JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 19, 11–26 (1993)

A New Performance Measure for Scheduling Independent Real-Time Tasks*

DAR-TZEN PENG AND KANG G. SHIN

Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, Michigan 48109-2122

A new performance measure for scheduling tasks in hard real-time systems is proposed and analyzed. The proposed measure is the maximum normalized task flowtime over all tasks, called the *system hazard*, where the normalized flowtime of a real-time task is the flowtime of the task divided by the period between its arrival time and deadline. The system hazard is a better measure than simply meeting task deadlines because it also indicates how early tasks as a whole can be completed before their deadlines. For a single processor with only independent periodic tasks, optimal scheduling algorithms with respect to the system hazard are derived for both static and dynamic cases. Two best bounds of processor utilization for these optimal algorithms are also computed. © 1993 Academic Press, Inc.

1. INTRODUCTION

The workload in a real-time system is composed of periodic and aperiodic tasks. Periodic tasks are the "base load" and invoked at fixed time intervals while aperiodic tasks are the "transient load," arriving randomly in response to environmental stimuli. In *hard* real-time systems such as missile navigation or robot control, execution of both periodic and aperiodic tasks must be not only logically correct but also completed in time. Specifically, there exists an associated deadline for each task before which the task must be completed.

Using different assumptions on the set of tasks to be scheduled and the set of processors to execute them, various scheduling algorithms have been proposed. (See [3] for an extensive survey.) It is important to note that all of these scheduling algorithms are concerned with one and only one objective: meeting task deadlines. A scheduling algorithm is said to be *optimal* if it generates a schedule in which every task can be completed before its deadline provided such a schedule exists. However, scheduling tasks with this objective alone has the following drawbacks:

- Prior knowledge of the task system based on which conventional scheduling algorithms were derived is not always available or accurate. For instance, the exact execution time of a task is difficult to obtain because of the uncertain behavior of loops and conditional branches in the task.
- It is impossible to evaluate the goodness of a schedule in terms of how early a task can be completed before its deadline. This information is important especially in scheduling periodic tasks in the presence of randomly arriving aperiodic tasks which must also be completed before their deadlines.

To remedy the above drawbacks, we propose a new performance measure, called the *system hazard*, as the objective function for scheduling real-time tasks. Specifically, the system hazard, denoted by Θ, is the maximum normalized task flowtime, where the flowtime (response time or turn-around time) of a task T_j is defined as the time period between the arrival (a_j) and the completion (c_j) of T_j. That is, $\Theta \triangleq \max_j (c_j - a_j)/(d_j - a_j)$, where d_j is the deadline of T_j. A schedule is said to be *optimal with respect to* (w.r.t.) Θ if it achieves the smallest possible value of Θ, denoted by Θ^*. Obviously, Θ depends only on the execution of tasks and can be used not only on a single processor but also on a multiprocessor/distributed system. Several insights can be drawn from Θ and Θ^* as follows. First, under the assumption that all task execution times are fixed (as with most existing scheduling algorithms), $\Theta^* \leq 1$ if and only if there exists at least one schedule under which all tasks can be completed before their deadlines. Thus, if $\Theta^* \leq 1$, an optimal schedule w.r.t. Θ is also optimal in terms of the ability to meet deadlines. Second, if task execution times are *random* rather than fixed, then Θ^* is a good measure for the system's inability of meeting task deadlines. Third, Θ^* can also be used to evaluate the goodness of task assignments in distributed real-time systems [10]. An assignment with

* The work reported in this paper was supported in part by the Office of Naval Research under Contract N00014-92-J-1080 and the National Science Foundation under Grant DMC-8721492. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the view of the funding agencies. Dar-Tzen Peng is now with Microelectronics and Technology Center, AlliedSignal Aerospace Company, 9140 Old Annapolis Road, Columbia, MD 21045-1998.

0743-7315/93 $5.00

lower Θ^* is superior to the one with higher Θ^* because the former results in a lower probability of each assigned task missing its deadline.

In this paper, we shall study the optimal preemptive resume scheduling algorithms and their associated processor utilizations for both periodic and aperiodic tasks by minimizing Θ. As was done in [7], this derivation is based on the assumption that (A1) tasks are to be scheduled on a single processor, and (A2) periodic and aperiodic tasks are independent of each other. A2 means that no precedence constraints exist between any two tasks except for those among periodic tasks implied by their invocation times.

For periodic tasks, both static and dynamic scheduling algorithms are considered. According to the definitions used in [7], a scheduling algorithm is said to be *static* (*dynamic*) if all invocations of a task are (may be) assigned the same priority (different priorities). Thus in a static scheduling algorithm, if task T_i is given priority over task T_j, then an invocation of T_i has priority over all invocations of T_j. In a dynamic scheduling algorithm, on the other hand, two invocations of the same task may be assigned different priorities. We shall prove that the *rate-monotonic scheduling* (RMS) algorithm—which was proven to be optimal in meeting deadlines [7]—is also an optimal static algorithm w.r.t. Θ for periodic tasks. For the dynamic case however, the *earliest due date* (EDD) scheduling algorithm—which is optimal w.r.t. many other performance measures—is shown to be not optimal w.r.t. Θ. That is, an optimal schedule derived by minimizing Θ not only meets all the deadlines that can be met by the EDD algorithm, but also offers additional benefits. For aperiodic tasks, we shall show that optimal on-line scheduling algorithms are non-existent except for some special cases.

The rest of the paper is organized as follows. In Section 2, we consider the optimal static scheduling algorithms for periodic tasks as well as achievable processor utilization bounds. The dynamic version of the subject in Section 2 is dealt with in Section 3. On-line scheduling algorithms for aperiodic tasks are treated in Section 4, where, rather than deriving processor utilization bounds, simple mechanisms are proposed to check whether or not a randomly arriving aperiodic task can be completed with not greater than the prespecified system hazard. Finally, the paper concludes with Section 5.

2. OPTIMAL STATIC SCHEDULING OF PERIODIC TASKS

Let $T = \{T_i : 1 \le i \le m\}$ be the set of m periodic tasks to be scheduled on a processor, where each task T_i repeats itself with period p_i during the entire mission. Let $I = [0, L)$ be a *planning cycle*, where L is the least common multiple (LCM) of all p_i's. For simplicity, we assume throughout the paper that all tasks are invoked simultaneously at the beginning of a planning cycle. The vth invocation of T_i, denoted by T_{iv}, is triggered at time $(v-1)p_i$ and has to be completed before its next invocation time vp_i. We want to derive a scheduling algorithm that minimizes

$$\Theta = \max_{\substack{1 \le v \le L/p_i \\ T_i \in T}} (c_{iv} - a_{iv})/p_i,$$

where c_{iv} and a_{iv} are the completion time and invocation time of T_{iv}, respectively.

2.1. Optimal Static Scheduling Algorithm

Liu and Layland [7] showed that the rate-monotonic scheduling (RMS) algorithm—which simply assigns task priorities inversely proportional to task invocation periods—is optimal in the sense that it generates a *feasible* schedule provided such a schedule exists.[1] One might conjecture that the RMS algorithm is also optimal w.r.t. Θ provided at least a feasible schedule exists. In what follows, we shall prove this conjecture using two useful results. The first result follows directly from [7] and is stated below without proof.

LEMMA 1. *In a feasible schedule, the maximum flow-time of a task occurs when the task is invoked simultaneously with all other higher-priority tasks.*

Lemma 1 is obvious because, in a feasible schedule, the completion of a task will be delayed most if all other higher-priority tasks are invoked at the same time when the task is invoked. Further, note that this time of simultaneous task invocations is assumed to be the beginning of each planning cycle. Lemma 1 is very useful for the analysis of the performance of a static scheduling algorithm. Specifically, to check if all task (invocation) deadlines are met under a particular algorithm, all we need to do is to check whether or not the task deadlines are met only for the first invocations of all tasks.

Lemma 2 [1, 6] stated below is the other result to be used for proving the optimality of the RMS algorithm. Consider a set of tasks to be scheduled on a single processor, where each task with a fixed execution time must be completed before its deadline. The earliest due date (EDD) scheduling algorithm is one that always selects the unscheduled task with the nearest deadline.

LEMMA 2. *If there exists a schedule where all[2] tasks meet their deadlines, then so can the EDD scheduling algorithm.*

[1] A schedule is said to be feasible if all invocations of every task in the schedule can be completed in time.

[2] In fact, the same conclusion holds as long as the number of tasks missing deadlines is at most 1.

Lemma 2 can be proved easily by using *pairwise exchange* of tasks [1]. Lemmas 1 and 2 lead to the following theorem, which assures the optimality of the RMS algorithm w.r.t. Θ.

THEOREM 1. *If a feasible schedule exists for $m > 0$ tasks, then the RMS algorithm is optimal w.r.t. Θ.*

Proof. From Lemma 1, we need to consider the first invocation T_{i1} of task T_i, $i = 1, 2, \cdots, m$, only. For notational simplicity in the proof, T_{i1} is simply denoted as the task T_i with arrival time $a_i = 0$ and deadline $d_i = p_i$. Let Θ^* be the minimum of Θ obtainable with any static scheduling algorithm. To prove the theorem, we need to show that the RMS algorithm achieves Θ^*.

From our definition of Θ above, a schedule with system hazard Θ^* is one where each task T_i meets its "revised" deadline $\Theta^* p_i$. However, from Lemma 2, the EDD scheduling algorithm also generates such a schedule as long as the task selection criterion is based on these revised deadlines. In other words, Θ^* is also achieved by this revised EDD algorithm, which is exactly the same as the RMS algorithm because each revised deadline is a constant (i.e., Θ^*) proportion of its task invocation period. **Q.E.D.**

Note that the existence of a feasible schedule is necessary for Lemma 1 and Theorem 1. For example, consider $T = \{T_1, T_2\}$ with $p_1 = 2$, $p_2 = 3$, $e_1 = 1$, and $e_2 = 2$, where Lemma 1 is no longer true and Θ^* is not even existent.

2.2. Achievable Processor Utilization Bounds

As was done in [7], the *utilization* U of a single processor system with m periodic tasks T_i's is defined as $U = \sum_{i=1}^{m} e_i/p_i$. Obviously, a higher U means a better processor utilization and $U \le 1$ must hold provided every invocation of a task can be completed before its deadline. It was shown in [7] that for $\Theta = 1$ there exist two bounds of U: $U_l = m(2^{1/m} - 1)$ and $U_h = 1$. Specifically, given any m tasks with $U \le U_l$, there always exists a feasible schedule for these tasks. On the other hand, if $U > U_h$, then no feasible schedule exists for these tasks. If $U_l < U \le U_h$, then a feasible schedule may, or may not, exist depending on p_i's and e_i's. In what follows, we shall generalize these results under the condition of $\Theta \le 1$. That is, for any given value of $0 < \Theta \le 1$, we want to derive the corresponding values of U_l and U_h. (The results of [7] are thus a special case of ours since theirs were derived for the case of $\Theta = 1$.) For any m tasks with $U \le U_l$, there always exists a feasible schedule with the maximum normalized flowtime (of all invocations of all tasks) not exceeding Θ. On the other hand, if $U > U_h$, then no such feasible schedule exists for these tasks. If $U_l < U \le U_h$, then such a feasible schedule may, or may not, exist depending on the values of p_i's and e_i's.

Notice that for any $\Theta \in (0, 1]$, U_l and U_h should be derived under the RMS algorithm—which is optimal w.r.t. Θ by Theorem 1—because the RMS algorithm yields the maximal values of U_l and U_h among all static scheduling algorithms. Also, the values of U_l and U_h corresponding to any $0 < \Theta < 1$ will be shown to be smaller than those in [7], which were derived for the case of $\Theta = 1$. Throughout the rest of the paper, a schedule is said to be *feasible* for a given $\Theta \in (0, 1]$ if it results in a system hazard not exceeding Θ.

2.2.1. Deriving U_l. U_l is first derived for two tasks T_1 and T_2 with $p_1 \le p_2$. This result will then be extended for an arbitrary number of tasks. To derive U_l, two cases need to be considered:[3]

$$\text{(C1)} \qquad \left\lfloor \frac{\Theta p_2}{p_1} \right\rfloor p_1 \le \Theta p_2 < \left\lfloor \frac{\Theta p_2}{p_1} \right\rfloor p_1 + \Theta p_1,$$

where

$$\left\lfloor \frac{\Theta p_2}{p_1} \right\rfloor \ge 1 \text{ (Fig. 1a)},$$

and

$$\text{(C2)} \qquad \left\lfloor \frac{\Theta p_2}{p_1} \right\rfloor p_1 + \Theta p_1 \le \Theta p_2 < \left\lceil \frac{\Theta p_2}{p_1} \right\rceil p_1,$$

where

$$\left\lfloor \frac{\Theta p_2}{p_1} \right\rfloor \ge 0 \text{ (Fig. 1b)}.$$

As was done in [7], U_l and U_h are derived with the processor fully utilized. A processor is said to be *fully utilized* by a set T of tasks for a given Θ under a feasible schedule if increasing the execution time of any one task in T makes the schedule infeasible.

C1. In order for T_1's normalized flowtime not to exceed the given value of Θ, $e_1 \le \Theta p_1$, but e_1 may or may not exceed

$$\xi \triangleq \Theta p_2 - \left\lfloor \frac{\Theta p_2}{p_1} \right\rfloor p_1.$$

When $e_1 \le \xi$, the largest e_2 that allows for at least one feasible schedule is

$$\Theta p_2 - \left\lceil \frac{\Theta p_2}{p_1} \right\rceil e_1,$$

[3] $\lfloor x \rfloor$ represents the largest integer $\le x$, whereas $\lceil x \rceil$ the smallest integer $\ge x$. Thus, $\lceil x \rceil = \lfloor x \rfloor$ if x is an integer. Otherwise, $\lceil x \rceil = \lfloor x \rfloor + 1$.

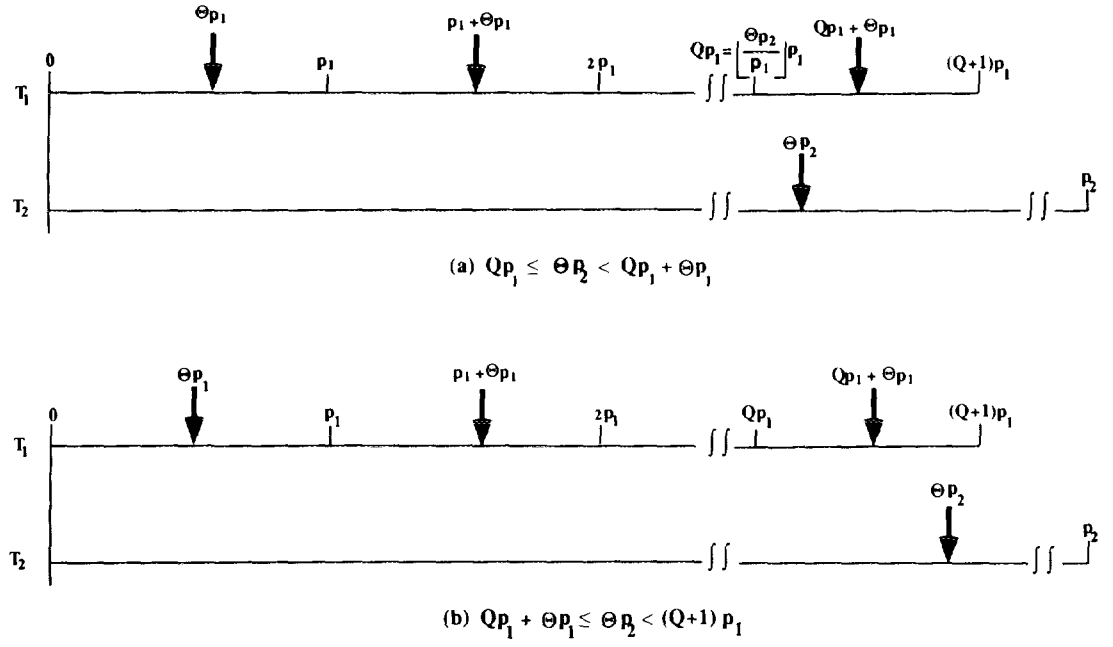


FIG. 1. Different relations between Θp_1 and Θp_2.

and the corresponding utilization is

$$U = \frac{e_1}{p_1} + \frac{e_2}{p_2} = \frac{e_1}{p_1} + \Theta - \frac{e_1}{p_2}\left\lceil \frac{\Theta p_2}{p_1} \right\rceil = \Theta + e_1\left\{\frac{1}{p_1} - \frac{1}{p_2}\left\lceil \frac{\Theta p_2}{p_1} \right\rceil\right\}. \quad (1)$$

On the other hand, when $e_1 > \xi$, e_2 must be no larger than

$$(p_1 - e_1)\left\lfloor \frac{\Theta p_2}{p_1} \right\rfloor,$$

and the corresponding utilization becomes

$$U = \frac{e_1}{p_1} + \frac{p_1 - e_1}{p_2}\left\lfloor \frac{\Theta p_2}{p_1} \right\rfloor = \frac{p_1}{p_2}\left\lfloor \frac{\Theta p_2}{p_1} \right\rfloor + e_1\left\{\frac{1}{p_1} - \frac{1}{p_2}\left\lfloor \frac{\Theta p_2}{p_1} \right\rfloor\right\}. \quad (2)$$

The U in Eq. (2) is a nondecreasing function of e_1 because

$$\frac{1}{p_1} \geq \frac{1}{p_2}\left\lfloor \frac{\Theta p_2}{p_1} \right\rfloor.$$

Thus, the behavior of U in Eq. (1) as a function of e_1 determines U_1. Specifically, if the U in Eq. (1) is also a nondecreasing function of e_1, then U_1 occurs at $e_1 = 0$. If U is a nonincreasing function of e_1, then U_1 occurs at

$$e_1 = \xi = \Theta p_2 - \left\lfloor \frac{\Theta p_2}{p_1} \right\rfloor p_1.$$

C2. Similarly to the case of $e_1 \leq \xi$ in C1, the largest allowable e_2 is

$$\Theta p_2 - \left\lceil \frac{\Theta p_2}{p_1} \right\rceil e_1,$$

and the corresponding utilization is

$$U = \Theta + e_1\left\{\frac{1}{p_1} - \frac{1}{p_2}\left\lceil \frac{\Theta p_2}{p_1} \right\rceil\right\},$$

which is a nondecreasing function of e_1 for the following reason. Since, by assumption,

$$\left\lfloor \frac{\Theta p_2}{p_1} \right\rfloor p_1 + \Theta p_1 \leq \Theta p_2, \quad \left\lfloor \frac{\Theta p_2}{p_1} \right\rfloor \leq \Theta\left(\frac{p_2}{p_1} - 1\right) \leq \frac{p_2}{p_1} - 1,$$

implying that

$$\frac{1}{p_1} - \frac{1}{p_2}\left\lceil \frac{\Theta p_2}{p_1} \right\rceil \geq 0.$$

Thus, $U_1 = \Theta$ occurs at $e_1 = 0$.
More formally, we have the following theorem.

THEOREM 2. *For a given* $\Theta \le 1$ *and two tasks* T_1 *and* T_2,

$$U_l = \begin{cases} \Theta & \text{if } \Theta \le 0.5 \\ 2(\sqrt{2\Theta} - 1) + 1 - \Theta & \text{if } 0.5 < \Theta \le 1. \end{cases}$$

Proof. We prove this theorem by deriving U_l under both C1 and C2, and choosing the smaller of the two. To find U_l under C1, it is necessary to examine the behavior of U in Eq. (1). First, consider the case of $\Theta \le 0.5$. Since

$$\frac{\Theta p_2}{p_1} \ge \left\lfloor \frac{\Theta p_2}{p_1} \right\rfloor, \frac{p_2}{p_1} \ge \frac{1}{\Theta} \left\lfloor \frac{\Theta p_2}{p_1} \right\rfloor$$
$$\ge 2 \left\lfloor \frac{\Theta p_2}{p_1} \right\rfloor \ge 1 + \left\lfloor \frac{\Theta p_2}{p_1} \right\rfloor \ge \left\lceil \frac{\Theta p_2}{p_1} \right\rceil.$$

Thus,

$$\frac{1}{p_1} \ge \frac{1}{p_2} \left\lceil \frac{\Theta p_2}{p_1} \right\rceil,$$

indicating that the U of Eq. (1) is a nondecreasing function of $e_1 \in [0, \Theta p_1]$. Thus under C1, $U_l = \Theta$ occurs at $e_1 = 0$. The theorem follows for $\Theta \le 0.5$ because $U_l = \Theta$ also holds under C2.

Next, consider the case of $\Theta > 0.5$. It can be easily shown that there always exists at least a (p_1, p_2) pair such that

$$\frac{1}{p_1} < (\ge) \frac{1}{p_2} \left\lceil \frac{\Theta p_2}{p_1} \right\rceil.$$

In other words, the U of Eq. (1) could be nonincreasing (nondecreasing) in e_1, depending on the values of p_1 and p_2. To derive U_l under C1 for $\Theta > 0.5$, one has to find U_l's for these two possibilities and then choose the smaller of them as U_l under C1 as follows. Since $U_l = \Theta$ if U in Eq. (1) is nondecreasing in e_1, we now consider the possibility of Eq. (1) being nonincreasing in e_1. If Eq. (1) is nonincreasing, then $U_l = U|_{e_1 = \xi}$ or, from Eq. (1) or (2),

$$U_l = \frac{p_1}{p_2} \left\lfloor \frac{\Theta p_2}{p_1} \right\rfloor + \left\{ \Theta p_2 - p_1 \left\lfloor \frac{\Theta p_2}{p_1} \right\rfloor \right\} \left\{ \frac{1}{p_1} - \frac{1}{p_2} \left\lfloor \frac{\Theta p_2}{p_1} \right\rfloor \right\}$$
$$= \frac{p_1}{p_2} \left(\left\lfloor \frac{\Theta p_2}{p_1} \right\rfloor \right)^2 + \left\{ \frac{p_1}{p_2} - \Theta - 1 \right\} \left\lfloor \frac{\Theta p_2}{p_1} \right\rfloor + \frac{\Theta p_2}{p_1}. \tag{3}$$

For a given Θ, Eq. (3) is an expression for U_l under C1 for a given (p_1, p_2) pair. To find U_l under C1, one must find the minimum of the above U_l among all possible (p_1, p_2) pairs. To find a special (p_1, p_2) pair which minimizes U_l, let $\Theta p_2/p_1 = Q + R$, where Q is a positive integer representing the quotient and $0 \le R < 1$ representing the remainder of $\Theta p_2/p_1$. Then, U_l in Eq. (3) becomes

$$U_l = \frac{\Theta}{Q + R} Q^2 + \left(\frac{\Theta}{Q + R} - \Theta - 1 \right) Q + (Q + R)$$
$$= \frac{\Theta Q^2 + \Theta Q + (R - \Theta Q)(Q + R)}{Q + R}$$
$$= \frac{(R + \Theta - R\Theta) Q + R^2}{Q + R} \tag{4}$$
$$= (R + \Theta - R\Theta) - \frac{R(\Theta - R\Theta)}{Q + R}$$

Given Θ and R, U_l in Eq. (4) is nondecreasing in Q because $\Theta - R\Theta \ge 0$. Therefore, the minimum of U_l must occur at $Q = 1$. Substituting $Q = 1$ into Eq. (4), U_l becomes $(R^2 + (1 - \Theta)R + \Theta)/(R + 1)$. To further minimize U_l w.r.t. R, we take the derivative of U_l w.r.t. R and find $R = R^*$ such that $dU_l/dR|_{R=R^*} = 0$. It turns out that $R^* = \sqrt{2\Theta} - 1$ and the corresponding minimum of U_l is $2(\sqrt{2\Theta} - 1) + 1 - \Theta$. To prove that $2(\sqrt{2\Theta} - 1) + 1 - \Theta$ is acceptable, we have to show that $(Q, R) = (1, R^*)$ is in the domain where Eq. (1) is indeed nonincreasing. In other words, for any given $\Theta \ge 0.5$, we want to show that

$$\frac{1}{p_1} - \frac{1}{p_2} \left\lceil \frac{\Theta p_2}{p_1} \right\rceil \le 0$$

at $(Q, R) = (1, R^*)$. Note that

$$\frac{1}{p_1} - \frac{1}{p_2} \left\lceil \frac{\Theta p_2}{p_1} \right\rceil = \frac{1}{p_2} \left\{ \frac{p_2}{p_1} - (Q + 1) \right\}$$
$$= \frac{1}{p_2} \left\{ \frac{1}{\Theta} (Q + R) - (Q + 1) \right\}.$$

Substituting $(1, \sqrt{2\Theta} - 1)$ for (Q, R), the above becomes

$$\frac{1}{p_2} \left\{ \frac{1}{\Theta} (1 + \sqrt{2\Theta} - 1) - (1 + 1) \right\} = \frac{1}{p_2} \left(\frac{1}{\Theta} \sqrt{2\Theta} - 2 \right)$$
$$= \frac{1}{p_2} \left(\sqrt{\frac{2}{\Theta}} - 2 \right) \le 0$$

provided $\Theta > 0.5$. Thus, the U_l for $\Theta \ge 0.5$ under C1 is $\min \{2(\sqrt{2\Theta} - 1) + 1 - \Theta, \Theta\} = 2(\sqrt{2\Theta} - 1) + 1 - \Theta$. Choosing the smaller of this U_l and that under C2, the U_l in case of $\Theta > 0.5$ becomes $\min \{2(\sqrt{2\Theta} - 1) + 1 - \Theta, \Theta\} = 2(\sqrt{2\Theta} - 1) + 1 - \Theta$. **Q.E.D.**

Notice that the results presented in [7] can be obtained from Theorem 2 by letting $\Theta = 1$. The results of Theorem 2 are extended below for $m \ge 2$ tasks. Theorem 3 establishes the results for $\Theta \le 0.5$, whereas Lemmas 3–4, Theorems 4–5 establishes those for $0.5 < \Theta \le 1$.

THEOREM 3. *For $m \geq 2$ periodic tasks $\{T_i\}$ and a given $\Theta \leq 0.5$, $U_l = \Theta$.*

Proof. This theorem can also be proved by pairwise exchanges. Let $p_1 \leq p_2 \leq \cdots \leq p_m$. To derive U_l, we again assume that the RMS algorithm is used and the processor is fully utilized.

Construct new task execution times, $e_1' = 0$, $e_i' = e_i$, $i = 2, 3, \cdots, m-1$, and a particular $e_m' \geq e_m$, such that the processor is fully utilized. We want to show that $U' = \sum_{i=1}^{m} e_i'/p_i \leq U = \sum_{i=1}^{m} e_i/p_i$. Specifically, we need to prove that $e_m'/p_m \leq e_1/p_1 + e_m/p_m$. Since

$$e_m' \leq e_m + \left\lceil \frac{\Theta p_m}{p_1} \right\rceil e_1$$

must hold,

$$\frac{e_m'}{p_m} \leq \frac{e_m}{p_m} + \frac{e_1}{p_m}\left\lceil \frac{\Theta p_m}{p_1} \right\rceil \leq \frac{e_m}{p_m} + \frac{e_1}{p_1}$$

because, as shown in Theorem 2,

$$\frac{1}{p_1} \geq \frac{1}{p_m}\left\lceil \frac{\Theta p_m}{p_1} \right\rceil$$

provided $\Theta \leq 0.5$. In other words, U_l must occur at $e_1 = 0$. The above arguments can be applied repeatedly between e_2 and e_m, between e_3 and e_m, and so on. Finally, we conclude that U_l must occur when $e_1 = e_2 = \cdots = e_{m-1} = 0$, $e_m = \Theta p_m$, and thus, $U_l = \Theta p_m/p_m = \Theta$. **Q.E.D.**

Before extending Theorem 3 to the case of $\Theta > 0.5$, it is necessary to prove the following important lemma, which identifies a case where the U_l can be found in Theorem 4.

LEMMA 3. *For a set T of $m \geq 2$ tasks and a given $\Theta > 0.5$, if $p_{m-1} \leq \Theta p_m$ and $p_m \leq 2p_1$, then the minimum processor utilization occurs when (see Fig. 2),*

$$e_i = e_i^* = \begin{cases} p_{i+1} - p_i & i = 1, \cdots, m-2 \\ \Theta p_m - p_{m-1} & i = m-1 \\ \Theta p_m - 2\sum_{i=1}^{m-1} e_i & i = m. \end{cases} \tag{5}$$

Proof. Assume $p_1 < p_2 < \cdots < p_m$. (As will be clear in the following steps of proof, the lemma is also true when $p_i = p_{i+1}$, $1 \leq i < m$.) Let $\{e_i\}$ be an arbitrary set of execution times for which the processor is fully utilized (with utilization U). We want to show that U must be greater than that corresponding to the set of e_i^* values in Eq. (5). This is done by considering each task sequentially as follows.

First, consider T_1. If $e_1 < p_2 - p_1$, then construct new task execution times $\{e_i'\}$ such that $e_1' = p_2 - p_1$, $e_i' = e_i$,

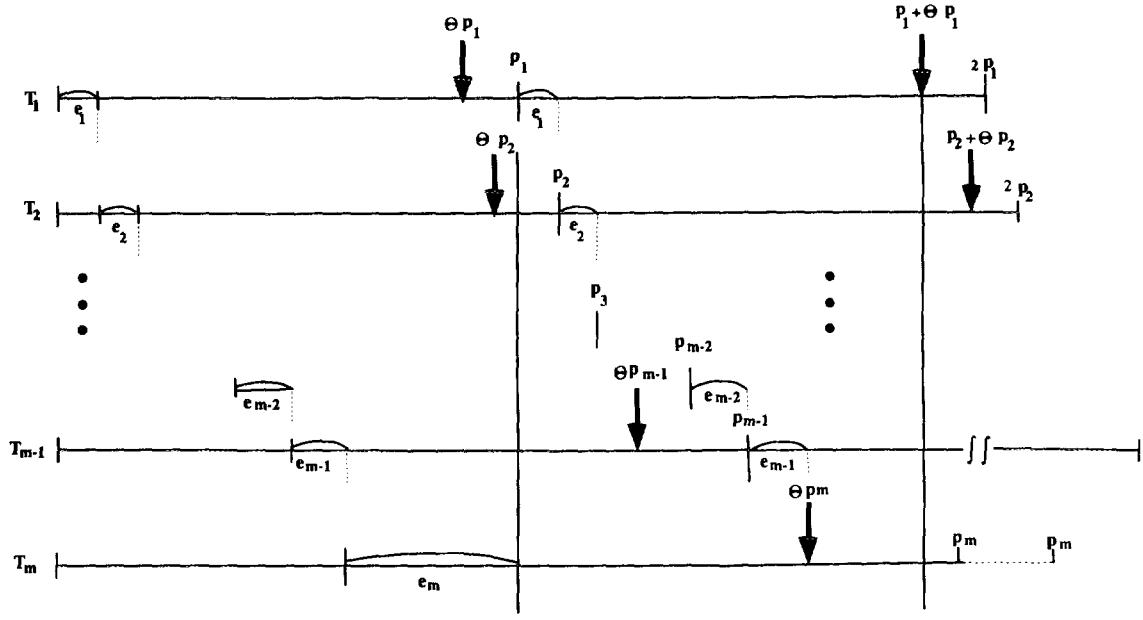


FIG. 2. e_i's which minimize U.

$i = 2, 3, \cdots, m-1$. In order to let $\{e_i'\}$ fully utilize the processor, $e_m' = e_m - 2[(p_2 - p_1) - e_1] > 0$ (Fig. 2). We need to show that the processor utilization U' corresponding to $\{e_i'\}$ is smaller than U, i.e., $e_1'/p_1 + e_m'/p_m < e_1/p_1 + e_m/p_m$. This inequality holds because

$$\frac{(p_2 - p_1) - e_1}{p_1} < \frac{2[(p_2 - p_1) - e_1]}{p_m}$$

since, by assumption, $p_1 > p_m/2$. On the other hand, if $e_1 > p_2 - p_1$, then the new execution times $\{e_i'\}$ must be constructed such that $e_1' = p_2 - p_1$, $e_2' = e_2 + [e_1 - (p_2 - p_1)]$, and $e_i' = e_i$, $i = 3, 4, \cdots, m$ to again fully utilize the processor (Fig. 2). Likewise, to show that the new utilization U' is less than U, we have to show that $[e_1 - (p_2 - p_1)]/p_1 > [e_1 - (p_2 - p_1)]/p_2$, which is true since $p_1 < p_2$. For the case where $e_1 = p_2 - p_1$ or after constructing $\{e_i'\}$ as above, we move on to consider T_2 based on the newly constructed $\{e_i'\}$. While considering T_2, we likewise increase (decrease) e_2' to $e_2'' = p_3 - p_2$ and decrease e_m' (increase e_3') twice the (the same) amount that e_2' has been increased (decreased) if $e_2' <(>) p_3 - p_2$. For the same reason as the case of T_1, $U'' < U'$ must hold, where U'' represents this newly generated utilization with T_2 considered. This process can be repeatedly applied to T_3, T_4, $\cdots$ and up to T_{m-2} such that the utilization is reduced each time. Finally, we consider T_{m-1} and conclude that given $e_i = p_{i+1} - p_i$, $i = 1, 2, \cdots, m-2$, $e_{m-1} = \Theta p_m - p_{m-1}$ must hold to minimize U. Since the utilization for any set of task execution times can be reduced with the above procedure until Eq. (5) is satisfied, the lemma follows. **Q.E.D.**

Under the conditions of lemma 3, U_1 can be derived as in the following theorem.

THEOREM 4. *For $m \geq 2$ tasks and a given $\Theta > 0.5$, if $p_{m-1} \leq \Theta p_m$ and $p_m \leq 2p_1$, then $U_1 = m[(2\Theta)^{1/m} - 1] + 1 - \Theta$.*

Proof. From Lemma 3, if $p_{m-1} \leq \Theta p_m$, $p_m \leq 2p_1$, and e_i's satisfy Eq. (5), then the resulting utilization will be minimized. We now derive U_1 by searching for the minimal utilization among all possible p_i's in the domain where $p_{m-1} \leq \Theta p_m$ and $p_m \leq 2p_1$ hold. That is, we want to find the minimum of $U = \sum_{i=1}^{m} e_i/p_i$ while varying p_i's and yet satisfying Eq. (5).

To derive U_1, we use the idea in [7] and define variables $g_i = (\Theta p_m - p_i)/p_i$, $i = 1, 2, \cdots, m-1$, and thus, $p_i = \Theta p_m - g_i p_i = \Theta p_m/(1 + g_i)$. Expressing e_i's in terms of g_i's and p_i's as $e_i = p_{i+1} - p_i = g_i p_i - g_{i+1} p_{i+1}$, $i = 1, 2, \cdots, m-2$, $e_{m-1} = \Theta p_m - p_{m-1} = g_{m-1} p_{m-1}$, and $e_m = \Theta p_m - 2\sum_{i=1}^{m-1} e_i = \Theta p_m - 2(\Theta p_m - p_1) = -\Theta p_m + 2(\Theta p_m - g_1 p_1) = \Theta p_m - 2g_1 p_1$, we get

$$\begin{aligned} U = \sum_{i=1}^{m} \frac{e_i}{p_i} &= \sum_{i=1}^{m-2} \left(g_i - g_{i+1} \frac{p_{i+1}}{p_i} \right) + g_{m-1} + \Theta - 2g_1 \frac{p_1}{p_m} \\ &= \sum_{i=1}^{m-2} \left(g_i - g_{i+1} \frac{1 + g_i}{1 + g_{i+1}} \right) + g_{m-1} \\ &\quad + \Theta - 2\Theta \frac{g_1}{1 + g_1}. \end{aligned} \tag{6}$$

Note that if $g_1 = g_2 = \cdots = g_{m-1} = 0$ (i.e., $\Theta p_m = p_i$, $i = 1, 2, \cdots, m-1$), then $U = \Theta$. To derive U_1, Eq. (6) is minimized w.r.t. all g_i's. this is done by setting $\partial U/\partial g_i = 0$, $\forall\, i$ and solving the resulting simultaneous difference equations:

$$\begin{aligned} &1 - \frac{2\Theta}{(1 + g_1)^2} = \frac{g_2}{1 + g_2} \\ &1 - \frac{1 + g_{i-1}}{(1 + g_i)^2} = \frac{g_{i+1}}{1 + g_{i+1}}, \quad i = 2, 3, \cdots, m-2, \\ &1 - \frac{1 + g_{m-2}}{(1 + g_{m-1})^2} = 0. \end{aligned} \tag{7}$$

It can be shown that

$$g_i = (2\Theta)^{(m-i)/m} - 1, \quad i = 1, 2, \cdots, m-1, \tag{8}$$

solve Eq. (7) and minimize U of Eq. (6). Replacing g_i's of Eq. (6) with those of Eq. (8), one can get

$$U_1 = m\,[(2\Theta)^{1/m} - 1] + 1 - \Theta. \tag{9}$$

Similarly to the proof of Theorem 2, we still need to check if p_i's (or g_i's) in Eq. (8) are in the domain where $p_m < 2p_i$ holds. This can be easily done as follows. Since $g_i = (\Theta p_m - p_i)/p_i$ by definition, $\Theta p_m/p_i = (2\Theta)^{(m-i)/m}$ from Eq. (8). It follows that $p_m/p_i = (1/\Theta)\,(2\Theta)\,(2\Theta)^{-i/m} = 2(2\Theta)^{-i/m} \leq 2$, since, by assumption, $\Theta > 0.5$ **Q.E.D.**

Note that for a given $\Theta > 0.5$, the above U_1 is a decreasing function of m, and for $m = 2$, Eq. (9) gives the same U_1 as in Theorem 2. Also, Eq. (9) becomes the same result in [7] when $\Theta = 1$. It is interesting to see that $U_1 \rightarrow \log(2\Theta) + 1 - \Theta$ as $m \rightarrow \infty$.

Theorem 4 was proved under the restriction that $p_{m-1} \leq \Theta p_m$ and $p_m \leq 2p_1$. In Lemma 4 and Theorem 5 below, we relax this restriction step-by-step and then present the general results on U_1.

LEMMA 4. *For a set T of $m \geq 2$ tasks and a given $\Theta > 0.5$, if $p_{m-1} \leq \Theta p_m \leq p_1 + \Theta p_1$, then*[4] $U_1 = m[2\Theta)^{1/m} - 1] + 1 - \Theta$.

Proof. Suppose $p_m > 2p_i$, $i = 1, 2, \cdots, n$, and $p_m \leq 2p_i$, $i = n+1, n+2, \cdots, m$. Construct $\{e_i'\}$ such that

$$e_i' = \begin{cases} 0 & i = 1, 2, \cdots, n \\ p_{i+1} - p_i & i = n+1, n+2, \cdots, m-2 \\ \Theta p_m - p_{m-1} & i = m-1 \\ \Theta p_m - 2\sum_{i=1}^{m-1} e_i & i = m. \end{cases}$$

Then, from previous discussions and Lemma 3, the new utilization associated with $\{e_i'\}$ will not be greater than the original utilization. Further, by Theorem 4, the minimum utilization resulting from the $m - n$ remaining tasks becomes

$$(m-n)[(2\Theta)^{1/(m-n)} - 1] + 1 - \Theta \geq m[(2\Theta)^{1/m} - 1] + 1 - \Theta$$

because the U_1 of Eq. (9) is nonincreasing in m.

Q.E.D.

THEOREM 5. *For a set T of $m \geq 2$ tasks and a given $\Theta > 0.5$, $U_1 = m[(2\Theta)^{1/m} - 1] + 1 - \Theta$.*

Proof. The theorem is proved by showing that the U_1 obtained in Lemma 4 under the restriction $p_{m-1} \leq \Theta p_m \leq p_1 + \Theta p_1$ is the same as that obtained for the general case. Note that $p_{m-1} \leq \Theta p_m \leq p_1 + \Theta p_1$ if and only if $p_i \leq \Theta p_m \leq p_i + \Theta p_i$, $i = 1, 2, \cdots, m-1$ (see Fig. 2). Thus, for any $\{p_i\}$ we want to show the existence of $\{p_i'\}$ such that $p_i' \leq \Theta p_m \leq p_i' + \Theta p_i'$, $\forall\, i \neq m$, and the resulting utilization is not greater than the original utilization. This is done by the following three sequential steps. Steps 1 and 2 construct new periods and execution times such that $p_i \leq \Theta p_m \leq p_i + \Theta p_i$, $\forall\, i \neq m$ while reducing the processor utilization. Because of the way these new periods are constructed, the scheduling algorithm used to derive the utilization with the new periods may not be the RMS algorithm. Step 3 remedies this subtlety and completes the proof.

For any p_j that does not satisfy $p_j \leq \Theta p_m \leq p_j + \Theta p_j$, let $\Theta p_m / p_j = Q + R$, where $Q = \lfloor \Theta p_m / p_j \rfloor$ is the quotient and $0 \leq R < 1$ the remainder. Step 1 constructs p_j' (if any) such that $p_j' \leq \Theta p_m \leq 2p_j'$. Using the p_j''s, Step 2 constructs p_j'', if any, such that $p_j'' \leq \Theta p_m \leq p_j'' + \Theta p_j''$. Note that $Q = 1$ if $p_j \leq \Theta p_m < 2p_j$.

S1: The case of $Q = 0$ and $Q \geq 2$ are dealt with in this step. For $Q = 0$ or $\Theta p_m < p_j$, construct $\{p_i'\}$ such that $p_j' = p_j/2$ and $p_i' = p_i$, $\forall\, i \neq j$. Also construct $\{e_i'\}$ such that $e_j' = 0$ and $e_i' = e_i$, $\forall\, i \neq j$, $i \neq m$ and $e_m' = e_m + e_j$ to fully utilize the processor. We show that $Q' = 1$ and the utilization is reduced by using $\{p_i'\}$ and $\{e_i'\}$ as follows. By the assumption that $\Theta p_j \leq \Theta p_m < p_j$ or $2\Theta p_j/2 \leq \Theta p_m < 2p_j/2$, the relation $p_j' \leq 2\Theta p_j' \leq \Theta p_m < 2p_j'$, and thus, $Q' = 1$ must hold because $\Theta \geq 0.5$. Moreover, because $e_j/p_j \geq e_j/p_m$, the new utilization is not greater than the original utilization. Next for the case of $Q \geq 2$, construct the new periods $\{p_i'\}$ such that $p_j' = Qp_j$ and $p_i' = p_i$, $\forall\, i \neq j$. Also, construct the new execution times $\{e_i'\}$ such that $e_i' = e_i$, $\forall\, i \neq m$ and $e_m' = e_m + (Q-1)e_j$ to fully utilize the processor. For these new periods and execution times, $Q' = 1$. Again, the original utilization is reduced by these e_i's since

$$\begin{aligned} U - U' &= \left(\frac{e_j}{p_j} + \frac{e_m}{p_m}\right) - \left(\frac{e_j}{Qp_j} + \frac{e_m + (Q-1)e_j}{p_m}\right) \\ &= \frac{e_j}{p_j}\left(1 - \frac{1}{Q}\right) - \frac{(Q-1)e_j}{p_m} \\ &= (Q-1)e_j\left(\frac{1}{Qp_j} - \frac{1}{p_m}\right) \geq 0 \end{aligned}$$

because $Qp_j \leq \Theta p_m \leq p_m$.

In this step, p_j''s are constructed such that $Q' = 1$ or $p_j' \leq \Theta p_m < 2p_j'$ for each p_j'. However, there may still exist some p_j''s such that $p_j' + \Theta p_j' < \Theta p_m < 2p_j'$, which is undesirable for Theorem 4. Therefore in Step 2, new periods and execution times are constructed from these p_j''s such that the utilization is reduced further.

S2: Suppose $p_j' + \Theta p_j' < \Theta p_m$, $1 \leq j \leq m-1$, where p_j''s are the new periods constructed in S1. Construct $\{p_i''\}$ such that $p_i'' = p_i'$, $\forall\, i \neq j$ and $p_j'' \leq \Theta p_m \leq p_j'' + \Theta p_j''$. Also, construct $\{e_i''\}$ such that $e_j'' = 0$, $e_i'' = e_i'$, $\forall\, i \neq j$, $i \neq m$, and $e_m'' = e_m' + 2e_j'$ to fully utilize the processor. Then

$$\begin{aligned} U' - U'' &= \frac{e_j'}{p_j'} + \frac{e_m'}{p_m'} - \frac{2e_j' + e_m'}{p_m'} \\ &= \frac{e_j'}{p_j'} - \frac{2e_j'}{p_m'} \geq 0 \end{aligned}$$

because by assumption, $\Theta p_m = \Theta p_m' \geq \Theta p_j' + p_j'$, and thus, $p_m' \geq 2p_j'$.

U'' is obtained under the RMS algorithm based on the original set of periods $\{p_i\}$. In other words, the scheduling algorithm which we used to obtain U'' in Step 2 may not be the RMS algorithm based on $\{p_i''\}$ except that $p_i \leq p_j$ implies $p_i'' \leq p_j''$. In Step 3 below, the utilization V result-

[4] Note that the conditions $p_{m-1} \leq \Theta p_m$ and $p_m \leq 2p_1$ of Theorem 4 together imply $p_{m-1} \leq \Theta p_m \leq p_1 + \Theta p_1$. But the converse is not true. Thus, the domain described by conditions $p_{m-1} \leq \Theta p_m$ and $p_m \leq 2p_1$ is a subset of that described by $p_{m-1} \leq \Theta p_m \leq p_1 + \Theta p_1$.

ing from the RMS algorithm based on $\{p_i''\}$ is shown to be less than or equal to the utilization U resulting from the RMS algorithm based on $\{p_i\}$.

S3: Let W be the utilization associated with $\{p_i\}$ and $\{e_i\}$ under the RMS algorithm based on $\{p_i''\}$, i.e., the same priority assignment as V. Then, from the discussions of Steps 1 and 2, $V \le W$. On the other hand, $U \ge W$ because from Theorem 1, the RMS algorithm is optimal. Thus, $U \ge W \ge V$.

Based on the above discussions, we conclude that for any $\{p_i\}$, there always exists a new set $\{p_i''\}$ such that $p_i'' \le \Theta p_m \le p_i'' + \Theta p_i''$, $\forall\, i \ne m$, and the original utilization is reduced. Since $p_{m-1} \le \Theta p_m \le p_1 + \Theta p_1$ if and only if $p_i \le \Theta p_m \le p_i + \Theta p_i$, $\forall i \ne m$, the theorem directly follows from Lemma 4. **Q.E.D.**

2.2.2. Deriving U_h. Given $\Theta \le 1$, U_h is the *limit* of maximum processor utilization achievable among all task periods and their associated execution times. As was done for U_l, U_h is first derived for the case of only two tasks, and then extended for an arbitrary number of tasks.

We need to consider the same two cases C1 and C2 used for deriving U_l. Further, the properties of the utilization in Eqs. (1) and (2) need to be used in the following theorem.

THEOREM 6. *For any two tasks and a given $\Theta \le 1$, $U_h = 1 - (1 - \Theta)^2$ occurs when Θp_2 is a multiple of p_1.*

Proof. The theorem is proved by considering both C1 and C2. For C1, U is expressed as either Eq. (1) or (2), depending on whether $e_1 \le \xi$ or $e_1 > \xi$. Moreover, Eq. (2) is monotonically nondecreasing in e_1 while Eq. (1) could be monotonically nondecreasing or nonincreasing in e_1. Therefore, given p_1 and p_2, the maximum of U must occur at either $e_1 = 0$ or $e_1 = \Theta p_1$, the two extreme points of e_1. It can be easily derived from Eq. (1) that if $e_1 = 0$ then $U = \Theta$, and from Eq. (2) that if $e_1 = \Theta p_1$ then

$$U = \frac{e_1}{p_1} + \frac{e_2}{p_2} = \frac{p_1}{p_2}\left\lfloor \frac{\Theta p_2}{p_1} \right\rfloor + \Theta p_1 \left\{ \frac{1}{p_1} - \frac{1}{p_2}\left\lfloor \frac{\Theta p_2}{p_1} \right\rfloor \right\} \tag{10}$$

$$= \Theta + (1 - \Theta)\frac{p_1}{p_2}\left\lfloor \frac{\Theta p_2}{p_1} \right\rfloor \ge \Theta.$$

That is, given p_1 and p_2, the maximum U occurs at $e_1 = \Theta p_1$. To derive U_h under C1, the largest value of U in Eq. (10) over all possible p_1's and p_2's is determined as follows. The U in Eq. (10) will be maximized when

$$\frac{p_1}{p_2}\left\lfloor \frac{\Theta p_2}{p_1} \right\rfloor$$

is maximized, i.e.,

$$\left\lfloor \frac{\Theta p_2}{p_1} \right\rfloor = \frac{\Theta p_2}{p_1},$$

or equivalently, Θp_2 is a multiple of p_1. Under C1,

$$U_h = \Theta + (1 - \Theta)\frac{p_1}{p_2}\frac{\Theta p_2}{p_1} = 1 - (1 - \Theta)^2.$$

For C2, the achievable utilization U is again expressed as Eq. (1), which, unlike that of C1, is a monotonically nondecreasing function of e_1. Therefore, for given p_1 and p_2, the maximum of U occurs at $e_1 = \Theta p_1$, and

$$U = \Theta + \Theta p_1\left\{\frac{1}{p}p_1 - \frac{1}{p_2}\left\lceil \frac{\Theta p_2}{p_1} \right\rceil\right\} \tag{11}$$

$$= 2\Theta - \Theta\frac{p_1}{p_2}\left\lceil \frac{\Theta p_2}{p_1} \right\rceil.$$

To derive U_h under C2, we likewise identify the largest value of U in Eq. (11) over all possible p_1's and p_2's as follows. The U in Eq. (11) will be maximized when

$$\frac{p_1}{p_2}\left\lceil \frac{\Theta p_2}{p_1} \right\rceil$$

is minimized, i.e.,

$$\left\lceil \frac{\Theta p_2}{p_1} \right\rceil = \frac{\Theta p_2}{p_1},$$

or equivalently, Θp_2 is a multiple of p_1. Under C2,

$$U_h = 2\Theta - \Theta\frac{p_1}{p_2}\frac{\Theta p_2}{p_1}$$

$$= 1 - (1 - \Theta)^2,$$

which is the same as that under C1. **Q.E.D.**

Before deriving U_h for an arbitrary number of tasks, consider a set $\{p_i\}$ of periods such that both Θp_j and p_j, $j = 2, 3, \cdots, m - 1$, are multiples of p_i, $i = 1, 2, \cdots, j - 1$, and Θp_m is a multiple of p_i, $\forall\, i \ne m$ (see Fig. 3). Thus, let $\Theta p_j = \alpha_{j,i} p_i$ and $\Theta p_m = \alpha_{m,i} p_i$, where $\alpha_{j,i}$'s and $\alpha_{m,i}$'s are positive integers. Then we have the following lemma, which identifies a case where U_h can be derived in Theorem 7.

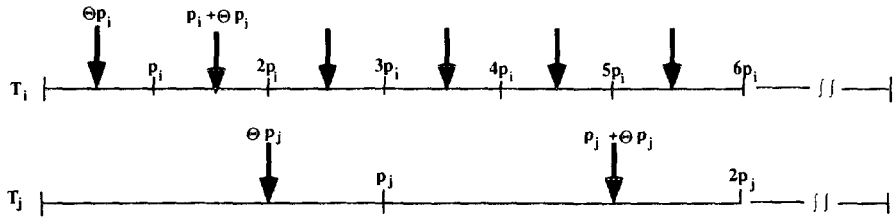


FIG. 3. Both Θp_j and p_j are multiples of p_i.

LEMMA 5. *For $m \geq 2$ tasks with periods given above and given $\Theta \leq 1$, the maximum U occurs at*

$$e_1 = \Theta p_1,$$
$$e_j = \Theta p_j - \sum_{i=1}^{j-1} \alpha_{j,i} e_i, \qquad j \geq 2. \tag{12}$$

Proof. Consider tasks in the order of $T_1, T_2, \cdots, T_m$. Suppose there exists T_j such that e_j does not satisfy (12) while all e_i, $1 \leq i < j$, satisfy (12). In particular, $e_j < \Theta p_j$ if $j = 1$, and $e_j < \Theta p_j - \sum_{i=1}^{j-1} \alpha_{j,i} e_i$ if $j \geq 2$. Then, construct $\{e_i'\}$ such that (i) Eq. (12) is now satisfied for e_i', $\forall i \leq j$, and (ii) the completion times of all invocations of other T_i's, $\forall i > j$, remain unchanged within the interval $[0, \Theta p_m)$. Specifically,

$$e_i' = e_i, \quad i = 1, 2, \cdots, j-1,$$
$$e_j' = e_j + \Delta_j,$$
$$e_{j+1}' = e_{j+1} - \Delta_{j+1} = e_{j+1} - \alpha_{j+1,j} \Delta_j,$$
$$e_i' = e_i - \Delta_i$$
$$= e_i - \left\{ \alpha_{i,j} \Delta_j - \sum_{k=1}^{i-j-1} \alpha_{i,j+k} \Delta_{j+k} \right\},$$
$$i = j+2, j+3, \cdots, m,$$

where $\Delta_j > 0$ is the *increase* in e_j to satisfy Eq. (12), and $\Delta_i \geq 0$, $i \geq j+1$, is the corresponding net *decrease*[5] in e_i to keep the completion times of all T_i's unchanged within $[0, \Theta p_m)$. A simple algebraic manipulation leads to $\Delta_i = (p_i/p_j)\, \Delta_j \Theta (1-\Theta)^{i-j-1}$, $\forall i \geq j+1$. It follows that

$$U' - U = \frac{\Delta_j}{p_j} - \sum_{i=j+1}^{m} \frac{\Delta_i}{p_i}$$
$$= \frac{\Delta_j}{p_j} (1-\Theta)^{m-j} \geq 0,$$

where U and U' are the processor utilizations associated with $\{e_i\}$ and $\{e_i'\}$, respectively. If the above arguments are applied repeatedly over all e_i's until Eq. (12) is satisfied, then one can increase the processor utilization monotonically until the maximum U is reached. **Q.E.D.**

With the above lemma, the following theorem can be easily proved.

THEOREM 7. *Let T be a set of $m \geq 2$ tasks with periods $p_1 < p_2 < \cdots < p_m$. If both Θp_j and p_j, $j = 2, 3, \cdots, m-1$, are multiples of p_i, $\forall i < j$, and Θp_m is a multiple of p_i, $\forall i < m$, then $U_h = 1 - (1-\Theta)^m$.*

Proof. From the results of Lemma 5 and Eq. (12), the maximum achievable utilization U can be derived as

$$U = \sum_{j=1}^{m} \frac{e_j}{p_j} = \Theta + \sum_{j=2}^{m} \Theta \left(1 - \sum_{i=1}^{j-1} \frac{e_i}{p_i}\right)$$
$$= \Theta(1 + (1-\Theta) + (1-\Theta)^2 + \cdots + (1-\Theta)^{m-1}) \tag{13}$$
$$= 1 - (1-\Theta)^m. \qquad \textbf{Q.E.D.}$$

Let U be the utilization for a given $\Theta \leq 1$. Then, it is not difficult to see that a new period p_j' of T_j can always be constructed such that both $\Theta p_j'$ and p_j' are multiples of p_i, $\forall i < j$, provided each of such p_i's and Θ are rational numbers. Using this observation, we have the following lemma—which is necessary to show in Theorem 8 that the U_h obtained in Theorem 7 is indeed the U_h for arbitrary m tasks.

LEMMA 6. *If p_m is modified to p_m' such that $\Theta p_m'$ is a multiple of p_i, $\forall i < m$, then the utilization corresponding to the new periods is greater than that to the original periods.*

Proof. Consider T_m and a particular T_i, $i \neq m$, while ignoring all the other tasks. From Theorem 6, for any $e_i \leq \Theta p_i$, the maximum utilization occurs when Θp_m is a multiple of p_i. Thus, the function

$$f_i(\Theta p_m) \equiv \frac{\Theta p_m - E_i(\Theta p_m)}{p_m} \leq \Theta \left(1 - \frac{e_i}{p_i}\right), \tag{14}$$

where $E_i(\Theta p_m)$ represents the total execution time of T_i within $[0, \Theta p_m)$ and the RHS of the above inequality is the part of utilization contributed by T_m as Θp_m is a multi-

[5] A temporary negative e_i' will not affect the proof of the theorem.

ple of p_i. Considering each (T_i, T_m) pair will lead to

$$\sum_{i=1}^{m-1} f_i(\Theta p_m) = \sum_{i=1}^{m-1} \frac{\Theta p_m - E_i(\Theta p_m)}{p_m}$$

$$= (m-2)\Theta + \frac{\Theta p_m - \sum_{i=1}^{m-1} E_i(\Theta p_m)}{p_m}$$

$$\le (m-2)\Theta + \Theta\left(1 - \sum_{i=1}^{m-1} \frac{e_i}{p_i}\right).$$

It follows that

$$\frac{\Theta p_m - \sum_{i=1}^{m-1} E_i(\Theta p_m)}{p_m} \le \Theta\left(1 - \sum_{i=1}^{m-1} \frac{e_i}{p_i}\right). \qquad (15)$$

Notice that the LHS of inequality (15) represents the part of U contributed by T_m with the original period p_m, whereas the RHS denotes that contributed by T_m as $\Theta p'_m$ is a multiple of p_i, $\forall i < m$. Since the part of utilization contributed by other tasks are the same, the lemma follows. **Q.E.D.**

Now, for a general $\{p_i\}$, we want to show the existence of a new set of periods which satisfies the property in Theorem 7 and increases processor utilization.

THEOREM 8. *For a set T of $m \ge 2$ tasks and a given $\Theta \le 1$, $U_h = 1 - (1 - \Theta)^m$.*

Proof. Consider tasks in order of T_m, T_{m-1}, ..., and T_1. Without loss of generality, Θp_m can be assumed to be a multiple of p_i, $\forall i < m$. (If not, from Lemma 6 a new p'_m can always be constructed such that $\Theta p'_m$ is a multiple of all other p_i's while increasing the processor utilization.)

Let $T_j \in T$ be the first task where both Θp_j and p_j, $2 \le j \le m-1$, are not multiples of p_i, $\forall i < j$. Then, construct a new p'_j such that both $\Theta p'_j$ and p'_j are multiples of p_i, $\forall i < j$. Assume that $p'_j = \sigma p_j$, where σ is a positive real number and p_j the original period of T_j. For all the other tasks, set

$$p'_j = \begin{cases} p_i, & i = 1, 2, \cdots, j-1, \\ \sigma p_i, & i = j+1, \cdots, m. \end{cases}$$

It is worth pointing out that after the above modification, $\Theta p'_m$, $\Theta p'_i$ and p'_i, $i = j+1, \cdots, m-1$, are still multiples of p'_k, $k = 1, 2, \cdots, j-1$.

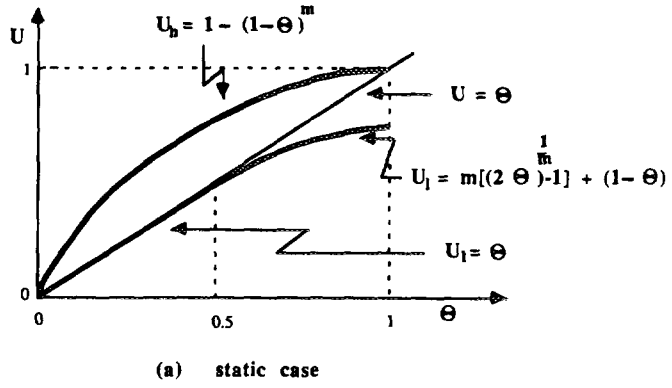
On the other hand, each e'_i is constructed as follows. First, set $e'_i = e_i$, $\forall i < j$, to keep the part of utilization contributed by such T_i's unchanged. Second, e'_j is constructed such that e'_j and e'_i, $i = 1, 2, \cdots, j-1$, fully utilize the processor. From Lemma 6, $U'_j = \Theta - \Theta\sum_{i=1}^{j-1} U'_i \ge U_j$, where U'_j is the new utilization contributed by T_j and U_j the old utilization by T_j. Finally, constructed e'_i, $i = j+1, j+2, \cdots, m$, such that e'_i and p'_k, $k = 1, 2, \cdots, i-1$, fully utilize the processor. Likewise, $U'_i = \Theta - \Theta\sum_{k=1}^{i-1} U'_k$. Let $\Delta = U'_j - U_j \ge 0$, then a simple algebraic manipulation leads to $U' - U = \Delta(1-\Theta)^{m-j} \ge 0$, where $U \equiv \sum_{i=1}^{m} U_i$ and $U' \equiv \sum_{i=1}^{m} U'_i$.

The above procedure can be repeatedly applied to p_{j-1}, p_{j-2} and so on, until each Θp_j and p_j, $2 \le j \le m-1$, are multiples of p_i, $\forall i < j$, so as to increase the utilization monotonically. Thus, this theorem follows from Theorem 7. **Q.E.D.**

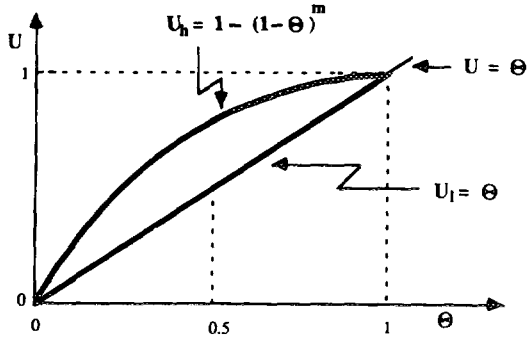
In Theorem 8, each of $\{p_i\}$ and Θ must be rational numbers such that both $\Theta p'_j$ and p'_j can be multiples of p_i, $i = 1, 2, \cdots, j-1$. Otherwise, p'_j must be constructed to make $\Theta p'_j$ and p'_j as close to multiples of p_i's as possible. This is the reason why U_h is termed "the limit of the maximum" of U. U_l and U_h are shown in Fig. 4a.

3. OPTIMAL DYNAMIC SCHEDULING OF PERIODIC TASKS

For the static scheduling algorithms discussed in the last section, priorities assigned to all invocations of a task are the same. For the dynamic algorithms to be addressed in this section however, different priorities may be assigned to different invocations of the same task. An optimal dynamic algorithm can naturally outperform its



(a) static case



(b) dynamic case

FIG. 4. U_l and U_h as functions of Θ.

static counterpart because it requires less stringent constraints in scheduling tasks.

3.1. Optimal Dynamic Scheduling Algorithm

To develop an optimal dynamic scheduling algorithm w.r.t. Θ, it is necessary to introduce an optimal single-machine scheduling algorithm developed in [2]. A set of independent[6] jobs with arbitrary release times is to be scheduled on a single machine so as to minimize the maximum job completion cost, where the cost associated with each job can be any monotone nondecreasing function of its competition time. As can be seen from the following steps, the computational complexity of the algorithm is $O(N^2)$, where N is the number of jobs to be scheduled.

SA1. Arrange the jobs in nondecreasing order of their release times to create a set of disjoint blocks of jobs. For example, suppose jobs X, Y and Z are released at $t = 0$, 2, 15, respectively, and 5 units of time are required to complete each of X and Y. Then, two blocks of jobs {X,Y} and {Z} will be created.

SA2. Consider a block B with block completion time $t(B)$. Let B' be the set of jobs in B which do not precede any other job in B. Select a job l from B' such that $f_l(t(B))$ is the minimum, where $f_l(t)$ is the nondecreasing cost function of job l if it is completed at t. This implies that l be the last job to be completed in B.

SA3. Create subblocks of jobs in the set $B - \{l\}$ by arranging the jobs in nondecreasing order of release times as in SA1. The time interval(s) allotted to l is (are) then the difference between the interval of B and the interval(s) allotted to these subblocks.

SA4. For each subblock, repeat SA2 and SA3 until time slot(s) is (are) allotted to every job.

By setting the cost function $f_{iv}(t)$ of T_{iv}, the vth invocation of T_i, as $f_{iv}(t) := (t - a_{iv})/p_i$, the above algorithm can be simplified to schedule periodic tasks while minimizing the system hazard as follows. Because T_{iv} must be completed before $T_{i(v+1)}$, only one invocation from each of the m tasks needs to be considered in Step SA2 in order to determine the last invocation to be completed in each block. Therefore, the computational complexity can be reduced to $O(mn)$, where n is the total number of invocations for the m tasks in T within a planning cycle.

For example, consider two tasks, T_1 and T_2, with $p_1 = 10$, $e_1 = 3$, $p_2 = 30$ and $e_2 = 8$. That is, a total of four invocations, T_{11}, T_{12}, T_{13} and T_{21}, need to be scheduled within the planning cycle $I = [0, 30)$. As the above algorithm is applied to this example, two blocks $B_1 = \{T_{11}, T_{21}, T_{12}\}$ in the interval [0, 14], and $B_2 = \{T_{13}\}$ in [20, 23] are created from Step SA1. From SA2, [20, 23] is trivially allotted to T_{13}. For B_1 however, since T_{11} must be completed before T_{12}, only T_{12} and T_{21} need to be considered as to which of them should be completed last in B_1. From SA2, it turns out that $f_{12}(14) = (14 - 10)/10 = 0.4 < 14/30 = f_{21}(14)$. Therefore, T_{12} must be completed last in B_1. Deleting T_{12} from B_1 and arranging T_{11} and T_{21} as described in SA3, a subblock $B_{11} = \{T_{11}, T_{21}\}$ in [0, 11] is created, meaning that the slot [11, 14] is allotted to T_{12}. Repeating the above steps on B_{11}, it follows that T_{11} must be completed before T_{21}, and the resulting system hazard $\Theta = 0.4$, which is the normalized flowtime of T_{12}. Notice that if the EDD algorithm [1]—which simply schedules the "ready" invocations in the nondecreasing order of their deadlines—is used in the example, then the resulting Θ becomes 14/30 (the normalized flowtime of T_{21}). Therefore, the EDD scheduling algorithm is *not*[7] an optimal algorithm in the sense of minimizing Θ.

3.2. Achievable Processor Utilization Bounds

As in the static case, we are also interested in deriving the two utilization bounds, U_l and U_h. Recall that both U_l and U_h must be derived with the processor fully utilized under the above optimal dynamic scheduling algorithm. However, it is not always easy to derive U_l and U_h based on this optimal dynamic scheduling algorithm. The following theorem remedies this difficulty by showing that U_l and U_h can also be derived under the EDD scheduling algorithm after properly modifying the deadlines of the task invocations to be scheduled.

THEOREM 9. *Given any $\Theta \leq 1$, U_l and U_h derived under the optimal dynamic scheduling algorithm are the same as those derived under the EDD scheduling algorithm with $(v - 1)p_i + \Theta p_i$ as T_{iv}'s deadline, $\forall i$ and v.*

Proof. Given Θ, let U_l' and U_h' denote the two bounds obtainable under the above EDD algorithm. Then, we want to show that $U_l = U_l'$ and $U_h = U_h'$. Let $\{p_i\}$ and $\{e_i\}$ be the sets of periods and execution times where $U = U_l$ is obtained under the optimal dynamic scheduling algorithm. That is, given $\Theta \geq 1$, each T_{iv} can be completed by the time $(v - 1)p_i + \Theta p_i$, $\forall i$ and v. However, by using $(v - 1)p_i + \Theta p_i$ as the deadline of T_{iv}, the EDD algorithm must also be able to generate a schedule under which T_{iv} can be completed before $(v - 1)p_i + \Theta p_i$, meaning that $U_l' \geq U_l$. By switching the roles of U_l' and U_l in the above argument, we can show that $U_l' \leq U_l$. Thus, $U_l = U_l'$. Similarly, $U_h = U_h'$. **Q.E.D.**

By using Theorem 9, U_l and U_h are derived in the next two subsections.

[6] The algorithm in [2] also schedules dependent jobs.

[7] One might argue that for each T_{iv}, if we use the revised deadline $(v - 1)p_i + \Theta^* p_i$ instead of the original deadline vp_i, then the EDD algorithm is optimal. However, this is true (see Theorem 9) only had we known Θ^*, which, unfortunately, is yet to be found.

3.2.1. Deriving U_l. The following lemma follows directly from [7].

LEMMA 7. *Let T be a set of m tasks with periods $\{p_i\}$ and execution times $\{e_i\}$. Then, using vp_i as T_{iv}'s deadline, the EDD scheduling algorithm is feasible w.r.t. the deadline if and only if the processor utilization*

$$U \equiv \sum_{i=1}^{m} \frac{e_i}{p_i} \leq 1.$$

LEMMA 8. *Given $\Theta \leq 1$, the EDD scheduling algorithm with $(v-1)p_i + \Theta p_i$ as T_{iv}'s deadline is feasible w.r.t. Θ if*

$$U = \sum_{i=1}^{m} \frac{e_i}{p_i} \leq \Theta.$$

Proof. For any set T of m tasks with $U = \sum_{i=1}^{m} e_i/p_i \leq \Theta$, we want to prove that there always exists at least a schedule in which each T_{iv} can be completed by its deadline, $(v-1)p_i + \Theta p_i$. Consider a modified task set T' of T where T' is the same as T except that the execution time e_i' in T' is e_i/Θ instead of the original e_i in T. Since $U' = U/\Theta \leq 1$, where U' is the processor utilization for T', there always exists a schedule S' in which each T_{iv} of T' can be completed before vp_i. However, since $e_i = \Theta e_i'$, there must exist a schedule in which each T_{iv} of T can be completed by its deadline $(v-1)p_i + \Theta p_i$. Since the EDD scheduling algorithm is optimal in meeting deadlines, the theorem follows. **Q.E.D.**

U_l can now be derived as in the following theorem.

THEOREM 10. *For a set T of $m \geq 1$ tasks and a given $\Theta \leq 1$, $U_l = \Theta$.*

Proof. The theorem can be proved from the following three facts. First, from Lemma 8, a feasible schedule for the given Θ can always be derived from the EDD algorithm as long as the processor utilization does not exceed Θ. Second, consider a particular set of execution times such that $e_i = 0$, $\forall i \neq m$; for this particular instance of $\{e_i\}$, no feasible schedule with a system hazard not exceeding Θ may exist if $e_m > \Theta p_m$. That is, given $\Theta \leq 1$, there exists an instance of T where no feasible schedule with a system hazard not exceeding Θ may exist should the processor utilization exceed Θ. Based on the above two facts, $\Theta = U_l'$, where U_l' is the U_l under the EDD scheduling algorithm with $(v-1)p_i = \Theta p_i$ as T_{iv}'s deadline. Finally, from Theorem 9, U_l—which must be obtained under the optimal dynamic scheduling algorithm—is the same as U_l'. **Q.E.D.**

Notice that for $\Theta \leq 0.5$, the same U_l is obtained under either the optimal dynamic or optimal static scheduling algorithm. If $0.5 < \Theta \leq 1$ however, the U_l associated with the optimal dynamic algorithm is greater than that associated with the optimal static algorithm.

3.2.2. Deriving U_h. Consider a task $T_i \in T$. Since T_{iv} arrives at time $(v-1)p_i$ and must be completed by its deadline $(v-1)p_i + \Theta p_i$, T_i can only be executed during the set of intervals $E_i \equiv \bigcup_{v=1}^{L/p_i} E_{iv}$, where $E_{iv} \equiv [(v-1)p_i, (v-1)p_i + \Theta p_i)$. $E \equiv \bigcup_{i=1}^{m} E_i$ is the *executable time zone* (ETZ) of T, which represents the set of time intervals within $[0, L)$ during which tasks of T can be executed. Let $|E|$ represent the total length of E. Then the *executable time ratio* (*ETR*) of T is defined as $|E|/L$. Several insights can be drawn from the notion of *ETR* as follows. First, $ETR \leq 1$. Second, if $U > ETR$, then no feasible schedule exists for the given Θ. Finally, it was also shown in Theorem 8 that for any $\Theta \leq 1$, the maximum *ETR* occurs when both Θp_j and p_j, $j = 2, 3, \cdots, m-1$, are multiples of p_i, $\forall i < j$, and Θp_m is a multiple of p_i, $\forall i < m$. Or, the supremum of *ETR* among all instances of T with m tasks is $1 - (1-\Theta)^m$.

U_h can now be obtained as in the following theorem.

THEOREM 11. *For a set T of $m \geq 1$ tasks and a given $\Theta \leq 1$, $U_h = 1 - (1-\Theta)^m$.*

Proof. From the definition of U_h, we need to show that (i) if $U > 1 - (1-\Theta)^m$, then T is never schedulable w.r.t. Θ, and (ii) there exists at least a feasible instance of T if $U < 1 - (1-\Theta)^m$. The claim in (i) is obvious because $1 - (1-\Theta)^m$ is the supremum of *ETR* among all instances of T. Thus, no feasible instance of T may exist if $U > 1 - (1-\Theta)^m$. On the other hand, the claim in (ii) has already been shown to be true in Theorem 8. In particular, given $U < 1 - (1-\Theta)^m$, the feasible instance of T is the same as that for the static scheduling algorithm as derived in Theorem 8. **Q.E.D.**

Note that U_h for the optimal dynamic scheduling algorithm happens to be the same as that for its static counterpart. U_l and U_h are shown in Fig. 4b.

4. ON-LINE SCHEDULING ALGORITHMS

So far, we have derived optimal scheduling algorithms under the assumptions that the arrival times, execution times and deadlines of all tasks (invocations) are known in advance. Both the optimal static algorithm derived in Section 2 and the optimal dynamic algorithm in Section 3 are thus *off-line* algorithms. When the above information on each task is not known until it actually arrives, an *on-line* algorithm must be found.

An on-line scheduling algorithm, if any, is said to be *optimal* if it always generates a schedule which is as good as that generated by its off-line dynamic counterpart. Note that the existence of an optimal on-line algorithm depends not only on the scheduling objective but also on

the characteristics of the randomly arriving tasks. For example, in case of a single processor, it has been shown in [4, 9] that the on-line EDD scheduling algorithm is always optimal in meeting task deadlines.

In this section, we consider a system in which, in addition to periodic tasks, aperiodic tasks arrive randomly. While the information of all periodic tasks is assumed known a priori to the scheduler, the information of each aperiodic task is unknown until its actual arrival. We shall prove that, except for some extremely simple cases, no optimal on-line scheduling algorithms w.r.t. Θ in such a system exist. Thus, instead of deriving the two processor utilization bounds U_l and U_h, we shall describe simple on-line mechanisms which are useful to determine whether or not an arriving aperiodic task can be completed with a system hazard not exceeding Θ without disturbing the execution of periodic tasks. These mechanisms can be used to implement load-sharing strategies [11, 14] where tasks are transferred from over-loaded processing nodes (PNs) to under-loaded PNs. A PN may use the mechanisms proposed here to decide on whether to execute an arriving task locally or transfer it to another PN for execution.

4.1. Unavailability of Optimal On-Line Scheduling Algorithms

The following theorem shows that an optimal on-line scheduling algorithm w.r.t. the system hazard does not always exist.

THEOREM 12. *No optimal on-line scheduling algorithms w.r.t. Θ exist in a system where aperiodic tasks arrive randomly with general execution times and deadlines.*

Proof. First, considering a system in which there exists no periodic tasks, we shall prove the theorem by contradiction. Suppose there is an optimal on-line scheduler and consider the following scenario. At time t, there exist two aperiodic tasks T_1 and T_2. T_1 arrived at the time $t - 20$ with deadline $t + 20$ and remaining execution time 3, while T_2 arrived at t with deadline $t + 10$ and execution time 3. Starting from t, the optimal scheduler will schedule these two tasks on a single processor. Assume no tasks arrive in $[t, t + 3)$ and consider the following two cases depending on which of T_1 and T_2 is executed in $[t, t + 3)$.

Case 1. T_1 is executed in $[t, t + 3)$. In this case, T_1 is completed at $t + 3$ and the remaining execution time of T_2 is 3. Now, consider the scenario where T_3 arrives at $t + 3$ with execution time 4 and deadline $t + 8$, and T_3 is the only task that will ever arrive at, or after, $t + 3$. Then, $\Theta = \max\{0.575, 1.0, 0.8\} = 1.0$, which is achieved by executing T_3 in $[t + 3, t + 7)$ and T_2 in $[t + 7, t + 10)$. However, if we have had at t the knowledge of task arrivals at, or after, $t + 3$, the minimum system hazard would become $\Theta = \max\{0.75, 0.3, 0.8\} = 0.8$, which is achieved by executing these three tasks in order of T_2, T_3 and T_1.

Case 2. T_2 is executed in $[t, t + 3)$. In this case, T_2 is completed at $t + 3$ and the remaining execution time of T_1 is 3. Now, consider the scenario where no task will ever arrive at, or after, $t + 3$. Then, the minimum system hazard is $\Theta = \max\{0.65, 0.3\} = 0.65$. However, if at t we have known that no task will ever arrive at, or after, $t + 3$, the minimum system hazard would become $\Theta = \max\{0.575, 0.6\} = 0.6$, which is achieved by executing these two tasks in the order of T_1 and T_2.

From the above discussion, we conclude that the assumed optimal on-line scheduler cannot always generate a schedule as good as that generated by an optimal off-line scheduler. The theorem follows since a system with only aperiodic tasks is a special case of the general system in which both periodic and aperiodic tasks exist. **Q.E.D.**

Even though an optimal on-line scheduling algorithm does not exist, there are special cases where such an algorithm exists. For example, in a system where each (periodic or aperiodic) task has the same deadline since its arrival, the on-line EDD scheduler described above is optimal.

4.2. On-Line Determination of the Schedulability for an Arriving Aperiodic Task

Scheduling aperiodic tasks in the presence of periodic ones is important in real-time applications because of the unpredictable and time-critical nature of aperiodic tasks. A solution approach which is most commonly seen in the open literature [8, 12, 13] suggests the following. First, some minimal interarrival time is assumed to exist for all aperiodic tasks. Second, treat the aperiodic task stream as a periodic task with a period of the minimal interarrival time. Third, use the RMS algorithm or variations thereof to determine whether or not the resulting schedule is feasible with $\Theta = 1$. Despite its simplicity, this approach has the following drawbacks:

D1. Since there may exist different types of aperiodic tasks, it may not be realistic to assume a minimal interarrival time between an aperiodic task of one type and that of another.

D2. As pointed out and analyzed in [5], a system with very low processor utilization might result from the conservative nature of this approach.

In what follows, simple on-line mechanisms which are more realistic and could result in better processor utilization are described. They are presented only under the following three different scheduling algorithms for the

tasks involved: (A1) static scheduling of periodic tasks with the lowest priority given to aperiodic tasks, (A2) static scheduling of periodic tasks with the highest priority given to aperiodic tasks, and (A3) dynamic scheduling of all tasks. For all three algorithms, consider the current time instant t when an aperiodic task with execution time e and deadline $t + d$ arrives. Also, assume that all tasks (the periodic tasks and the aperiodic ones already accepted) residing in the processor before t are guaranteed to be completed in time with a system hazard not exceeding Θ.

For A1, the RMS algorithm is used for periodic tasks, and aperiodic tasks are assigned the lowest priority (the FCFS rule is used among aperiodic tasks). Let $R(t) \leq t$ denote the cumulated processor idle time in $[0, t]$ if there are only periodic tasks in the processor. Also, let $CET(t)$ be the total cumulated remaining execution time of all aperiodic tasks accepted by time t. Then, the processor may accept the arriving aperiodic task only if $R(t + \Theta d) - R(t) \geq e + CET(t)$.

A2 is similar to A1 except that the highest priority is assigned to the aperiodic tasks. The arriving aperiodic task would be accepted only if (i) it can be completed by $t + \Theta d$ and (ii) the resulting normalized flowtime is not greater than Θ for each unfinished periodic task invocation within the planning cycle. Specifically, the planning cycle *involved* is the current planning cycle if at time t there exists at least an unfinished invocation within the cycle. Then, the processor may accept the arriving task only if $\Theta d \geq CET(t) + e$ and $\max_{T_{iv}}(c_{iv} - a_{iv})/p_i \leq \Theta$, where c_{iv}, a_{iv} and p_i are the completion time, arrival times and period, respectively, of T_{iv} for all unfinished T_{iv}'s within the planning cycle.

A3 uses the on-line EDD algorithm for both periodic and aperiodic tasks. Upon arrival of an aperiodic task at time t, A3 uses the on-line EDD algorithm to check if all tasks can be completed with a system hazard not exceeding the pre-specified Θ. If they can, the arriving task is accepted. In addition to the unfinished aperiodic tasks already accepted, the tasks involved in this check includes the arriving aperiodic task and all unfinished periodic tasks.

Note that whether to accept an arriving task w.r.t. Θ is equivalent to treating $t + \Theta d$ as the deadline of the arriving task. Depending on the specific need of the system, a different deadline such as $t + d$ may also be used in the above algorithms for an arriving task.

5. CONCLUSION

Scheduling tasks to complete before their deadlines is an important design issue in hard real-time systems. In this paper, we have proposed and analyzed a new performance measure, called the *system hazard*, for scheduling real-time tasks. The system hazard can be used not only for meeting deadlines but also for measuring how early each task can be completed. Further, since Θ depends only on the execution of tasks, it can be used on a single processor or a multiprocessor/distributed system.

For a single processor with only periodic tasks, optimal static and dynamic scheduling algorithms are derived. In the static case, the RMS algorithm is shown to be optimal w.r.t. the system hazard. In the dynamic case however, the EDD scheduling algorithm is shown to be not optimal w.r.t. the system hazard. More importantly, two best bounds of processor utilization are derived for these two cases.

Since optimal on-line algorithms w.r.t. the system hazard are non-existent for all but some special cases, simple mechanisms are derived to check whether or not an arriving aperiodic task can be completed with a system hazard not exceeding the prespecified value. These mechanisms are useful in a system where load-sharing strategies are adopted to improve system performance. An over-loaded PN may use the mechanisms to determine whether or not to transfer the arriving task to an under-loaded PN for execution.

APPENDIX: SUMMARY OF NOTATION AND SYMBOLS

$a_{iv}(c_{iv})$	The arrival (completion) time of T_{iv}. $a_{iv} = (v - 1)p_i$.
$CET(t)$	The total cumulated remaining execution time of all aperiodic tasks accepted by time t.
E	The executable time zone (ETZ) of T representing the set of time intervals during which tasks in T can be executed.
E_i	The set of time intervals during which T_i can be executed.
$E_{iv} \equiv [(v - 1)p_i, (v - 1)p_i + \Theta p_i]$	The time interval during which T_{iv}, the vth invocation of T_i, can be executed.
e_i	The execution time of T_i.
$ETR \equiv \|E\|/L$	The executable time ratio of T.

$I \equiv [0, L)$	A planning cycle of T, where L is the LCM of all task periods in T.
m	The number of tasks in T.
p_i	The invocation period of T_i.
$R(t)$	$R(t) \leq t$ is the cumulated processor idle time in $[0, t]$ when there are only periodic tasks on the processor.
$T \equiv \{T_i\}$	The set of periodic tasks to be scheduled.
U	The processor utilization of T. $U \equiv \sum_{i=1}^{m} e_i/p_i$.
$U_h(U_l)$	The high (low) processor utilization bound of T.
$\Theta(\Theta^*)$	The (minimum obtainable) system hazard of T. $\Theta \equiv \max_{\substack{1 \leq v \leq L/p_i \\ T_i \in T}} (c_{iv} - a_{iv})/p_i$.

REFERENCES

1. K. R. Baker, *Introduction to Sequencing and Scheduling*. Wiley, New York, 1974.
2. K. R. Baker, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, Preemptive scheduling of a single machine to minimize maximum cost subject to release dates and precedence constraints. *Oper. Res.* **31,** 2 (Mar–Apr 1983), 381–386.
3. C. C. Cheng, J. A. Stankovic, and K. Ramamritham, Scheduling algorithms for hard real-time systems—A brief survey. *Real-Time Systems Newsletter* **3,** 2 (Summer 1987), pp. 1–24.
4. M. Dertouzos, Control robotics: The procedural control of physical processes. *Proc. of the IFIP Congress*, 1974, pp. 807–813.
5. K. D. Gordon, L. W. Dowdy, J. Baldo, Jr., and K. J. Rappoport, Scheduling aperiodic tasks with hard deadlines in a rate monotonic framework. *Proc. Sixth IEEE Workshop on Real-Time Operating Systems and Software*, 1989, pp. 1–5.
6. J. Y.-T. Leung, and J. Whitehead, On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation* **2,** (1982), 237–250.
7. C. L. Liu and J. W. Layland, Scheduling algorithms for multiprogramming in a hard real-time environment. *J. Assoc. Comput. Mach.* **20,** 1 (1973), 46–61.
8. J. P. Lehoczky, L. Sha, and J. K. Strosnider, Enhanced aperiodic responsiveness in hard real-time environments. *IEEE Real-Time Systems Symposium*, Dec. 1987, pp. 271–270.
9. A. K.-L. Mok, *Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment*, Ph. D. Dissertation, M.I.T., Cambridge, 1983.
10. D. Peng and K. G. Shin, Static allocation of periodic tasks with precedence constraints in distributed real-time systems. *Proc. IEEE, 9th Int. Conf. Distrib. Compu. Syst.* June 1989, pp. 190–198.
11. K. G. Shin and Y.-C. Chang, Load sharing in distributed real-time systems with state-change broadcasts,"*IEEE Trans. Comput.* **C-38,** 8 (Aug. 1989), 1124–1142.
12. L. Sha, J. P. Lehoczky and R. Rajkumar, Solutions for some practical problems in prioritized preemptive scheduling. *IEEE Real-Time Systems Symposium*, Dec. 1986, pp. 181–191.
13. B. Sprunt, L. Sha, and J. P. Lehoczky, Aperiodic task scheduling for real-time systems. *The Journal of Real-Time Systems*, Kluwer, Dordrecht, 1989, pp. 27–60.
14. Y. T. Wang and R. J. T. Morris, Load sharing in distributed systems, *IEEE Trans. Comput.* **C-34,** 3 (Mar. 1985), pp. 329–336.

Received March 1992; revised June 1992; accepted June 18, 1992

DAR-TZEN PENG received his BSEE from National Cheng-Kung University in 1974, his M.S. degree in Management Science from National Chiao-Tung University in 1976, and his Ph.D. degree in computer science and engineering from the University of Michigan in 1989. Since 1989, he has been with the AlliedSignal Microelectronics and Technology Center as a member of technical staff. Currently, he is involved in the research and design of distributed fault-tolerant real-time computing systems. His research interests include fault-tolerant real-time computing and computer networks. Dr. Peng is a member of the ACM and IEEE Computer Society.

KANG G. SHIN is Professor and Chair of Computer Science and Engineering Division, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, Michigan. He has authored/coauthored over 190 technical papers (about 90 of these in archival journals) and several book chapters in the areas of distributed real-time computing and control, fault-tolerant computing, computer architecture, and robotics and automation. In 1987, he received the Outstanding IEEE Transactions on Automatic Control Paper Award for a paper on robot trajectory planning. In 1989, he also received the Research Excellence Award from The University of Michigan. In 1985, he founded the Real-Time Computing Laboratory, where he and his colleagues are currently building a 19-node hexagonal mesh multicomputer, called **HARTS**, to validate various architectures and analytic results in the area of distributed real-time computing. He received the B.S. degree in electronics engineering from Seoul National University, Seoul, Korea in 1970, and both the M.S. and Ph.D. degrees in electrical engineering from Cornell University, Ithaca, New York in 1976 and 1978, respectively. From 1978 to 1982 he was on the faculty of Rensselaer Polytechnic Institute, Troy, New York. He has held visiting positions at the U.S. Air Force Flight Dynamics Laboratory, AT&T Bell Laboratories, Computer Science Division within the Department of Electrical Engineering and Computer Science at UC Berkeley, and International Computer Science Institute, Berkeley, CA. He is an IEEE fellow, was the Program Chairman of the 1986 IEEE Real-Time Systems Symposium (RTSS), the General Chairman of the 1987 RTSS, the Guest Editor of the 1987 August special issue of *IEEE Transactions on Computers* on Real-Time Systems, and is a program co-chair for the 1992 *International Conference on Parallel Processing*. He currently chairs the IEEE Technical Committee on Real-Time Systems, is a distinguished visitor of the Computer Society of the IEEE, an Editor of *IEEE Trans. on Parallel and Distributed Computing*, and an area editor of *International Journal of Time-Critical Computing Systems*.