 \leq \ldots \leq C_n \leq 2C_1 .$$

According to the definition of α_i, we have

$$\alpha_i = C_{i+1} + \ldots + C_{i+m} \text{ for } i = 1, 2, \ldots, n-m \text{, and}$$

$$\alpha_i = C_{i+1} + \ldots + C_n + 2C_1 + \ldots + 2C_{i-(n-m)} \text{ for } i = n-m+1, \ldots, n .$$

In other words, the minimum of $U(x)$ is achieved when the task periods satisfy

$$T_i = C_i + C_{i+1} + \ldots + C_{i+m} \text{ for } i = 1, 2, \ldots, n-m \text{, and}$$

$$T_i = C_i + C_{i+1} + \ldots + C_n + 2C_1 + \ldots + 2C_{i-(n-m)} \text{ for } i = n-m+1, \ldots, n .$$

The minimization problem of $U = \sum_{i=1}^{n} C_i / T_i$ now becomes a convex optimization problem.

Finally, we solve the problem by using one of the standard method.

$$\sum_{i=1}^{n} C_i / T_i = \sum_{i=1}^{n-m} \frac{C_i}{\sum_{j=0}^{m} C_{i+j}} + \sum_{i=n-m+1}^{n} \frac{C_i}{\sum_{j=i}^{n} C_j + \sum_{j=1}^{i-(n-m)} 2C_j} \tag{7}$$

Let us define

$$x_i = \log \frac{C_{i+1}}{C_i} \tag{8}$$

for $i = 1, 2, \ldots, n-1$, and

$$x_n = \log \frac{2C_1}{C_n} . \tag{9}$$

Then $\sum_{i=1}^{n} x_i = 1$.

We want to minimize

$$U = \sum_{i=1}^{n-m+1} \frac{1}{1 + \sum_{j=0}^{m-1} 2^{\sum_{k=0}^{j} x_{i+k}}} +$$

$$\sum_{i=n-m+2}^{n} \frac{1}{1 + \sum_{j=0}^{n-i} 2^{\sum_{k=0}^{j} x_{i+k}} + \sum_{j=1}^{i-(n-m+1)} 2^{\sum_{k=i}^{n} x_k + \sum_{k=1}^{j} x_k}} \tag{10}$$

subject to

$$x_i > 0, i = 1, 2, \ldots, n \tag{11}$$

$$\sum_{i=1}^{n} x_i = 1. \tag{12}$$

Since the function and its conditions are symmetric under permutation of indices, the minimum is achieved at $x_i = 1/n$.

Therefore $U = n / \sum_{i=0}^{m} 2^{i/n}$.

Next we show that there indeed exists some task sets such that the above bound is achieved. In other words, the given bound is tight.

Let ε be an arbitrarily small positive number and a be a positive number. Then for a task set given by

$$\tau_i = (C_i, T_i) = \left(a2^{i/n} + \varepsilon, a2^{i/n}\left(\sum_{j=0}^{m} 2^{j/n} \right) \right),$$

for $i = 1, 2, \ldots, n$, any $(m + 1)$ of the n tasks cannot be scheduled on a single processor. ■

Theorem 2: *For any given set of n tasks* $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, 2, \ldots, n\}$*, no more than* $\min(n, \lceil 1/\{\log[1 + n(2^{1/n} - 1)/U] - 1/n\} \rceil)$ *processors are required in an optimal schedule, such that the tasks can be feasibly scheduled by the Rate-Monotonic algorithm, where* $U = \sum_{i=1}^{n} C_i/T_i$.

Proof: For any given set of n tasks with a utilization of $U = \sum_{i=1}^{n} C_i/T_i$, it is apparent that at most n processors are needed for the feasible scheduling of the tasks by the Rate-Monotonic algorithm, as long as $C_i/T_i \le 1$ for $i = 1, 2, \ldots, n$.

What we want to find out is, with regard to n and U (i.e., the set of the task sets each with n tasks and a utilization of U), the maximum numbers of processors that are necessary to feasibly schedule the task sets in all the optimal schedules. In other words, we are trying to find the maximum of processors used in an optimal schedule in the worst cases. The worst cases are thus defined as such that for some task sets with n tasks each and a utilization of U, the number of processors required by these task sets is no smaller than those required by any set of n tasks with the same utilization of U.

Before we can proceed further, we need to define a procedure for finding the minimum number of processors for any given set Σ of n tasks with a utilization of U, i.e., the number of processors in an optimal schedule. In other words, we will design the optimal algorithm for scheduling a set of periodic tasks, the one which always returns the minimum number of processors for any given set of tasks. By describing such algorithm, we actually define a canonical form for the optimal schedule, to be used in the proof that follows. Note that the Rate-Monotonic algorithm is employed by each processor as its scheduling algorithm.

(1) $m = 1$; $S \leftarrow \Sigma$;

(2) Find the largest subset $S_m \subseteq S$ of tasks such that

(i) any $(m + 1)$ tasks in S_m cannot be scheduled on a single processor;

(ii) any i tasks in S_m cannot be scheduled with any $(m - i + 1)$ tasks in $S - S_m$ for $i = 1, 2, \ldots, m$;

(iii) for each task in S_m, there exist some groups of $(m - 1)$ tasks in S_m that these m

tasks can be feasibly scheduled on a single processor.

Then $S \leftarrow S - S_m$.

We give the following procedure that can compute such S_m:

(a) $S_m \leftarrow \{\}$;

(b) Rename the task set S such that the $|S|$ tasks are indexed from 1 to $|S|$.

For all $j = 1, 2, \ldots, m$, $\tau_{i_j} \in S$, $i_j = 1, \ldots, |S|$, and $i_k \neq i_l$ with $k, l \in [1 \ldots m]$, repeat the following until either $S = \{\}$ or the conditions (I) and (II) are not true.

If

(I) these m tasks $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ can be scheduled on a processor; and

(II) any i tasks among the m tasks $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ cannot be scheduled with any $(m - i + 1)$ tasks in $S - S_m$ on a processor for $i = 1, 2, \ldots, m$

then $S_m \leftarrow S_m + \{\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}\}$ and $S \leftarrow S - \{\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}\}$

(3) If $S \neq \{\}$, then $m \leftarrow m + 1$ and goto (2).

From the algorithm, it is apparent that in the final schedule, if l is the maximum number of tasks assigned on a processor, then any $(l + 1)$ tasks among the n tasks cannot be scheduled on a processor.

If $U > \dfrac{n}{1 + 2^{1/n}}$, then in the worst cases, any two of the n tasks cannot be scheduled in a single processor. Hence, the number of processors required for the scheduling of this set is n. Since

$$\left\lceil \frac{1}{\log[1 + n(2^{1/n} - 1)/U] - 1/n} \right\rceil > n$$

for $U \geq \dfrac{n}{1 + 2^{1/n}}$, the theorem holds.

If $U \leq \dfrac{n}{1 + 2^{1/n}}$, we claim that in the worst cases, the maximum numbers of processors in the optimal schedules, that are required for the feasible scheduling of the task sets, say P, is given by $\left\lceil \dfrac{n}{m} \right\rceil$, where m is determined by finding the smallest m such that

$$U \geq \frac{n}{\sum_{i=0}^{m} 2^{i/n}}. \tag{13}$$

Let us note that such a number of m does exist, since $U \leq n / \sum_{i=0}^{m} 2^{i/n}$ for $m = 1$, and the function $f(m) = n / \sum_{i=0}^{m} 2^{i/n}$ is a monotonically decreasing function with regard to m. Furthermore, by solving inequality (13), we obtain

$$m \geq n \log\left[1 + \frac{n(2^{1/n} - 1)}{U}\right] - 1. \tag{14}$$

Hence,

$$P = \left\lceil \frac{n}{m} \right\rceil \leq \left\lceil \frac{1}{\log\,[1 + n\,(2^{1/n} - 1)\,/U] - 1/n} \right\rceil. \tag{15}$$

Now suppose that the claim is not true, i.e., there exists a worst case where the number of processors required is Q such that $Q > P$. Then let n_i be the number of processors on each of which i tasks are assigned and k and l are the minimum and the maximum number of tasks assigned to a processor, respectively. Then $Q = \sum_{i=k}^{l} n_i$ and $n = \sum_{i=k}^{l} i n_i$.

If $l < m$, i.e., each processor is assigned less than m tasks, then since any $(l + 1)$ of the n tasks cannot be scheduled on a processor, the total utilization of the task set must be greater than $n/\sum_{i=0}^{l} 2^{i/n}$. Since m is the smallest number such that $U \geq n/\sum_{i=0}^{m} 2^{i/n}$, therefore $U < n/\sum_{i=0}^{l} 2^{i/n}$. This indicates that the task set has a greater utilization and thus a contradiction is introduced.

If $l > m$, i.e., some processors are assigned more than m tasks, then since any $(l + 1)$ of the n tasks cannot be scheduled on a processor, the total utilization of the task set must be greater than $\frac{n}{\sum_{i=0}^{l} 2^{i/n}}$. Since m is the smallest number such that $U \geq \frac{n}{\sum_{i=0}^{m} 2^{i/n}}$,

$$U \geq \frac{n}{\sum_{i=0}^{m} 2^{i/n}} > \frac{n}{\sum_{i=0}^{l} 2^{i/n}}.$$

This results in a contradiction.

Therefore, the theorem must be true. ■

III. Earliest Deadline First Scheduling of Periodic Tasks

Before establishing the first result for the Earliest Deadline First algorithm, we restate a result that was first proven in [9] and will serve as a basis for our proof.

Lemma 2: *A set of n tasks* $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$ *can be feasibly scheduled by the Earliest Deadline First algorithm if and only if* $\sum_{i=1}^{n} C_i/T_i \leq 1$.

Theorem 3: *For a set of n tasks, if any* $m + 1$ *of the n tasks cannot be feasibly scheduled on a processor by the Earliest Deadline First algorithm, then the total utilization of the n tasks must be greater than* $n/(m+1)$, *where* $n \geq m + 1$, $n \geq 1$ *and* $m \geq 0$.

Proof: Let the set of n tasks be $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$, where C_i and T_i are the computation time and the period of the task τ_i. Note that the theorem is true when either $n = 1$

or $m = 0$. Hence we need only to consider the case where $n \geq 2$ and $m \geq 1$.

Since any $(m + 1)$ of n tasks cannot be feasibly scheduled on a processor by the Earliest Deadline First algorithm, then

$$\sum_{j=1}^{m+1} u_{i_j} > 1 \tag{16}$$

for all $j = 1, 2, \ldots, m + 1$, $i_j \in [1 \ldots n]$, $i_k \neq i_l$, and $k, l \in [1 \ldots (m + 1)]$, where $u_i = C_i / T_i$. Note that there are a total of $\binom{n}{m+1}$ inequalities in (16).

Summing up the inequalities in (16) yields

$$\binom{n-1}{m} \sum_{i=1}^{n} u_i > \binom{n}{m+1}. \tag{17}$$

Hence, $\sum_{i=1}^{n} u_i > n / (m + 1)$ ■

Now we are ready to state the second result for the Earliest Deadline First algorithm.

Theorem 4: *For any given set of n tasks* $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$, *no more than* $\min(n, \lceil U + U^2 / (n - U) \rceil)$ *processors are required in an optimal schedule, such that the task set can be feasibly with the Earliest Deadline First algorithm, where* $U = \sum_{i=1}^{n} C_i / T_i$.

Proof: For any given set of n tasks with a utilization of $U = \sum_{i=1}^{n} C_i / T_i$, it is apparent that at most n processors are needed for the feasible scheduling of the tasks by the Earliest Deadline First algorithm, as long as $C_i / T_i \leq 1$ for $i = 1, 2, \ldots, n$.

What we want to find out is, with regard to n and U (i.e., the set of the task sets each with n tasks and a utilization of U), the maximum numbers of processors that are necessary to feasibly schedule the task sets in all the optimal schedules. In other words, we are trying to find the maximum of processors used in an optimal schedule in the worst cases. The worst cases are thus defined as such that for some task sets with n tasks each and a utilization of U, the number of processors required by these task sets is no smaller than those required by any set of n tasks with the same utilization of U.

Before we can proceed further, we need to define a procedure for finding the minimum number of processors for any given set Σ of n tasks with a utilization of U. In other words, we will design the optimal algorithm for scheduling a set of periodic tasks, the one which always returns the minimum number of processors for any given set of tasks. By describing such algorithm, we actually define a canonical form for the optimal schedule, to be used in the proof that follows. Note that the Earliest Deadline First algorithm is employed by each processor as its scheduler.

An optimal algorithm is given as follows:

(1) $m = 1$; $S \leftarrow \Sigma$;

(2) Find the largest subset $S_m \subseteq S$ of tasks such that

(i) any $(m + 1)$ tasks in S_m cannot be scheduled on a single processor;

(ii) any i tasks in S_m cannot be scheduled with any $(m - i + 1)$ tasks in $S - S_m$ for $i = 1, 2, \ldots, m$;

(iii) for each task in S_m, there exist some groups of $(m - 1)$ tasks in S_m that these m tasks can be feasibly scheduled on a single processor.

Then $S \leftarrow S - S_m$.

We give the following procedure that can compute such S_m:

(a) $S_m \leftarrow \{\}$;

(b) Rename the task set S such that the $|S|$ tasks are indexed from 1 to $|S|$.

For all $j = 1, 2, \ldots, m$, $\tau_{i_j} \in S$, $i_j = 1, \ldots, |S|$, and $i_k \neq i_l$ with $k, l \in [1 \ldots m]$, repeat the following until either $S = \{\}$ or the conditions (I) and (II) are not true.

If

(I) these m tasks $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ can be scheduled on a processor; and

(II) any i tasks among the m tasks $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ cannot be scheduled with any $(m - i + 1)$ tasks in $S - S_m$ on a processor for $i = 1, 2, \ldots, m$

then $S_m \leftarrow S_m + \{\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}\}$ and $S \leftarrow S - \{\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}\}$

(3) If $S \neq \{\}$, then $m \leftarrow m + 1$ and goto (2).

From the algorithm, it is apparent that in the final schedule, if l is the maximum number of tasks assigned on a processor, then any $(l + 1)$ tasks among the n tasks cannot be scheduled on a processor.

If $U > n/2$, then in the worst cases, any two of the n tasks cannot be scheduled in a single processor. Hence, the number of processors required for the scheduling of this set is n. Since

$$\lceil U + U^2/(n - U) \rceil > n$$

for $U \geq n/2$, the theorem holds.

If $U \leq n/2$, we claim that in the worst cases, the maximum numbers of processors in the optimal schedules, that are required for the feasible scheduling of the task sets, say P, is given by $\left\lceil \frac{n}{m} \right\rceil$, where m is determined by finding the smallest m such that

$$U \geq \frac{n}{m+1}. \tag{18}$$

Let us note that such a number of *m* does exist, since $U \leq n/(m+1)$ for *m* = 1, and the function $f(m) = n/(m+1)$ is a monotonically decreasing function with regard to *m*. Furthermore, by solving inequality (13), we obtain

$$m \geq \frac{n}{U} - 1. \tag{19}$$

Hence,

$$P = \left\lceil \frac{n}{m} \right\rceil \leq \left\lceil U + \frac{U^2}{n-U} \right\rceil. \tag{20}$$

Now suppose that the claim is not true, i.e., there exists a worst case where the number of processors required is *Q* such that *Q* > *P*. Then let n_i be the number of processors on each of which *i* tasks are assigned and *k* and *l* are the minimum and the maximum number of tasks assigned to a processor, respectively. Then $Q = \sum_{i=k}^{l} n_i$ and $n = \sum_{i=k}^{l} i n_i$.

If $l < m$, i.e., each processor is assigned less than *m* tasks, then since any (*l* + 1) of the *n* tasks cannot be scheduled on a processor, the total utilization of the task set must be greater than $n/(l+1)$. Since *m* is the smallest number such that $U \geq n/(m+1)$, therefore $U < n/(l+1)$. This indicates that the task set now has a greater utilization and thus a contradiction is introduced.

If $l > m$, i.e., some processors are assigned more than *m* tasks, then since any (*l* + 1) of the *n* tasks cannot be scheduled on a processor, the total utilization of the task set must be greater than $\frac{n}{l+1}$. Since *m* is the smallest number such that $U \geq \frac{n}{m+1}$,

$$U \geq \frac{n}{m+1} > \frac{n}{l+1}.$$

This results in a contradiction.

Therefore, the theorem must be true. ■

IV. Concluding Remarks

The discovery of the general schedulability condition as given in Theorem 1 for Rate-Monotonic scheduling was the direct result of our attempt to design and analyze effective heuristic algorithms for scheduling periodic tasks on a multiprocessor system. In analyzing various heuristic algorithms for their worst-case performance, we realize that for any given set of tasks, some upper

bounds on the number of processors must be established. However, we also learn from our proof that it is very difficult to establish such bounds, because it usually involves the solutions to non-convex optimization problems.

Further questions remain as how to derive meaningful conditions for periodic tasks which share resources in a multiprocessor environment. Also of interest to our research is the derivation of similar schedulability conditions for scheduling periodic tasks whose deadlines are shorter than their periods. A number of researchers have addressed this problem to some extents, see [1] for example.

Acknowledgments: I would like to thank Dr. Sang H. Son for his support.

References

[1] N.C. AUDSLEY, A. BURNS, M.F. RICHARDSON, K.W. TINDELL, AND A.J. WELLINGS. "Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling," *Software Engineering Journal* **8(5)**: 284-292 (1993).

[2] A. BURCHARD, J. LIEBEHERR, Y. OH, AND S.H. SON. "Assigning Real-Time Tasks to Homogeneous Multiprocessor Systems," *IEEE Transactions on Computer* (to appear).

[3] S. DAVARI AND S.K. DHALL. "An On Line Algorithm for Real-Time Tasks Allocation," *IEEE Real-Time Systems Symposium*, 194-200 (1986).

[4] S.K. DHALL AND C.L. LIU. "On a Real-Time Scheduling Problem," *Operations Research* **26**: 127-140 (1978).

[5] M.R. GAREY AND D.S. JOHNSON. *Computers and Intractability: A Guide to the Theory of NP-completeness*, W.H. Freeman and Company, NY, 1978.

[6] M. JOSEPH AND P. PANDYA. "Finding Response Times in a Real-Time System," *The Computer Journal* **29(5)**: 390-395 (1986).

[7] J. LEHOCZKY, L. SHA, AND Y. DING. "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," *IEEE Real-Time Symposium*, 166-171 (1989).

[8] J.Y.T. LEUNG AND J. WHITEHEAD. "On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks," *Performance Evaluation* **2**: 237-250 (1982).

[9] C.L. LIU AND J. LAYLAND. "Scheduling Algorithms for Multiprogramming in a Hard

Real-Time Environment,” *J. Assoc. Comput. Machinery* **10(1)**: 174-189 (1973).

[10] Y. OH AND S.H. SON. “Allocating Fixed-priority Periodic Tasks on Multiprocessor Systems,” *Journal of Real-Time Systems* (to appear).

[11] Y. OH. The Design and Analysis of Scheduling Algorithms for Real-Time and Fault-Tolerant Computer Systems. Ph.D. Dissertation, Dept. of Computer Science, University of Virginia, May 1994.

[12] P. SERLIN, “Scheduling of Time Critical Processes,” *Proceedings of the Spring Joint Computers Conference* **40**: 925-932 (1972).

[13] L. SHA, R. RAJKUMAR, AND J.P. LEHOCZKY. “Priority Inheritance Protocols: An Approach to Real-Time Synchronization,” *IEEE Transactions on Computers* **39(9)**: 1175-1185 (1990).

Scheduling Periodic Tasks In a Hard Real-Time Environment

Yingfeng Oh

Department of Computer Science
University of Virginia
Charlottesville, VA 22903
USA

Consider the traditional problem of scheduling a set of periodic tasks on a mulitprocessor system, where task deadlines must be guaranteed. We first derive a general schedulability condition for Rate-Monotonic, which reduces the uniprocessor schedulability condition obtained by Liu and Layland and by Serlin, and the multiprocessor schedulability condition recently derived by Burchard, Liebeherr, Oh, and Son to its two specific cases. Then a tight upper bound is obtained for the number of processors required in an optimal schedule for any given set of tasks with a fixed number of tasks and a fixed utilization. Finally, similar conditions are derived for the Earliest Deadline First scheduling. These conditions shed new light on the periodic task scheduling problem.

I. Introduction

In this paper, we consider the problem of scheduling a set of periodic tasks. The task set to be considered is defined as follows:

(1) The requests of each task are periodic, with constant interval between requests.

(2) The deadline constraints specify that each request must be completed before the next request of the same task occurs.

(3) The tasks are independent in that the requests of a task do not depend on the initiation or the completion of the requests of other tasks.

(4) The worst-case run-time (or computation time) for the request of a task is constant for the task. Run-time here refers to the time a processor takes to execute the request without interruption.

The question we are interested in is whether any given set of *n* tasks can be scheduled on *m* processors. In general, this decision problem is NP-complete [5]. Practical solutions to this problem often rely on some kinds of heuristic algorithms, which can deliver approximate solutions in

a short period of time. Heuristic solutions often trade computational time complexity for accuracy of solutions. The approach we take in this paper is to find a schedulability condition for any given set of tasks such that as long as the total utilization or load of the task set is under certain threshold number, the task set can be feasibly scheduled on a certain number of processors. The derivation of this bound subsumes the previous results on schedulability conditions derived by Liu and Layland [9], and by Serlin [12] for Rate-Monotonic scheduling on a single processor system, and by Burchard, Liebeherr, Oh, and Son [2] for Rate-Monotonic scheduling on a multiprocessor system. This tight bound can serve as the basis for constructing more effective heuristic algorithms and for proving tighter worst-case performance guarantee. For more details on how to use schedulability conditions in the design and analysis of heuristics algorithms, reader may refer to [11].

From the description of the task set, it follows that a task is completely defined by two numbers, the run-time of the requests and the request period. We shall denote a task τ_i by the ordered pair (C_i, T_i), where C_i is the computation time and T_i is the period of the requests of the task τ_i. The ratio C_i/T_i is called the utilization (or load) of the task τ_i, and the total utilization (or load) of a set of *n* tasks is given by $U = \sum_{i=1}^{n} C_i/T_i$. All the processors are identical in the sense that the run-time of a task remains the same across all processors.

Tasks can be scheduled for execution on a processor by using a priority-driven algorithm, in which each task is assigned a priority and the task with the highest priority is always the one to be executed. By assigning different priorities to tasks, we therefore determine the schedule of the execution of tasks. A priority assignment algorithm is fixed if the priority of a task remains fixed once it is assigned. Otherwise, it is a dynamic priority assignment algorithm. Here we concern ourselves with priority-driven algorithms only.

If a set of tasks can be scheduled such that all task deadlines can be met by some algorithms, then we say that the task set is *feasible*. If a set of periodic tasks can be feasibly scheduled on a single processor, then the *Rate-Monotonic* (or *RM*) [9] or *Intelligent Fixed Priority* algorithm [12] is optimal for fixed priority assignment, in the sense that no other fixed priority assignment algorithm can schedule a task set which cannot be scheduled by the RM algorithm. The RM algorithm assigns priorities to tasks according to their periods, where the priority of a task is in inverse relationship to its period. In other words, a task with a shorter period is assigned a higher priority. The execution of a low-priority task will be preempted if a high-priority task arrives. Liu and Layland proved that a set of *n* periodic tasks can be feasibly scheduled by the Rate-Monotonic algorithm if

the total utilization of the tasks is no more than a threshold number, which is given by $n\left(2^{1/n}-1\right)$.

One of the important properties of Rate-Monotonic scheduling is that for a single processor system where Rate-Monotonic algorithm is employed to schedule tasks, as long as the CPU utilization of the tasks lies below a certain bound, they will meet their deadlines without the programmer knowing exactly when any given request of a task is running. Even if a transient overload occurs, a fixed subset of the most frequently arrived tasks will still meet their deadlines as long as their total CPU utilization lies below a certain bound. This property puts the real-time software development on a sound analytical footing.

For dynamic priority assignment, the Earliest Deadline First (or EDF) algorithm [9] is optimal in the sense that no other dynamic priority assignment algorithm can schedule a task set which cannot be scheduled by the EDF algorithm. The request of a task is assigned the highest priority if its deadline is the closest. Furthermore, a set of periodic tasks can be feasibly scheduled on a single processor system by the EDF algorithm if and only if its total utilization is no more than one.

Although the schedulability condition, i.e., $\sum_{i=1}^{n} C_i/T_i \le n\,(2^{1/n}-1)$, given by Liu and Layland is simple and elegant, they are pessimistic in nature since the condition is derived under the worst case conditions. Several more efficient conditions were later derived [4, 2, 10, 11]. All these conditions are sufficient but not necessary. The necessary and sufficient condition to schedule a set of periodic tasks using fixed-priority algorithms was given by Joseph and Pandya [6], and by Lehoczky, Sha, and Ding [7].

The results about Rate-Monotonic scheduling are presented in Section II, while similiar results for Earliest Deadline First scheduling are given in Section III. We conclude this paper in Section IV by discussing some remaining issues.

II. Fundamental Conditions for Rate-Monotonic Scheduling

In Theorem 1, we present a general result about rate-monotonic scheduling. It puts a tight upper bound on the number of processors that are required to schedule a set of n tasks such that each task is guaranteed its deadline by the Rate-Monotonic algorithm.

Theorem 1: *For a set of n tasks, if any $m+1$ of the n tasks cannot be feasibly scheduled on a processor by the Rate-Monotonic algorithm, then the total utilization of the n tasks must be greater than $n/\sum_{i=0}^{m} 2^{i/n}$, where $n \ge m+1$, $n \ge 1$ and $m \ge 0$.*

If $m = 0$, then $n/\sum_{i=0}^{m} 2^{i/n} = n$. This is equivalent to saying that if any task cannot be

scheduled on a single processor by the Rate-Monotonic algorithm, then the total utilization of the n tasks must be greater than n. This is trivial true.

If $m = 1$, then $n/\sum_{i=0}^{m} 2^{i/n} = n / \sum_{i=0}^{1} 2^{i/n} = n / \left(2^{1/n} + 1\right)$. This is equivalent to saying that if any two tasks cannot be scheduled on a processor by the Rate-Monotonic algorithm, then the total utilization of the n tasks must be greater than $n / \left(2^{1/n} + 1\right)$. This reduces to the result obtained by Burchard, Liebeherr, Oh, and Son in [2].

If $n = m + 1$, then $n/\sum_{i=0}^{m} 2^{i/n} = n/\sum_{i=0}^{n-1} 2^{i/n} = n\left(2^{1/n} - 1\right)$. This is equivalent to saying that if any n tasks cannot be scheduled on a processor, then the total utilization of the n tasks must be greater than $n\left(2^{1/n} - 1\right)$. This reduces to the result originally obtained by Liu and Layland [9] and Serlin [12].

When n is large, i.e., $n \to \infty$, $n/\sum_{i=0}^{m} 2^{i/n} \to n/(m+1)$. This implies that compared with the utilization bound for the Earliest Deadline First scheduling, the bound given by Theorem 1 for the Rate-Monotonic scheduling is very favorable, i.e., very close to the “best” possible.

In order to prove Theorem 1, we need the following lemma that was proven in [2].

Lemma 1: *If a set of tasks $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$ cannot be scheduled on N processors, then the task set $\Sigma = \{\tau'_i = (C'_i, T'_i) \mid i = 1, 2, \ldots, n\}$ given by $C'_i = T'_i * C_i/T_i$, $T'_i = 2^{V_i}$, and $V_i = \log_2 T_i - \lfloor \log_2 T_i \rfloor$ cannot be scheduled on the N processor either.*

Proof of Theorem 1: Let the set of n tasks be $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$, where C_i and T_i are the computation time and the period of the task τ_i. Note that the theorem is true when either $n = 1$ or $m = 0$. Hence we need only to consider the case where $n \geq 2$ and $m \geq 1$.

According to Lemma 1, we can assume, without loss of generality, that

$$T_1 \leq T_2 \leq \ldots \leq T_n < 2T_1 \qquad (1)$$

Since no $m + 1$ of the n tasks can be scheduled together on a processor, the following conditions must hold according to the necessary and sufficient condition [6, 7]:

$$\begin{cases} C_{i_1} + C_{i_2} + \ldots + C_{i_m} + C_{i_{m+1}} > T_{i_1} \\ 2C_{i_1} + C_{i_2} + \ldots + C_{i_m} + C_{i_{m+1}} > T_{i_2} \\ \ldots\ldots\ldots \\ 2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_m} + C_{i_{m+1}} > T_{i_{m+1}} \end{cases} \qquad (2)$$

where $1 \leq i_1 < \ldots < i_m < i_{m+1} \leq n$.

We want to find the minimum of $U = \sum_{i=1}^{n} C_i/T_i$ subject to the constraints of (1), (2), and

(3).

$$0 < C_i \le T_i \qquad i = 1, \ldots, n \tag{3}$$

In order to ensure that the minimum is obtained at some point, we replace “>” by “≥”. This replacement will not affect the minimum.

We proceed in three steps to obtain the minimum of U:

(1) Fix the values $\bar{C} = (C_1, C_2, \ldots, C_n)$ and express $\bar{T} = (T_1, T_2, \ldots, T_n)$ in terms of $\bar{C}$ in the minimization problem.

(2) Reduce the minimization problem to a convex optimization problem by proving that $C_1 \le C_2 \le \ldots \le C_n \le 2C_1$ if the minimum of U is achieved.

(3) Solve the optimization problem using standard methods.

Let us define

$S_i = \{2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_m} | i_1 < \ldots < i_m, i_x < i, x \in [1 \ldots m]\} \cup$
$\{2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_{m-1}} + C_{i_m} | i_1 < \ldots < i_{m-1}, i_x < i, i_m > i, x \in [1 \ldots (m-1)]\} \cup$
$\ldots \cup \{C_{i_1} + C_{i_2} + \ldots + C_{i_m} | i_1 < \ldots < i_m, i_x > i, x \in [1 \ldots m]\}$,

where $i = 1, 2, \ldots, n$, $C_i > 0$, $n \ge m + 1$, and $1 \le i_x \le n$. The cardinality of each set S_i is given by $|S_i| = \binom{n-1}{m}$. In other words, there are $\binom{n-1}{m}$ inequalities associated with each T_i term that must be satisfied if any $(m+1)$ tasks cannot be scheduled on a single processor.

Let $\alpha_i = \min(S_i)$, i.e., α_i is the minimum member in value of the set S_i. If we view each member of the set S_i as a summation of *m* terms from $(C_1, C_2, \ldots, C_n)$, then α_i is the minimum summation among the $\binom{n-1}{m}$ ones.

Let us further define that for any *i* and *j* such that $i \in [1 \ldots n]$, $j \in [1 \ldots n]$, and $i \ne j$, if the term C_j appears in the summation of α_i, then we say that $C_j \in \alpha_i$ (note that α_i is not a set!). Otherwise, $C_j \notin \alpha_i$.

First, let us assume that $\bar{C} = (C_1, C_2, \ldots, C_n)$ is known.

Since

$$\frac{\partial U}{\partial T_i} = -\frac{C_i}{T_i^2} \tag{4}$$

U decreases as T_i increases. But the increase of T_i cannot exceed the limit that is imposed by the constraints in (2). In other words, *U* is minimized when

$T_i = C_i + \min(\{2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_m} | i_1 < \ldots < i_m, i_x < i, x \in [1 \ldots m]\} \cup$

$\{2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_{m-1}} + C_{i_m} \mid i_1 < \ldots < i_{m-1}, i_x < i, i_m > i, x \in [1\ldots(m-1)]\} \cup$
$\ldots \cup \{C_{i_1} + C_{i_2} + \ldots + C_{i_m} \mid i_1 < \ldots < i_m, i_x > i, x \in [1\ldots m]\}$),
for $i = 1, 2, \ldots, n$.

According to the definition of α_i, we rewrite T_i as $T_i = C_i + \alpha_i$. The minimization problem then becomes

$$U(\bar{C}, \bar{T}) = \sum_{i=1}^{n} C_i / T_i = \sum_{i=1}^{n} C_i / (C_i + m_i) . \qquad (5)$$

Next we show that the minimum of *U* is achieved at $C_1 \le C_2 \le \ldots \le C_n \le 2C_1$. This is accomplished by proving the following three claims.

Claim 1: For every $j \in [1\ldots n]$, there exists at least one index *i* such that

$$C_j \in \alpha_i \text{ or } 2C_j \in \alpha_i . \qquad (6)$$

Suppose that when the minimum of $U(\bar{C}, \bar{T})$ is achieved and (6) is not satisfied, i.e., for some index *j* there does not exist an index $i \ne j$ such that $C_j \in \alpha_i$ if $i < j$ or $2C_j \in \alpha_i$ if $i > j$. Then $U(\bar{C}, \bar{T})$ can be phrased exclusively in terms of $\bar{C}$. Since

$$\frac{\partial}{\partial C_j} U(\bar{C}, \bar{T}) = \frac{\alpha_j}{(C_j + \alpha_j)^2} > 0 ,$$

meaning that *U* increases as C_j increases, we can lower the value of *U* by lowering the value of C_j. Thus, condition (6) is satisfied for any index *j*.

Claim 2: For every C_i with $i \in [1\ldots n]$, there are at most *m* C_ks such that $C_i > C_k$ for $k > i$ i or $C_i > 2C_k$ for $k < i$.

Suppose that the contrary is true, i.e., there exists an index $i \in [1\ldots n]$ such that there are $l \ge m + 1$ C_ks such that $C_i > C_k$ for $k > i$ or $C_i > 2C_k$ for $k < i$. Then for any $k > i$, $C_i \notin \alpha_k$ because there are *l* terms that are smaller than C_i. Similarly, for any $k < i$, $2C_i \notin \alpha_k$. This is a contradiction to Claim 1. Hence Claim 2 must be true. A corollary of this claim is that if $C_i > C_{i+1}$, then $C_{i+1} \in \alpha_i$.

Claim 3: The minimum of *U* is achieved at $C_1 \le C_2 \le \ldots \le C_n \le 2C_1$.

Suppose that there exists a task set Σ' such that the minimum of *U* is achieved when, for some index $i \in [1\ldots n]$, $C_i > C_{i+1}$ for $i < n$ or $C_i > 2C_1$ for $i = n$. We will only present the proof for the case of $i < n$ since the proof for the case of $i = n$ is symmetric.

Claim 3.1: For such a task set, there must exist an index $k \ne i + 1$ such that either (1) $k \le i$, $C_i = C_k$, and $2C_k \in \alpha_{i+1}$; or (2) $k > i + 1$, $2C_i = C_k$, and $C_k \in \alpha_{i+1}$.

The core of the above claim is that the term $2C_i$ must be included in the summation of α_{i+1} if the value of $2C_i$ is unique. The bulk of the claim covers the case where there might be other C_k that is equal to C_i or $2C_i$ in value. Hence it is apparent that we need only to prove that $2C_i \in \alpha_{i+1}$ assuming that $2C_i$ is unique.

If $2C_i \notin \alpha_{i+1}$, then there are at least m C_ks such that $C_k < 2C_i$ for $k > i + 1$ or $C_k < C_i$ for $k < i$. Since $C_i > C_{i+1}$, there are at least (m + 1) such C_ks that are smaller than C_i. This is a contradiction to Claim 2. Therefore, the claim must be true.

Now we can construct a new task set Σ' from the task set Σ as follows: the computation times of the tasks are given by

$$C_1' = C_1$$

$$C_2' = C_2$$

$$\ldots\ldots$$

$$C_{i-1}' = C_{i-1}$$

$$C_i' = C_{i+1}$$

$$C_{i+1}' = C_i$$

$$C_{i+2}' = C_{i+2}$$

$$\ldots\ldots$$

$$C_n' = C_n$$

and the task periods are given by

$$T_1' = T_1$$

$$T_2' = T_2$$

$$\ldots\ldots$$

$$T_{i-1}' = T_{i-1}$$

$$T_i' = \alpha_i + C_i$$

$$T_{i+1}' = \alpha_{i+1} + C_{i+1} - C_i$$

$$T_{i+2}' = T_{i+2}$$

$$\ldots\ldots$$

$$T_n' = T_n .$$

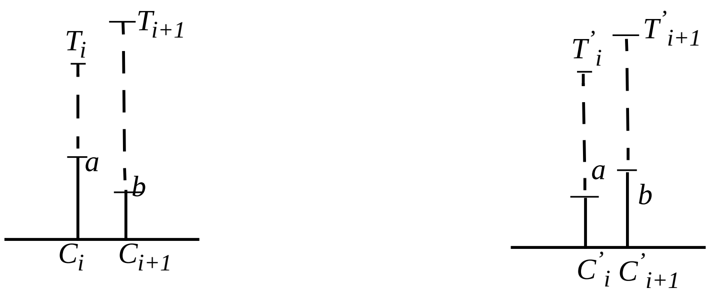

Figure 1: Relationship between two task sets

We want to prove that any $m + 1$ tasks within the newly constructed task set cannot be scheduled on a single processor and $U > U'$, where $U' = \sum_{i=1}^{n} C_i' / T_i'$. Note that this newly constructed task set may not satisfy Claim 1 (but it does not affect the validity of our argument).

First we want to assert that the tasks in Σ' are in the order of non-increasing task periods. From the construction, we need only to consider the order among the tasks T_{i-1}', T_i', T_{i+1}', and T_{i+2}', since the order among the rest of the tasks does not change.

From the definition, it is immediate that $T_{i-1}' \le T_i'$, since $T_i' = T_i$ and $T_{i+1}' < T_{i+1} \le T_{i+2}'$. Since $C_{i+1} \in \alpha_i$ and $2C_i \in \alpha_{i+1}$, the difference between α_{i+1} and α_i is given by $2C_i - C_{i+1}$, i.e., $\alpha_{i+1} - \alpha_i \ge 2C_i - C_{i+1}$. Hence $T_{i+1}' - T_i' = \alpha_{i+1} - \alpha_i + C_{i+1} - 2C_i \ge 0$. Therefore, $T_{i-1}' \le T_i' \le T_{i+1}' \le T_{i+2}'$, and the whole task set is in non-decreasing order of task periods.

Then let us prove that any $m + 1$ tasks from the task set Σ' cannot be scheduled on a single processor.

Obviously, for any l such that $l < i$ or $l > i + 1$, the inequalities for the new task set related to T_l' hold, since the exchange of the values of C_{i+1} and C_i does not affect the original inequalities. More specifically, for each T_l', there are $\binom{n-1}{m}$ inequalities to verify. According to the definition of α_l, if $\alpha_l' + C_l' \ge T_l'$, then the rest of the $\binom{n-1}{m} - 1$ inequalities holds. Since the exchange of the values of C_{i+1} and C_i does not affect the equalities $\alpha_l' = \alpha_l$ for $l < i$ or $l > i + 1$, we have $\alpha_l' + C_l' \ge T_l'$.

Now we shall verify that $\alpha_i' + C_i' \ge T_i'$ and $\alpha_{i+1}' + C_{i+1}' \ge T_{i+1}'$.

Case (i): We shall prove the claim that $\alpha_i' \ge \alpha_i - C_{i+1} + C_i$.

Since $C_i > C_{i+1}$, then $C_{i+1} \in \alpha_i$ according to Claim 2. Then $C_i \in \alpha_i'$ must be true, i.e., the term C_i (or a term C_x with $C_i = C_x$) must be included. Otherwise, for the original C_i in the old task set, there are $(m+1)$ C_ks such that $C_i > C_k$ for $k > i$ or $C_i > 2C_k$ for $k < i$ and the minimum of U is achieved.

Therefore the difference between α_i' and α_i is given by $C_{i+1} - C_i$, i.e., $\alpha_i' \geq \alpha_i - C_{i+1} + C_i$.

$$\alpha_i' + C_i' \geq \alpha_i - C_{i+1} + C_i + C_{i+1} = \alpha_i + C_i = T_i = T_i' .$$

Case (i+1): We shall prove that $\alpha_{i+1}' \geq \alpha_{i+1} - 2C_i + C_{i+1}$.

Since $C_i > C_{i+1}$, then $C_{i+1} \in \alpha_i$ according to Claim 2. Furthermore, there are at most m C_ks such that $C_i > C_k$ for $k > i$ or $C_i > 2C_k$ for $k < i$. Then there are at most $(m-1)$ C_ks such that $C_{i+1} > C_k$ for $k > i+1$ or $C_{i+1} > 2C_k$ for $k < i$

Since $C_i \in \alpha_{i+1}$, it follows that $2C_i' \in \alpha_{i+1}'$. The difference between α_{i+1}' and α_{i+1} is given by $2C_i' - 2C_i$, i.e., $\alpha_{i+1}' \geq \alpha_{i+1} - 2C_i + 2C_{i+1} \geq \alpha_{i+1} - 2C_i + C_{i+1}$.

$$\alpha_{i+1}' + C_{i+1}' \geq \alpha_{i+1} - 2C_i + C_{i+1} + C_i = \alpha_{i+1} + C_{i+1} - C_i = T_{i+1}' .$$

Therefore, any $(m+1)$ tasks in the new task set cannot be scheduled on a single processor by the rate-monotonic algorithm.

Finally, let us prove that $U > U'$.

$$U - U' = \left(\frac{C_i}{T_i} + \frac{C_{i+1}}{T_{i+1}}\right) - \left(\frac{C_i'}{T_i'} + \frac{C_{i+1}'}{T_{i+1}'}\right) = \left(\frac{C_i}{T_i} + \frac{C_{i+1}}{T_{i+1}}\right) - \left(\frac{C_{i+1}}{T_i} + \frac{C_i}{T_{i+1}'}\right)$$

Since $T_i = T_i' < T_{i+1}' < T_{i+1}$, $C_i > C_{i+1}$, and $T_i \geq C_i$, we have $U > U'$.

Therefore, the minimum of $U(x)$ is achieved when

$$C_1 \leq C_2 \leq \ldots \leq C_n \leq 2C_1 .$$

According to the definition of α_i, we have

$$\alpha_i = C_{i+1} + \ldots + C_{i+m} \text{ for } i = 1, 2, \ldots, n-m \text{, and}$$

$$\alpha_i = C_{i+1} + \ldots + C_n + 2C_1 + \ldots + 2C_{i-(n-m)} \text{ for } i = n-m+1, \ldots, n .$$

In other words, the minimum of $U(x)$ is achieved when the task periods satisfy

$$T_i = C_i + C_{i+1} + \ldots + C_{i+m} \text{ for } i = 1, 2, \ldots, n-m \text{, and}$$

$$T_i = C_i + C_{i+1} + \ldots + C_n + 2C_1 + \ldots + 2C_{i-(n-m)} \text{ for } i = n-m+1, \ldots, n .$$

The minimization problem of $U = \sum_{i=1}^{n} C_i / T_i$ now becomes a convex optimization problem.

Finally, we solve the problem by using one of the standard method.

$$\sum_{i=1}^{n} C_i / T_i = \sum_{i=1}^{n-m} \frac{C_i}{\sum_{j=0}^{m} C_{i+j}} + \sum_{i=n-m+1}^{n} \frac{C_i}{\sum_{j=i}^{n} C_j + \sum_{j=1}^{i-(n-m)} 2C_j} \tag{7}$$

Let us define

$$x_i = \log \frac{C_{i+1}}{C_i} \tag{8}$$

for $i = 1, 2, \ldots, n-1$, and

$$x_n = \log \frac{2C_1}{C_n} . \tag{9}$$

Then $\sum_{i=1}^{n} x_i = 1$.

We want to minimize

$$U = \sum_{i=1}^{n-m+1} \frac{1}{1 + \sum_{j=0}^{m-1} 2^{\sum_{k=0}^{j} x_{i+k}}} +$$

$$\sum_{i=n-m+2}^{n} \frac{1}{1 + \sum_{j=0}^{n-i} 2^{\sum_{k=0}^{j} x_{i+k}} + \sum_{j=1}^{i-(n-m+1)} 2^{\sum_{k=i}^{n} x_k + \sum_{k=1}^{j} x_k}} \tag{10}$$

subject to

$$x_i > 0, i = 1, 2, \ldots, n \tag{11}$$

$$\sum_{i=1}^{n} x_i = 1. \tag{12}$$

Since the function and its conditions are symmetric under permutation of indices, the minimum is achieved at $x_i = 1/n$.

Therefore $U = n / \sum_{i=0}^{m} 2^{i/n}$.

Next we show that there indeed exists some task sets such that the above bound is achieved. In other words, the given bound is tight.

Let ε be an arbitrarily small positive number and *a* be a positive number. Then for a task set given by

$$\tau_i = (C_i, T_i) = \left(a2^{i/n} + \varepsilon, a2^{i/n}\left(\sum_{j=0}^{m} 2^{j/n}\right)\right),$$

for $i = 1, 2, \ldots, n$, any $(m + 1)$ of the *n* tasks cannot be scheduled on a single processor. ■

Theorem 2: *For any given set of n tasks* $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, 2, \ldots, n\}$*, no more than* $\min(n, \lceil 1/\{\log[1 + n(2^{1/n} - 1)/U] - 1/n\}\rceil)$ *processors are required in an optimal schedule, such that the tasks can be feasibly scheduled by the Rate-Monotonic algorithm, where* $U = \sum_{i=1}^{n} C_i/T_i$.

Proof: For any given set of *n* tasks with a utilization of $U = \sum_{i=1}^{n} C_i/T_i$, it is apparent that at most *n* processors are needed for the feasible scheduling of the tasks by the Rate-Monotonic algorithm, as long as $C_i/T_i \le 1$ for $i = 1, 2, \ldots, n$.

What we want to find out is, with regard to *n* and *U* (i.e., the set of the task sets each with *n* tasks and a utilization of *U*), the maximum numbers of processors that are necessary to feasibly schedule the task sets in all the optimal schedules. In other words, we are trying to find the maximum of processors used in an optimal schedule in the worst cases. The worst cases are thus defined as such that for some task sets with *n* tasks each and a utilization of *U*, the number of processors required by these task sets is no smaller than those required by any set of *n* tasks with the same utilization of *U*.

Before we can proceed further, we need to define a procedure for finding the minimum number of processors for any given set Σ of *n* tasks with a utilization of *U*, i.e., the number of processors in an optimal schedule. In other words, we will design the optimal algorithm for scheduling a set of periodic tasks, the one which always returns the minimum number of processors for any given set of tasks. By describing such algorithm, we actually define a canonical form for the optimal schedule, to be used in the proof that follows. Note that the Rate-Monotonic algorithm is employed by each processor as its scheduling algorithm.

(1) $m = 1$; $S \leftarrow \Sigma$;

(2) Find the largest subset $S_m \subseteq S$ of tasks such that

(i) any $(m + 1)$ tasks in S_m cannot be scheduled on a single processor;

(ii) any *i* tasks in S_m cannot be scheduled with any $(m - i + 1)$ tasks in $S - S_m$ for $i = 1, 2, \ldots, m$;

(iii) for each task in S_m, there exist some groups of $(m - 1)$ tasks in S_m that these *m*

tasks can be feasibly scheduled on a single processor.

Then $S \leftarrow S - S_m$.

We give the following procedure that can compute such S_m:

(a) $S_m \leftarrow \{\}$;

(b) Rename the task set S such that the $|S|$ tasks are indexed from 1 to $|S|$.

For all $j = 1, 2, \ldots, m$, $\tau_{i_j} \in S$, $i_j = 1, \ldots, |S|$, and $i_k \neq i_l$ with $k, l \in [1 \ldots m]$, repeat the following until either $S = \{\}$ or the conditions (I) and (II) are not true.

If

(I) these m tasks $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ can be scheduled on a processor; and

(II) any i tasks among the m tasks $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ cannot be scheduled with any $(m - i + 1)$ tasks in $S - S_m$ on a processor for $i = 1, 2, \ldots, m$

then $S_m \leftarrow S_m + \{\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}\}$ and $S \leftarrow S - \{\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}\}$

(3) If $S \neq \{\}$, then $m \leftarrow m + 1$ and goto (2).

From the algorithm, it is apparent that in the final schedule, if l is the maximum number of tasks assigned on a processor, then any $(l + 1)$ tasks among the n tasks cannot be scheduled on a processor.

If $U > \frac{n}{1 + 2^{1/n}}$, then in the worst cases, any two of the n tasks cannot be scheduled in a single processor. Hence, the number of processors required for the scheduling of this set is n. Since

$$\left\lceil \frac{1}{\log\left[1 + n\left(2^{1/n} - 1\right)/U\right] - 1/n} \right\rceil > n$$

for $U \geq \frac{n}{1 + 2^{1/n}}$, the theorem holds.

If $U \leq \frac{n}{1 + 2^{1/n}}$, we claim that in the worst cases, the maximum numbers of processors in the optimal schedules, that are required for the feasible scheduling of the task sets, say P, is given by $\left\lceil \frac{n}{m} \right\rceil$, where m is determined by finding the smallest m such that

$$U \geq \frac{n}{\sum_{i=0}^{m} 2^{i/n}}. \tag{13}$$

Let us note that such a number of m does exist, since $U \leq n / \sum_{i=0}^{m} 2^{i/n}$ for $m = 1$, and the function $f(m) = n / \sum_{i=0}^{m} 2^{i/n}$ is a monotonically decreasing function with regard to m. Furthermore, by solving inequality (13), we obtain

$$m \geq n \log\left[1 + \frac{n\left(2^{1/n} - 1\right)}{U}\right] - 1. \tag{14}$$

Hence,

$$P = \left\lceil \frac{n}{m} \right\rceil \leq \left\lceil \frac{1}{\log\,[1 + n\,(2^{1/n} - 1)\,/\,U] \,-\, 1/n} \right\rceil. \tag{15}$$

Now suppose that the claim is not true, i.e., there exists a worst case where the number of processors required is Q such that $Q > P$. Then let n_i be the number of processors on each of which i tasks are assigned and k and l are the minimum and the maximum number of tasks assigned to a processor, respectively. Then $Q = \sum_{i=k}^{l} n_i$ and $n = \sum_{i=k}^{l} i n_i$.

If $l < m$, i.e., each processor is assigned less than m tasks, then since any $(l + 1)$ of the n tasks cannot be scheduled on a processor, the total utilization of the task set must be greater than $n/\sum_{i=0}^{l} 2^{i/n}$. Since m is the smallest number such that $U \geq n/\sum_{i=0}^{m} 2^{i/n}$, therefore $U < n/\sum_{i=0}^{l} 2^{i/n}$. This indicates that the task set has a greater utilization and thus a contradiction is introduced.

If $l > m$, i.e., some processors are assigned more than m tasks, then since any $(l + 1)$ of the n tasks cannot be scheduled on a processor, the total utilization of the task set must be greater than $\frac{n}{\sum_{i=0}^{l} 2^{i/n}}$. Since m is the smallest number such that $U \geq \frac{n}{\sum_{i=0}^{m} 2^{i/n}}$,

$$U \geq \frac{n}{\sum_{i=0}^{m} 2^{i/n}} > \frac{n}{\sum_{i=0}^{l} 2^{i/n}}.$$

This results in a contradiction.

Therefore, the theorem must be true. ■

III. Earliest Deadline First Scheduling of Periodic Tasks

Before establishing the first result for the Earliest Deadline First algorithm, we restate a result that was first proven in [9] and will serve as a basis for our proof.

Lemma 2: *A set of n tasks* $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$ *can be feasibly scheduled by the Earliest Deadline First algorithm if and only if* $\sum_{i=1}^{n} C_i/T_i \leq 1$.

Theorem 3: *For a set of n tasks, if any* $m + 1$ *of the n tasks cannot be feasibly scheduled on a processor by the Earliest Deadline First algorithm, then the total utilization of the n tasks must be greater than* $n/(m + 1)$, *where* $n \geq m + 1$, $n \geq 1$ *and* $m \geq 0$.

Proof: Let the set of n tasks be $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$, where C_i and T_i are the computation time and the period of the task τ_i. Note that the theorem is true when either $n = 1$

or $m = 0$. Hence we need only to consider the case where $n \geq 2$ and $m \geq 1$.

Since any $(m+1)$ of n tasks cannot be feasibly scheduled on a processor by the Earliest Deadline First algorithm, then

$$\sum_{j=1}^{m+1} u_{i_j} > 1 \tag{16}$$

for all $j = 1, 2, \ldots, m+1$, $i_j \in [1 \ldots n]$, $i_k \neq i_l$, and $k, l \in [1 \ldots (m+1)]$, where $u_i = C_i / T_i$. Note that there are a total of $\binom{n}{m+1}$ inequalities in (16).

Summing up the inequalities in (16) yields

$$\binom{n-1}{m} \sum_{i=1}^{n} u_i > \binom{n}{m+1} . \tag{17}$$

Hence, $\sum_{i=1}^{n} u_i > n / (m+1)$ ■

Now we are ready to state the second result for the Earliest Deadline First algorithm.

Theorem 4: *For any given set of n tasks* $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$, *no more than* $\min(n, \lceil U + U^2 / (n - U) \rceil)$ *processors are required in an optimal schedule, such that the task set can be feasibly with the Earliest Deadline First algorithm, where* $U = \sum_{i=1}^{n} C_i / T_i$.

Proof: For any given set of n tasks with a utilization of $U = \sum_{i=1}^{n} C_i / T_i$, it is apparent that at most n processors are needed for the feasible scheduling of the tasks by the Earliest Deadline First algorithm, as long as $C_i / T_i \leq 1$ for $i = 1, 2, \ldots, n$.

What we want to find out is, with regard to n and U (i.e., the set of the task sets each with n tasks and a utilization of U), the maximum numbers of processors that are necessary to feasibly schedule the task sets in all the optimal schedules. In other words, we are trying to find the maximum of processors used in an optimal schedule in the worst cases. The worst cases are thus defined as such that for some task sets with n tasks each and a utilization of U, the number of processors required by these task sets is no smaller than those required by any set of n tasks with the same utilization of U.

Before we can proceed further, we need to define a procedure for finding the minimum number of processors for any given set Σ of n tasks with a utilization of U. In other words, we will design the optimal algorithm for scheduling a set of periodic tasks, the one which always returns the minimum number of processors for any given set of tasks. By describing such algorithm, we actually define a canonical form for the optimal schedule, to be used in the proof that follows. Note that the Earliest Deadline First algorithm is employed by each processor as its scheduler.

An optimal algorithm is given as follows:

(1) $m = 1$; $S \leftarrow \Sigma$;

(2) Find the largest subset $S_m \subseteq S$ of tasks such that

- (i) any $(m + 1)$ tasks in S_m cannot be scheduled on a single processor;
- (ii) any i tasks in S_m cannot be scheduled with any $(m - i + 1)$ tasks in $S - S_m$ for $i = 1, 2, \ldots, m$;
- (iii) for each task in S_m, there exist some groups of $(m - 1)$ tasks in S_m that these m tasks can be feasibly scheduled on a single processor.

Then $S \leftarrow S - S_m$.

We give the following procedure that can compute such S_m:

(a) $S_m \leftarrow \{\}$;

(b) Rename the task set S such that the $|S|$ tasks are indexed from 1 to $|S|$.

For all $j = 1, 2, \ldots, m$, $\tau_{i_j} \in S$, $i_j = 1, \ldots, |S|$, and $i_k \neq i_l$ with $k, l \in [1 \ldots m]$, repeat the following until either $S = \{\}$ or the conditions (I) and (II) are not true.

If

(I) these m tasks $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ can be scheduled on a processor; and

(II) any i tasks among the m tasks $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ cannot be scheduled with any $(m - i + 1)$ tasks in $S - S_m$ on a processor for $i = 1, 2, \ldots, m$

then $S_m \leftarrow S_m + \{\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}\}$ and $S \leftarrow S - \{\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}\}$

(3) If $S \neq \{\}$, then $m \leftarrow m + 1$ and goto (2).

From the algorithm, it is apparent that in the final schedule, if l is the maximum number of tasks assigned on a processor, then any $(l + 1)$ tasks among the n tasks cannot be scheduled on a processor.

If $U > n/2$, then in the worst cases, any two of the n tasks cannot be scheduled in a single processor. Hence, the number of processors required for the scheduling of this set is n. Since

$$\lceil U + U^2/(n - U) \rceil > n$$

for $U \geq n/2$, the theorem holds.

If $U \leq n/2$, we claim that in the worst cases, the maximum numbers of processors in the optimal schedules, that are required for the feasible scheduling of the task sets, say P, is given by $\left\lceil \frac{n}{m} \right\rceil$, where m is determined by finding the smallest m such that

$$U \geq \frac{n}{m+1} . \tag{18}$$

Let us note that such a number of *m* does exist, since $U \leq n/(m+1)$ for *m* = 1, and the function $f(m) = n/(m+1)$ is a monotonically decreasing function with regard to *m*. Furthermore, by solving inequality (13), we obtain

$$m \geq \frac{n}{U} - 1 . \tag{19}$$

Hence,

$$P = \left\lceil \frac{n}{m} \right\rceil \leq \left\lceil U + \frac{U^2}{n - U} \right\rceil . \tag{20}$$

Now suppose that the claim is not true, i.e., there exists a worst case where the number of processors required is *Q* such that *Q* > *P*. Then let n_i be the number of processors on each of which *i* tasks are assigned and *k* and *l* are the minimum and the maximum number of tasks assigned to a processor, respectively. Then $Q = \sum_{i=k}^{l} n_i$ and $n = \sum_{i=k}^{l} i n_i$.

If $l < m$, i.e., each processor is assigned less than *m* tasks, then since any (*l* + 1) of the *n* tasks cannot be scheduled on a processor, the total utilization of the task set must be greater than $n/(l+1)$. Since *m* is the smallest number such that $U \geq n/(m+1)$, therefore $U < n/(l+1)$. This indicates that the task set now has a greater utilization and thus a contradiction is introduced.

If $l > m$, i.e., some processors are assigned more than *m* tasks, then since any (*l* + 1) of the *n* tasks cannot be scheduled on a processor, the total utilization of the task set must be greater than $\frac{n}{l+1}$. Since *m* is the smallest number such that $U \geq \frac{n}{m+1}$,

$$U \geq \frac{n}{m+1} > \frac{n}{l+1} .$$

This results in a contradiction.

Therefore, the theorem must be true. ■

IV. Concluding Remarks

The discovery of the general schedulability condition as given in Theorem 1 for Rate-Monotonic scheduling was the direct result of our attempt to design and analyze effective heuristic algorithms for scheduling periodic tasks on a multiprocessor system. In analyzing various heuristic algorithms for their worst-case performance, we realize that for any given set of tasks, some upper

bounds on the number of processors must be established. However, we also learn from our proof that it is very difficult to establish such bounds, because it usually involves the solutions to non-convex optimization problems.

Further questions remain as how to derive meaningful conditions for periodic tasks which share resources in a multiprocessor environment. Also of interest to our research is the derivation of similar schedulability conditions for scheduling periodic tasks whose deadlines are shorter than their periods. A number of researchers have addressed this problem to some extents, see [1] for example.

Acknowledgments: I would like to thank Dr. Sang H. Son for his support.

References

[1] N.C. AUDSLEY, A. BURNS, M.F. RICHARDSON, K.W. TINDELL, AND A.J. WELLINGS. "Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling," *Software Engineering Journal* **8(5)**: 284-292 (1993).

[2] A. BURCHARD, J. LIEBEHERR, Y. OH, AND S.H. SON. "Assigning Real-Time Tasks to Homogeneous Multiprocessor Systems," *IEEE Transactions on Computer* (to appear).

[3] S. DAVARI AND S.K. DHALL. "An On Line Algorithm for Real-Time Tasks Allocation," *IEEE Real-Time Systems Symposium*, 194-200 (1986).

[4] S.K. DHALL AND C.L. LIU. "On a Real-Time Scheduling Problem," *Operations Research* **26**: 127-140 (1978).

[5] M.R. GAREY AND D.S. JOHNSON. *Computers and Intractability: A Guide to the Theory of NP-completeness*, W.H. Freeman and Company, NY, 1978.

[6] M. JOSEPH AND P. PANDYA. "Finding Response Times in a Real-Time System," *The Computer Journal* **29(5)**: 390-395 (1986).

[7] J. LEHOCZKY, L. SHA, AND Y. DING. "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," *IEEE Real-Time Symposium*, 166-171 (1989).

[8] J.Y.T. LEUNG AND J. WHITEHEAD. "On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks," *Performance Evaluation* **2**: 237-250 (1982).

[9] C.L. LIU AND J. LAYLAND. "Scheduling Algorithms for Multiprogramming in a Hard

Real-Time Environment," *J. Assoc. Comput. Machinery* **10(1)**: 174-189 (1973).

[10] Y. OH AND S.H. SON. "Allocating Fixed-priority Periodic Tasks on Multiprocessor Systems," *Journal of Real-Time Systems* (to appear).

[11] Y. OH. The Design and Analysis of Scheduling Algorithms for Real-Time and Fault-Tolerant Computer Systems. Ph.D. Dissertation, Dept. of Computer Science, University of Virginia, May 1994.

[12] P. SERLIN, "Scheduling of Time Critical Processes," *Proceedings of the Spring Joint Computers Conference* **40**: 925-932 (1972).

[13] L. SHA, R. RAJKUMAR, AND J.P. LEHOCZKY. "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Transactions on Computers* **39(9)**: 1175-1185 (1990).

Scheduling Periodic Tasks In a Hard Real-Time Environment

Yingfeng Oh

Department of Computer Science
University of Virginia
Charlottesville, VA 22903
USA

Consider the traditional problem of scheduling a set of periodic tasks on a mulitprocessor system, where task deadlines must be guaranteed. We first derive a general schedulability condition for Rate-Monotonic, which reduces the uniprocessor schedulability condition obtained by Liu and Layland and by Serlin, and the multiprocessor schedulability condition recently derived by Burchard, Liebeherr, Oh, and Son to its two specific cases. Then a tight upper bound is obtained for the number of processors required in an optimal schedule for any given set of tasks with a fixed number of tasks and a fixed utilization. Finally, similar conditions are derived for the Earliest Deadline First scheduling. These conditions shed new light on the periodic task scheduling problem.

I. Introduction

In this paper, we consider the problem of scheduling a set of periodic tasks. The task set to be considered is defined as follows:

(1) The requests of each task are periodic, with constant interval between requests.

(2) The deadline constraints specify that each request must be completed before the next request of the same task occurs.

(3) The tasks are independent in that the requests of a task do not depend on the initiation or the completion of the requests of other tasks.

(4) The worst-case run-time (or computation time) for the request of a task is constant for the task. Run-time here refers to the time a processor takes to execute the request without interruption.

The question we are interested in is whether any given set of *n* tasks can be scheduled on *m* processors. In general, this decision problem is NP-complete [5]. Practical solutions to this problem often rely on some kinds of heuristic algorithms, which can deliver approximate solutions in

a short period of time. Heuristic solutions often trade computational time complexity for accuracy of solutions. The approach we take in this paper is to find a schedulability condition for any given set of tasks such that as long as the total utilization or load of the task set is under certain threshold number, the task set can be feasibly scheduled on a certain number of processors. The derivation of this bound subsumes the previous results on schedulability conditions derived by Liu and Layland [9], and by Serlin [12] for Rate-Monotonic scheduling on a single processor system, and by Burchard, Liebeherr, Oh, and Son [2] for Rate-Monotonic scheduling on a multiprocessor system. This tight bound can serve as the basis for constructing more effective heuristic algorithms and for proving tighter worst-case performance guarantee. For more details on how to use schedulability conditions in the design and analysis of heuristics algorithms, reader may refer to [11].

From the description of the task set, it follows that a task is completely defined by two numbers, the run-time of the requests and the request period. We shall denote a task τ_i by the ordered pair (C_i, T_i), where C_i is the computation time and T_i is the period of the requests of the task τ_i. The ratio C_i / T_i is called the utilization (or load) of the task τ_i, and the total utilization (or load) of a set of *n* tasks is given by $U = \sum_{i=1}^{n} C_i / T_i$. All the processors are identical in the sense that the run-time of a task remains the same across all processors.

Tasks can be scheduled for execution on a processor by using a priority-driven algorithm, in which each task is assigned a priority and the task with the highest priority is always the one to be executed. By assigning different priorities to tasks, we therefore determine the schedule of the execution of tasks. A priority assignment algorithm is fixed if the priority of a task remains fixed once it is assigned. Otherwise, it is a dynamic priority assignment algorithm. Here we concern ourselves with priority-driven algorithms only.

If a set of tasks can be scheduled such that all task deadlines can be met by some algorithms, then we say that the task set is *feasible*. If a set of periodic tasks can be feasibly scheduled on a single processor, then the *Rate-Monotonic* (or *RM*) [9] or *Intelligent Fixed Priority* algorithm [12] is optimal for fixed priority assignment, in the sense that no other fixed priority assignment algorithm can schedule a task set which cannot be scheduled by the RM algorithm. The RM algorithm assigns priorities to tasks according to their periods, where the priority of a task is in inverse relationship to its period. In other words, a task with a shorter period is assigned a higher priority. The execution of a low-priority task will be preempted if a high-priority task arrives. Liu and Layland proved that a set of *n* periodic tasks can be feasibly scheduled by the Rate-Monotonic algorithm if

the total utilization of the tasks is no more than a threshold number, which is given by $n\left(2^{1/n}-1\right)$.

One of the important properties of Rate-Monotonic scheduling is that for a single processor system where Rate-Monotonic algorithm is employed to schedule tasks, as long as the CPU utilization of the tasks lies below a certain bound, they will meet their deadlines without the programmer knowing exactly when any given request of a task is running. Even if a transient overload occurs, a fixed subset of the most frequently arrived tasks will still meet their deadlines as long as their total CPU utilization lies below a certain bound. This property puts the real-time software development on a sound analytical footing.

For dynamic priority assignment, the Earliest Deadline First (or EDF) algorithm [9] is optimal in the sense that no other dynamic priority assignment algorithm can schedule a task set which cannot be scheduled by the EDF algorithm. The request of a task is assigned the highest priority if its deadline is the closest. Furthermore, a set of periodic tasks can be feasibly scheduled on a single processor system by the EDF algorithm if and only if its total utilization is no more than one.

Although the schedulability condition, i.e., $\sum_{i=1}^{n} C_i/T_i \le n\,(2^{1/n}-1)$, given by Liu and Layland is simple and elegant, they are pessimistic in nature since the condition is derived under the worst case conditions. Several more efficient conditions were later derived [4, 2, 10, 11]. All these conditions are sufficient but not necessary. The necessary and sufficient condition to schedule a set of periodic tasks using fixed-priority algorithms was given by Joseph and Pandya [6], and by Lehoczky, Sha, and Ding [7].

The results about Rate-Monotonic scheduling are presented in Section II, while similiar results for Earliest Deadline First scheduling are given in Section III. We conclude this paper in Section IV by discussing some remaining issues.

II. Fundamental Conditions for Rate-Monotonic Scheduling

In Theorem 1, we present a general result about rate-monotonic scheduling. It puts a tight upper bound on the number of processors that are required to schedule a set of n tasks such that each task is guaranteed its deadline by the Rate-Monotonic algorithm.

Theorem 1: *For a set of n tasks, if any $m+1$ of the n tasks cannot be feasibly scheduled on a processor by the Rate-Monotonic algorithm, then the total utilization of the n tasks must be greater than $n/\sum_{i=0}^{m} 2^{i/n}$, where $n \ge m+1$, $n \ge 1$ and $m \ge 0$.*

If $m = 0$, then $n/\sum_{i=0}^{m} 2^{i/n} = n$. This is equivalent to saying that if any task cannot be

scheduled on a single processor by the Rate-Monotonic algorithm, then the total utilization of the n tasks must be greater than n. This is trivial true.

If $m = 1$, then $n/\sum_{i=0}^{m} 2^{i/n} = n / \sum_{i=0}^{1} 2^{i/n} = n / \left(2^{1/n} + 1\right)$. This is equivalent to saying that if any two tasks cannot be scheduled on a processor by the Rate-Monotonic algorithm, then the total utilization of the n tasks must be greater than $n / \left(2^{1/n} + 1\right)$. This reduces to the result obtained by Burchard, Liebeherr, Oh, and Son in [2].

If $n = m + 1$, then $n/\sum_{i=0}^{m} 2^{i/n} = n/\sum_{i=0}^{n-1} 2^{i/n} = n\left(2^{1/n} - 1\right)$. This is equivalent to saying that if any n tasks cannot be scheduled on a processor, then the total utilization of the n tasks must be greater than $n\left(2^{1/n} - 1\right)$. This reduces to the result originally obtained by Liu and Layland [9] and Serlin [12].

When n is large, i.e., $n \to \infty$, $n/\sum_{i=0}^{m} 2^{i/n} \to n/(m+1)$. This implies that compared with the utilization bound for the Earliest Deadline First scheduling, the bound given by Theorem 1 for the Rate-Monotonic scheduling is very favorable, i.e., very close to the “best” possible.

In order to prove Theorem 1, we need the following lemma that was proven in [2].

Lemma 1: *If a set of tasks $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$ cannot be scheduled on N processors, then the task set $\Sigma = \{\tau'_i = (C'_i, T'_i) \mid i = 1, 2, \ldots, n\}$ given by $C'_i = T'_i * C_i/T_i$, $T'_i = 2^{V_i}$, and $V_i = \log_2 T_i - \lfloor \log_2 T_i \rfloor$ cannot be scheduled on the N processor either.*

Proof of Theorem 1: Let the set of n tasks be $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$, where C_i and T_i are the computation time and the period of the task τ_i. Note that the theorem is true when either $n = 1$ or $m = 0$. Hence we need only to consider the case where $n \geq 2$ and $m \geq 1$.

According to Lemma 1, we can assume, without loss of generality, that

$$T_1 \leq T_2 \leq \ldots \leq T_n < 2T_1 \tag{1}$$

Since no $m + 1$ of the n tasks can be scheduled together on a processor, the following conditions must hold according to the necessary and sufficient condition [6, 7]:

$$\begin{cases} C_{i_1} + C_{i_2} + \ldots + C_{i_m} + C_{i_{m+1}} > T_{i_1} \\ 2C_{i_1} + C_{i_2} + \ldots + C_{i_m} + C_{i_{m+1}} > T_{i_2} \\ \quad\cdots\cdots\cdots \\ 2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_m} + C_{i_{m+1}} > T_{i_{m+1}} \end{cases} \tag{2}$$

where $1 \leq i_1 < \ldots < i_m < i_{m+1} \leq n$.

We want to find the minimum of $U = \sum_{i=1}^{n} C_i/T_i$ subject to the constraints of (1), (2), and

(3).

$$0 < C_i \le T_i \qquad i = 1, \ldots, n \tag{3}$$

In order to ensure that the minimum is obtained at some point, we replace “>” by “≥”. This replacement will not affect the minimum.

We proceed in three steps to obtain the minimum of U:

(1) Fix the values $\bar{C} = (C_1, C_2, \ldots, C_n)$ and express $\bar{T} = (T_1, T_2, \ldots, T_n)$ in terms of $\bar{C}$ in the minimization problem.

(2) Reduce the minimization problem to a convex optimization problem by proving that $C_1 \le C_2 \le \ldots \le C_n \le 2C_1$ if the minimum of U is achieved.

(3) Solve the optimization problem using standard methods.

Let us define

$S_i = \{2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_m} | i_1 < \ldots < i_m, i_x < i, x \in [1\ldots m]\} \cup$
$\{2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_{m-1}} + C_{i_m} | i_1 < \ldots < i_{m-1}, i_x < i, i_m > i, x \in [1\ldots(m-1)]\} \cup$
$\ldots \cup \{C_{i_1} + C_{i_2} + \ldots + C_{i_m} | i_1 < \ldots < i_m, i_x > i, x \in [1\ldots m]\}$,

where $i = 1, 2, \ldots, n$, $C_i > 0$, $n \ge m + 1$, and $1 \le i_x \le n$. The cardinality of each set S_i is given by $|S_i| = \binom{n-1}{m}$. In other words, there are $\binom{n-1}{m}$ inequalities associated with each T_i term that must be satisfied if any $(m + 1)$ tasks cannot be scheduled on a single processor.

Let $\alpha_i = \min(S_i)$, i.e., α_i is the minimum member in value of the set S_i. If we view each member of the set S_i as a summation of m terms from $(C_1, C_2, \ldots, C_n)$, then α_i is the minimum summation among the $\binom{n-1}{m}$ ones.

Let us further define that for any i and j such that $i \in [1\ldots n]$, $j \in [1\ldots n]$, and $i \ne j$, if the term C_j appears in the summation of α_i, then we say that $C_j \in \alpha_i$ (note that α_i is not a set!). Otherwise, $C_j \notin \alpha_i$.

First, let us assume that $\bar{C} = (C_1, C_2, \ldots, C_n)$ is known.

Since

$$\frac{\partial U}{\partial T_i} = -\frac{C_i}{T_i^2} \tag{4}$$

U decreases as T_i increases. But the increase of T_i cannot exceed the limit that is imposed by the constraints in (2). In other words, U is minimized when

$T_i = C_i + \min(\{2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_m} | i_1 < \ldots < i_m, i_x < i, x \in [1\ldots m]\} \cup$

$\{2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_{m-1}} + C_{i_m} | i_1 < \ldots < i_{m-1}, i_x < i, i_m > i, x \in [1\ldots(m-1)]\} \cup$
$\ldots \cup \{C_{i_1} + C_{i_2} + \ldots + C_{i_m} | i_1 < \ldots < i_m, i_x > i, x \in [1\ldots m]\}$),
for $i = 1, 2, \ldots, n$.

According to the definition of α_i, we rewrite T_i as $T_i = C_i + \alpha_i$. The minimization problem then becomes

$$U(\bar{C}, \bar{T}) = \sum_{i=1}^{n} C_i / T_i = \sum_{i=1}^{n} C_i / (C_i + m_i) \ . \qquad (5)$$

Next we show that the minimum of *U* is achieved at $C_1 \le C_2 \le \ldots \le C_n \le 2C_1$. This is accomplished by proving the following three claims.

Claim 1: For every $j \in [1\ldots n]$, there exists at least one index *i* such that

$$C_j \in \alpha_i \text{ or } 2C_j \in \alpha_i. \qquad (6)$$

Suppose that when the minimum of $U(\bar{C}, \bar{T})$ is achieved and (6) is not satisfied, i.e., for some index *j* there does not exist an index $i \ne j$ such that $C_j \in \alpha_i$ if $i < j$ or $2C_j \in \alpha_i$ if $i > j$. Then $U(\bar{C}, \bar{T})$ can be phrased exclusively in terms of $\bar{C}$. Since

$$\frac{\partial}{\partial C_j} U(\bar{C}, \bar{T}) = \frac{\alpha_j}{(C_j + \alpha_j)^2} > 0 \ ,$$

meaning that *U* increases as C_j increases, we can lower the value of *U* by lowering the value of C_j. Thus, condition (6) is satisfied for any index *j*.

Claim 2: For every C_i with $i \in [1\ldots n]$, there are at most *m* C_ks such that $C_i > C_k$ for $k > i$ i or $C_i > 2C_k$ for $k < i$.

Suppose that the contrary is true, i.e., there exists an index $i \in [1\ldots n]$ such that there are $l \ge m+1$ C_ks such that $C_i > C_k$ for $k > i$ or $C_i > 2C_k$ for $k < i$. Then for any $k > i$, $C_i \notin \alpha_k$ because there are *l* terms that are smaller than C_i. Similarly, for any $k < i$, $2C_i \notin \alpha_k$. This is a contradiction to Claim 1. Hence Claim 2 must be true. A corollary of this claim is that if $C_i > C_{i+1}$, then $C_{i+1} \in \alpha_i$.

Claim 3: The minimum of *U* is achieved at $C_1 \le C_2 \le \ldots \le C_n \le 2C_1$.

Suppose that there exists a task set Σ' such that the minimum of *U* is achieved when, for some index $i \in [1\ldots n]$, $C_i > C_{i+1}$ for $i < n$ or $C_i > 2C_1$ for $i = n$. We will only present the proof for the case of $i < n$ since the proof for the case of $i = n$ is symmetric.

Claim 3.1: For such a task set, there must exist an index $k \ne i+1$ such that either (1) $k \le i$, $C_i = C_k$, and $2C_k \in \alpha_{i+1}$; or (2) $k > i+1$, $2C_i = C_k$, and $C_k \in \alpha_{i+1}$.

The core of the above claim is that the term $2C_i$ must be included in the summation of α_{i+1} if the value of $2C_i$ is unique. The bulk of the claim covers the case where there might be other C_k that is equal to C_i or $2C_i$ in value. Hence it is apparent that we need only to prove that $2C_i \in \alpha_{i+1}$ assuming that $2C_i$ is unique.

If $2C_i \notin \alpha_{i+1}$, then there are at least m C_ks such that $C_k < 2C_i$ for $k > i+1$ or $C_k < C_i$ for $k < i$. Since $C_i > C_{i+1}$, there are at least $(m+1)$ such C_ks that are smaller than C_i. This is a contradiction to Claim 2. Therefore, the claim must be true.

Now we can construct a new task set Σ' from the task set Σ as follows: the computation times of the tasks are given by

$$C'_1 = C_1$$

$$C'_2 = C_2$$

$$\ldots\ldots$$

$$C'_{i-1} = C_{i-1}$$

$$C'_i = C_{i+1}$$

$$C'_{i+1} = C_i$$

$$C'_{i+2} = C_{i+2}$$

$$\ldots\ldots$$

$$C'_n = C_n$$

and the task periods are given by

$$T'_1 = T_1$$

$$T'_2 = T_2$$

$$\ldots\ldots$$

$$T'_{i-1} = T_{i-1}$$

$$T'_i = \alpha_i + C_i$$

$$T'_{i+1} = \alpha_{i+1} + C_{i+1} - C_i$$

$$T'_{i+2} = T_{i+2}$$

$$\ldots\ldots$$

$$T'_n = T_n .$$

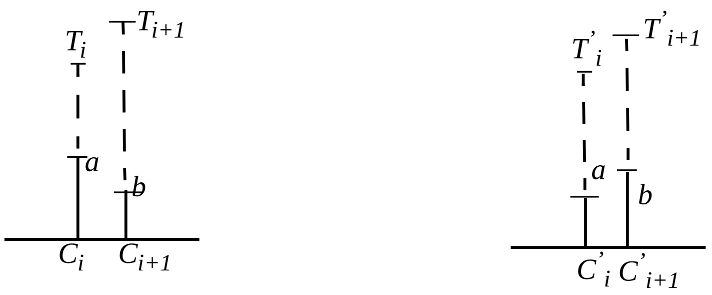

Figure 1: Relationship between two task sets

We want to prove that any $m + 1$ tasks within the newly constructed task set cannot be scheduled on a single processor and $U > U'$, where $U' = \sum_{i=1}^{n} C'_i / T'_i$. Note that this newly constructed task set may not satisfy Claim 1 (but it does not affect the validity of our argument).

First we want to assert that the tasks in Σ' are in the order of non-increasing task periods. From the construction, we need only to consider the order among the tasks T'_{i-1}, T'_i, T'_{i+1}, and T'_{i+2}, since the order among the rest of the tasks does not change.

From the definition, it is immediate that $T'_{i-1} \le T'_i$, since $T'_i = T_i$ and $T'_{i+1} < T_{i+1} \le T'_{i+2}$. Since $C_{i+1} \in \alpha_i$ and $2C_i \in \alpha_{i+1}$, the difference between α_{i+1} and α_i is given by $2C_i - C_{i+1}$, i.e., $\alpha_{i+1} - \alpha_i \ge 2C_i - C_{i+1}$. Hence $T'_{i+1} - T'_i = \alpha_{i+1} - \alpha_i + C_{i+1} - 2C_i \ge 0$. Therefore, $T'_{i-1} \le T'_i \le T'_{i+1} \le T'_{i+2}$, and the whole task set is in non-decreasing order of task periods.

Then let us prove that any $m + 1$ tasks from the task set Σ' cannot be scheduled on a single processor.

Obviously, for any l such that $l < i$ or $l > i + 1$, the inequalities for the new task set related to T'_l hold, since the exchange of the values of C_{i+1} and C_i does not affect the original inequalities. More specifically, for each T'_l, there are $\binom{n-1}{m}$ inequalities to verify. According to the definition of α_l, if $\alpha'_l + C'_l \ge T_l{}'$, then the rest of the $\binom{n-1}{m} - 1$ inequalities holds. Since the exchange of the values of C_{i+1} and C_i does not affect the equalities $\alpha'_l = \alpha_l$ for $l < i$ or $l > i + 1$, we have $\alpha'_l + C'_l \ge T_l{}'$.

Now we shall verify that $\alpha'_i + C'_i \ge T'_i$ and $\alpha'_{i+1} + C'_{i+1} \ge T'_{i+1}$.

Case (i): We shall prove the claim that $\alpha'_i \ge \alpha_i - C_{i+1} + C_i$.

Since $C_i > C_{i+1}$, then $C_{i+1} \in \alpha_i$ according to Claim 2. Then $C_i \in \alpha_i'$ must be true, i.e., the term C_i (or a term C_x with $C_i = C_x$) must be included. Otherwise, for the original C_i in the old task set, there are $(m+1)$ C_ks such that $C_i > C_k$ for $k > i$ or $C_i > 2C_k$ for $k < i$ and the minimum of U is achieved.

Therefore the difference between α_i' and α_i is given by $C_{i+1} - C_i$, i.e., $\alpha_i' \geq \alpha_i - C_{i+1} + C_i$.

$$\alpha_i' + C_i' \geq \alpha_i - C_{i+1} + C_i + C_{i+1} = \alpha_i + C_i = T_i = T_i' .$$

Case (i+1): We shall prove that $\alpha_{i+1}' \geq \alpha_{i+1} - 2C_i + C_{i+1}$.

Since $C_i > C_{i+1}$, then $C_{i+1} \in \alpha_i$ according to Claim 2. Furthermore, there are at most m C_ks such that $C_i > C_k$ for $k > i$ or $C_i > 2C_k$ for $k < i$. Then there are at most $(m-1)$ C_ks such that $C_{i+1} > C_k$ for $k > i+1$ or $C_{i+1} > 2C_k$ for $k < i$

Since $C_i \in \alpha_{i+1}$, it follows that $2C_i' \in \alpha_{i+1}'$. The difference between α_{i+1}' and α_{i+1} is given by $2C_i' - 2C_i$, i.e., $\alpha_{i+1}' \geq \alpha_{i+1} - 2C_i + 2C_{i+1} \geq \alpha_{i+1} - 2C_i + C_{i+1}$.

$$\alpha_{i+1}' + C_{i+1}' \geq \alpha_{i+1} - 2C_i + C_{i+1} + C_i = \alpha_{i+1} + C_{i+1} - C_i = T_{i+1}' .$$

Therefore, any $(m+1)$ tasks in the new task set cannot be scheduled on a single processor by the rate-monotonic algorithm.

Finally, let us prove that $U > U'$.

$$U - U' = \left(\frac{C_i}{T_i} + \frac{C_{i+1}}{T_{i+1}}\right) - \left(\frac{C_i'}{T_i'} + \frac{C_{i+1}'}{T_{i+1}'}\right) = \left(\frac{C_i}{T_i} + \frac{C_{i+1}}{T_{i+1}}\right) - \left(\frac{C_{i+1}}{T_i} + \frac{C_i}{T_{i+1}'}\right)$$

Since $T_i = T_i' < T_{i+1}' < T_{i+1}$, $C_i > C_{i+1}$, and $T_i \geq C_i$, we have $U > U'$.

Therefore, the minimum of $U(x)$ is achieved when

$$C_1 \leq C_2 \leq \ldots \leq C_n \leq 2C_1 .$$

According to the definition of α_i, we have

$$\alpha_i = C_{i+1} + \ldots + C_{i+m} \text{ for } i = 1, 2, \ldots, n-m \text{, and}$$

$$\alpha_i = C_{i+1} + \ldots + C_n + 2C_1 + \ldots + 2C_{i-(n-m)} \text{ for } i = n-m+1, \ldots, n .$$

In other words, the minimum of $U(x)$ is achieved when the task periods satisfy

$$T_i = C_i + C_{i+1} + \ldots + C_{i+m} \text{ for } i = 1, 2, \ldots, n-m \text{, and}$$

$$T_i = C_i + C_{i+1} + \ldots + C_n + 2C_1 + \ldots + 2C_{i-(n-m)} \text{ for } i = n-m+1, \ldots, n .$$

The minimization problem of $U = \sum_{i=1}^{n} C_i / T_i$ now becomes a convex optimization problem.

Finally, we solve the problem by using one of the standard method.

$$\sum_{i=1}^{n} C_i / T_i = \sum_{i=1}^{n-m} \frac{C_i}{\sum_{j=0}^{m} C_{i+j}} + \sum_{i=n-m+1}^{n} \frac{C_i}{\sum_{j=i}^{n} C_j + \sum_{j=1}^{i-(n-m)} 2C_j} \tag{7}$$

Let us define

$$x_i = \log \frac{C_{i+1}}{C_i} \tag{8}$$

for $i = 1, 2, \ldots, n-1$, and

$$x_n = \log \frac{2C_1}{C_n} . \tag{9}$$

Then $\sum_{i=1}^{n} x_i = 1$.

We want to minimize

$$U = \sum_{i=1}^{n-m+1} \frac{1}{1 + \sum_{j=0}^{m-1} 2^{\sum_{k=0}^{j} x_{i+k}}} +$$

$$\sum_{i=n-m+2}^{n} \frac{1}{1 + \sum_{j=0}^{n-i} 2^{\sum_{k=0}^{j} x_{i+k}} + \sum_{j=1}^{i-(n-m+1)} 2^{\sum_{k=i}^{n} x_k + \sum_{k=1}^{j} x_k}} \tag{10}$$

subject to

$$x_i > 0, i = 1, 2, \ldots, n \tag{11}$$

$$\sum_{i=1}^{n} x_i = 1. \tag{12}$$

Since the function and its conditions are symmetric under permutation of indices, the minimum is achieved at $x_i = 1/n$.

Therefore $U = n / \sum_{i=0}^{m} 2^{i/n}$.

Next we show that there indeed exists some task sets such that the above bound is achieved. In other words, the given bound is tight.

Let ε be an arbitrarily small positive number and *a* be a positive number. Then for a task set given by

$$\tau_i = (C_i, T_i) = \left(a2^{i/n} + \varepsilon, a2^{i/n}\left(\sum_{j=0}^{m} 2^{j/n}\right)\right),$$

for $i = 1, 2, \ldots, n$, any $(m + 1)$ of the *n* tasks cannot be scheduled on a single processor. ■

Theorem 2: *For any given set of n tasks* $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, 2, \ldots, n\}$, *no more than* $\min(n, \lceil 1/\{\log[1 + n(2^{1/n} - 1)/U] - 1/n\}\rceil)$ *processors are required in an optimal schedule, such that the tasks can be feasibly scheduled by the Rate-Monotonic algorithm, where* $U = \sum_{i=1}^{n} C_i/T_i$.

Proof: For any given set of *n* tasks with a utilization of $U = \sum_{i=1}^{n} C_i/T_i$, it is apparent that at most *n* processors are needed for the feasible scheduling of the tasks by the Rate-Monotonic algorithm, as long as $C_i/T_i \le 1$ for $i = 1, 2, \ldots, n$.

What we want to find out is, with regard to *n* and *U* (i.e., the set of the task sets each with *n* tasks and a utilization of *U*), the maximum numbers of processors that are necessary to feasibly schedule the task sets in all the optimal schedules. In other words, we are trying to find the maximum of processors used in an optimal schedule in the worst cases. The worst cases are thus defined as such that for some task sets with *n* tasks each and a utilization of *U*, the number of processors required by these task sets is no smaller than those required by any set of *n* tasks with the same utilization of *U*.

Before we can proceed further, we need to define a procedure for finding the minimum number of processors for any given set Σ of *n* tasks with a utilization of *U*, i.e., the number of processors in an optimal schedule. In other words, we will design the optimal algorithm for scheduling a set of periodic tasks, the one which always returns the minimum number of processors for any given set of tasks. By describing such algorithm, we actually define a canonical form for the optimal schedule, to be used in the proof that follows. Note that the Rate-Monotonic algorithm is employed by each processor as its scheduling algorithm.

(1) $m = 1$; $S \leftarrow \Sigma$;

(2) Find the largest subset $S_m \subseteq S$ of tasks such that

(i) any $(m + 1)$ tasks in S_m cannot be scheduled on a single processor;

(ii) any *i* tasks in S_m cannot be scheduled with any $(m - i + 1)$ tasks in $S - S_m$ for $i = 1, 2, \ldots, m$;

(iii) for each task in S_m, there exist some groups of $(m - 1)$ tasks in S_m that these *m*

tasks can be feasibly scheduled on a single processor.

Then $S \leftarrow S - S_m$.

We give the following procedure that can compute such S_m:

(a) $S_m \leftarrow \{\}$;

(b) Rename the task set S such that the $|S|$ tasks are indexed from 1 to $|S|$.

For all $j = 1, 2, \ldots, m$, $\tau_{i_j} \in S$, $i_j = 1, \ldots, |S|$, and $i_k \neq i_l$ with $k, l \in [1 \ldots m]$, repeat the following until either $S = \{\}$ or the conditions (I) and (II) are not true.

If

(I) these m tasks $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ can be scheduled on a processor; and

(II) any i tasks among the m tasks $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ cannot be scheduled with any $(m - i + 1)$ tasks in $S - S_m$ on a processor for $i = 1, 2, \ldots, m$

then $S_m \leftarrow S_m + \{\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}\}$ and $S \leftarrow S - \{\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}\}$

(3) If $S \neq \{\}$, then $m \leftarrow m + 1$ and goto (2).

From the algorithm, it is apparent that in the final schedule, if l is the maximum number of tasks assigned on a processor, then any $(l + 1)$ tasks among the n tasks cannot be scheduled on a processor.

If $U > \frac{n}{1 + 2^{1/n}}$, then in the worst cases, any two of the n tasks cannot be scheduled in a single processor. Hence, the number of processors required for the scheduling of this set is n. Since

$$\left\lceil \frac{1}{\log\left[1 + n\left(2^{1/n} - 1\right)/U\right] - 1/n} \right\rceil > n$$

for $U \geq \frac{n}{1 + 2^{1/n}}$, the theorem holds.

If $U \leq \frac{n}{1 + 2^{1/n}}$, we claim that in the worst cases, the maximum numbers of processors in the optimal schedules, that are required for the feasible scheduling of the task sets, say P, is given by $\left\lceil \frac{n}{m} \right\rceil$, where m is determined by finding the smallest m such that

$$U \geq \frac{n}{\sum_{i=0}^{m} 2^{i/n}}. \qquad (13)$$

Let us note that such a number of m does exist, since $U \leq n / \sum_{i=0}^{m} 2^{i/n}$ for $m = 1$, and the function $f(m) = n / \sum_{i=0}^{m} 2^{i/n}$ is a monotonically decreasing function with regard to m. Furthermore, by solving inequality (13), we obtain

$$m \geq n \log\left[1 + \frac{n\left(2^{1/n} - 1\right)}{U}\right] - 1. \qquad (14)$$

Hence,

$$P = \left\lceil \frac{n}{m} \right\rceil \le \left\lceil \frac{1}{\log\,[1 + n\,(2^{1/n} - 1)\,/\,U] - 1/n} \right\rceil. \qquad (15)$$

Now suppose that the claim is not true, i.e., there exists a worst case where the number of processors required is Q such that $Q > P$. Then let n_i be the number of processors on each of which i tasks are assigned and k and l are the minimum and the maximum number of tasks assigned to a processor, respectively. Then $Q = \sum_{i=k}^{l} n_i$ and $n = \sum_{i=k}^{l} i n_i$.

If $l < m$, i.e., each processor is assigned less than m tasks, then since any $(l + 1)$ of the n tasks cannot be scheduled on a processor, the total utilization of the task set must be greater than $n / \sum_{i=0}^{l} 2^{i/n}$. Since m is the smallest number such that $U \ge n / \sum_{i=0}^{m} 2^{i/n}$, therefore $U < n / \sum_{i=0}^{l} 2^{i/n}$. This indicates that the task set has a greater utilization and thus a contradiction is introduced.

If $l > m$, i.e., some processors are assigned more than m tasks, then since any $(l + 1)$ of the n tasks cannot be scheduled on a processor, the total utilization of the task set must be greater than $\frac{n}{\sum_{i=0}^{l} 2^{i/n}}$. Since m is the smallest number such that $U \ge \frac{n}{\sum_{i=0}^{m} 2^{i/n}}$,

$$U \ge \frac{n}{\sum_{i=0}^{m} 2^{i/n}} > \frac{n}{\sum_{i=0}^{l} 2^{i/n}}.$$

This results in a contradiction.

Therefore, the theorem must be true. ■

III. Earliest Deadline First Scheduling of Periodic Tasks

Before establishing the first result for the Earliest Deadline First algorithm, we restate a result that was first proven in [9] and will serve as a basis for our proof.

Lemma 2: *A set of n tasks* $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$ *can be feasibly scheduled by the Earliest Deadline First algorithm if and only if* $\sum_{i=1}^{n} C_i / T_i \le 1$.

Theorem 3: *For a set of n tasks, if any* $m + 1$ *of the n tasks cannot be feasibly scheduled on a processor by the Earliest Deadline First algorithm, then the total utilization of the n tasks must be greater than* $n / (m + 1)$, *where* $n \ge m + 1$, $n \ge 1$ *and* $m \ge 0$.

Proof: Let the set of n tasks be $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$, where C_i and T_i are the computation time and the period of the task τ_i. Note that the theorem is true when either $n = 1$

or $m = 0$. Hence we need only to consider the case where $n \geq 2$ and $m \geq 1$.

Since any $(m+1)$ of n tasks cannot be feasibly scheduled on a processor by the Earliest Deadline First algorithm, then

$$\sum_{j=1}^{m+1} u_{i_j} > 1 \tag{16}$$

for all $j = 1, 2, \ldots, m+1$, $i_j \in [1 \ldots n]$, $i_k \neq i_l$, and $k, l \in [1 \ldots (m+1)]$, where $u_i = C_i / T_i$. Note that there are a total of $\binom{n}{m+1}$ inequalities in (16).

Summing up the inequalities in (16) yields

$$\binom{n-1}{m} \sum_{i=1}^{n} u_i > \binom{n}{m+1}. \tag{17}$$

Hence, $\sum_{i=1}^{n} u_i > n / (m+1)$ ■

Now we are ready to state the second result for the Earliest Deadline First algorithm.

Theorem 4: *For any given set of n tasks* $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$, *no more than* $\min(n, \lceil U + U^2/(n-U) \rceil)$ *processors are required in an optimal schedule, such that the task set can be feasibly with the Earliest Deadline First algorithm, where* $U = \sum_{i=1}^{n} C_i / T_i$.

Proof: For any given set of n tasks with a utilization of $U = \sum_{i=1}^{n} C_i / T_i$, it is apparent that at most n processors are needed for the feasible scheduling of the tasks by the Earliest Deadline First algorithm, as long as $C_i / T_i \leq 1$ for $i = 1, 2, \ldots, n$.

What we want to find out is, with regard to *n* and *U* (i.e., the set of the task sets each with *n* tasks and a utilization of *U*), the maximum numbers of processors that are necessary to feasibly schedule the task sets in all the optimal schedules. In other words, we are trying to find the maximum of processors used in an optimal schedule in the worst cases. The worst cases are thus defined as such that for some task sets with *n* tasks each and a utilization of *U*, the number of processors required by these task sets is no smaller than those required by any set of *n* tasks with the same utilization of *U*.

Before we can proceed further, we need to define a procedure for finding the minimum number of processors for any given set Σ of *n* tasks with a utilization of *U*. In other words, we will design the optimal algorithm for scheduling a set of periodic tasks, the one which always returns the minimum number of processors for any given set of tasks. By describing such algorithm, we actually define a canonical form for the optimal schedule, to be used in the proof that follows. Note that the Earliest Deadline First algorithm is employed by each processor as its scheduler.

An optimal algorithm is given as follows:

```
(1) $m = 1$; $S \leftarrow \Sigma$;
(2) Find the largest subset $S_m \subseteq S$ of tasks such that
    (i)   any $(m + 1)$ tasks in $S_m$ cannot be scheduled on a single processor;
    (ii)  any $i$ tasks in $S_m$ cannot be scheduled with any $(m - i + 1)$ tasks in $S - S_m$ for
          $i = 1, 2, \ldots, m$;
    (iii) for each task in $S_m$, there exist some groups of $(m - 1)$ tasks in $S_m$ that these $m$
          tasks can be feasibly scheduled on a single processor.
    Then $S \leftarrow S - S_m$.
    We give the following procedure that can compute such $S_m$:
    (a) $S_m \leftarrow \{\}$;
    (b) Rename the task set $S$ such that the $|S|$ tasks are indexed from 1 to $|S|$.
        For all $j = 1, 2, \ldots, m$, $\tau_{i_j} \in S$, $i_j = 1, \ldots, |S|$, and $i_k \neq i_l$ with $k, l \in [1 \ldots m]$,
        repeat the following until either $S = \{\}$ or the conditions (I) and (II) are not true.
            If
            (I)  these $m$ tasks $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ can be scheduled on a processor; and
            (II) any $i$ tasks among the $m$ tasks $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ cannot be scheduled with any
                 $(m - i + 1)$ tasks in $S - S_m$ on a processor for $i = 1, 2, \ldots, m$
            then $S_m \leftarrow S_m + \{\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}\}$ and $S \leftarrow S - \{\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}\}$
(3) If $S \neq \{\}$, then $m \leftarrow m + 1$ and goto (2).
```

From the algorithm, it is apparent that in the final schedule, if l is the maximum number of tasks assigned on a processor, then any $(l + 1)$ tasks among the n tasks cannot be scheduled on a processor.

If $U > n/2$, then in the worst cases, any two of the n tasks cannot be scheduled in a single processor. Hence, the number of processors required for the scheduling of this set is n. Since

$$\lceil U + U^2/(n - U) \rceil > n$$

for $U \geq n/2$, the theorem holds.

If $U \leq n/2$, we claim that in the worst cases, the maximum numbers of processors in the optimal schedules, that are required for the feasible scheduling of the task sets, say P, is given by $\left\lceil \frac{n}{m} \right\rceil$, where m is determined by finding the smallest m such that

$$U \geq \frac{n}{m+1}. \tag{18}$$

Let us note that such a number of *m* does exist, since $U \leq n/(m+1)$ for *m* = 1, and the function $f(m) = n/(m+1)$ is a monotonically decreasing function with regard to *m*. Furthermore, by solving inequality (13), we obtain

$$m \geq \frac{n}{U} - 1. \tag{19}$$

Hence,

$$P = \left\lceil \frac{n}{m} \right\rceil \leq \left\lceil U + \frac{U^2}{n-U} \right\rceil. \tag{20}$$

Now suppose that the claim is not true, i.e., there exists a worst case where the number of processors required is *Q* such that *Q* > *P*. Then let n_i be the number of processors on each of which *i* tasks are assigned and *k* and *l* are the minimum and the maximum number of tasks assigned to a processor, respectively. Then $Q = \sum_{i=k}^{l} n_i$ and $n = \sum_{i=k}^{l} i n_i$.

If $l < m$, i.e., each processor is assigned less than *m* tasks, then since any (*l* + 1) of the *n* tasks cannot be scheduled on a processor, the total utilization of the task set must be greater than $n/(l+1)$. Since *m* is the smallest number such that $U \geq n/(m+1)$, therefore $U < n/(l+1)$. This indicates that the task set now has a greater utilization and thus a contradiction is introduced.

If $l > m$, i.e., some processors are assigned more than *m* tasks, then since any (*l* + 1) of the *n* tasks cannot be scheduled on a processor, the total utilization of the task set must be greater than $\frac{n}{l+1}$. Since *m* is the smallest number such that $U \geq \frac{n}{m+1}$,

$$U \geq \frac{n}{m+1} > \frac{n}{l+1}.$$

This results in a contradiction.

Therefore, the theorem must be true. ■

IV. Concluding Remarks

The discovery of the general schedulability condition as given in Theorem 1 for Rate-Monotonic scheduling was the direct result of our attempt to design and analyze effective heuristic algorithms for scheduling periodic tasks on a multiprocessor system. In analyzing various heuristic algorithms for their worst-case performance, we realize that for any given set of tasks, some upper

bounds on the number of processors must be established. However, we also learn from our proof that it is very difficult to establish such bounds, because it usually involves the solutions to non-convex optimization problems.

Further questions remain as how to derive meaningful conditions for periodic tasks which share resources in a multiprocessor environment. Also of interest to our research is the derivation of similar schedulability conditions for scheduling periodic tasks whose deadlines are shorter than their periods. A number of researchers have addressed this problem to some extents, see [1] for example.

Acknowledgments: I would like to thank Dr. Sang H. Son for his support.

References

[1] N.C. AUDSLEY, A. BURNS, M.F. RICHARDSON, K.W. TINDELL, AND A.J. WELLINGS. "Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling," *Software Engineering Journal* **8(5)**: 284-292 (1993).

[2] A. BURCHARD, J. LIEBEHERR, Y. OH, AND S.H. SON. "Assigning Real-Time Tasks to Homogeneous Multiprocessor Systems," *IEEE Transactions on Computer* (to appear).

[3] S. DAVARI AND S.K. DHALL. "An On Line Algorithm for Real-Time Tasks Allocation," *IEEE Real-Time Systems Symposium*, 194-200 (1986).

[4] S.K. DHALL AND C.L. LIU. "On a Real-Time Scheduling Problem," *Operations Research* **26**: 127-140 (1978).

[5] M.R. GAREY AND D.S. JOHNSON. *Computers and Intractability: A Guide to the Theory of NP-completeness*, W.H. Freeman and Company, NY, 1978.

[6] M. JOSEPH AND P. PANDYA. "Finding Response Times in a Real-Time System," *The Computer Journal* **29(5)**: 390-395 (1986).

[7] J. LEHOCZKY, L. SHA, AND Y. DING. "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," *IEEE Real-Time Symposium*, 166-171 (1989).

[8] J.Y.T. LEUNG AND J. WHITEHEAD. "On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks," *Performance Evaluation* **2**: 237-250 (1982).

[9] C.L. LIU AND J. LAYLAND. "Scheduling Algorithms for Multiprogramming in a Hard

Real-Time Environment,” *J. Assoc. Comput. Machinery* **10(1)**: 174-189 (1973).

[10] Y. OH AND S.H. SON. “Allocating Fixed-priority Periodic Tasks on Multiprocessor Systems,” *Journal of Real-Time Systems* (to appear).

[11] Y. OH. The Design and Analysis of Scheduling Algorithms for Real-Time and Fault-Tolerant Computer Systems. Ph.D. Dissertation, Dept. of Computer Science, University of Virginia, May 1994.

[12] P. SERLIN, “Scheduling of Time Critical Processes,” *Proceedings of the Spring Joint Computers Conference* **40**: 925-932 (1972).

[13] L. SHA, R. RAJKUMAR, AND J.P. LEHOCZKY. “Priority Inheritance Protocols: An Approach to Real-Time Synchronization,” *IEEE Transactions on Computers* **39(9)**: 1175-1185 (1990).

Scheduling Periodic Tasks In a Hard Real-Time Environment

Yingfeng Oh

Department of Computer Science
University of Virginia
Charlottesville, VA 22903
USA

Consider the traditional problem of scheduling a set of periodic tasks on a mulitprocessor system, where task deadlines must be guaranteed. We first derive a general schedulability condition for Rate-Monotonic, which reduces the uniprocessor schedulability condition obtained by Liu and Layland and by Serlin, and the multiprocessor schedulability condition recently derived by Burchard, Liebeherr, Oh, and Son to its two specific cases. Then a tight upper bound is obtained for the number of processors required in an optimal schedule for any given set of tasks with a fixed number of tasks and a fixed utilization. Finally, similar conditions are derived for the Earliest Deadline First scheduling. These conditions shed new light on the periodic task scheduling problem.

I. Introduction

In this paper, we consider the problem of scheduling a set of periodic tasks. The task set to be considered is defined as follows:

(1) The requests of each task are periodic, with constant interval between requests.

(2) The deadline constraints specify that each request must be completed before the next request of the same task occurs.

(3) The tasks are independent in that the requests of a task do not depend on the initiation or the completion of the requests of other tasks.

(4) The worst-case run-time (or computation time) for the request of a task is constant for the task. Run-time here refers to the time a processor takes to execute the request without interruption.

The question we are interested in is whether any given set of *n* tasks can be scheduled on *m* processors. In general, this decision problem is NP-complete [5]. Practical solutions to this problem often rely on some kinds of heuristic algorithms, which can deliver approximate solutions in

a short period of time. Heuristic solutions often trade computational time complexity for accuracy of solutions. The approach we take in this paper is to find a schedulability condition for any given set of tasks such that as long as the total utilization or load of the task set is under certain threshold number, the task set can be feasibly scheduled on a certain number of processors. The derivation of this bound subsumes the previous results on schedulability conditions derived by Liu and Layland [9], and by Serlin [12] for Rate-Monotonic scheduling on a single processor system, and by Burchard, Liebeherr, Oh, and Son [2] for Rate-Monotonic scheduling on a multiprocessor system. This tight bound can serve as the basis for constructing more effective heuristic algorithms and for proving tighter worst-case performance guarantee. For more details on how to use schedulability conditions in the design and analysis of heuristics algorithms, reader may refer to [11].

From the description of the task set, it follows that a task is completely defined by two numbers, the run-time of the requests and the request period. We shall denote a task τ_i by the ordered pair (C_i, T_i), where C_i is the computation time and T_i is the period of the requests of the task τ_i. The ratio C_i/T_i is called the utilization (or load) of the task τ_i, and the total utilization (or load) of a set of *n* tasks is given by $U = \sum_{i=1}^{n} C_i/T_i$. All the processors are identical in the sense that the run-time of a task remains the same across all processors.

Tasks can be scheduled for execution on a processor by using a priority-driven algorithm, in which each task is assigned a priority and the task with the highest priority is always the one to be executed. By assigning different priorities to tasks, we therefore determine the schedule of the execution of tasks. A priority assignment algorithm is fixed if the priority of a task remains fixed once it is assigned. Otherwise, it is a dynamic priority assignment algorithm. Here we concern ourselves with priority-driven algorithms only.

If a set of tasks can be scheduled such that all task deadlines can be met by some algorithms, then we say that the task set is *feasible*. If a set of periodic tasks can be feasibly scheduled on a single processor, then the *Rate-Monotonic* (or *RM*) [9] or *Intelligent Fixed Priority* algorithm [12] is optimal for fixed priority assignment, in the sense that no other fixed priority assignment algorithm can schedule a task set which cannot be scheduled by the RM algorithm. The RM algorithm assigns priorities to tasks according to their periods, where the priority of a task is in inverse relationship to its period. In other words, a task with a shorter period is assigned a higher priority. The execution of a low-priority task will be preempted if a high-priority task arrives. Liu and Layland proved that a set of *n* periodic tasks can be feasibly scheduled by the Rate-Monotonic algorithm if

the total utilization of the tasks is no more than a threshold number, which is given by $n\left(2^{1/n}-1\right)$.

One of the important properties of Rate-Monotonic scheduling is that for a single processor system where Rate-Monotonic algorithm is employed to schedule tasks, as long as the CPU utilization of the tasks lies below a certain bound, they will meet their deadlines without the programmer knowing exactly when any given request of a task is running. Even if a transient overload occurs, a fixed subset of the most frequently arrived tasks will still meet their deadlines as long as their total CPU utilization lies below a certain bound. This property puts the real-time software development on a sound analytical footing.

For dynamic priority assignment, the Earliest Deadline First (or EDF) algorithm [9] is optimal in the sense that no other dynamic priority assignment algorithm can schedule a task set which cannot be scheduled by the EDF algorithm. The request of a task is assigned the highest priority if its deadline is the closest. Furthermore, a set of periodic tasks can be feasibly scheduled on a single processor system by the EDF algorithm if and only if its total utilization is no more than one.

Although the schedulability condition, i.e., $\sum_{i=1}^{n} C_i/T_i \leq n\,(2^{1/n}-1)$, given by Liu and Layland is simple and elegant, they are pessimistic in nature since the condition is derived under the worst case conditions. Several more efficient conditions were later derived [4, 2, 10, 11]. All these conditions are sufficient but not necessary. The necessary and sufficient condition to schedule a set of periodic tasks using fixed-priority algorithms was given by Joseph and Pandya [6], and by Lehoczky, Sha, and Ding [7].

The results about Rate-Monotonic scheduling are presented in Section II, while similiar results for Earliest Deadline First scheduling are given in Section III. We conclude this paper in Section IV by discussing some remaining issues.

II. Fundamental Conditions for Rate-Monotonic Scheduling

In Theorem 1, we present a general result about rate-monotonic scheduling. It puts a tight upper bound on the number of processors that are required to schedule a set of n tasks such that each task is guaranteed its deadline by the Rate-Monotonic algorithm.

Theorem 1: *For a set of n tasks, if any $m+1$ of the n tasks cannot be feasibly scheduled on a processor by the Rate-Monotonic algorithm, then the total utilization of the n tasks must be greater than $n/\sum_{i=0}^{m} 2^{i/n}$, where $n \geq m+1$, $n \geq 1$ and $m \geq 0$.*

If $m = 0$, then $n/\sum_{i=0}^{m} 2^{i/n} = n$. This is equivalent to saying that if any task cannot be

scheduled on a single processor by the Rate-Monotonic algorithm, then the total utilization of the *n* tasks must be greater than *n*. This is trivial true.

If $m = 1$, then $n / \sum_{i=0}^{m} 2^{i/n} = n / \sum_{i=0}^{1} 2^{i/n} = n / \left(2^{1/n} + 1\right)$. This is equivalent to saying that if any two tasks cannot be scheduled on a processor by the Rate-Monotonic algorithm, then the total utilization of the *n* tasks must be greater than $n / \left(2^{1/n} + 1\right)$. This reduces to the result obtained by Burchard, Liebeherr, Oh, and Son in [2].

If $n = m + 1$, then $n / \sum_{i=0}^{m} 2^{i/n} = n / \sum_{i=0}^{n-1} 2^{i/n} = n\left(2^{1/n} - 1\right)$. This is equivalent to saying that if any *n* tasks cannot be scheduled on a processor, then the total utilization of the *n* tasks must be greater than $n\left(2^{1/n} - 1\right)$. This reduces to the result originally obtained by Liu and Layland [9] and Serlin [12].

When *n* is large, i.e., $n \to \infty$, $n / \sum_{i=0}^{m} 2^{i/n} \to n / (m + 1)$. This implies that compared with the utilization bound for the Earliest Deadline First scheduling, the bound given by Theorem 1 for the Rate-Monotonic scheduling is very favorable, i.e., very close to the "best" possible.

In order to prove Theorem 1, we need the following lemma that was proven in [2].

Lemma 1: *If a set of tasks* $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$ *cannot be scheduled on N processors, then the task set* $\Sigma = \{\tau_i' = (C_i', T_i') \mid i = 1, 2, \ldots, n\}$ *given by* $C_i' = T_i' * C_i / T_i$, $T_i' = 2^{V_i}$, *and* $V_i = \log_2 T_i - \lfloor \log_2 T_i \rfloor$ *cannot be scheduled on the N processor either.*

Proof of Theorem 1: Let the set of *n* tasks be $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$, where C_i and T_i are the computation time and the period of the task τ_i. Note that the theorem is true when either $n = 1$ or $m = 0$. Hence we need only to consider the case where $n \geq 2$ and $m \geq 1$.

According to Lemma 1, we can assume, without loss of generality, that

$$T_1 \leq T_2 \leq \ldots \leq T_n < 2T_1 \tag{1}$$

Since no $m + 1$ of the *n* tasks can be scheduled together on a processor, the following conditions must hold according to the necessary and sufficient condition [6, 7]:

$$\begin{cases} C_{i_1} + C_{i_2} + \ldots + C_{i_m} + C_{i_{m+1}} > T_{i_1} \\ 2C_{i_1} + C_{i_2} + \ldots + C_{i_m} + C_{i_{m+1}} > T_{i_2} \\ \cdots\cdots\cdots \\ 2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_m} + C_{i_{m+1}} > T_{i_{m+1}} \end{cases} \tag{2}$$

where $1 \leq i_1 < \ldots < i_m < i_{m+1} \leq n$.

We want to find the minimum of $U = \sum_{i=1}^{n} C_i / T_i$ subject to the constraints of (1), (2), and

(3).

$$0 < C_i \le T_i \qquad i = 1, \ldots, n \tag{3}$$

In order to ensure that the minimum is obtained at some point, we replace ">" by "≥". This replacement will not affect the minimum.

We proceed in three steps to obtain the minimum of U:

(1) Fix the values $\bar{C} = (C_1, C_2, \ldots, C_n)$ and express $\bar{T} = (T_1, T_2, \ldots, T_n)$ in terms of $\bar{C}$ in the minimization problem.

(2) Reduce the minimization problem to a convex optimization problem by proving that $C_1 \le C_2 \le \ldots \le C_n \le 2C_1$ if the minimum of *U* is achieved.

(3) Solve the optimization problem using standard methods.

Let us define

$S_i = \{2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_m} | i_1 < \ldots < i_m, i_x < i, x \in [1 \ldots m]\} \cup$
$\{2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_{m-1}} + C_{i_m} | i_1 < \ldots < i_{m-1}, i_x < i, i_m > i, x \in [1 \ldots (m-1)]\} \cup$
$\ldots \cup \{C_{i_1} + C_{i_2} + \ldots + C_{i_m} | i_1 < \ldots < i_m, i_x > i, x \in [1 \ldots m]\}$,

where $i = 1, 2, \ldots, n$, $C_i > 0$, $n \ge m + 1$, and $1 \le i_x \le n$. The cardinality of each set S_i is given by $|S_i| = \binom{n-1}{m}$. In other words, there are $\binom{n-1}{m}$ inequalities associated with each T_i term that must be satisfied if any $(m + 1)$ tasks cannot be scheduled on a single processor.

Let $\alpha_i = \min(S_i)$, i.e., α_i is the minimum member in value of the set S_i. If we view each member of the set S_i as a summation of *m* terms from $(C_1, C_2, \ldots, C_n)$, then α_i is the minimum summation among the $\binom{n-1}{m}$ ones.

Let us further define that for any *i* and *j* such that $i \in [1 \ldots n]$, $j \in [1 \ldots n]$, and $i \ne j$, if the term C_j appears in the summation of α_i, then we say that $C_j \in \alpha_i$ (note that α_i is not a set!). Otherwise, $C_j \notin \alpha_i$.

First, let us assume that $\bar{C} = (C_1, C_2, \ldots, C_n)$ is known.

Since

$$\frac{\partial U}{\partial T_i} = -\frac{C_i}{T_i^2} \tag{4}$$

U decreases as T_i increases. But the increase of T_i cannot exceed the limit that is imposed by the constraints in (2). In other words, *U* is minimized when

$T_i = C_i + \min(\{2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_m} | i_1 < \ldots < i_m, i_x < i, x \in [1 \ldots m]\} \cup$

$\{2C_{i_1} + 2C_{i_2} + \ldots + 2C_{i_{m-1}} + C_{i_m} | i_1 < \ldots < i_{m-1}, i_x < i, i_m > i, x \in [1 \ldots (m-1)]\} \cup$
$\ldots \cup \{C_{i_1} + C_{i_2} + \ldots + C_{i_m} | i_1 < \ldots < i_m, i_x > i, x \in [1 \ldots m]\})$,
for $i = 1, 2, \ldots, n$.

According to the definition of α_i, we rewrite T_i as $T_i = C_i + \alpha_i$. The minimization problem then becomes

$$U(\bar{C}, \bar{T}) = \sum_{i=1}^{n} C_i / T_i = \sum_{i=1}^{n} C_i / (C_i + m_i) . \tag{5}$$

Next we show that the minimum of *U* is achieved at $C_1 \le C_2 \le \ldots \le C_n \le 2C_1$. This is accomplished by proving the following three claims.

Claim 1: For every $j \in [1 \ldots n]$, there exists at least one index *i* such that

$$C_j \in \alpha_i \text{ or } 2C_j \in \alpha_i. \tag{6}$$

Suppose that when the minimum of $U(\bar{C}, \bar{T})$ is achieved and (6) is not satisfied, i.e., for some index *j* there does not exist an index $i \ne j$ such that $C_j \in \alpha_i$ if $i < j$ or $2C_j \in \alpha_i$ if $i > j$. Then $U(\bar{C}, \bar{T})$ can be phrased exclusively in terms of $\bar{C}$. Since

$$\frac{\partial}{\partial C_j} U(\bar{C}, \bar{T}) = \frac{\alpha_j}{(C_j + \alpha_j)^2} > 0 ,$$

meaning that *U* increases as C_j increases, we can lower the value of *U* by lowering the value of C_j. Thus, condition (6) is satisfied for any index *j*.

Claim 2: For every C_i with $i \in [1 \ldots n]$, there are at most *m* C_ks such that $C_i > C_k$ for $k > i$ i or $C_i > 2C_k$ for $k < i$.

Suppose that the contrary is true, i.e., there exists an index $i \in [1 \ldots n]$ such that there are $l \ge m + 1$ C_ks such that $C_i > C_k$ for $k > i$ or $C_i > 2C_k$ for $k < i$. Then for any $k > i$, $C_i \notin \alpha_k$ because there are *l* terms that are smaller than C_i. Similarly, for any $k < i$, $2C_i \notin \alpha_k$. This is a contradiction to Claim 1. Hence Claim 2 must be true. A corollary of this claim is that if $C_i > C_{i+1}$, then $C_{i+1} \in \alpha_i$.

Claim 3: The minimum of *U* is achieved at $C_1 \le C_2 \le \ldots \le C_n \le 2C_1$.

Suppose that there exists a task set Σ' such that the minimum of *U* is achieved when, for some index $i \in [1 \ldots n]$, $C_i > C_{i+1}$ for $i < n$ or $C_i > 2C_1$ for $i = n$. We will only present the proof for the case of $i < n$ since the proof for the case of $i = n$ is symmetric.

Claim 3.1: For such a task set, there must exist an index $k \ne i + 1$ such that either (1) $k \le i$, $C_i = C_k$, and $2C_k \in \alpha_{i+1}$; or (2) $k > i + 1$, $2C_i = C_k$, and $C_k \in \alpha_{i+1}$.

The core of the above claim is that the term $2C_i$ must be included in the summation of α_{i+1} if the value of $2C_i$ is unique. The bulk of the claim covers the case where there might be other C_k that is equal to C_i or $2C_i$ in value. Hence it is apparent that we need only to prove that $2C_i \in \alpha_{i+1}$ assuming that $2C_i$ is unique.

If $2C_i \notin \alpha_{i+1}$, then there are at least m C_ks such that $C_k < 2C_i$ for $k > i + 1$ or $C_k < C_i$ for $k < i$. Since $C_i > C_{i+1}$, there are at least $(m + 1)$ such C_ks that are smaller than C_i. This is a contradiction to Claim 2. Therefore, the claim must be true.

Now we can construct a new task set Σ' from the task set Σ as follows: the computation times of the tasks are given by

$$C'_1 = C_1$$

$$C'_2 = C_2$$

$$\ldots\ldots$$

$$C'_{i-1} = C_{i-1}$$

$$C'_i = C_{i+1}$$

$$C'_{i+1} = C_i$$

$$C'_{i+2} = C_{i+2}$$

$$\ldots\ldots$$

$$C'_n = C_n$$

and the task periods are given by

$$T'_1 = T_1$$

$$T'_2 = T_2$$

$$\ldots\ldots$$

$$T'_{i-1} = T_{i-1}$$

$$T'_i = \alpha_i + C_i$$

$$T'_{i+1} = \alpha_{i+1} + C_{i+1} - C_i$$

$$T'_{i+2} = T_{i+2}$$

$$\ldots\ldots$$

$$T'_n = T_n.$$

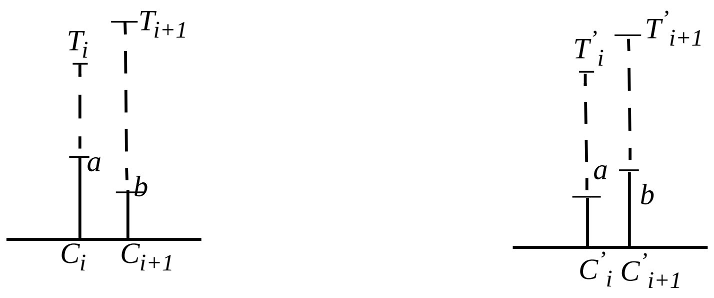

Figure 1: Relationship between two task sets

We want to prove that any $m+1$ tasks within the newly constructed task set cannot be scheduled on a single processor and $U > U'$, where $U' = \sum_{i=1}^{n} C'_i / T'_i$. Note that this newly constructed task set may not satisfy Claim 1 (but it does not affect the validity of our argument).

First we want to assert that the tasks in Σ' are in the order of non-increasing task periods. From the construction, we need only to consider the order among the tasks T'_{i-1}, T'_i, T'_{i+1}, and T'_{i+2}, since the order among the rest of the tasks does not change.

From the definition, it is immediate that $T'_{i-1} \leq T'_i$, since $T'_i = T_i$ and $T'_{i+1} < T_{i+1} \leq T'_{i+2}$. Since $C_{i+1} \in \alpha_i$ and $2C_i \in \alpha_{i+1}$, the difference between α_{i+1} and α_i is given by $2C_i - C_{i+1}$, i.e., $\alpha_{i+1} - \alpha_i \geq 2C_i - C_{i+1}$. Hence $T'_{i+1} - T'_i = \alpha_{i+1} - \alpha_i + C_{i+1} - 2C_i \geq 0$. Therefore, $T'_{i-1} \leq T'_i \leq T'_{i+1} \leq T'_{i+2}$, and the whole task set is in non-decreasing order of task periods.

Then let us prove that any $m+1$ tasks from the task set Σ' cannot be scheduled on a single processor.

Obviously, for any l such that $l < i$ or $l > i+1$, the inequalities for the new task set related to T'_l hold, since the exchange of the values of C_{i+1} and C_i does not affect the original inequalities. More specifically, for each T'_l, there are $\binom{n-1}{m}$ inequalities to verify. According to the definition of α_l, if $\alpha'_l + C'_l \geq T_l'$, then the rest of the $\binom{n-1}{m} - 1$ inequalities holds. Since the exchange of the values of C_{i+1} and C_i does not affect the equalities $\alpha'_l = \alpha_l$ for $l < i$ or $l > i+1$, we have $\alpha'_l + C'_l \geq T_l'$.

Now we shall verify that $\alpha'_i + C'_i \geq T'_i$ and $\alpha'_{i+1} + C'_{i+1} \geq T'_{i+1}$.

Case (i): We shall prove the claim that $\alpha'_i \geq \alpha_i - C_{i+1} + C_i$.

Since $C_i > C_{i+1}$, then $C_{i+1} \in \alpha_i$ according to Claim 2. Then $C_i \in \alpha_i'$ must be true, i.e., the term C_i (or a term C_x with $C_i = C_x$) must be included. Otherwise, for the original C_i in the old task set, there are $(m+1)$ C_ks such that $C_i > C_k$ for $k > i$ or $C_i > 2C_k$ for $k < i$ and the minimum of U is achieved.

Therefore the difference between α_i' and α_i is given by $C_{i+1} - C_i$, i.e., $\alpha_i' \geq \alpha_i - C_{i+1} + C_i$.

$$\alpha_i' + C_i' \geq \alpha_i - C_{i+1} + C_i + C_{i+1} = \alpha_i + C_i = T_i = T_i' .$$

Case (i+1): We shall prove that $\alpha_{i+1}' \geq \alpha_{i+1} - 2C_i + C_{i+1}$.

Since $C_i > C_{i+1}$, then $C_{i+1} \in \alpha_i$ according to Claim 2. Furthermore, there are at most m C_ks such that $C_i > C_k$ for $k > i$ or $C_i > 2C_k$ for $k < i$. Then there are at most $(m-1)$ C_ks such that $C_{i+1} > C_k$ for $k > i+1$ or $C_{i+1} > 2C_k$ for $k < i$

Since $C_i \in \alpha_{i+1}$, it follows that $2C_i' \in \alpha_{i+1}'$. The difference between α_{i+1}' and α_{i+1} is given by $2C_i' - 2C_i$, i.e., $\alpha_{i+1}' \geq \alpha_{i+1} - 2C_i + 2C_{i+1} \geq \alpha_{i+1} - 2C_i + C_{i+1}$.

$$\alpha_{i+1}' + C_{i+1}' \geq \alpha_{i+1} - 2C_i + C_{i+1} + C_i = \alpha_{i+1} + C_{i+1} - C_i = T_{i+1}' .$$

Therefore, any $(m+1)$ tasks in the new task set cannot be scheduled on a single processor by the rate-monotonic algorithm.

Finally, let us prove that $U > U'$.

$$U - U' = \left(\frac{C_i}{T_i} + \frac{C_{i+1}}{T_{i+1}}\right) - \left(\frac{C_i'}{T_i'} + \frac{C_{i+1}'}{T_{i+1}'}\right) = \left(\frac{C_i}{T_i} + \frac{C_{i+1}}{T_{i+1}}\right) - \left(\frac{C_{i+1}}{T_i} + \frac{C_i}{T_{i+1}'}\right)$$

Since $T_i = T_i' < T_{i+1}' < T_{i+1}$, $C_i > C_{i+1}$, and $T_i \geq C_i$, we have $U > U'$.

Therefore, the minimum of $U(x)$ is achieved when

$$C_1 \leq C_2 \leq \ldots \leq C_n \leq 2C_1 .$$

According to the definition of α_i, we have

$$\alpha_i = C_{i+1} + \ldots + C_{i+m} \text{ for } i = 1, 2, \ldots, n-m \text{, and}$$

$$\alpha_i = C_{i+1} + \ldots + C_n + 2C_1 + \ldots + 2C_{i-(n-m)} \text{ for } i = n-m+1, \ldots, n .$$

In other words, the minimum of $U(x)$ is achieved when the task periods satisfy

$$T_i = C_i + C_{i+1} + \ldots + C_{i+m} \text{ for } i = 1, 2, \ldots, n-m \text{, and}$$

$$T_i = C_i + C_{i+1} + \ldots + C_n + 2C_1 + \ldots + 2C_{i-(n-m)} \text{ for } i = n-m+1, \ldots, n .$$

The minimization problem of $U = \sum_{i=1}^{n} C_i/T_i$ now becomes a convex optimization problem.

Finally, we solve the problem by using one of the standard method.

$$\sum_{i=1}^{n} C_i/T_i = \sum_{i=1}^{n-m} \frac{C_i}{\sum_{j=0}^{m} C_{i+j}} + \sum_{i=n-m+1}^{n} \frac{C_i}{\sum_{j=i}^{n} C_j + \sum_{j=1}^{i-(n-m)} 2C_j} \tag{7}$$

Let us define

$$x_i = \log \frac{C_{i+1}}{C_i} \tag{8}$$

for $i = 1, 2, \ldots, n-1$, and

$$x_n = \log \frac{2C_1}{C_n} . \tag{9}$$

Then $\sum_{i=1}^{n} x_i = 1$.

We want to minimize

$$U = \sum_{i=1}^{n-m+1} \frac{1}{1 + \sum_{j=0}^{m-1} 2^{\sum_{k=0}^{j} x_{i+k}}} + \sum_{i=n-m+2}^{n} \frac{1}{1 + \sum_{j=0}^{n-i} 2^{\sum_{k=0}^{j} x_{i+k}} + \sum_{j=1}^{i-(n-m+1)} 2^{\sum_{k=i}^{n} x_k + \sum_{k=1}^{j} x_k}} \tag{10}$$

subject to

$$x_i > 0, i = 1, 2, \ldots, n \tag{11}$$

$$\sum_{i=1}^{n} x_i = 1. \tag{12}$$

Since the function and its conditions are symmetric under permutation of indices, the minimum is achieved at $x_i = 1/n$.

Therefore $U = n/\sum_{i=0}^{m} 2^{i/n}$.

Next we show that there indeed exists some task sets such that the above bound is achieved. In other words, the given bound is tight.

Let ε be an arbitrarily small positive number and *a* be a positive number. Then for a task set given by

$$\tau_i = (C_i, T_i) = \left(a2^{i/n} + \varepsilon, a2^{i/n}\left(\sum_{j=0}^{m} 2^{j/n}\right)\right),$$

for $i = 1, 2, \ldots, n$, any $(m + 1)$ of the *n* tasks cannot be scheduled on a single processor. ■

Theorem 2: *For any given set of n tasks* $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, 2, \ldots, n\}$, *no more than* $\min(n, \lceil 1/\{\log[1 + n(2^{1/n} - 1)/U] - 1/n\}\rceil)$ *processors are required in an optimal schedule, such that the tasks can be feasibly scheduled by the Rate-Monotonic algorithm, where* $U = \sum_{i=1}^{n} C_i/T_i$.

Proof: For any given set of *n* tasks with a utilization of $U = \sum_{i=1}^{n} C_i/T_i$, it is apparent that at most *n* processors are needed for the feasible scheduling of the tasks by the Rate-Monotonic algorithm, as long as $C_i/T_i \le 1$ for $i = 1, 2, \ldots, n$.

What we want to find out is, with regard to *n* and *U* (i.e., the set of the task sets each with *n* tasks and a utilization of *U*), the maximum numbers of processors that are necessary to feasibly schedule the task sets in all the optimal schedules. In other words, we are trying to find the maximum of processors used in an optimal schedule in the worst cases. The worst cases are thus defined as such that for some task sets with *n* tasks each and a utilization of *U*, the number of processors required by these task sets is no smaller than those required by any set of *n* tasks with the same utilization of *U*.

Before we can proceed further, we need to define a procedure for finding the minimum number of processors for any given set Σ of *n* tasks with a utilization of *U*, i.e., the number of processors in an optimal schedule. In other words, we will design the optimal algorithm for scheduling a set of periodic tasks, the one which always returns the minimum number of processors for any given set of tasks. By describing such algorithm, we actually define a canonical form for the optimal schedule, to be used in the proof that follows. Note that the Rate-Monotonic algorithm is employed by each processor as its scheduling algorithm.

(1) $m = 1$; $S \leftarrow \Sigma$;

(2) Find the largest subset $S_m \subseteq S$ of tasks such that

- (i) any $(m + 1)$ tasks in S_m cannot be scheduled on a single processor;
- (ii) any *i* tasks in S_m cannot be scheduled with any $(m - i + 1)$ tasks in $S - S_m$ for $i = 1, 2, \ldots, m$;
- (iii) for each task in S_m, there exist some groups of $(m - 1)$ tasks in S_m that these *m*

tasks can be feasibly scheduled on a single processor.

Then $S \leftarrow S - S_m$.

We give the following procedure that can compute such S_m:

(a) $S_m \leftarrow \{\}$;

(b) Rename the task set S such that the $|S|$ tasks are indexed from 1 to $|S|$.

For all $j = 1, 2, \ldots, m$, $\tau_{i_j} \in S$, $i_j = 1, \ldots, |S|$, and $i_k \neq i_l$ with $k, l \in [1 \ldots m]$, repeat the following until either $S = \{\}$ or the conditions (I) and (II) are not true.

If

(I) these m tasks $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ can be scheduled on a processor; and

(II) any i tasks among the m tasks $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ cannot be scheduled with any $(m - i + 1)$ tasks in $S - S_m$ on a processor for $i = 1, 2, \ldots, m$

then $S_m \leftarrow S_m + \{\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}\}$ and $S \leftarrow S - \{\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}\}$

(3) If $S \neq \{\}$, then $m \leftarrow m + 1$ and goto (2).

From the algorithm, it is apparent that in the final schedule, if l is the maximum number of tasks assigned on a processor, then any $(l + 1)$ tasks among the n tasks cannot be scheduled on a processor.

If $U > \frac{n}{1 + 2^{1/n}}$, then in the worst cases, any two of the n tasks cannot be scheduled in a single processor. Hence, the number of processors required for the scheduling of this set is n. Since

$$\left\lceil \frac{1}{\log\left[1 + n\left(2^{1/n} - 1\right)/U\right] - 1/n} \right\rceil > n$$

for $U \geq \frac{n}{1 + 2^{1/n}}$, the theorem holds.

If $U \leq \frac{n}{1 + 2^{1/n}}$, we claim that in the worst cases, the maximum numbers of processors in the optimal schedules, that are required for the feasible scheduling of the task sets, say P, is given by $\left\lceil \frac{n}{m} \right\rceil$, where m is determined by finding the smallest m such that

$$U \geq \frac{n}{\sum_{i=0}^{m} 2^{i/n}}. \tag{13}$$

Let us note that such a number of m does exist, since $U \leq n / \sum_{i=0}^{m} 2^{i/n}$ for $m = 1$, and the function $f(m) = n / \sum_{i=0}^{m} 2^{i/n}$ is a monotonically decreasing function with regard to m. Furthermore, by solving inequality (13), we obtain

$$m \geq n \log\left[1 + \frac{n\left(2^{1/n} - 1\right)}{U}\right] - 1. \tag{14}$$

Hence,

$$P = \left\lceil \frac{n}{m} \right\rceil \le \left\lceil \frac{1}{\log\,[1 + n\,(2^{1/n} - 1)\,/\,U] - 1/n} \right\rceil . \qquad (15)$$

Now suppose that the claim is not true, i.e., there exists a worst case where the number of processors required is Q such that $Q > P$. Then let n_i be the number of processors on each of which i tasks are assigned and k and l are the minimum and the maximum number of tasks assigned to a processor, respectively. Then $Q = \sum_{i=k}^{l} n_i$ and $n = \sum_{i=k}^{l} i n_i$.

If $l < m$, i.e., each processor is assigned less than m tasks, then since any $(l + 1)$ of the n tasks cannot be scheduled on a processor, the total utilization of the task set must be greater than $n / \sum_{i=0}^{l} 2^{i/n}$. Since m is the smallest number such that $U \ge n / \sum_{i=0}^{m} 2^{i/n}$, therefore $U < n / \sum_{i=0}^{l} 2^{i/n}$. This indicates that the task set has a greater utilization and thus a contradiction is introduced.

If $l > m$, i.e., some processors are assigned more than m tasks, then since any $(l + 1)$ of the n tasks cannot be scheduled on a processor, the total utilization of the task set must be greater than $\frac{n}{\sum_{i=0}^{l} 2^{i/n}}$. Since m is the smallest number such that $U \ge \frac{n}{\sum_{i=0}^{m} 2^{i/n}}$,

$$U \ge \frac{n}{\sum_{i=0}^{m} 2^{i/n}} > \frac{n}{\sum_{i=0}^{l} 2^{i/n}} .$$

This results in a contradiction.

Therefore, the theorem must be true. ■

III. Earliest Deadline First Scheduling of Periodic Tasks

Before establishing the first result for the Earliest Deadline First algorithm, we restate a result that was first proven in [9] and will serve as a basis for our proof.

Lemma 2: *A set of n tasks* $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$ *can be feasibly scheduled by the Earliest Deadline First algorithm if and only if* $\sum_{i=1}^{n} C_i / T_i \le 1$.

Theorem 3: *For a set of n tasks, if any* $m + 1$ *of the n tasks cannot be feasibly scheduled on a processor by the Earliest Deadline First algorithm, then the total utilization of the n tasks must be greater than* $n / (m + 1)$, *where* $n \ge m + 1$, $n \ge 1$ *and* $m \ge 0$.

Proof: Let the set of n tasks be $\Sigma = \{\tau_i = (C_i, T_i) \mid i = 1, \ldots, n\}$, where C_i and T_i are the computation time and the period of the task τ_i. Note that the theorem is true when either $n = 1$

or $m = 0$. Hence we need only to consider the case where $n \geq 2$ and $m \geq 1$.

Since any $(m + 1)$ of n tasks cannot be feasibly scheduled on a processor by the Earliest Deadline First algorithm, then

$$\sum_{j=1}^{m+1} u_{i_j} > 1 \tag{16}$$

for all $j = 1, 2, \ldots, m + 1$, $i_j \in [1 \ldots n]$, $i_k \neq i_l$, and $k, l \in [1 \ldots (m + 1)]$, where $u_i = C_i / T_i$. Note that there are a total of $\binom{n}{m+1}$ inequalities in (16).

Summing up the inequalities in (16) yields

$$\binom{n-1}{m} \sum_{i=1}^{n} u_i > \binom{n}{m+1}. \tag{17}$$

Hence, $\sum_{i=1}^{n} u_i > n / (m + 1)$ ■

Now we are ready to state the second result for the Earliest Deadline First algorithm.

Theorem 4: *For any given set of n tasks* $\Sigma = \{ \tau_i = (C_i, T_i) \mid i = 1, \ldots, n \}$, *no more than* $\min(n, \lceil U + U^2 / (n - U) \rceil)$ *processors are required in an optimal schedule, such that the task set can be feasibly with the Earliest Deadline First algorithm, where* $U = \sum_{i=1}^{n} C_i / T_i$.

Proof: For any given set of *n* tasks with a utilization of $U = \sum_{i=1}^{n} C_i / T_i$, it is apparent that at most *n* processors are needed for the feasible scheduling of the tasks by the Earliest Deadline First algorithm, as long as $C_i / T_i \leq 1$ for $i = 1, 2, \ldots, n$.

What we want to find out is, with regard to *n* and *U* (i.e., the set of the task sets each with *n* tasks and a utilization of *U*), the maximum numbers of processors that are necessary to feasibly schedule the task sets in all the optimal schedules. In other words, we are trying to find the maximum of processors used in an optimal schedule in the worst cases. The worst cases are thus defined as such that for some task sets with *n* tasks each and a utilization of *U*, the number of processors required by these task sets is no smaller than those required by any set of *n* tasks with the same utilization of *U*.

Before we can proceed further, we need to define a procedure for finding the minimum number of processors for any given set Σ of *n* tasks with a utilization of *U*. In other words, we will design the optimal algorithm for scheduling a set of periodic tasks, the one which always returns the minimum number of processors for any given set of tasks. By describing such algorithm, we actually define a canonical form for the optimal schedule, to be used in the proof that follows. Note that the Earliest Deadline First algorithm is employed by each processor as its scheduler.

An optimal algorithm is given as follows:

(1) $m = 1$; $S \leftarrow \Sigma$;

(2) Find the largest subset $S_m \subseteq S$ of tasks such that

(i) any $(m + 1)$ tasks in S_m cannot be scheduled on a single processor;

(ii) any i tasks in S_m cannot be scheduled with any $(m - i + 1)$ tasks in $S - S_m$ for $i = 1, 2, \ldots, m$;

(iii) for each task in S_m, there exist some groups of $(m - 1)$ tasks in S_m that these m tasks can be feasibly scheduled on a single processor.

Then $S \leftarrow S - S_m$.

We give the following procedure that can compute such S_m:

(a) $S_m \leftarrow \{\}$;

(b) Rename the task set S such that the $|S|$ tasks are indexed from 1 to $|S|$.

For all $j = 1, 2, \ldots, m$, $\tau_{i_j} \in S$, $i_j = 1, \ldots, |S|$, and $i_k \neq i_l$ with $k, l \in [1 \ldots m]$, repeat the following until either $S = \{\}$ or the conditions (I) and (II) are not true.

If

(I) these m tasks $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ can be scheduled on a processor; and

(II) any i tasks among the m tasks $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ cannot be scheduled with any $(m - i + 1)$ tasks in $S - S_m$ on a processor for $i = 1, 2, \ldots, m$

then $S_m \leftarrow S_m + \{\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}\}$ and $S \leftarrow S - \{\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}\}$

(3) If $S \neq \{\}$, then $m \leftarrow m + 1$ and goto (2).

From the algorithm, it is apparent that in the final schedule, if l is the maximum number of tasks assigned on a processor, then any $(l + 1)$ tasks among the n tasks cannot be scheduled on a processor.

If $U > n/2$, then in the worst cases, any two of the n tasks cannot be scheduled in a single processor. Hence, the number of processors required for the scheduling of this set is n. Since

$$\lceil U + U^2/(n - U) \rceil > n$$

for $U \geq n/2$, the theorem holds.

If $U \leq n/2$, we claim that in the worst cases, the maximum numbers of processors in the optimal schedules, that are required for the feasible scheduling of the task sets, say P, is given by $\left\lceil \frac{n}{m} \right\rceil$, where m is determined by finding the smallest m such that

$$U \geq \frac{n}{m+1}. \tag{18}$$

Let us note that such a number of *m* does exist, since $U \leq n/(m+1)$ for *m* = 1, and the function $f(m) = n/(m+1)$ is a monotonically decreasing function with regard to *m*. Furthermore, by solving inequality (13), we obtain

$$m \geq \frac{n}{U} - 1. \tag{19}$$

Hence,

$$P = \left\lceil \frac{n}{m} \right\rceil \leq \left\lceil U + \frac{U^2}{n-U} \right\rceil. \tag{20}$$

Now suppose that the claim is not true, i.e., there exists a worst case where the number of processors required is *Q* such that *Q* > *P*. Then let n_i be the number of processors on each of which *i* tasks are assigned and *k* and *l* are the minimum and the maximum number of tasks assigned to a processor, respectively. Then $Q = \sum_{i=k}^{l} n_i$ and $n = \sum_{i=k}^{l} i n_i$.

If $l < m$, i.e., each processor is assigned less than *m* tasks, then since any (*l* + 1) of the *n* tasks cannot be scheduled on a processor, the total utilization of the task set must be greater than $n/(l+1)$. Since *m* is the smallest number such that $U \geq n/(m+1)$, therefore $U < n/(l+1)$. This indicates that the task set now has a greater utilization and thus a contradiction is introduced.

If $l > m$, i.e., some processors are assigned more than *m* tasks, then since any (*l* + 1) of the *n* tasks cannot be scheduled on a processor, the total utilization of the task set must be greater than $\frac{n}{l+1}$. Since *m* is the smallest number such that $U \geq \frac{n}{m+1}$,

$$U \geq \frac{n}{m+1} > \frac{n}{l+1}.$$

This results in a contradiction.

Therefore, the theorem must be true. ■

IV. Concluding Remarks

The discovery of the general schedulability condition as given in Theorem 1 for Rate-Monotonic scheduling was the direct result of our attempt to design and analyze effective heuristic algorithms for scheduling periodic tasks on a multiprocessor system. In analyzing various heuristic algorithms for their worst-case performance, we realize that for any given set of tasks, some upper

bounds on the number of processors must be established. However, we also learn from our proof that it is very difficult to establish such bounds, because it usually involves the solutions to non-convex optimization problems.

Further questions remain as how to derive meaningful conditions for periodic tasks which share resources in a multiprocessor environment. Also of interest to our research is the derivation of similar schedulability conditions for scheduling periodic tasks whose deadlines are shorter than their periods. A number of researchers have addressed this problem to some extents, see [1] for example.

Acknowledgments: I would like to thank Dr. Sang H. Son for his support.

References

[1] N.C. AUDSLEY, A. BURNS, M.F. RICHARDSON, K.W. TINDELL, AND A.J. WELLINGS. "Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling," *Software Engineering Journal* **8(5)**: 284-292 (1993).

[2] A. BURCHARD, J. LIEBEHERR, Y. OH, AND S.H. SON. "Assigning Real-Time Tasks to Homogeneous Multiprocessor Systems," *IEEE Transactions on Computer* (to appear).

[3] S. DAVARI AND S.K. DHALL. "An On Line Algorithm for Real-Time Tasks Allocation," *IEEE Real-Time Systems Symposium*, 194-200 (1986).

[4] S.K. DHALL AND C.L. LIU. "On a Real-Time Scheduling Problem," *Operations Research* **26**: 127-140 (1978).

[5] M.R. GAREY AND D.S. JOHNSON. *Computers and Intractability: A Guide to the Theory of NP-completeness*, W.H. Freeman and Company, NY, 1978.

[6] M. JOSEPH AND P. PANDYA. "Finding Response Times in a Real-Time System," *The Computer Journal* **29(5)**: 390-395 (1986).

[7] J. LEHOCZKY, L. SHA, AND Y. DING. "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," *IEEE Real-Time Symposium*, 166-171 (1989).

[8] J.Y.T. LEUNG AND J. WHITEHEAD. "On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks," *Performance Evaluation* **2**: 237-250 (1982).

[9] C.L. LIU AND J. LAYLAND. "Scheduling Algorithms for Multiprogramming in a Hard

Real-Time Environment," *J. Assoc. Comput. Machinery* **10(1)**: 174-189 (1973).

[10] Y. OH AND S.H. SON. "Allocating Fixed-priority Periodic Tasks on Multiprocessor Systems," *Journal of Real-Time Systems* (to appear).

[11] Y. OH. The Design and Analysis of Scheduling Algorithms for Real-Time and Fault-Tolerant Computer Systems. Ph.D. Dissertation, Dept. of Computer Science, University of Virginia, May 1994.

[12] P. SERLIN, "Scheduling of Time Critical Processes," *Proceedings of the Spring Joint Computers Conference* **40**: 925-932 (1972).

[13] L. SHA, R. RAJKUMAR, AND J.P. LEHOCZKY. "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Transactions on Computers* **39(9)**: 1175-1185 (1990).