Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines

John P. Lehoczky[1]
Department of Statistics
Carnegie Mellon University
Pittsburgh, PA 15213

ABSTRACT

This paper considers the problem of fixed priority scheduling of periodic tasks with arbitrary deadlines. A general criterion for the schedulability of such a task set is given. Worst case bounds are given which generalize the Liu and Layland bound. The results are shown to provide a basis for developing predictable distributed real-time systems.

1 Introduction

The problem of scheduling periodic tasks with hard deadlines on a uniprocessor was first studied by Liu and Layland [5] in 1973. Their paper gave a worst case performance analysis of the rate monotonic scheduling algorithm for their scheduling problem, the optimal fixed priority scheduling algorithm. The rate monotonic theory has been greatly generalized since the original Liu and Layland paper. It now addresses practical issues such as the mixture of periodic and aperiodic tasks, task synchronization, stochastic execution time and processing important tasks in cases of transient overload, see [9, 10]. With few exceptions (see for example [2, 4, 6, 11]), results for the rate monotonic scheduling algorithm have been developed assuming task deadlines are equal to task periods. Indeed, only [11] allows deadlines to exceed the task periods, and it addresses only the optimality of a modification of the rate monotonic algorithm. Consequently, there is a need to extend the theory to cover this case.

In spite of this generality obtained for fixed priority scheduling in the uniprocessor case, only a limited amount of work has been done in the distributed system case. In the distributed case, periodic tasks require a sequence of resources, and the task deadline is in the form of an end-to-end deadline. For example, a periodic process may involve data capture and computation on processor 1, sending the results over a network to processor 2, additional computation followed by display on processor 2. The process is periodic, and the allowable latency time between data capture and display must be no greater than some deadline. There has been some work in the distributed or multi-stage case including the work of Bettati and Liu [1], the work on cycle stealing by Rajkumar, Sha and Lehoczky [8, 10] and the work on multiprocessor task synchronization by Rajkumar [7], but much work needs to be done on developing a fixed priority scheduling theory for the distributed case. In particular, we would like to be able to solve the distributed scheduling problem without resorting to a handcrafted timeline schedule with all its drawbacks (see for example [9]). Such a theory would begin with periodic distributed tasks, each requiring a sequence of resources and associated amounts of processing time on each resource and an end-to-end deadline. The theory would allow one to determine analytically whether a scheduling algorithm such as the rate monotonic algorithm can meet the deadlines of a set of periodic distributed tasks. One might also develop worst case scheduling bounds.

In the uniprocessor case, the Liu and Layland bound is $\log_e 2 = .693$, meaning that if periodic utilization is kept below this level, the optimal fixed priority algorithm will meet all deadlines under all task phasings. The worst case value of .693 is respectably large, and the average case value is often nearly .90 [3]. On the other hand, in the distributed case the worst case utilization levels can be extremely low if the task set is not restricted in some fashion. Consider the following example.

Example 1

Suppose we have two periodic tasks, both of which first use resource 1 followed by resource 2. Let task 1 have period T and computation requirement C_1 and C_2 respectively on these two resources. Suppose task 2 also has period T. If $C_1 + C_2 = T$, then task 1 cannot be interrupted or it will miss its deadline. If these two tasks have the same phasing, then if task 2 has an arbitrarily small processing requirement on each resource, either task 1 or task 2 must miss its deadline. The utilization on resource 1 can be made

[1]Sponsored in part by the Office of Naval Research under contract N00014-84-K-0734, in part by the Naval Ocean Systems Center under contract N66001-87-C-0155, and in part by the Systems Integration Division of IBM Corporation under University Agreement Y-278067.

arbitrarily close to C_i/T for $i = 1, 2$. These utilization sum to 1, thus the worst case *total* utilization on the two resources is 1. The minimum of the two resources utilizations is at most .5. Similar examples show that this quantity drops to 1/3 if we consider 3 resources and becomes $1/r$ for r resources. □

The above example shows that the worst case utilization is too low to be useful, and one must modify the scheduling problem in some way, for example by restricting task utilizations or phasing. A more useful approach is to recognize that if a task requires the use of multiple resources and it must contend for access to each resource, then more than one period should be available for the total task processing. For example, if three resources are needed, we might allow a task with period T initiated at time 0 to have a deadline of 3T. Note that deadline postponement does not prevent new jobs of that task from being initiated. If the first job finished just before 3T, two other jobs (initiated at T and 2T) would already be in progress.

While deadline postponement is a sensible approach, the ability to postpone a deadline depends on the maximum task latency allowed by the application. If a task corresponds to monitoring a control system and responding to a critical condition, then there is a maximum allowable response time, hence a maximum deadline postponement. One simple approach to distributed scheduling is to take the end-to-end deadline and create artificial intermediate deadlines for the processing at each resource. In this way, the distributed problem is decomposed into a set of independent single resource scheduling problems. To do this decomposition in the optimal way, one may wish to use a relatively long artificial deadline for a heavily used resource and a relatively short artificial deadline for a lightly used resource. This means that the resulting single resource scheduling problems will have tasks with deadlines which are different from task periods, a problem not studied by Liu and Layland. Some partial results in this direction were obtained by Leung and Whitehead [4], Lehoczky and Sha [2] and Peng and Shin [6]. To carry out this approach to distributed scheduling, we must have an exact understanding of the tradeoffs between schedulability and deadline postponement in the uniprocessor case, and that is the topic of this paper.

This paper is organized as follows. In Section 2 we introduce the scheduling problem and derive an exact schedulability criterion for an arbitrary fixed priority scheduling algorithm. Section 3 derives worst case bounds, and Section 4 offers a summary and concluding remarks.

2 Scheduling Periodic Tasks with Modified Deadlines

We consider a set of n periodic tasks, $\tau_1, \ldots, \tau_n$. Each task is characterized by four components, (C_n, T_n, D_n, I_n) where

C_i = deterministic computation requirement of each job of τ_i.

T_i = period of τ_i.

D_i = deadline of τ_i.

I_i = phasing of τ_i relative to some fixed time origin.

The j^{th} job of τ_i is ready at time $I_i + (j-1)T_i$, and the C_i units of computation required have a deadline of $I_i + (j-1)T_i + D_i$. We consider fixed priority scheduling algorithms and seek a criterion to determine whether the scheduling algorithm is able to meet all the deadlines of all the jobs in the task set. Throughout this paper we assume the scheduling algorithms are preemptive and ignore all preemption overhead.

To determine if a scheduling algorithm can meet all the task deadlines, it is useful to identify the task phasing which results in the longest response time for any job of a particular task τ_i. We introduce an arbitrary fixed priority scheduling algorithm and assume the priority ordering $\tau_1 < \cdots < \tau_n$ where τ_1 is highest priority and τ_n is lowest. If $T_1 \leq T_2 \leq \cdots \leq T_n$, then this corresponds to rate monotonic scheduling, while if $D_1 \leq D_2 \leq \cdots \leq D_n$ this corresponds to deadline monotonic scheduling. Nevertheless, the results in this section are general for fixed priority scheduling.

To determine the worst case response time for a task with priority level i, we introduce the concept of a level-i busy period.

Definition

A level-i busy period is a time interval [a,b] within which jobs of priority i or higher are processed throughout [a,b] but no jobs of level i or higher are processed in $(a-\epsilon, a)$ or $(b, b+\epsilon)$ for sufficiently small $\epsilon > 0$.

All response times of the jobs of task i are a part of some level-i busy period, thus we need only identify the phasing of $\tau_1, \ldots, \tau_i$ which creates the longest task i response time. For rate monotonic scheduling, Liu and Layland proved that the longest response time for τ_i occurs when the task phasing creates a critical instant, i.e., $I_1 = \cdots = I_n = 0$, and the longest response time for τ_i is associated with the first job, thus only the deadline of the first job needs to be checked to infer that all deadlines of τ_i are met. This result remains true when $D_i \leq T_i$; however, more care is required if

$D_i > T_i$. Consider the following example due to Ye Ding.

Example 2

Let $n = 2$ with $C_1 = 52, T_1 = 100, D_1 = 110$ and $C_2 = 52, T_2 = 140, D_2 = 154$. Here $D_1/T_1 = D_2/T_2 = 1.1$, so both the rate monotonic and deadline monotonic scheduling algorithms accord highest priority to τ_1. With this priority assignment, the task set is not schedulable. Task 1 will be processed during $[0, 52], [100, 152]$ and $[200, 252]$. The first job of task 2 will be completed at time 156 and misses its deadline at 154. If one were to accord the highest priority to τ_2, then it would be processed during $[0, 52], [140, 192]$ and $[280, 332]$. This means the first job of τ_1 will finish at 104, the second at 208 and the third at 260 completing the busy period. The three task 1 response times are 104, 108 and 60 respectively. Each meets its deadline, thus the task set can be scheduled with this priority assignment. It should also be pointed out that the response time of the second job is longer than for the first, thus the deadlines of all the jobs in the busy period must be checked. If one considered only the first job of task 1, one would draw the erroneous conclusion that $D_1 = 104$ would be sufficient for the task set to be schedulable with this priority ordering. □

Example 3

Consider the case of $n = 2$, $C_1 = 26$, $T_1 = 70$, $C_2 = 62$, $T_2 = 100$, $U = .9914$. Let τ_1 have highest priority in accordance with the rate monotonic algorithm. We ignore the deadlines for the moment. Assuming that both tasks are initiated at time 0, one can find the level−2 busy period to be [0, 696]. The table below gives the response times of τ_2 jobs during this busy period.

Arrival of τ_2 job	Completion Time	Response Time
0	114	114
100	202	102
200	316	116
300	404	104
400	518	118
500	606	106
600	696	96

Task τ_1 will meet all of its deadlines provided $D_1 \geq 26$ or $\Delta_1 = D_1/T_1 \geq .371$. The longest response time for τ_2 occurs for the fifth job of τ_2 during the busy period. Consequently, all deadlines of τ_2 will be met provided $D_2 \geq 118$ or $\Delta_2 = D_2/T_2 \geq 1.18$. The non-monotonic behavior of the response times of task 2 illustrates that all response times must be checked for all jobs processed during the busy period. □

Theorem 1

The longest response time for a job of τ_i occurs during a level-i busy period initiated by a critical instant, $I_1 = \cdots = I_i = 0$.

Proof

Let $[0, b]$ be a level-i busy period, and suppose $I_i > 0$. Only tasks having higher priority than τ_i are processed during $[0, I_i)$, thus if I_i were changed to any value in $[0, I_i)$, each job of τ_i in $[0, b]$ would finish at the same time, thus increasing each of the τ_i response times. The maximum response occurs when I_i is as small as possible, namely $I_i = 0$. If $I_j > 0$ for some $j < i$, then reducing I_j serves to increase (or leave unchanged) the processing requirements of τ_j during $[0, t]$ for every $t \epsilon [0, b)$, thus increasing (or leaving unchanged) the response time of τ_i jobs. The longest response time is achieved by setting I_j to their smallest values, that is, $I_1 = \cdots = I_i = 0$. □

Throughout the rest of this paper we assume the worst case phasing. Under this assumption, it is simple to write an exact criterion for the schedulability of a periodic task set $\tau_1, \ldots, \tau_n$ with tasks listed in fixed priority order. For checking τ_m, we define

$$W_m(k, x) = \min_{t \leq x} \left(\left(\sum_{j=1}^{m-1} C_j \left\lceil \frac{t}{T_j} \right\rceil + kC_m \right) / t \right) .$$

The quantity $\Sigma_{j=1}^{m-1} c_j \lceil \frac{t}{T_j} \rceil + C_m$ gives the total cumulative processor demands made by all jobs of $\tau_1, \ldots, \tau_{m-1}$ and the first job of τ_m during $[0, t]$. Jobs associated with task $\tau_{m+1}, \ldots, \tau_n$ can be ignored, because these jobs have lower priority than τ_m and can be preempted. The first job of τ_m will meet its deadline if and only if this quantity is less than or equal to t for some $t \leq D_n$, because at such a time the processor will have completed all of C_m and all required higher priority work. Indeed, the smallest value of t for which $\Sigma_{j=1}^{m-1} C_j \lceil \frac{t}{T_j} \rceil + C_m = t$ is the time at which this job is completed. In addition, the level-m busy period which started at time 0 will end with the completion of the first job of τ_m if there is no more processing at level m or higher to be done.

These two conditions can be reexpressed as $W_m(1, D_m) \leq 1$ for deadline fulfillment of the first job of τ_m and $W_m(1, T_m) \leq 1$ for the end of the level-m busy period. If $W_m(1, D_m) \leq 1$ but $W_m(1, T_m) > 1$, then the first job of τ_m meets its deadline, but the busy period continues beyond T_m, because there is additional work at level m from later jobs of τ_m yet to be done. One must now consider the second job of τ_m. This can be done by replacing τ_m by τ_m' having computation requirement $2C_m$ and deadline $T_m + D_m$. Thus the second deadline is satisfied if and

only if $W_m(2, T_m + D_m) \leq 1$. If $W_m(2, 2T_m) > 1$, additional jobs of τ_m must be checked. If we define $N_m = \min\{k \mid W_m(k, kT_m) \leq 1\}$, then exactly N_m task τ_m jobs are part of the level-m busy period. Note that N_m is finite, because the total processor utilization is less than 1. Schedulability of τ_m is determined by

$$\max_{k \leq N_m} W_m(k, (k-1)T_m + D_m) \leq 1. \qquad (2.1)$$

One must check that each of the tasks in the task set is schedulable, thus we require (2.1) to hold for each $m, 1 \leq m \leq n$, that is

$$\max_{1 \leq m \leq n} \max_{k \leq N_m} W_m(k, (k-1)T_m + D_m) \leq 1. \qquad (2.2)$$

The criterion (2.2) holds for all fixed priority scheduling algorithms assuming the worst case task phasing. Lui and Layland proved that if $T_i = D_i$, $i \leq i \leq n$, then the rate monotonic priority ordering is optimal among all fixed priority orderings in the sense that if a task set can be scheduled by some fixed priority algorithm it can also be scheduled by the rate monotonic algorithm. Leung and Whitehead [4] proved the inverse deadline (deadline monotonic) ordering is optimal when $D_i \leq T_i$. As shown by the earlier example, the optimality of the deadline or rate monotonic ordering fails when $D_i > T_i$. Shih, Liu and Liu [11] introduced the *modified rate monotonic algorithm* for the case of $D_i > T_i$ and proved some optimality results. In the next section, we will develop results for the rate monotonic algorithm. This will correspond to the deadline monotonic algorithm when $D_i = \Delta T_i$, i.e., when all deadlines are the same constant fraction or multiple of the task periods.

3 Worst Case Utilization Bounds

The criterion given by equation (2.2) to determine if a periodic task set with general deadlines can be scheduled by any particular fixed priority scheduling algorithm. It can be used to create worst case scheduling bounds for the rate monotonic scheduling algorithm for the special case with $D_i = \Delta T_i$, $1 \leq i \leq n$. We assume $T_1 < T_2 < \cdots < T_n$ and assign τ_i higher priority than τ_j if and only if $T_i < T_j$. We first find *full utilization* task sets. These are task sets which meet all deadlines under rate monotonic scheduling, but if the computation requirement of any of the tasks is increased, one of the jobs of the task set will miss its deadline. We then seek the full utilization task set having minimal utilization, $\Sigma_{i=1}^{n} C_i/T_i = U_n^*$. If the utilization of any arbitrary task set consisting of n tasks is kept below U_n^*, then all the deadlines of all the tasks will be met under all task phasings. If the utilization is above U_n^*, then equation (2.2) must be used to determine task set schedulability.

To find the worst case utilization bounds for rate monotonic scheduling for a task set of size n having a common deadline postponement factor Δ, we first simplify by considering task sets which fully utilizes $[0, \Delta T_n]$ under the worst case phasing. Thus the task set will be considered if the first job of τ_n meets its deadline but any increase in any of the computation requirements would cause this deadline to be missed. We will next find the minimum utilization task set from this collection. Finally, we will verify that all the deadlines of all tasks are met during the level-n busy period for this worst case task set. Example 3 gave a task set for which the first response time was not the longest. It turns out that such task sets have relatively high utilization levels. Recall that for Example 3 the task set consisted of two tasks with $C_1 = 26, T_1 = 70, C_2 = 62, T_2 = 100, U = .9914, \Delta = 1.18$. Consider the modified task set with the periods unchanged and $C_1 = 48, C_2 = 22, U = .9057, \Delta = 1.18$. Here, the interval $[0, 118]$ is fully utilized by a task set with smaller utilization. One can reduce the total utilization further by setting $C_1 = 41, T_1 = 77, C_2 = 36, T_2 = 100, U = .8925, \Delta = 1.18$. The latter task set also fully utilizes $[0, 118]$ and all deadlines are met throughout the busy period. Indeed, the first job of τ_2 is completed at time 77 before its period of 100. If one were instead to let $\Delta = 1.14$, corresponding to the response time of the first job of τ_2 in the task set of Example 3, one could modify this task set to $C_1 = 38.5, T_1 = 75.5, C_2 = 37, T_2 = 100, U = .8799, \Delta = 1.14$ and achieve a full utilization task set with far lower utilization.

Liu and Layland found the worst case utilization bounds for $\Delta = 1$ and our analysis follows the same basic pattern as their derivation. The first step is to restrict the period ratios, T_n/T_1. Liu and Layland proved that attention can be restricted to $T_n/T_1 < 2$ for $\Delta = 1$. This result remains true when $\Delta > 1$. If $T_n/T_1 \geq 2$, then one can replace τ_1 by τ_1' with $C_1' = 2C_1, T_1' = 2T_1$ leaving utilization unchanged. The modified task set fully utilizes the processor during $[0, \Delta T_n]$ and some other processing might have to be reduced to guarantee all deadlines are still met. Consequently, attention can be restricted to $T_n/T_1 < 2$. When $\Delta > 1$, we want to impose tighter restrictions on T_n/T_1. Suppose $\Delta T_n = qT_1 + r$ with $q = \lfloor \Delta T_n/T_1 \rfloor$ and $0 \leq r < T_1$. If $\frac{q}{q-1} T_1 \leq T_n$, and we replace (C_1, T_1) by $(\frac{q}{q-1} C_1 , \frac{q}{q-1} T_1)$, the modified task set utilization is unchanged. The modified task set also fully utilizes $[0, \Delta T_n]$, and some deadline might

now be missed because the total processing for task 1 requested in $[0, \Delta T_n]$ changes from $(q+1)C_1$ to $q\frac{q}{q-1}C_1 > (q+1)C_1$.

It is not obvious that the modified task set fully utilizes $[0, \Delta T_n]$. We illustrate why this is true with an example. Suppose $T_n = 1, \frac{1}{2} < T_1 < \frac{2}{3}$ and $\Delta = 2$, let $S = \sum_{i=2}^{n} C_i$. The modified task set consists of τ_1' with $C_1' = \frac{3}{2}C_1$ and $T_1' = \frac{3}{2}T_1$. The original task set filled $[0, C_1]$, $[T_1, T_1 + C_1]$, $[2T_1, 2T_1 + C_1]$ and $[3T_1, 3T_1 + C_1]$ with task 1 processing. Moreover, $[0, 2S + 3C_1]$ is busy with processing from the first three jobs of τ_1 and the first two of $\tau_2, \ldots, \tau_n$. Consequently, $[2S + 3C_1, 3T_1]$ is busy from the third jobs of $\tau_2, \ldots, \tau_n$. Therefore, $[S + \frac{3}{2}C_1, \frac{3}{2}T_1]$ would be busy with the second jobs of $\tau_2, \ldots, \tau_n$.

The total demand for processing is at least as great for the modified task set as for the original task set once $\frac{3}{2}T_1$ is reached. The only interval in question is $[S + \frac{3}{2}C_1, \frac{3}{2}T_1]$, and this has been shown to be busy. This argument can be extended to $\Delta > 2$.

If $\frac{q}{q-1}T_1 \leq T_n$, then the task set can be modified. Consequently we can restrict attention to task sets satisfying $T_n/T_1 < q/(q-1)$. This reduces to the condition $T_n/T_1 < \frac{\Delta+1}{\Delta}$ for Δ an integer and $T_n/T_1 < \frac{\lceil\Delta\rceil}{\lfloor\Delta\rfloor}$ for non-integer $\Delta > 1$.

We will find the following lemma to be useful in calculating the worst case utilization. It first appeared as Lemma 4.3 in [12].

Lemma 2

Let $x > 0$, $y > 1$ and $R_i > 0$, $1 \leq i \leq n$. Define $\mathbf{R} = (R_i, \ldots, R_n)$ and let $H(\mathbf{R}) = \sum_{i=1}^{n} R_i + x/\prod_{i=1}^{n} R_i - (n+1)$ and $\mathbf{S}_y = \{\mathbf{R} \mid R_i \geq 1, \prod_{i=1}^{n} R_i \leq y\}$. Then defining $x^{n/(n+1)} = x_n$

$$\min_{\mathbf{R} \in \mathbf{S}_y} H(\mathbf{R}) = \begin{cases} (n+1)(x^{1/(n+1)} - 1) & \text{if } x_n \leq y \\ n(y^{1/n} - 1) - 1 + x/y & \text{if } y < x_n \end{cases}$$

and

$$\lim_{n \to \infty} \min_{\mathbf{R} \in \mathbf{S}_y} H(\mathbf{R}) = \begin{cases} \log_e x & \text{if } x \leq y \\ \log_e y + (x-y)/y & \text{if } x > y. \end{cases}$$

3.1 The Case of Integer Values of Δ

Let us first consider the case in which Δ is an integer, $\tau_1, \ldots, \tau_n$ fully utilizes $[0, \Delta T_n]$ and $T_n/T_1 < (\Delta+1)/\Delta$. Following Liu and Layland Theorem 4 exactly, the worst case task set is given by

$$\begin{aligned} C_i &= \Delta T_{i+1} - \Delta T_i, \quad 1 \leq i \leq n-1 \\ C_n &= \Delta(\Delta+1)T_1 - \Delta^2 T_n, \end{aligned}$$

with corresponding utilization

$$\begin{aligned} &U_n(\Delta) \\ &= \Delta\left(\tfrac{T_2 - T_1}{T_1} + \cdots + \tfrac{T_n - T_{n-1}}{T_{n-1}} + \tfrac{(\Delta+1)T_1 - \Delta T_n}{T_n}\right) \\ &= \Delta\left(\sum_{i=1}^{n-1} R_i + (\Delta+1)/\prod_{i=1}^{n-1} R_i - n - (\Delta-1)\right). \end{aligned}$$

where $R_i = T_{i+1}/T_i$.

One can minimize this using Lemma 2 with $x = \Delta + 1$ and $y = (\Delta+1)/\Delta$. If $\Delta = 1$, then $x = y$ and $x^{(n-1)/n} < y$. If $\Delta \geq 2$, then $y < x^{(n-1)/n}$, so we have the worst case bound given by

$$n\left((\Delta+1)^{1/n} - 1\right) = n\left(2^{1/n} - 1\right), \Delta = 1,$$

$$\Delta\left((n-1)\left(\left(\frac{\Delta+1}{\Delta}\right)^{1/(n-1)} - 1\right)\right), \Delta = 2, 3, \ldots . \quad (3.1)$$

Letting $n \to \infty$ we derive the asymptotic worst case bound

$$U_\infty(\Delta) = \Delta \log_e\left(\frac{\Delta+1}{\Delta}\right), \Delta = 1, 2, \ldots . \quad (3.2)$$

It is easily checked that all deadlines are met throughout the level-n busy period for this task set. Note that $T_k = \left(\frac{\Delta+1}{\Delta}\right)^{(k-1)/(n-1)}$, consequently $C_n = 0$ and all the deadlines of τ_n are met.

This bound is graphed on Figure 1. One can see that if $\Delta = 2$, the case in which tasks are given an extra period within which to complete processing, the worst case bound increases from .693 to .811. Increasing Δ to 3 raises the worst case bound to .863. Simulation studies on the breakdown utilization for periodic task sets on a uniprocessor allowing deadline postponement show the average case schedulability to be at least 95%, often 100% with $\Delta = 2$. Consequently, it should rarely be necessary to require $\Delta > 2$.

3.2 The Case of Noninteger Values of Δ

The analysis for noninteger values of Δ is surprisingly messy. One would hope that the formula given in (3.2) would apply for non-integer Δ as well as for integer values of Δ. Unfortunately, this is not correct. Some fragmentary results are known. In particular, if $0 \leq \Delta \leq 1$, then Lehoczky and Sha [2] and Peng and Shin [6] derived

$$U_\infty(\Delta) = \begin{cases} \Delta & \text{if } 0 \leq \Delta \leq \frac{1}{2} \\ \log_e(2\Delta) + 1 - \Delta & \text{if } \frac{1}{2} \leq \Delta \leq 1 \end{cases} \quad (3.3)$$

We present the analysis for arbitrary Δ only in the case of a large number of tasks. Let $\lfloor\Delta\rfloor = k$, assume $T_n/T_1 < (k+1)/k$, set $T_n = 1$ and define $C(t)$

= total processing requirements of all tasks with periods $\leq t$. $C(t)$ is a nondecreasing right-continuous function. For finite values of $n, C(t)$ is a step function having jumps at the task periods and jump heights equal to the processing requirements of the various tasks. Letting $n \to \infty$ allows us to consider continuous functions $C(t)$ having a processing requirement density function $C'(t)$, discrete processing requirements and mixtures of these two cases. The cumulative processing requirement function $C(t)$ is similar to a cumulative distribution function used to describe a probability distribution, except $C(1)$ need not equal 1. Let $D = C(1) - \lim_{t \to 1^-} C(t)$. This is positive if and only if there is a discrete processing requirement for the task with longest period 1. From earlier results we know that the worst case task set has periods lying between $\frac{k}{k+1}$ and 1, and $C(\frac{k}{k+1}) = 0$. Let $C(1) = S$, the total processing requirement of the task set. The task set utilization is given by

$$U = \int_{\frac{k}{k+1}}^{1} \frac{dC(t)}{t}, \tag{3.4}$$

where the integral is an ordinary Stieltjes integral.

Condition (2.1) and (2.2) can be rewritten using the function $C(t)$. Since we are considering only the first deadline of the lowest priority task, the relevant condition (2.1) becomes

$$\int_{k/k+1}^{1-} \lceil \frac{t}{s} \rceil \, dC(s) + D \geq t \,, \; 0 \leq t \leq \Delta \tag{3.5}$$

with equality for some $0 \leq t \leq \Delta$. We wish to find a cumulative processing function $C(t)$ which minimizes U given by (3.4) among all $C(t)$ satisfying the constraint (3.5). If all deadlines are met for this cumulative processing function, its corresponding utilization will provide the required worst case bound. Since the goal is to minimize (3.4), one wants to choose $C(t)$ as small as possible. The solution to this minimization problem is very messy, so we first consider a special case, namely $1 < \Delta \leq 3/2$. Let $C(1) \geq 1/2$ be given. It follows that (3.5) is satisfied for $0 \leq t \leq C(1)$. This observation allows us to set $C(t) = 0 \,, \; 0 \leq t \leq C(1)$. This makes (3.5) an equality for $t = C(1)$. For $t > C(1), C(t)$ must grow at least at rate 1 to satisfy (3.5). If, however, one sets $C'(t) = 2$, then the left-hand side of (3.5) will increase at rate 2. Furthermore, if $C'(t) = 2$ for $C(1) \leq t \leq \frac{\Delta}{2}$, this will create a fully utilized processor during $[2C(1), \Delta]$. Setting $C'(t) = 0$ for $\frac{\Delta}{2} \leq t \leq \Delta - C(1)$ bring (3.5) back to an equality condition when $t = \Delta - C(1)$.

To satisfy (3.5) during $[\Delta - C(1), 1)$, we set $C'(t) = 1$. The cumulative processing function constructed thus far satisfies

$$C'(t) = \begin{cases} 2 & \text{if } C(1) \leq t \leq \frac{\Delta}{2} \,, \\ 1 & \text{if } \Delta - C(1) \leq t < 1 \,, \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

Consequently

$$\begin{aligned} \int_{1/2}^{1-} dC(t) &= 2\left(\frac{\Delta}{2} - C(1)\right) + 1 - (\Delta - C(1)) \\ &= 1 - C(1) \end{aligned} \tag{3.7}$$

In addition

$$C(1) = D + \int_{1/2}^{1-} dC(t) = D + 1 - C(1) \,, \tag{3.8}$$

therefore

$$D = 2C(1) - 1. \tag{3.9}$$

We summarize the worst case cumulative processing requirement function

$$C(t) = \begin{cases} 0 & \text{if } t \leq C(1) \\ 2(t - C(1)) & \text{if } C(1) \leq t \leq \frac{\Delta}{2} \\ \Delta - 2C(1) & \text{if } \frac{\Delta}{2} \leq t \leq \Delta - C(1) \\ t - C(1) & \text{if } \Delta - C(1) \leq t < 1 \\ C(1) & \text{if } \Delta = 1 \,. \end{cases} \tag{3.10}$$

This task set distribution corresponds to uniform task periods over $[C(1), \frac{\Delta}{2}]$ with processing requirement of rate 2, uniform periods over $[\Delta - C(1), 1)$ with processing requirement of rate 1 and a lowest priority task with period 1 and processing requirement $2C(1) - 1$. The task set utilization is given by

$$\int_{C(1)}^{\frac{\Delta}{2}} \frac{2}{t} dt + \int_{\Delta - C(1)}^{1} \frac{dt}{t} + \frac{2C(1) - 1}{1} \tag{3.11}$$

$$= 2 \log_e \left(\frac{\Delta}{2C(1)}\right) + \log_e \left(\frac{1}{\Delta - C(1)}\right) + 2C(1) - 1. \tag{3.12}$$

Finally one can optimize over $C(1), \frac{1}{2} \leq C(1) \leq 1$, to find the minimum value in (3.12). The optimum choice of $C(1)$ is the smallest root of the quadratic equation

$$S^2 - (\Delta + \frac{3}{2})S + \Delta = 0. \tag{3.13}$$

It remains to check that all deadlines are met. All processing on the first jobs of all of the tasks is done by $t = C(1)$, all processing on the second jobs is completed by $2C(1)$ and the busy period ends at $t = \Delta$. Consequently, all deadlines are easily met, although any increase in $C(t)$ will cause a deadline to be missed.

The case of general Δ is similar in spirit but even more complicated. We sketch the broad outlines of the

derivation. We again introduce a task set processing requirement function specified by $C(t)$ as before. Here, our earlier discussion shows that $C(t) = 0$ if $t \leq \frac{k}{k+1}$ where $\Delta \in [k, k+1]$. We let $S = C(1)$ and note that S must satisfy $\frac{k}{k+1} \leq S \leq \frac{\Delta}{k+1}$. The minimal processing requirement is determined by applying the full utilization conditions (2.1) and (2.2) first to $t \in [(k+1)S\,,\,\Delta]$ and then to $t \in [kS\,,\,(k+1)S]$. Finally, $C(t)$ is completely specified by putting sufficient weight on tasks with period equal to 1 achieve a total processing requirement equal to S.

To enforce full utilization over $[(k+1)S\,,\,\Delta]$, we require

$$(k+1)S + \int_S^{\frac{t}{k+1}} dC(u) \geq t \qquad \text{for } (k+1)S \leq t \leq \Delta \tag{3.14}$$

or

$$\int_S^t dC(u) \geq (k+1) \qquad \text{for } S \leq t \leq \tfrac{\Delta}{k+1} \tag{3.15}$$

To minimize $C(t)$, we set

$$C'(t) = k+1 \qquad S \leq t \leq \frac{\Delta}{k+1} \tag{3.16}$$

Using (3.16) in a full utilization equation for $t \in [kS\,,\,(k+1)S]$ we find

$$kS + (k+1)\left(\frac{\Delta}{k+1} - S\right) + \int_{\frac{\Delta}{k+1}}^{t/k} dC(t) \geq t\,,\; kS \leq t \leq (k+1)S \tag{3.17}$$

This condition breaks into two distinct candidates for the minimum utilization task set:

Case 1: $S \leq \Delta - k$

The minimum utilization task set is:

$$C'(t) = \begin{cases} k+1 & t \in [S\,,\,\frac{\Delta}{k+1}] \\ 0 & t \in (\frac{\Delta}{k+1}\,,\,1) \end{cases} \tag{3.18}$$

and $(k+2)S - \Delta$ units of processing at period 1. The corresponding utilization is given by

$$(k+1)\log_e(\Delta/S(k+1)) + (k+2)S - \Delta. \tag{3.19}$$

Equation (3.19) is minimized by setting $S = \frac{k+1}{k+2}$ which is feasible if $\Delta > k + 1 - 1/(k+2)$.

Case 2: $\Delta - k \leq S$

The minimum utilization task set is given by

$$C'(t) = \begin{cases} k+1 & t \in [S\,,\,\frac{\Delta}{k+1}] \\ k & t \in [\frac{\Delta - S}{k}\,,\,1] \end{cases} \tag{3.20}$$

and $(k+1)S - k$ units of processing at period 1. The corresponding utilization is given by

$$(k+1)\log_e\left(\frac{\Delta}{S(k+1)}\right) + k\log_e\left(\frac{k}{\Delta - S}\right) + (k+1)S - k. \tag{3.21}$$

Equation (3.21) is minimized by setting S equal to the smallest root of

$$S^2 - (\Delta + (2k+1)/(k+1))S + \Delta = 0. \tag{3.22}$$

Again this task set must be checked to ensure that all deadlines are satisfied. The deadlines of the first jobs are all met by $t = S$. The processor begins to have idle capacity at $t - \Delta$, but any increase in the $C(t)$ function will cause the deadline at Δ to be missed.

We summarize the worst case utilization bounds associated with task sets $\tau_1, \ldots, \tau_n$ with $D_k = \Delta T_k$ for $n \to \infty$. Let $\Delta \in [k, k+1]$, $k = 0, 1, 2, \ldots$. If $k \leq \Delta \leq k + 1 - 1/(k+2)$, then

$$U_\infty^* = (k+1)\log_e(\Delta/S(k+1)) - k\log_e((\Delta - S)/k) + (k+1)S + k \tag{3.23}$$

If $k + 1 - 1/(k+2) \leq \Delta \leq k+1$, then

$$U_\infty^* = (k+1)\log_e((k+2)\Delta/(k+1)^2) + (k+1) - \Delta \tag{3.24}$$

where S is the smallest root of

$$S^2 - S[\Delta + (2k+1)/(k+1)] + \Delta = 0.$$

For example

$$U_\infty^* = \begin{cases} \Delta & \text{if } \Delta \in [0, \frac{1}{2}] \\ \log_e(2\Delta) + 1 - \Delta & \text{if } \Delta \in [\frac{1}{2}, 1] \\ H(\Delta) & \text{if } \Delta \in [1, \frac{5}{3}] \\ 2\log(\frac{3}{4}\Delta) + 2 - \Delta & \text{if } \Delta \in [\frac{5}{3}, 2] \end{cases} \tag{3.25}$$

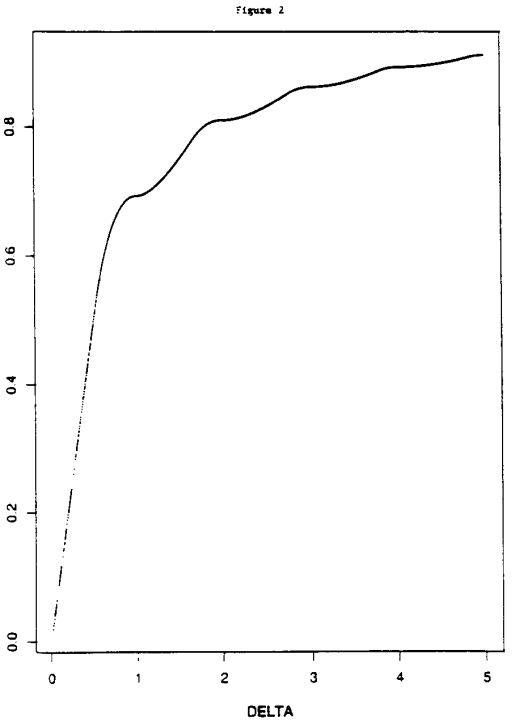
where $H(\Delta) = 2\log_e(\Delta/2S) - \log_e(\Delta - S) + 2S - 1$ and S is the smallest root of $S^2 - (\Delta + \frac{3}{2})S + \Delta = 0$.

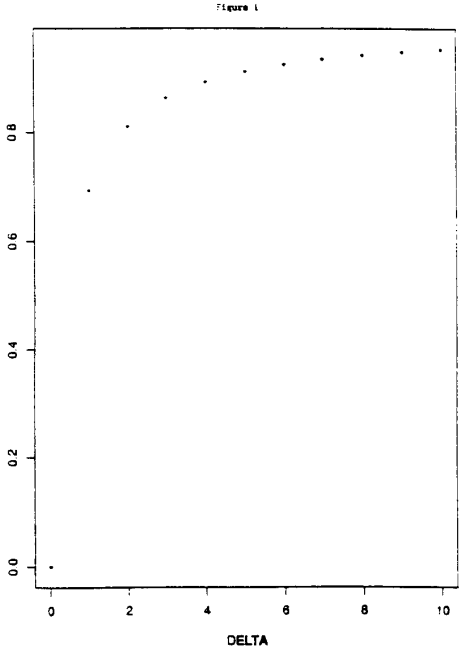
A graph of the worst case utilization bound is given in Figure 2 for $\Delta \in [0, 5]$. A blown up version is given for $\Delta \in [\frac{1}{2}, 2]$ in Figure 3. It is interesting to observe the irregular behavior between the integer arguments. The "S-Shape" behavior suggest that the most significant increases in worst case utilization come in the middle of an interval $[k, k+1]$ rather at the integer values themselves where the curve is flat.

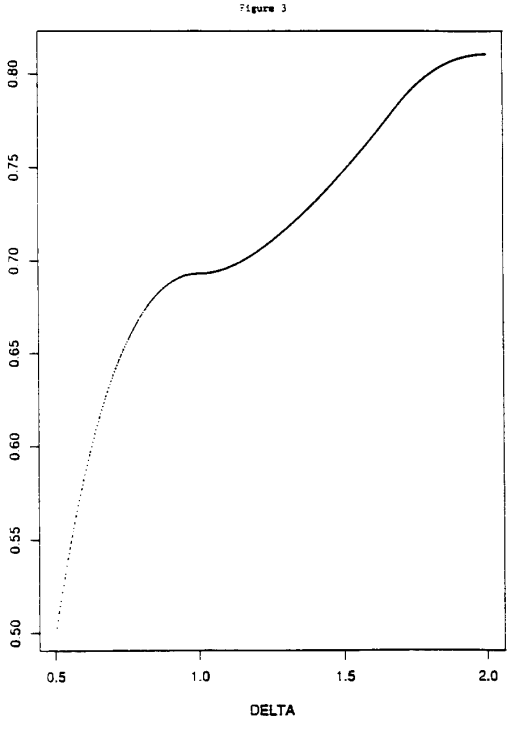
4 Summary

In this paper, we have developed an exact schedulability criterion for the fixed priority scheduling of periodic tasks with arbitrary deadlines. In the case of rate monotonic scheduling, we developed worst case bounds generalizing the original bounds of Liu and Layland in that the tasks are allowed to have deadlines $D_n = \Delta T_n$ for any $\Delta > 0$. The bounds show that when one additional period ($\Delta = 2$) is given to tasks to complete their computation requirement, the worst case schedulable utilization increases from .693 to .811. Simulation studies show that the average schedulable utilization increases from .88 to over .95 and often reaches 1.00.

These worst case bounds are useful in developing a fixed priority scheduling theory for distributed real-time systems. If a distributed task has a sequence of resource requirements and an end-to-end deadline, one can decompose the task into a set of tasks using a single resource and having an artificial intermediate deadline. The standard rate monotonic theory can be applied to determine the schedulability of each resource. Moreover, using the results in this paper, one can adjust the intermediate deadlines and still check schedulability. This adjustment process can lead to better overall schedulability. While this is quite simple and certainly not as powerful as a true understanding of the distributed scheduling problem, it will serve as a good first step to the construction of predictable distributed systems.







REFERENCES

[1] Bettati, R. and Liu, J.W.S., "Algorithms for end-to-end scheduling to meet deadlines," Report No. UIUCDCS-R-90-1594, Department of Computer Science, University of Illinois.

[2] Lehoczky, J.P., and Sha, L., "Performance of real-time bus scheduling algorithms," *ACM Performance Evaluation Review*, **14**, 1986.

[3] Lehoczky, J.P., Sha, L. and Ding, Y., "The rate monotonic scheduling algorithm: exact characterization and average case behavior," *IEEE Real-Time Systems Symposium*, 1989, 166–171.

[4] Leung, J. and Whitehead, J., "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance Evaluation*, **2**, 1982, 237–50

[5] Liu, C.L. and Layland, J.W., "Scheduling algorithms for multiprogramming in a hard real-time environment," *JACM*, **20**, 1973, 460–61.

[6] Peng, D-T. and Shin, K.G., "A new performance measure for scheduling independent real-time tasks," Technical Report, Real-Time Computing Laboratory, University of Michigan, 1989.

[7] Rajkumar, R., "Task synchronization in real-time systems," Ph. D. Dissertation, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1989.

[8] Rajkumar, R., Sha, L. and Lehoczky, J.P., "On countering the effects of cycle-stealing in a hard real-time environment," *IEEE Real-Time Systems Symposium*, 1987, 2–11.

[9] Sha, L. and Goodenough, J., "Real-time scheduling theory and Ada," *Computer*, **23** No. 4, 1990, 53–62.

[10] Sha, L., Lehoczky, J.P. and Rajkumar, R., "Solutions for some practical problems in prioritized preemptive scheduling," *IEEE Real-Time Systems Symposium*, 1986.

[11] Shih, W.K., Liu, J.W.S. and Liu, C.L., "Scheduling periodic jobs with deferred deadlines," Report No. UIUCDCS-R-90-1593, University of Illinois, 1990.

[12] Strosnider, J., Lehoczky, J.P. and Sha, L., "The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments," Technical Report, Carnegie Mellon University, 1989.