An Analysis of Fixed-Priority Schedulability on a Multiprocessor TR-050201

Theodore P. Baker
Department of Computer Science
Florida State University
Tallahassee, FL 32306-4530
e-mail: baker@cs.fsu.edu

Abstract.

A new feasibility test for preemptive scheduling of periodic or sporadic real-time tasks on a single-queue m-server system allows for arbitrary fixed task priorities and arbitrary deadlines. For the special case when deadline equals period and priorities are rate monotonic, any set of tasks with maximum individual task utilization $u_{\max}$ is feasible if the total utilization does not exceed $m(1 - u_{\max})/2 + u_{\max}$.

Keywords: deadline monotonic, fixed priority, multiprocessor, rate monotonic, real time, scheduling, symmetric multiprocessing, utilization bound

1. Introduction

Starting at least as early as the Safeguard anti-ballistic missile system[3] and continuing up to the present[15, 11], high performance real-time embedded systems have relied on multiprocessor architectures. With the trend toward multi-core architectures in the current and next generation of microprocessors[6, 5], embedded applications of multiprocessors are likely to become much more common.

The understanding of real-time multiprocessor scheduling has lagged behind that of single-processor scheduling. Over the three decades since Liu and Layland's 1973 seminal analysis of rate monotonic and deadline scheduling[7], the theory of fixed-priority single-processor scheduling has been refined, extended, and generalized, to the point that it is now very well understood [14]. During those same decades comparatively little attention was paid to the possibility of extending the analysis to multiple processors.

Much of the analysis of multiprocessor scheduling that has been done has focussed on a partitioned model, in which tasks are assigned statically to processors[4, 12, 8, 9]. The alternative, global scheduling,

was shown in 1978 by Dhall and Liu[4] to have very poor worst-case performance. A task set may have utilization arbitrarily close to 1 and still not be schedulable on m processors using rate monotonic or earliest-deadline-first scheduling. By comparison, even though optimal partitioning is NP complete, heuristic partitioning algorithms can do much better. For example, using rate monotonic local scheduling and a simple first-fit-decreasing partitioning heuristic, a utilization level of at least $m(2^{1/2}-1)$ is always feasible for an m-processor system[12]. This is not only much better than the worst case for global scheduling; is is not far from the limit of $(m+1)/2$ which Andersson *et al.* showed applies to all fixed-job-priority multiprocessor scheduling algorithms, partitioned or not[1].

Recently, progress has been made in understanding global multiprocessor scheduling, based on a re-evaluation of Dhall's result. Dhall worst-case example has two kinds of tasks: "heavy" ones, with high ratio of computation time to deadline, and "light" ones, with low ratio of computation time to deadline. It is the mixing of those two kinds of tasks that causes a problem. A scheduling policy that segregates the heavy tasks from the light ones, on disjoint sets of CPU's, would have no problem with Dhall's example. Examination of further examples leads one to conjecture that such a hybrid scheduling policy would not miss any deadlines until a fairly high level of CPU utilization is achieved, and might even permit the use of simple utilization-based schedulability tests.

In 2001 Andersson, Baruah, and Jonsson[1] examined the preemptive scheduling of periodic tasks on multiprocessors, and showed that any system of independent periodic tasks for which the utilization of every individual task is at most $m/(3m-2)$ can be scheduled successfully on m processors using rate monotonic scheduling if the total utilization is at most $m^2/(3m-1)$. Baruah and Goossens[2] proved a similar result, showing that a total utilization of at least $m/3$ can be achieved if the individual task utilizations do not exceed 1/3. Andersson, Baruah, and Johnsson proposed a hybrid scheduling algorithm, called RM-US$[m/(3m-2)]$, which gives higher priority to tasks with utilizations above $m/(3m-2)$, that is able to successfully schedule *any* set of independent periodic tasks with total utilization up to $m^2/(3m-1)$.

This paper further advances the theoretical understanding of global fixed-priority multiprocessor schedulability. The main contribution is a new analysis concept, called the $(\mu, k-1)$ *busy interval*. By analyzing the workload of a $(\mu, k-1)$ busy interval one can relax several of the assumptions made in the prior analyses cited above, and derive a more general schedulability test. The principal consequences are:

1. Applicability to all fixed priority assignments, rather than just rate monotonic.

2. Applicability to tasks with arbitrary deadlines, rather than just those where deadline equals period.

3. Applicability of the RM-US hybrid model to systems with higher utilization levels, by setting the cut-off between "heavy" and "light" at any desired utilization level, rather than just 1/3.

These results provide a theoretical means of verifying the feasibility of task sets on multiprocessor systems that make use of global scheduling, at non-trivial utilization levels and without arbitrary constraints on priorities and deadlines. By reducing the gap in guaranteed-feasible utilization levels between the partitioned and global scheduling approaches, they also suggest that perhaps global scheduling should be given more serious consideration for real-time systems. (The broader question of whether global scheduling is preferable to partitioned scheduling, either theoretically or pragmatically, remains a subject for further research.)

The rest of the paper presents the derivation of the theory. Section 2 defines the problem formally, and outlines the overall approach. Section 3 derives a lower bound on the workload contributions of competing tasks in an interval where a task misses a deadline. Section 4 defines the notion of $(\mu, k-1)$ busy interval and derives an upper bound on the workload contributions of competing tasks in any such interval. Section 5 combines the upper and lower bounds on workload to obtain schedulability tests, including a utilization bound for rate monotonic scheduling, and compares the performance of those tests on some randomly generated task sets. Section 6 shows that the utilization bound is not tight. Section 7 reviews in more detail the connections to prior work, including an application to determining an "optimal" value of λ for RM-US[λ] scheduling. Section 8 summarizes and concludes.

2. Definition of the Problem

Suppose one is given a set of N simple independent sporadic tasks $\tau_1, \ldots, \tau_N$, where each task τ_i has minimum inter-release time (called *period* for short) T_i, worst case computation time c_i, and relative deadline d_i, where $c_i \leq d_i$, and $c_i \leq T_i$. Each task generates a sequence of *jobs*, each of whose release time is separated from that of its predecessor by at least T_i. No special assumptions are made about the first release time of each task.

Time is represented by the domain of rational numbers. Square brackets and parentheses are used to distinguish whether time intervals include their endpoints. For example the time interval $[t_1, t_2)$ contains

the time values greater than or equal to t_1 and less than t_2. All of the intervals $[t_1, t_2]$, $[t_1, t_2)$, $(t_1, t_2]$ and (t_1, t_2) are said to be of *length* $t_2 - t_1$.

The objective of this paper is to formulate a simple test for schedulability of a task set, expressed in terms of the periods, deadlines, and worst-case computation times of the tasks, such that if the test is passed no deadlines will be missed. The problem is approached by analyzing the minimum processor load that is needed over an interval of time to cause a missed deadline.

DEFINITION 1. Let $S = \{\tau_1, \ldots, \tau_N\}$ be any task set. A *release-time assignment* for S is a function $r : \{1, \ldots, N\} \times \mathbb{N} \to \mathit{Time}$. The value $r(i, j)$ is interpreted as the release time of the jth job of τ_i. All task releases are required to be separated by at least the task period, *i.e.*, $r(i, j) + T_i \leq r(i, j + 1)$.

The jobs of each task must be executed sequentially, and all jobs are scheduled on m identical processors according to a global preemptive fixed-priority policy, where task τ_i always has priority over task τ_{i+1}. Here "global" means that jobs are assigned to processors dynamically, so that whenever there are m or fewer jobs ready they will all be executing, and whenever there are more than m jobs ready there will be m jobs executing, all with priority higher than or equal to the priorities of the jobs that are not executing.

Since it is not the intent of this paper to compare the efficiency of global versus partitioned scheduling, and for the sake of simplicity of analysis, the abstract computational model does not include any execution time penalty for preemption or for interprocessor task migration. In a real system there will be some penalty for interrupting the execution of a processor, and there may be some penalty for reloading the cache of the new processor with instructions/data of the task (if the cache of the old processor has not already been overwritten by intervening other tasks). The cache penalty is variable, and in practical applications will add to the margin of error in schedulability analysis that already exists due to variability in task execution times and cache effects due to preemption on a single processor. However, if one prefers to take the task migration penalty into account explicitly, the analysis presented in this paper could be modified to account for the worst-case preemption and migration costs by adding an appropriately chosen constant to the execution time c_i of each task.

DEFINITION 2. Given a task set S and a release-time assignment r, the *work* W_i done by task τ_i over a time interval $[t - \Delta, t)$ is the actual amount of computation time that is used by jobs of τ_i in the interval, and the *load* due to task τ_i is W_i/Δ. (This definition differs slightly from some other work on

demand analysis by counting only work that is *actually executed* in the interval.) Wherever the notation W_i is used the release time assignment and time interval will be clear from context.

For any integer k, $1 \leq k \leq N$, the *level k work* with release-time assignment r is $\sum_{i \leq k} W_i$, and the *level k load* is $\sum_{i \leq k} W_i / \Delta$.

DEFINITION 3. For a given task set and release time assignment, a *first missed deadline* is a time t at which some task misses a deadline and such that no task misses a deadline before t. If t is a first missed deadline and task τ_k misses a deadline at time t, then t is em a first missed deadline of task τ_k.

If a task set misses a deadline for some release time assignment then it has a unique first missed deadline for that release time assignment. If one can find a lower bound on the processor load over an interval leading up to every first missed deadline, and one can guarantee that a given set of tasks could not possibly generate so much load in such an interval, that would be sufficient to serve as a proof of schedulability.

3. Lower Bound on Load

One can establish a lower bound on the level $k-1$ load of the interval ending at a first missed deadline of task τ_k and starting with the release time of the corresponding job, by observing that, since the job does not complete by the end of the interval, the lengths of all the subintervals in which the job does not execute must exceed its slack time. This fact is well known, and is the basis of the prior analysis by Phillips *et al.*[13] and others. It is illustrated Figure 1 for the case where $m = 3$ and $d_k \leq T_k$. The diagonally shaded rectangles indicate blocks of time during which τ_k executes. The dotted rectangles indicate times during which all m processors must be busy executing other jobs that contribute to the load for this interval. It is easy to see that the total level $k-1$ work of the interval $[t - d_k, t)$ must be at least $m(d_k - x)$, where $x \leq c_k$ is the amount of time that τ_k executes in the interval.

To allow for the possibility that $d_k \geq T_k$, one needs to consider intervals that may include more than one job of τ_k. Figure 2 shows the release times and deadlines of two such jobs $\tau_{k,1}$ and $\tau_{k,2}$. The job $\tau_{k,1}$ is delayed by two blocks of higher priority interference. This interference is not enough to cause $\tau_{k,1}$ to miss its deadline, but (because jobs of the same task must be executed sequentially) it delays the start of $\tau_{k,2}$ enough to cause that job to miss its deadline.

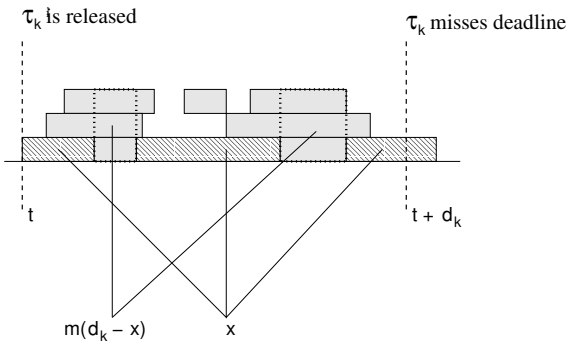

Figure 1. All processors must be busy whenever τ_k is not executing.

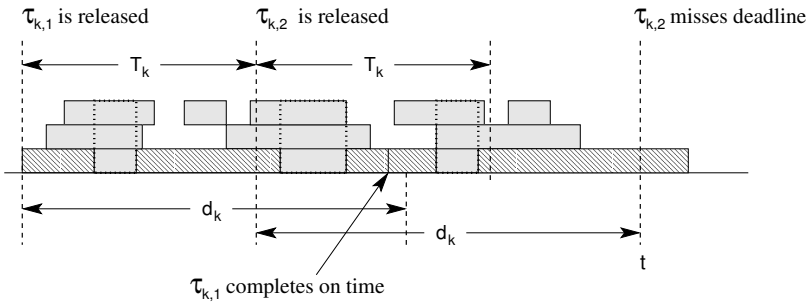

Figure 2. More than one job of τ_k may execute in a τ_k-busy interval if $d_k \geq T_k$.

DEFINITION 4. A job is *backlogged* at a time t if it is released before time t and has nonzero execution time remaining at time t. A task is backlogged if it has a backlogged job.

For any task τ_k, a time interval $[t', t)$ is *τ_k-busy* if there are backlogged jobs of τ_k continually throughout the interval (t', t).

LEMMA 5. *For a given a set of tasks S and a given release-time assignment r, if τ_k is backlogged at time t then there is a unique τ_k-busy interval $[t - \Delta, t)$ such that:*

1. *There are no backlogged jobs of τ_k at time $t - \Delta$.*
2. *There is a job of τ_k released at time $t - \Delta$.*

Proof. Let t'' be the latest time before t at which τ_k is not backlogged. There must be such a time, since τ_k is not backlogged at the system start time. Let t' be the next release time of τ_k on or after t''. There must be such a time, since τ_k is backlogged at time t. The interval $[t', t)$ satisfies the definition of τ_k-busy. The value t' is unique, since τ_k is not backlogged at time t' and τ_k is backlogged at all times from t' through t.

□

The unique interval guaranteed by Lemma 5 is called the *maximal* τ_k-busy interval ending at t. If $d_k \leq T_k$ this interval cannot be longer than d_k, but if $d_k > T_k$ the interval may be arbitrarily longer than d_k. For example, consider a system of $m+1$ tasks, such that $T_1 = \cdots = T_{m+1} = 1$, $d_1 = \cdots = d_m = 1$, $d_{m+1} = 2$, $c_1 = \cdots = c_m = \epsilon$, and $c_{m+1} = 1$. The first m tasks create a block of interference of duration ϵ for every release of τ_{m+1}, so that each successive job of τ_{m+1} completes later by ϵ. The first job of task τ_{m+1} to miss its deadline will be the jth job, where j is the least integer greater than $1/\epsilon$. It will miss its deadline at time $jT_{m+1} + d_{m+1} = j + 1$, since $j\epsilon > d_{m+1} - T_{m+1} = 1$. One can make j arbitrarily large by choosing ϵ small enough.

LEMMA 6 (lower bound on load). *If t is a first missed deadline of τ_k then the maximal τ_k-busy interval $[t - \Delta, t)$ has level $(k-1)$ load greater than $m(1 - \frac{c_k}{\min\{T_k, d_k\}})$.*

Proof. Let x be the amount of time τ_k executes in the interval $[t - \Delta, t)$. Since τ_k is continually backlogged over the interval, the only times that τ_k does not execute are the times that all m processors are executing jobs of tasks with higher priority than τ_k. The level $(k-1)$ work of $[t - \Delta, t)$ must be at least equal to the work of these higher priority tasks, *i.e.*,

$$\sum_{i<k} \frac{W_i}{\Delta} \geq \frac{m(\Delta - x)}{\Delta} = m(1 - \frac{x}{\Delta}) \tag{1}$$

Let j be the number of jobs of τ_k that execute in the interval. The amount x of time that τ_k executes in the interval is bounded as follows:

$$x < jc_k \tag{2}$$

Since τ_k is not backlogged at the start of the interval, all of the j jobs are released on or after $t - \Delta$ and not later than $t - d_k$ (because t is a missed deadline), and the release times are separated by at least T_k, so:

$$(j-1)T_k + d_k \leq \Delta \tag{3}$$

It follows from (1-3) that

$$\sum_{i<k} \frac{W_i}{\Delta} > m(1 - \frac{jc_k}{(j-1)T_k + d_k})$$

Let $f : \mathbb{N} \rightarrow \mathit{Time}$ be the function defined by $f(j) = \frac{jc_k}{(j-1)T_k+d_k}$. The objective here is to find an upper bound for $f(j)$, subject to the available constraints. There are two cases to consider:

1. If $d_k \leq T_k$ then $\frac{c_k}{d_k} \geq \frac{c_k}{T_k}$, f is non-increasing with respect to j, and since $j \geq 1$,

$$f(j) \leq f(1) = \frac{c_k}{d_k}$$

2. If $d_k > T_k$ then $\frac{c_k}{d_k} < \frac{c_k}{T_k}$, f is increasing with respect to j, and

$$f(j) \leq \lim_{j \rightarrow \infty} f(j) = \frac{c_k}{T_k}$$

Putting the above two cases together, it follows that:

$$\sum_{i<k} \frac{W_i}{\Delta} > m(1 - \frac{c_k}{\min\{d_k, T_k\}})$$

□

4. Upper Bound on Load

The next step is to derive an upper bound on the level $(k-1)$ load of an interval leading up to a first missed deadline of task τ_k. If one can find such an upper bound β_k, it will follow that schedulability of a task system can be guaranteed by checking that β_k is less than the minimum level $(k-1)$ load needed to cause a missed deadline of τ_k. The upper bound will be defined as the sum of individual upper bounds on the load due to each task that can preempt τ_k in the interval.

The work W_i done by task τ_i in any interval $[t - \Delta, t)$ is clearly bounded by the length Δ of the interval and may include:

1. a portion of the execution times of zero or more jobs that are released before $t - \Delta$ but are unable to complete by that time, which are called *carried-in* jobs;

2. the full execution times c_i of zero or more jobs that are released on or after time $t - \Delta$ and complete by time t;

3. a portion of the execution time of at most one that is released at some time $t - \delta$, $0 < \delta \leq \Delta$, but is unable to complete by time t.

To bound the size of the carried-in contribution of τ_i the maximal τ_k-busy interval is extended downward as far as possible while still maintaining the level $(k-1)$ load that caused τ_k to miss its deadline at time t.

DEFINITION 7. A time interval $[t', t)$ is $(\mu, k-1)$ *busy* if the level $(k-1)$ load is greater than μ. It is a *maximal* $(\mu, k-1)$ busy interval if it is $(\mu, k-1)$ busy and there is no $t'' < t'$ such that $[t'', t)$ is also $(\mu, k-1)$ busy.

LEMMA 8. *If t is a first missed deadline of τ_k and $0 < \mu \leq m(1 - \frac{c_k}{\min\{T_k, d_k\}})$ then there is a unique maximal $(\mu, k-1)$ busy interval $[t - \hat{\Delta}, t)$, and $\hat{\Delta} \geq d_k$.*

Proof. By Lemma 5, there is a unique maximal τ_k-busy interval $[t', t)$. Since t is a missed deadline for τ_k, the length of this τ_k-busy interval is at least d_k. By Lemma 6, the level $(k-1)$ load of this interval is greater than μ. Therefore, the set of all starting points $t'' < t'$ of $(\mu, k-1)$ busy intervals $[t'', t)$ is non-empty. This set must have a minimal member, since there are no backlogged jobs at the start time of the system. Let $\hat{\Delta} = t - t''$ for this minimum value t'' and the lemma is satisfied.

□

DEFINITION 9. Given a task set S, a release-time assignment r, a task τ_k that has a first missed deadline at time t, and a value μ, the unique interval $[t - \hat{\Delta}, t)$ that is guaranteed by Lemma 8 is called *the $(\mu, k-1)$ busy interval of τ_k*. From this point on, let $[t - \hat{\Delta}, t)$ denote such an interval.

The next step in the analysis is to find an upper bound on the load $\frac{W_i}{\hat{\Delta}}$ due to each task τ_i in a $(\mu, k-1)$ busy interval. The only interesting cases are those where τ_i makes a contribution to the the level $(k-1)$, that is, where $i < k$.

LEMMA 10 (upper bound on load). *If t is a first missed deadline of task τ_k, $0 < \mu \leq m(1 - \frac{c_k}{\min\{T_k, d_k\}})$, and $[t - \hat{\Delta}, t)$ is the corresponding $(\mu, k-1)$ busy interval, then the contribution $W_i/\hat{\Delta}$ of each task τ_i, $i < k$, to the load of the interval is strictly bounded above by the function $\beta_{\mu,k}(i)$ defined by Table I.*

Table I. Cases for $\beta_{\mu,k}(i)$, the upper bound on $\frac{W_i}{\hat{\Delta}}$.

Case	$\beta_{\mu,k}(i)$
$\frac{m-\mu}{m-1} \geq \frac{c_i}{T_i}$	$\frac{c_i}{T_i}(1 + \frac{T_i - c_i}{d_k})$
$\frac{m-\mu}{m-1} < \frac{c_i}{T_i}$	$\frac{c_i}{T_i}(1 + \frac{T_i - c_i}{d_k}) + \frac{d_i}{d_k}(\frac{c_i}{T_i} - \frac{m-\mu}{m-1})$

Proof.

Let j be the number of jobs of τ_i that execute in the interval. If $j = 0$ the lemma is satisfied trivially. Therefore, it is only necessary to consider the case where $j \geq 1$.

Let $\epsilon > 0$ be the amount of time that the last of these j jobs of τ_i executes in the interval, and let $t - \delta$ be the release time of this job. Observe that $\epsilon \leq \delta$ and $\epsilon \leq c_i$.

Let $t - \hat{\Delta} - \phi$ be the release time of the first job of τ_i that is released before $t - \hat{\Delta}$ and executes in the interval, if such exists; otherwise, let $\phi = 0$.

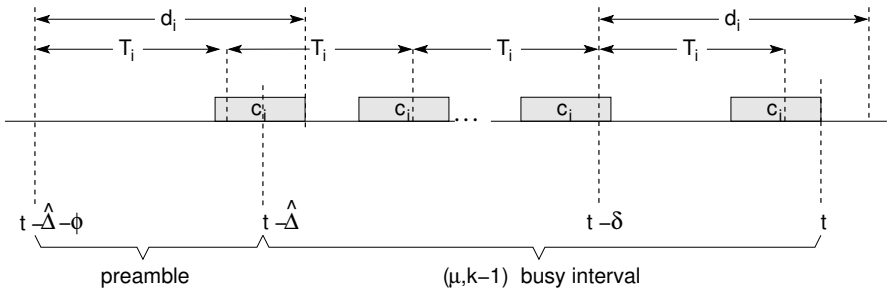

Figure 3. Preamble of $(\mu, k-1)$ busy interval.

If $\phi > 0$ the interval $[t - \hat{\Delta} - \phi, t - \hat{\Delta})$ is non-empty and must be τ_i-busy. Call this the *preamble* with respect to t_i of $[t - \hat{\Delta}, t)$. In this case reasoning similar to that of Lemma 6 can be used to bound the amount of work that τ_i may carry from the preamble into $[t - \hat{\Delta}, t)$, as follows.

Let x be the total amount of time spent executing jobs of τ_i in the preamble and let y be the sum of the lengths of all the subintervals within the preamble where all m processors are simultaneously executing jobs that preempt τ_i. Since τ_i can execute when and only when there are less than m processors executing jobs that preempt τ_i, $y = \phi - x$. It follows that the total amount of level i work executed in the preamble must be at least $my + x$. Putting this together with the fact that the interval $[t - \hat{\Delta}, t)$ is

$(\mu, k-1)$ busy, one can conclude that the level $(k-1)$ work of the entire interval $[t - \hat{\Delta} - \phi, t)$ is at least $\mu\hat{\Delta} + my + x$.

By the definition of maximal $(\mu, k-1)$ busy, the level $(k-1)$ load of the interval $[t - \hat{\Delta} - \phi, t)$ is less than μ. It follows that if $\phi > 0$

$$\begin{aligned} \mu\hat{\Delta} + my + x &< \mu(\phi + \hat{\Delta}) \\ m(\phi - x) + x = my + x &< \mu\phi \\ x(1 - m) &> \phi(\mu - m) \\ x &\geq \frac{m - \mu}{m - 1}\phi \end{aligned}$$

If $\phi = 0$ the preamble is empty and so it followss that in all cases $x \geq \frac{m-\mu}{m-1}\phi$.

By subtracting out $\frac{m-\mu}{m-1}\phi$ as a lower bound on the work x of τ_i that is done in the preamble, one can obtain the following bound on the load due to τ_i in the $(\mu, k-1)$ busy interval $[t - \hat{\Delta}, t)$:

$$\frac{W_i}{\hat{\Delta}} \leq \frac{c_i(j-1) + \epsilon - \frac{m-\mu}{m-1}\phi}{\hat{\Delta}} \tag{4}$$

Because of the minimum separation constraint,

$$\begin{aligned} (j-1)T_i + \delta &\leq \hat{\Delta} + \phi \\ (j-1) &\leq \frac{\hat{\Delta} + \phi - \delta}{T_i} \end{aligned}$$

and so

$$\frac{W_i}{\hat{\Delta}} \leq \frac{c_i\frac{\hat{\Delta}+\phi-\delta}{T_i} + \epsilon - \frac{m-\mu}{m-1}\phi}{\hat{\Delta}}$$

The expression on the right of the inequality above is increasing with respect to ϵ. Since $\epsilon \leq c_i$, the value of the expression is never greater than when $\epsilon = c_i$.

The same expression is decreasing with respect to δ. By definition, $\delta \geq \epsilon$. It follows that the maximum value of the expression on the right side of the inequality is never greater than when $\delta = \epsilon = c_i$.

$$\frac{W_i}{\hat{\Delta}} \leq \frac{c_i\frac{\hat{\Delta}+\phi-c_i}{T_i} + c_i - \frac{m-\mu}{m-1}\phi}{\hat{\Delta}} \tag{5}$$

$$= \frac{c_i}{T_i}(1 + \frac{T_i - c_i}{\hat{\Delta}}) + \frac{\phi(\frac{c_i}{T_i} - \frac{m-\mu}{m-1})}{\hat{\Delta}} \tag{6}$$

The value of the expression on the right side of the inequality above may be increasing or decreasing with respect to ϕ, depending on whether $\frac{c_i}{T_i} \leq \frac{m-\mu}{m-1}$.

Case 1: If $\frac{c_i}{T_i} \leq \frac{m-\mu}{m-1}$ the value of the expression on the right of inequality (6) is non-increasing with respect to ϕ, and since $\phi \geq 0$, the global maximum is achieved when $\phi = 0$. It follows that

$$\frac{W_i}{\hat{\Delta}} < \frac{c_i}{T_i}(1 + \frac{T_i - c_i}{\hat{\Delta}}) \leq \frac{c_i}{T_i}(1 + \frac{T_i - c_i}{d_k}) \tag{7}$$

Case 2: If $\frac{c_i}{T_i} > \frac{m-\mu}{m-1}$ the value of the expression on the right of inequality (6) is increasing with respect to ϕ. Since there are no missed deadlines prior to t and the job released at time $t - \hat{\Delta} - \phi$ does not complete by $t - \hat{\Delta}$, $\phi < d_i$. It follows that

$$\frac{W_i}{\hat{\Delta}} < \frac{c_i}{T_i}(1 + \frac{T_i - c_i}{\hat{\Delta}}) + \frac{d_i(\frac{c_i}{T_i} - \frac{m-\mu}{m-1})}{\hat{\Delta}}$$

Since $\frac{c_i}{T_i} > \frac{m-\mu}{m-1}$ and $\hat{\Delta} \geq d_k$,

$$\frac{W_i}{\hat{\Delta}} < \frac{c_i}{T_i}(1 + \frac{T_i - c_i}{d_k}) + \frac{d_i(\frac{c_i}{T_i} - \frac{m-\mu}{m-1})}{d_k} \tag{8}$$

The upper bounds for $\frac{W_i}{\hat{\Delta}}$ derived in each of the cases above correspond to the definition of $\beta_{\mu,k}(i)$ in Table I.

□

5. Schedulability Tests

Based on the above analysis, one can now prove the following theorem, which provides a sufficient condition for schedulability.

THEOREM 11 ($O(N^3)$ schedulability test). *A set of sporadic tasks $S = \{\tau_1, \ldots, \tau_N\}$ is schedulable on m processors using preemptive fixed-priority scheduling if, for every task τ_k, $k = m+1, \ldots, N$, there exists a positive value $\mu \leq m(1 - \frac{c_k}{\min\{T_k, d_k\}})$ such that*

$$\sum_{i=1}^{k-1} \beta_{\mu,k}(i) \leq \mu \tag{9}$$

where $\beta_{\mu,k}(i)$ is as defined in Table I.

Proof. The proof is by contradiction. Suppose there are a task set S and a release time assignment r for which some task τ_k has a first missed deadline at time t. By the priority ordering, $k > m$. Let $[t - \hat{\Delta}, t)$ be the $(\mu, k-1)$ busy interval guaranteed by Lemma 8. By definition of $(\mu, k-1)$ busy,

$$\sum_{i=1}^{k-1} \frac{W_i}{\hat{\Delta}} > \mu$$

By Lemma 10, $\frac{W_i}{\hat{\Delta}} \leq \beta_{\mu,k}(i)$, for $i = 1, \ldots, k-1$. It follows that

$$\sum_{i=1}^{k-1} \beta_{\mu,k}(i) \geq \sum_{i=1}^{k-1} \frac{W_i}{\hat{\Delta}} > \mu$$

The above is a contradiction of (9).

□

To use the condition above as a schedulability test, it might seem necessary to consider all possible values of μ for each k. On the contrary, the only values of μ that need to be considered are the upper bound and the points at which the function $\beta_{\mu,k}(i)$ is discontinuous with respect to the parameter μ. That is, at the points

$$\mu_i = m - \frac{c_i}{T_i}(m-1)$$

for $i = 1, \ldots, k$, and

$$\mu_{\max} = m(1 - \frac{c_k}{\min\{T_k, d_k\}})$$

The computational complexity of checking (9) for all such values of of μ for each value of k is $O(N^3)$. Therefore, it is referred to here as the $O(N^3)$ schedulability test. If one is willing to sacrifice some precision for a faster test, one can check fewer values of μ, resulting in an $O(N^2)$ schedulability test. Any one of the values of μ checked in the $O(N^3)$ test would be sufficient, but using the largest possible value gives better results in most cases.

COROLLARY 12 ($O(N^2)$ test). *A set of sporadic tasks $\tau_1, \ldots, \tau_N$ is schedulable on m processors if for every task τ_k, $k = m+1, \ldots, N$,*

$$\sum_{i=1}^{k-1} \frac{c_i}{T_i}(1 + \frac{T_i - c_i}{d_k}) \leq m(1 - \lambda_k) \tag{10}$$

where $\lambda_k = \frac{c_k}{\min\{T_k, d_k\}}$.

Proof.

The proof is by application of Theorem 11. Let $\mu = m(1 - \lambda_k)$. It follows that $\frac{m-\mu}{m-1} > \lambda_k$. Therefore, if condition (10) is satisfied then

$$\sum_{i=1}^{k-1} \beta_{\mu,k}(i) = \sum_{i=1}^{k-1} \frac{c_i}{T_i}(1 + \frac{T_i - c_i}{d_k})$$

If this condition is satisfied for $k = m + 1, \ldots, N$ then, by Theorem 11, the task set must be schedulable.

□

If one is willing to sacrifice some more precision for a stil faster test, there is an $O(N)$.

COROLLARY 13 ($O(N)$ schedulability test). *A set of sporadic tasks $\tau_1, \ldots, \tau_N$ is schedulable on m processors if*

$$\sum_{i=1}^{N-1} \frac{c_i}{T_i}(1 + \frac{T_i - c_i}{d_{min}}) \ \leq \ m(1 - \lambda_{\max}) \tag{11}$$

where $\lambda_{\max} = \max\{\frac{c_i}{\min\{T_i, d_i\}} \ \mid \ i = 1, \ldots, N\}$, *and* $d_{min} = \min\{d_k \ \mid \ i = 1, \ldots, N\}$.

Proof.

Corollary 13 is proved by application of Theorem 11. Let $\mu = m(1 - \lambda_{\max})$. It follows that $\frac{m-\mu}{m-1} > \frac{c_k}{\min\{T_k, d_k\}}$ for $k = 1, \ldots, N$. Therefore,

$$\sum_{i=1}^{k-1} \beta_{\mu,k}(i) = \sum_{i=1}^{k-1} \frac{c_i}{T_i}(1 + \frac{T_i - c_i}{d_k})$$

Since $k \leq N$ and $d_k \leq d_{\min}$,

$$\sum_{i=1}^{k-1} \beta_{\mu,k}(i) \leq \sum_{i=1}^{N-1} \frac{c_i}{T_i}(1 + \frac{T_i - c_i}{d_{\min}})$$

If condition (11) is satisfied then, for all $k = m + 1, \ldots, N$,

$$\sum_{i=1}^{k} \beta_{\mu,k}(i) \leq m(1 - \lambda_{\max}) \leq m(1 - \frac{c_k}{\min\{T_k, d_k\}})$$

By Theorem 11, the task set must be schedulable.

□

If one assumes the deadline of each task is equal to its period Theorem 11 also yields a lower bound on the minimum achievable utilization for rate monotonic scheduling.

COROLLARY 14 (utilization test). *A set of sporadic tasks, all with deadline equal to period, is guaranteed to be schedulable on m processors using preemptive rate monotonic scheduling if*

$$\sum_{i=1}^{N} \frac{c_i}{T_i} \leq \frac{m}{2}(1 - u_{\max}) + u_{\max} \tag{12}$$

where $u_{\max} = \max\{\frac{c_i}{T_i} \mid i = 1, \ldots, N\}$.

Proof.

The proof is similar to that of Corollary 13. Since deadline equals period, $\lambda_{\max} = u_{\max}$. Let $\mu = m(1-\lambda_{\max}) = m(1-u_{\max})$. It follows that $\frac{m-\mu}{m-1} > \frac{c_k}{\min\{T_k, d_k\}}$ for $k = 1, \ldots, N$. Therefore, since $d_k = T_k$,

$$\sum_{i=1}^{k-1} \beta_{\mu,k}(i) = \sum_{i=1}^{k-1} \frac{c_i}{T_i}(1 + \frac{T_i - c_i}{T_k})$$

By the rate monotonic ordering of task priorities, $T_i \leq T_k$ for $i < k$, and so

$$\frac{c_i}{T_i}(1 + \frac{T_i - c_i}{T_k}) \geq \frac{c_i}{T_i}(1 + \frac{T_i - c_i}{T_i}) = 2\frac{c_i}{T_i} - (\frac{c_i}{T_i})^2 \tag{13}$$

By substitution of (13) into (12), and using the fact that $u_{\max} \geq \frac{c_k}{T_k}$, it follows that

$$\begin{aligned} \sum_{i=1}^{k-1} \beta_{\mu,k}(i) &\leq \sum_{i=1}^{k-1}(2\frac{c_i}{T_i} - (\frac{c_i}{T_i})^2) \\ &\leq 2\sum_{i=1}^{N} \frac{c_i}{T_i} - 2u_{\max} \end{aligned}$$

Suppose condition (12) is satisfied. It follows that, for $k = 1, \ldots, N$,

$$\sum_{i=1}^{k} \beta_{\mu,k}(i) \leq 2(\frac{m}{2}(1 - u_{\max}) + u_{\max}) - 2u_{\max} = \mu$$

Since $\mu \leq m(1 - \frac{c_k}{\min\{T_k, d_k\}})$ for $k = m+1, \ldots, N$, by Theorem 11, the task set must be schedulable.

□

The above theorem and its corollaries provide three general fixed-priority schedulability tests, one of complexity $O(N^3)$, one of complexity $O(N^2)$, and one of complexity $O(N)$, and a utilization-bound test for rate monotonic scheduling in the case where deadline equals period. It is natural to wonder how the performances of these tests compare. In order to address this question, simulation experiments were conducted, using randomly generated task sets.

The performance measure chosen for these simulations is the ratio A/B, where A is the number of task sets that a given test verifies as schedulable and B is the number of task sets for which no missed deadlines occur over the course of a simulated execution. The simulated execution is of periodic tasks, all released at time zero. If each task τ_k reaches a level $(k-1)$ idle point without missing a deadline, the simulation test is considered to have passed. Although this pseudo-polynomial-time test is a sufficient test for schedulability on a single processor, due to Liu and Layland's critical zone theorem, the critical zone property does not extend to multiprocessors, so it is only a necessary (not sufficient) test for multiprocessor schedulability. However, it is still useful as a filter for weeding out of the test sample many task sets that are clearly unschedulable.

Simulations were conducted for various numbers of processors, task sizes, and priority assignment rules, and with various algorithms for randomly generating the task periods, deadlines, and execution times. In the interest of brevity, just a few of the results are presented here, all for the case of $m = 4$ processors. The results shown are for simulations of 1,000,000 task sets. Times were represented as integers. The procedure was to generate a random set of $m + 1$ tasks, run all the schedulability tests on that set, then add a randomly generated task to the set, run all the schedulability tests on the new set, *etc.*, until the set grew to a size that it missed a deadline when simulated with all tasks released together at time zero. The procedure was then repeated. This method of generating random task sets produces a fairly uniform distribution of total utilizations, except at the two extremes. Figure 4 shows the distributions of task set utilizations in two experiments with four processors. One used rate monotonic scheduling and deadline equal to period, and the other used deadline monotonic scheduling and randomly chosen deadlines uniformly distributed over the range between the execution time and the period. The distributions are sparse at the low ends because sets with low utilization mostly had fewer than four tasks. Such task sets were thrown out of the sample because they are trivially schedulable. The

distributions are sparse at the high ends because most of the sets with high utilization missed a deadline in the simulation test, and so were also thrown out of the sample because they are unschedulable.

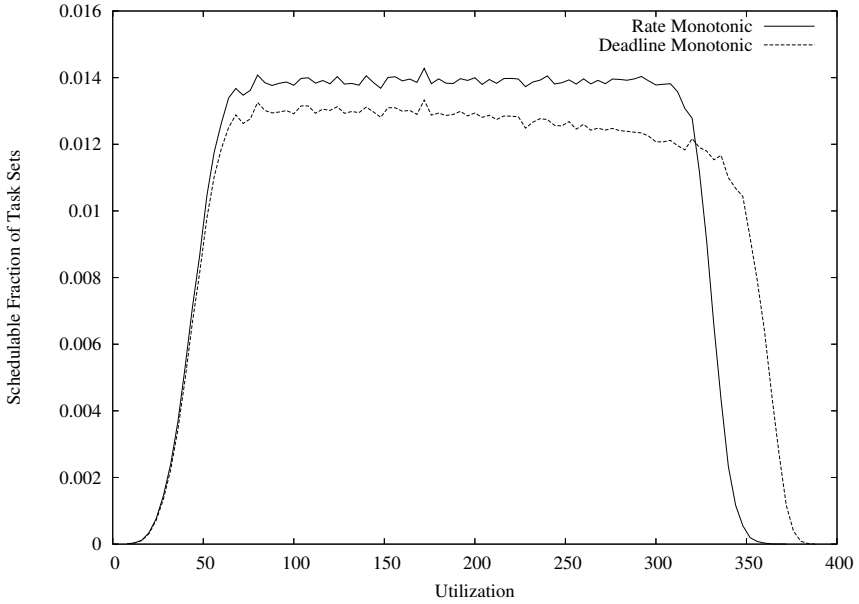

Figure 4. Distribution of total utilizations in experiments.

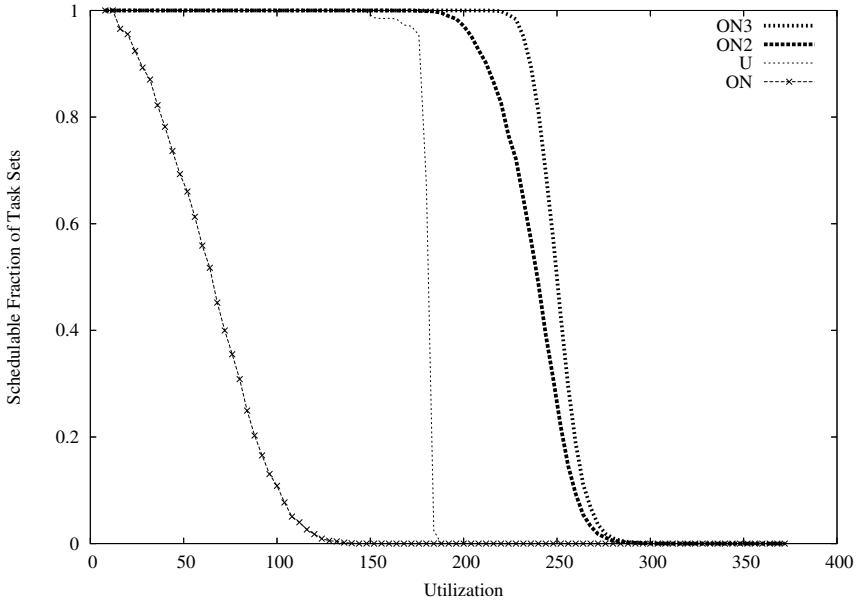

Figure 5. Deadline equals period and rate monotonic priorities.

Figure 5 shows the performances of the schedulability tests for a four processor system when deadline equals period and priorities are rate-monotonic. Figure 6 shows the performances for randomly chosen deadlines and deadline-monotonic priorities. The vertical axis corresponds to the fraction of the total

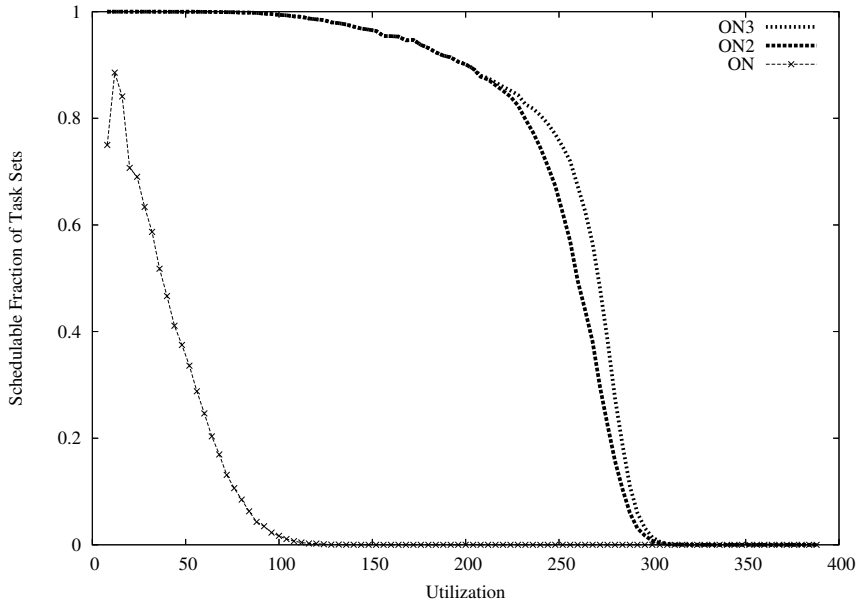

Figure 6. Random deadlines and deadline monotonic priorities.

number of task sets that each schedulability test reports to be schedulable, for each utilization level. The higher-complexity tests clearly are able to identify a larger number of schedulable task sets at the higher utilization levels. In all experiments, the $O(N)$ test performed much worse than the $O(N^2)$ and $O(N^3)$ tests. The $O(N^3)$ performed consistently better than the $O(N^2)$ test, but the margin was not large. The utilization-bound test performed well where it could be applied, but the extension of this test to pre-period deadlines using padding (not shown here) performed extremely poorly, much worse than the $O(N)$ test.

6. Untightness of Utilization Bound

The lower bound on the minimum achievable utilization given by Corollary 14 is close, but it is not tight. The following theorem provides an estimate of how tight it is.

THEOREM 15 (upper bound on minimum achievable RM utilization). *There exist task sets that are not feasible with preemptive RM scheduling on m processors and have utilization arbitrarily close to $u_{\max} + m \ln(\frac{2}{1+u_{\max}})$, where $u_{\max}$ is the maximum single-task utilization.*

Proof.

The goal of the proof is to show that there is an infinite sequence of task sets $S'_{m,1}, S'_{m,2}, S'_{m,3}, \ldots$, each of which is not schedulable on m processors and for which

$$\lim_{k\to\infty} U(S_{m,k}) = u_{\max} + m\ln(\frac{2}{1+u_{\max}})$$

Instead of constructing such a sequence of unschedulable task sets directly, it will be sufficient to construct an infinite sequence of *barely schedulable* task sets $S_{m,1}, S_{m,2}, S_{m,3}, \ldots$, whose utilizations converge to the desired limit. Then, for each k, an unschedulable task set $S'_{m,k}$ can be obtained from $S_{m,k}$ by increasing the execution time c_N of the lowest-priority task by an amount that converges to zero for sufficiently large k. Since the limit of the amounts by which the task execution times are increased is zero, the limit of the utilizations of the sequence of unschedulable task sets $S'_{m,k}$ is the same as the limit of the utilizations of the barely schedulable task sets $S_{m,k}$. Therefore, to prove the theorem it is sufficient to construct the sequence $S_{m,1}, S_{m,2}, S_{m,3}, \ldots$ of barely schedulable task sets.

The construction of the barely schedulable task sets and the proof of the limiting utilization are derived from Liu and Layland[7]. The differences are that there are m processors instead of one, and the utilization of the longest-period task is bounded by $u_{\max}$.

Let $p_i = (\frac{2}{1+u_{\max}})^{\frac{i}{k}}$ for $i = 1, \ldots, k+1$. The periods of the $N = mk+1$ tasks in $S_{m,k}$ are defined to be

$$T_{(i-1)m+1} = T_{(i-1)m+2} = \cdots = T_{i\cdot m} = p_i \text{ for } i = 1, \ldots, k$$

$$T_N = p_{k+1}$$

and the execution times of the tasks are defined to be

$$c_{(i-1)m+1} = c_{(i-1)m+2} = \cdots = c_{i\cdot m} = p_{i+1} - p_i \text{ for } i = 1, \ldots, k$$

$$c_N = 2p_1 - p_{k+1} = T_N - 2\sum_{i=1}^{k}(p_{i+1} - p_i)$$

As shown schematically in Figure 7, when all the tasks are released together at time zero they execute as k blocks of m, with task τ_N being forced to execute during the time the other tasks are idle. Such a task set is barely schedulable. Any increase in the execution time of task τ_N will cause it to miss its deadline. The utilization of such a task set is

$$U(S_{m,k}) = \frac{2p_1 - p_{k+1}}{p_{k+1}} + \sum_{i=1}^{k} m\frac{p_{i+1} - p_i}{p_i}$$

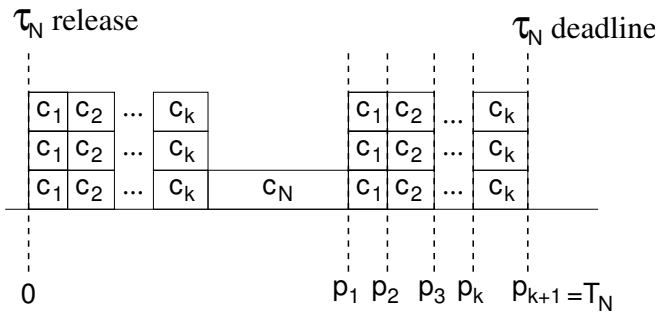

Figure 7. Task set that is barely schedulable.

$$
\begin{aligned}
&= 2(\prod_{i=1}^{k} \frac{p_i}{p_{i+1}}) - 1 + m(\sum_{i=1}^{k} \frac{p_{i+1}}{p_i}) - mk \\
&= 2(\prod_{i=1}^{k} (\frac{1+u_{\max}}{2})^{\frac{1}{k}}) - 1 + m(\sum_{i=1}^{k} (\frac{2}{1+u_{\max}})^{\frac{1}{k}}) - mk \\
&= 2(\frac{1+u_{\max}}{2}) - 1 + mk(\frac{2}{1+u_{\max}})^{\frac{1}{k}}) - mk \\
&= u_{\max} + mk((\frac{2}{1+u_{\max}})^{\frac{i}{k}} - 1)
\end{aligned}
$$

L'Hôpital's Rule can be applied to find the limit of the above expression for large k, which is

$$\lim_{k\to\infty} U(S_{m,k}) = u_{\max} + m \ln(\frac{2}{1+u_{\max}})$$

□

The actual minimum achievable RM utilization is somewhere between the lower bound given by Corollary 14 and the upper bound given by Theorem 15.

7. Relation to Prior Work

Andersson, Baruah, and Jonsson[1] defined a periodic task set $\{\tau_1, \tau_2, \ldots, \tau_N\}$ to be a *light system on m processors* if it satisfies the following properties:

1. $\sum_{i=1}^{N} \frac{c_i}{T_i} \leq \frac{m^2}{3m-2}$
2. $\frac{c_i}{T_i} \leq \frac{m}{3m-2}$, for $1 \leq i \leq n$.

They then proved the following theorem:

THEOREM 16 (Andersson, Baruah, Jonsson). *Any periodic task system that is light on m processors is scheduled to meet all deadlines on m processors by the preemptive rate monotonic scheduling algorithm.*

The above result is a special case of Corollary 14. If one takes $u_{\max} = m/(3m-2)$, it follows that the system of tasks is schedulable to meet deadlines if

$$\sum_{i=1}^{N} \frac{c_i}{T_i} \leq \frac{m}{2}(1 - \frac{m}{3m-2}) + \frac{m}{3m-2} = \frac{m^2}{3m-2}$$

Baruah and Goossens[2] proved the following similar result.

COROLLARY 17 (Baruah & Goossens). *A set of tasks, all with deadline equal to period, is guaranteed to be schedulable on m processors using rate monotonic scheduling if* $\frac{c_i}{T_i} \leq 1/3$ *for* $i = 1, \ldots, N$ *and*

$$\sum_{i=1}^{N} \frac{c_i}{T_i} \leq m/3$$

This, too, follows from a special case of Corollary 14. If one takes $u_{\max} = 1/3$, it follows that the system of tasks is schedulable to meet deadlines if

$$\sum_{i=1}^{N-1} \frac{c_i}{T_i} \leq \frac{m}{2}(1 - 1/3) + 1/3 = m/3 + 1/3$$

The results presented in this paper generalize and extend the above cited results in the following ways:

1. Theorem 11 can be applied to tasks with arbitrary deadlines. This is important for systems where some tasks have tighter deadlines, due to bounded jitter requirements, and other tasks have looser deadlines, due to buffering.

2. Theorem 11 can be applied to any set of sporadic tasks, without an arbitrary upper bound on individual task utilizations.

3. If the maximum utilization of all tasks is very low, Corollary 14 can guarantee higher levels of total utilization than $m^2/(3m-2)$ without missed deadlines. For example, if $u_{\max} \leq 1/4$ the system is guaranteed by Corollary 14 to be schedulable up to utilization $\frac{3}{8}m + 1/4$, as compared to Baruah and Goossens' $m^2/(3m-2)$.

4. So long as the total utilization is lower than $\frac{m}{2}(1 - u_{\max}) + u_{\max}$ Corollary 14 can accommodate tasks with utilization higher than $m/(3m - 2)$.

Andersson, Baruah, and Jonsson[1] proposed the following hybrid scheduling algorithm:

ALGORITHM 1. RM-US[λ]

(heavy task rule) If $\frac{c_i}{T_i} > \lambda$ then schedule τ_i's jobs at maximum priority.

(light task rule) If $\frac{c_i}{T_i} \leq \lambda$ then schedule τ_i's jobs according to their normal rate monotonic priorities.

They then proved that Algorithm RM-US[$m/(3m - 2)$] correctly schedules on m processors any periodic task system whose total utilization is at most $m^2/(3m-2)$. The proof is based on the observation that the upper bound on total utilization guarantees the number of heavy tasks cannot exceed m. The essence of the argument is that Algorithm RM-US[$m/(3m - 2)$] can do no worse than scheduling each of the heavy tasks on its own processor, and then scheduling the remainder (which must must be light on the remaining processors) using RM.

Corollary 14 extends the analysis of RM-US[λ] to other values of λ. Let ξ be the number of tasks with utilization greater than or equal to λ. If $\xi < m$, the algorithm can do no worse than scheduling the ξ highest utilization tasks on dedicated processors and scheduling the $N - \xi$ lowest-utilization tasks on the remaining $m - \xi$ processors. The corollary guarantees that the latter tasks can be scheduled unless their combined utilization exceeds $\frac{m-\xi}{2}(1 - \lambda) + \lambda$. That is, no deadline can be missed unless the total utilization, including both heavy and light tasks, exceeds

$$\frac{m - \xi}{2}(1 - \lambda) + (\xi + 1)\lambda$$

The value of the expression above is decreasing with respect to λ for $\lambda < 1/3$ and is increasing with respect to λ for $\lambda > 1/3$. It follows that the optimal value of λ for use with RM-US *and the utilization bound test* is $1/3$, in which case the worst-case guaranteed feasible utilization is $\frac{m+1}{3}$. Of course, the value $\lambda = 1/3$ is optimal only with respect to verifying schedulability using the utilization test, which is not tight.

Lundberg[10] has reasoned that the true optimum value of λ for RM-US[λ] is approximately 0.3748225282. He asserts that a task experiences the maximum competing load when there is 'block interference similar to that shown in Figure 1, and argues that the worst-case block interference occurs with task sets similar

to the family of examples in the proof of Theorem 15, whose limiting utilization is $\lambda + m \ln(\frac{2}{1+\lambda})$. If Lundberg's reasoning can be made rigorous, it would follow that $u_{\max} + m \ln(\frac{2}{1+u_{\max}})$ is the actual worst-case RM utilization bound.

The schedulability tests presented in Section 5 also suggest the following generalization of the RM-US idea:

ALGORITHM 2. RM-Hybrid[ξ]

Choose $N - \xi$ tasks with the lowest utilizations and call those the *light* tasks; call the rest of the tasks the *heavy* tasks. Schedule the ξ heavy tasks maximum priority, and the other tasks according to deadline-monotonic priorities.

The number ξ of heavy tasks is chosen so that the remaining $N - \xi$ tasks set can be verified as schedulable on the remaining $m - \xi$ processors, according to whatever schedulability test is available. For example, if one uses the $O(N^3)$ test, one would choose the smallest ξ such that the $N - \xi$ lowest-utilization tasks pass the $O(N^3)$ test for schedulability on $m - \xi$ processors. If such a ξ exists, it follows that the task set is schedulable.

8. Conclusions

Efficiently computable schedulability tests have been given for general fixed-priority scheduling on a homogeneous multiprocessor system, with arbitrary deadlines. These improve on previously known multiprocessor RM schedulability conditions by relaxing the assumptions of rate monotonic priorities and deadline being equal to period.

For the case where period equals deadline this analysis gives a simple lower bound on the minimum achievable utilization. That is, a system of independent periodic or sporadic tasks can be scheduled by RM scheduling to meet all deadlines if the total utilization is at most $\frac{m}{2}(1 - u_{\max}) + u_{\max}$, where $u_{\max}$ is the maximum of the individual task utilizations. This result can be used to verify the RM schedulability of systems of tasks with sufficiently low individual processor utilization, or combined with a hybrid scheduling policy to verify the schedulability of systems with a few high-utilization tasks. It can be applied statically, or applied dynamically as an admission test. This improves on previously known

utilization-based multiprocessor RM schedulability tests, by allowing both higher total utilizations and higher individual task utilizations. In addition to the new lower bound on the minimum achievable RM utilization, an upper bound of $u_{\max} + m \ln(\frac{2}{1+u_{\max}})$ has been derived.

The existence of these schedulability tests makes verification of single-queue (global) fixed-priority multi-processor schedulability an option for certain classes of task sets. This may be of immediate interest for real-time applications on popular symmetric multiprocessing operating systems, such as Linux and Sun Microsystems' Solaris, which support global scheduling as the default. Removing one of the reasons often given for favoring a queue-per-processor (partitioned) approach to multiprocessor scheduling also opens the way for additional research.

Further study is needed into the comparative strengths of global versus partitioned scheduling. One question is about the average-case performance. The worst-case utilization bounds seem very close, but is that also true of the average case? A second question is about the implementation overhead. Global scheduling has higher overhead in at least two respects: the contention delay and the synchronization overhead for a single dispatching queue is higher than for per-processor queues; the cost of resuming a task may be higher if it is on a different processor (due to interprocessor interrupt handling and cache reloading) than on the processor where it last executed. The latter cost can be quite variable, since it depends on the actual portion of a task's memory that remains in cache when the task resumes execution, and how much of that remnant will be referenced again before it is overwritten. It seems that only experimentation with actual implementations can determine how serious are these overheads, and how they balance against any advantages global scheduling may have for on-time completion of tasks.

Acknowledgement

The author is thankful to the anonymous reviewers for their constructive comments, which improved the quality of this paper. He is especially thankful to the reviewer who observed that most of the results in the paper apply to arbitrary fixed priority assignments, and the reviewer who suggested adding some simulation results.

References

1. B. Andersson, S. Baruah, and J. Jonsson. Static-priority scheduling on multiprocessors. In *Proc. 22nd IEEE Real-Time Systems Symposium*, pages 193–202, London, UK, December 2001.

2. S. Baruah and Joel Goossens. Rate-monotonic scheduling on uniform multiprocessors. *IEEE Trans. Computers*, 52(7):966–970, July 2003.
3. Willian Gardner Bell. Department of the army, historical summary fiscal year 1971 [online]. 1973. Available from: `http://www.army.mil/cmh-pg/books/DAHSUM/1971/chIV.htm`.
4. S. K. Dhall and C. L. Liu. On a real-time scheduling problem. *Operations Research*, 26(1):127–140, February 1978.
5. Intel Developer Support Forum. HT and pause vs. halt [online]. 2004. Available from: `http://softwareforums.intel.com/ids/board/print?board.id=42&message.id=%548`.
6. Arik Hesseldahl. Broadcom unveils new multicore chip [online]. October 2004. Available from: `http://www.forbes.com/enterprisetech/2004/10/04/cx_ah_1004chips.html`.
7. C. L. Liu and J. W. Layland. Scheduling alghorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, January 1973.
8. J. M. Lopez, J. L. Diaz, and D. F. Garcia. Minimum and maximum utilization bounds for multiprocessor RM scheduling. In *Proc. 13th Euromicro Conf. Real-Time Systems*, pages 67–75, Delft, Netherlands, June 2001.
9. J. M. Lopez, J. L. Diaz, M. Garcia, and D. F. Garcia. Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems. In *Proc. 12th Euromicro Conf. Real-Time Systems*, pages 25–33, 2000.
10. L. Lundberg. Analyzing fixed-priority global multiprocessor scheduling. In *Proc. 8th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 145–153, San Jose, CA, USA, 2002. IEEE Computer Society.
11. M. Meyer, G. Finger, H. Mehrgan, J. Stegmeier, and A. F. M. Moorwood. The ESO infrared detector high-speed array control and processing electronics IRACE [online]. November 2004. Available from: `http://www.eso.org/projects/iridt/irace/pdffiles/Irace_from_messenger_n%o_86.pdf`.
12. D. I. Oh and T. P. Baker. Utilization bounds for N-processor rate monotone scheduling with stable processor assignment. *Real Time Systems*, 15(2):183–193, September 1998.
13. C. A. Phillips, C. Stein, E. Torng, and J Wein. Optimal time-critical scheduling via resource augmentation. In *Proc. 29th Annual ACM Symposium on Theory of Computing*, pages 140–149, El Paso, TX, 1997. ACM.
14. L. Sha, T. Abdelzaher, K. E. Årzén, A. Cervin, T. P. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok. Real time scheduling theory: A historical perspective. *Real-Time Systems*, 28(2–3):101–155, November 2004.
15. Mercury Computer Systems. Mercury computer systems4 multicomputers selected for development phase of F-35 joint strike fighter program [online]. February 2004. Available from: `http://www.dedicated-systems.com/VPR/layout/display/pr.asp?PRID=6810`.