An Analysis of Deadline-Monotonic Schedulability on a Multiprocessor

FSU Computer Science Technical Report TR-030301

Theodore P. Baker

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

3 March 2003*

Abstract

A new schedulability test is presented for preemptive deadline-monotonic scheduling of periodic or sporadic real-time tasks on a single-queue m-server system. This generalizes and extends a result of Andersson, Baruah, and Jonsson, for the case where deadline equals period, which showed all deadlines will be met if the total utilization is at most $m^2/(3m-1)$ and the utilization of each task is at most $m/(3m-2)$. The new condition does not require that the task deadline be equal to the period, and can be used to verify schedulability for task sets with higher total utilizations or lower individual task utilizations. In addition to the lower bound on the minimum achievable utilization, an upper bound of $\lambda + m\ln(\frac{2}{1+\lambda})$ is derived.

1 Introduction

This paper derives a simple sufficient condition for schedulability of systems of periodic or sporadic tasks in a multiprocessor preemptive fixed-priority scheduling environment. In 1973 Liu and Layland[13] showed that the optimal way of assigning priorities to periodic tasks in such a system is rate monotonic. With rate monotonic (RM) scheduling tasks are assigned priorities according to their rates, with higher priority going to tasks with shorter periods. Liu and Layland showed further that there is a lower bound on the utilization at which systems of periodic tasks may miss deadlines under preemptive RM scheduling, called the *minimum achievable utilization*. A set of n periodic tasks is guaranteed to to meet deadlines on a single processor under RM scheduling if the system utilization is no greater than $n(2^{1/n}-1)$. This is a conservative but useful test for schedulability.

The ideas in [13] have been extended by many others, and developed into a comprehensive toolkit for the design and analysis of real time systems[11]. One of the extensions is deadline monotonic scheduling. Deadline monotonic (DM) scheduling is the generalization of rate monotonic scheduling to systems of tasks with pre-period and post-period deadlines, in which tasks with shorter relative deadlines are given higher priority.

The cornerstone of rate-monotonic and deadline-monotonic schedulability analysis is an observation in [13], called the *critical zone* property. That is, the worst-case response time for any periodic task is achieved when it

*Revision *date* : 2003/03/2615 : 30 : 20.

is released simultaneously with all the higher priority tasks. Unfortunately, the critical zone property only holds for single-processor systems.

Dhall and Liu[6] cast doubts on the value of rate monotonic scheduling for multiprocessor systems when they showed that RM scheduling can give very poor utilization on multiprocessor systems. For example, consider a set of periodic tasks $\tau_1, \ldots, \tau_{m+1}$ with periods and relative deadlines $T_1 = d_1 = 2mx, \ldots, T_m = d_m = 2mx, T_{m+1} = d_{m+1} = 2mx + 1$, and worst-case execution times $c_1 = 1, \ldots, c_m = 1, c_{m+1} = mx + 1$, all released at time zero. If τ_{m+1} is allowed to monopolize one processor, all the jobs can be completed by their deadlines on m processors, as shown in Figure 1. However, a rate monotonic scheduler for m processors would schedule $\tau_1 \ldots \tau_m$ ahead of τ_{m+1}, causing τ_{m+1} to miss its deadline, as shown in Figure 2.

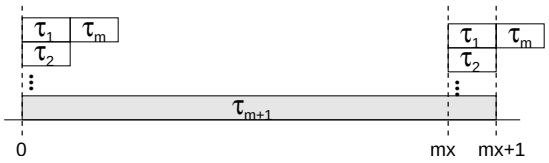

Figure 1: All jobs can be scheduled if τ_{m+1} is given its own processor.

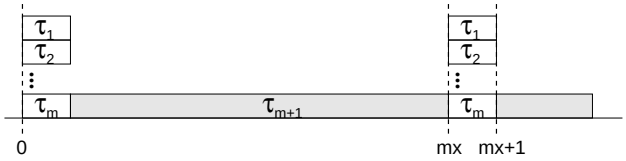

Figure 2: τ_{m+1} misses deadline with RM scheduling.

The total utilization of this set of tasks is

$$\begin{aligned} U &= (\sum_{i=1}^{m} \frac{c_i}{T_i}) + \frac{c_{m+1}}{T_{m+1}} \\ &= m\frac{1}{2mx} + \frac{2mx+1}{2mx+1} \\ &= 1 + \frac{1}{2x} \end{aligned}$$

By choosing x sufficiently large, U can be made as close to one as desired, *i.e.*, all but one of the m processors will be idle almost all of the time.

Generalizing from such examples, one is tempted to conjecture that there might be no useful utilization bound test for rate monotonic scheduling, and even that RM scheduling may not be a good real-time scheduling policy for multiprocessor systems. However, neither conclusion is actually justified.

Looking at the example above we observed that there are two kinds of tasks: "hard" ones, with high ratio of compute time to deadline, and "easy" ones, with low ratio of compute time to deadline. It is the mixing of those two kinds of tasks that causes a problem. A scheduling policy that segregates the large tasks from the small tasks, on disjoint sets of CPU's, would have no problem with this example. Examination of further examples led us to conjecture that such a segregated scheduling policy would not miss any deadlines until a very high level of CPU utilization is achieved, and may even permit the use of simple utilization-based schedulability tests.

Perhaps motivated by reasoning similar to ours, Andersson, Baruah, and Jonsson[1] recently 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 monotone scheduling if the total utilization is at most $m^2/(3m-1)$. They then proposed a modified scheduling algorithm that gives higher priority to tasks with utilizations above $m/(3m-2)$, which is able to successfully schedule *any* set of independent periodic tasks with total utilization up to $m^2/(3m-1)$. They derived these results indirectly, using a general theorem relating schedulability on sets of processors of different speeds by Phillips, Stein, Torng, and Wein[15]. A similar result, showing that a total utilization of at least $m/3$ can be achieved if the individual task utilizations do not exceed $1/3$, also based on [15], is proved by Baruah and Goossens[5].

We approached the problem in a different and more direct way, based on our analysis of earliest-deadline-first scheduling [3]. This leads to a different and more general schedulability condition, of which the results of [1] and [5] are special cases. The rest of this paper presents the derivation of our schedulability condition, discusses its application, and explains how it relates to the above cited prior work.

2 Definition of the Problem

Suppose one is given a set of n simple independent periodic tasks $\tau_1, \ldots, \tau_n$, where each task τ_i has minimum interrelease time T_i (called *period* for short), worst case compute time c_i, and relative deadline d_i, where $c_i \leq d_i \leq T_i$. 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 rational numbers. A time interval $[t_1, t_2)$, $t_1 \neq t_2$, is said to be of length $t_2 - t_1$ and contains the time values greater than or equal to t_1 and less than t_2.

Note that what we call a periodic task here is sometimes called a sporadic task. In this regard we follow Jane Liu[14], who observed that defining periodic tasks to have interrelease times exactly equal to the period "has led to the common misconception that scheduling and validation algorithms based on the periodic task model are applicable only when every periodic task is truly periodic ... in fact most existing results remain correct as long as interrelease times of jobs in each task are bounded from below by the period of the task".

In this paper we assume that the jobs of a set of periodic tasks are scheduled on m processors according to a preemptive fixed priority policy, with dynamic processor assignment. That is, 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 jobs that are not executing. For notational convenience, the tasks are numbered in order of increasing priority, so that task τ_1 has the highest priority.

Our objective is to formulate a simple test for schedulability, expressed in terms of the periods, deadlines, and worst-case compute times of the tasks, such that if the test is passed one can rest assured that no deadlines will be missed.

Our approach closely parallels the approach used in [3] to analyze EDF schedulability. We will consider a first failure of scheduling for the given set of tasks. That is, consider a sequence of job release times and execution times, consistent with the interrelease and execution time constraints, that produces a schedule with the earliest possible missed deadline. Find the first point in this schedule at which a deadline is missed. Let τ_k be the task of a job that misses its deadline at this first point. Let t be the release time of this job of τ_k. We call τ_k the *problem task*, the job of τ_k released at time t the *problem job*, and the time interval $[t, t + d_k)$ the *problem window*.

Definition 1 (demand) *The* demand *due to task τ_i over a time interval is the total amount of computation that would need to be completed within the interval for all the deadlines of τ_i within the interval to be met. For any task τ_k, the* competing demand *of τ_k is the sum of the demand due to all the other tasks τ_i that can preempt τ_k in the interval (i.e., such that $i < k$). The* combined demand *with respect to τ_k is the sum of the demand due to τ_k and the competing demand.*

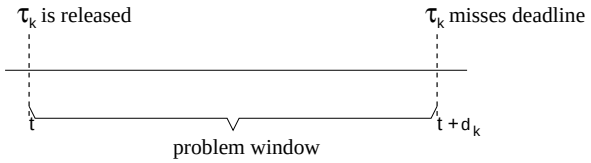

Figure 3: The problem window.

Definition 2 (load) *The* load *due to task τ_i over an interval $[t, t+\Delta)$ is W_i/Δ, where W_i is the demand due to τ_i over the interval. For any task τ_k the* competing load *of τ_k is the sum of the loads W_i/Δ over all the competing tasks τ_i, $i < k$. The* combined load *with respect to τ_k is the sum of the load due to τ_k and the competing load.*

If we can find a lower bound on the competing load that is necessary for a job to miss its deadline, and we can guarantee that a given set of tasks could not possibly generate so much load in the problem window, that would be sufficient to serve as a schedulability condition.

3 Lower Bound on Load

A lower bound on the competing load in a problem window can be established using the following well known argument, which is also used by [15]:

> Since the problem job misses its deadline, the sum of the lengths of all the time intervals in which the problem job does not execute must exceed its slack time, $d_k - c_k$.

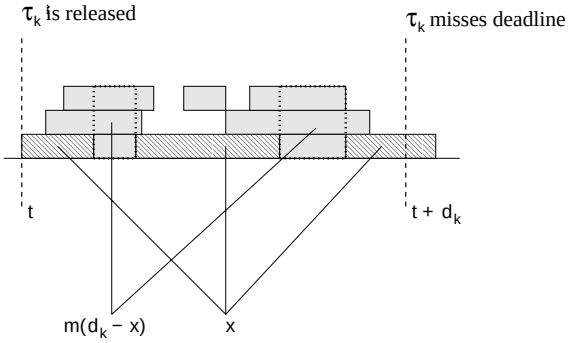

Figure 4: All processors must be busy whenever τ_k is not executing.

This situation is illustrated for $m = 3$ processors in Figure 4. The diagonally shaded rectangles indicate times during which τ_k executes. The dotted rectangles indicate times during which all m processors must be busy executing other jobs that are able to preempt that of τ_k in this interval.

Lemma 3 (lower bound on load) *If W/d_k is the combined load of the window $[t, t + d_k)$, where $t + d_k$ is a missed deadline of τ_k, then*

$$\frac{W}{d_k} > m(1 - \frac{c_k}{d_k}) + \frac{c_k}{d_k}$$

Proof: Let x be the amount of time that the problem job executes in the interval $[t, t+d_k)$. Since τ_k misses its deadline at $t+d_k$, we know that $x < c_k$. A processor is never idle while a job is waiting to execute. Therefore, during the problem window, whenever the problem job is not executing all m processors must be busy executing other jobs that can preempt τ_k. The sum of the lengths of all the intervals in the problem window for which all m processors are executing jobs belonging to the competing demand of the window must be at least $d_k - x$, so we have $\sum_{i \neq k} W_i > m(d_k - x)$. Since $x < c_k$ and the value of the expression on the right is decreasing with respect to x, we have $\sum_{i \neq k} W_i > m(d_k - c_k)$ and $\sum_{i \neq k} \frac{W_i}{d_k} > 1 - \frac{c_k}{d_k}$. Adding in the load of τ_k, we have

$$\frac{W}{d_k} = \sum_{i=1}^{k} \frac{W_i}{d_k} > m(1 - \frac{c_k}{d_k}) + \frac{c_k}{d_k}$$

□

Note that the argument used in Lemma 3 is very coarse. It does not take into account any special characteristics of the scheduling policy other than that it never idles a processor while a job is waiting to execute. This kind of scheduling algorithm is called a "busy algorithm" by Phillips *et al.*[15].

4 Upper Bound on Load

We now try to derive an upper bound on the load of a window leading up to a missed deadline. If we can find such an upper bound $\beta > W/\Delta$ it will follow from Lemma 3 that the condition $\beta \leq m(1 - \frac{c_k}{d_k}) + \frac{c_k}{d_k}$ is sufficient to guarantee schedulability. The upper bound β on W/Δ is the sum of individual upper bounds β_i on the load W_i/Δ due to each individual task τ_i that can preempt τ_k in the window, and β_k for τ_k itself. It then follows that

$$\frac{W}{\Delta} = \sum_{i=1}^{k} \frac{W_i}{\Delta} < \sum_{i=1}^{n} \beta_i$$

While our immediate interest is in a problem window, it turns out that one can obtain a tighter schedulability condition by considering an extension of the problem window. Therefore, we look for a bound on the load of an arbitrary downward extension $[t, t+\Delta)$ of a problem window. We call this extension the *window of interest*, or just the *window* for short.

We divide the window of interest into three parts, which we call the *head*, the *body*, and the *tail* of the window with respect to τ_i, as shown in Figures 5-7. The contribution W_i of τ_i to the demand in the window of interest is the sum of the contributions of the head, the body, and the tail. To obtain an upper bound on W_i we look at each of these contributions, starting with the head.

4.1 Demand of the Head

The *head* is the initial segment of the window up to the earliest possible release time (if any) of τ_i within or beyond the beginning of the window. More precisely, the head of the window is the interval $[t, t + \min\{\Delta, T_i - \phi)\})$, such that there is a job of task τ_i that is released at time $t' = t - \phi$, $t < t' + T_i < t + T_i$, $0 \leq \phi < T_i$. We call such a job, if one exists, the *carried-in job* of the window with respect to τ_i. The rest of the window is the body and tail, which are formally defined closer to where they are used, in Section 4.3.

Figures 5 and 6 show windows with carried-in jobs. If the interrelease constraint prevents any releases of τ_i within the window, the head comprises the entire window, as shown in Figure 5. Otherwise, the head is an initial segment of the window, as shown in Figure 6. If there is no carried-in job, as shown in Figure 7, the head is said to be null.

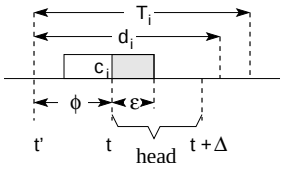

Figure 5: Window with head only ($\phi + \Delta \leq T_i$).

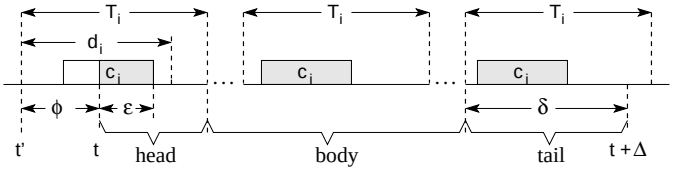

Figure 6: Window with head, body, and tail.

The carried-in job has two impacts on the demand in the window:

1. It constrains the time of the first release of τ_i (if any) in the window, to be no earlier than $t + T_i - \phi$.
2. It may contribute to W_i.

If there is a carried-in job, the contribution of the head to W_i is the residual compute time of the carried-in job at the beginning of the window, which we call the *carry-in*. If there is no carried-in job, the head makes no contribution to W_i.

Definition 4 (carry-in) *The* carry-in *of τ_i at time t is the residual compute time of the last job of task τ_i released before before t, if any, and is denoted by the symbol ϵ. This is illustrated in Figures 5 and 6.*

It is not hard to find a coarse upper bound for ϵ. If there are no missed deadlines before the window of interest (or if we assume that uncompleted jobs are aborted when they miss a deadline) no demand of the job of τ_i released at time t' can be carried past $t' + d_i$. In particular, $\epsilon = 0$ unless $\phi < d_i$. Therefore, for determining an upper bound on ϵ, we only need to look at cases where $\phi < d_i$. Since the carry-in must be completed between times $t' + \phi$ and $t' + d_i$, it follows that $\epsilon \leq d_i - \phi$.

The amount of carry-in is also bounded above by c_i, the full compute time of τ_i, but that bound is too coarse to be useful. The larger the value of ϕ the longer is the time available to complete the carried-in job before the beginning of the window, and the smaller should be the value of ϵ. We make this reasoning more precise in Lemmas 5 and 9.

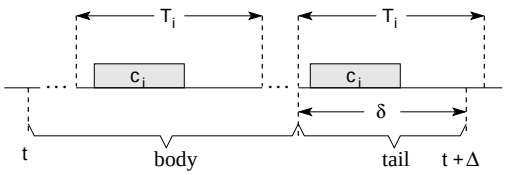

Figure 7: Window with body and tail only ($\phi = 0$).

Lemma 5 (carry-in bound) *If t' is the last release time of τ_i before t, $\phi = t - t'$, and y is the sum of the lengths of all the intervals in $[t', t)$ where all m processors are executing jobs that can preempt τ_i, then*

1. *If the carry-in ϵ of task τ_i at time t is nonzero, then $\epsilon = c_i - (\phi - y)$.*
2. *The combined load of the interval $[t', t)$ with respect to τ_i is at least $(m-1)(\phi - c_i + \epsilon)/\phi + 1$.*

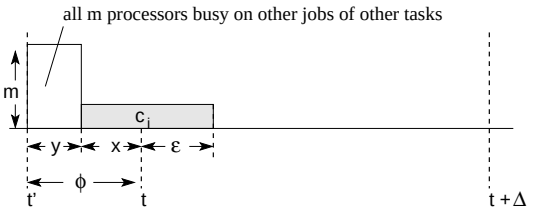

Figure 8: Carry-In depends on competing demand.

Proof: Suppose τ_i has nonzero carry-in. Let x be the amount of time that τ_i executes in the interval $[t', t' + \phi)$. For example, see Figure 8. Then, by the definition of ϵ, $\epsilon = c_i - x$. Since the job of τ_i does not complete in the interval, whenever τ_i is not executing during the interval all m processors must be executing other jobs that can preempt that job of τ_i. This has two consequences:

1. $x = \phi - y$, and so $\epsilon = c_i - (\phi - y)$
2. The combined load of the interval $[t', t' + \phi)$ with respect to τ_i is at least $(my + (\phi - y))/\phi$.

From the first observation above, we have $y = \phi - c_i + \epsilon$. Putting these two facts together gives

$$\frac{my + (\phi - y)}{\phi} = (m-1)\frac{y}{\phi} + 1 = (m-1)\frac{\phi - c_i + \epsilon}{\phi} + 1$$

□

4.2 Busy Interval

Since the size of the carry-in, ϵ, of a given task depends on the specific window and on the schedule leading up to the beginning of the window, it seems that bounding ϵ closely depends on being able to restrict the window of interest. Previous analysis of single-processor schedulability (*e.g.*, [13, 4, 7, 12]) bounded carry-in to zero by considering the *busy interval* leading up to a missed deadline, *i.e.*, the interval between the first time t at which a task τ_k misses a deadline and the last time before t at which there are no pending jobs that can preempt τ_k. By definition, no demand that can preempt τ_k is carried into the busy interval. By modifying the definition of busy interval slightly, we can also apply it here.

Definition 6 (λ-busy) *A time interval is* λ-busy *with respect to task τ_k if the combined load with respect to τ_k in the interval is strictly greater than $m(1-\lambda) + \lambda$. A* downward extension *of an interval is an interval that has an earlier starting point and shares the same endpoint. A* maximal λ-busy downward extension *of a λ-busy interval is a downward extension of the interval that is λ-busy and has no proper downward extensions that are λ-busy.*

Lemma 7 (λ-busy extension) *Any problem interval for task τ_k has a unique maximal λ-busy downward extension for $\lambda = \frac{c_k}{d_k}$.*

Proof:

Let $[t_0, t_0 + d_k)$ be any problem window for τ_k. By Lemma 3 the problem window is λ-busy, so the set of λ-busy downward extensions of the problem window is non-empty. The system has some start time, before which no task is released, so the set of all λ-busy downward extensions of the problem window is finite. The set is totally ordered by length. Therefore, it has a unique maximal element. □

Definition 8 (busy window) *For any problem window, the unique maximal $\frac{c_k}{d_k}$-busy downward extension whose existence is guaranteed by Lemma 7 is called the* busy window*, and denoted in the rest of this paper by $[t, t + \Delta)$.*

Observe that the busy window for τ_k contains a problem window for τ_k, and so $\Delta \geq d_k$.

Lemma 9 (λ-busy carry-in bound) *Let $[t, t + \Delta)$ be a λ-busy window window. Let $t - \phi$ be the last release time of τ_i before time t. If $\phi \geq d_i$ the carry-in of τ_i at t is zero. If the carry-in of τ_i at t is nonzero it is between zero and $c_i - \lambda\phi$.*

Proof: The proof follows from Lemma 5 and the definition of λ-busy. □

4.3 Completing the Analysis

We are looking for a close upper bound on the contribution W_i of each task τ_i to the demand in a particular window of time. We have bounded the contribution to W_i of the head of the window. We are now ready to derive a bound on the whole of W_i , including the contributions of head, body, and tail.

The *tail* of a window with respect to a task τ_i is the final segment, beginning with the release time of the *carried-out job* of τ_i in the window (if any). The *carried-out job* has a release time within the window and an earliest-permitted next release time beyond the window. That is, if the release time of the carried-out job is t'', $t'' < t + \Delta < t'' + T_i$. If there is no such job, then the tail of the window is null. We use the symbol δ to denote the length of the tail, as shown in Figures 6 and 7.

The *body* is the middle segment of the window, *i.e.*, the portion that is not in the head or the tail. Like the head and the tail, the body may be null (provided the head and tail are not also null).

Unlike the contribution of the head, the contributions of the body and tail to W_i do not depend on the schedule leading up to the window. They depend only on the release times within the window, which in turn are constrained by the period T_i and by the release time of the carried-in job of τ_i (if any).

Let n be the number of jobs of τ_i released in the body and tail. If both body and tail are null, $\Delta = \delta - \phi$, $n = 0$, and the contribution of the body and tail is zero. Otherwise, the body and or the tail is non-null, the combined length of the body and tail is $\Delta + \phi - T_i = (n - 1)T_i + \delta$, and $n \geq 1$.

Lemma 10 (combined demand) *For any busy window $[t, t+\Delta)$ of task τ_k (i.e., the maximal λ-busy downward extension of a problem window) and any task τ_i, the demand W_i of τ_i in the busy window is no greater than*

$$nc_i - \max\{0, c_i - \phi\lambda\}$$

where $n = \lfloor(\Delta - \delta_i)/T_i\rfloor + 1$, $\phi = nT_i + \delta_i - \Delta$, $\delta_i = c_i$ if $i < k$, and $\delta_i = d_i$ if $i = k$.

Proof:

We will first consider the case where $i < k$. We will identify a worst-case situation, where W_i achieves the largest possible value for a given value of Δ. For simplicity, we will risk overbounding W_i by considering a wide range of possibilities, which might include some cases that would not occur in a specific busy window. We will start by considering all conceivable patterns of release times, then narrow down the set of possibilities by stages, until only the worst case is left.

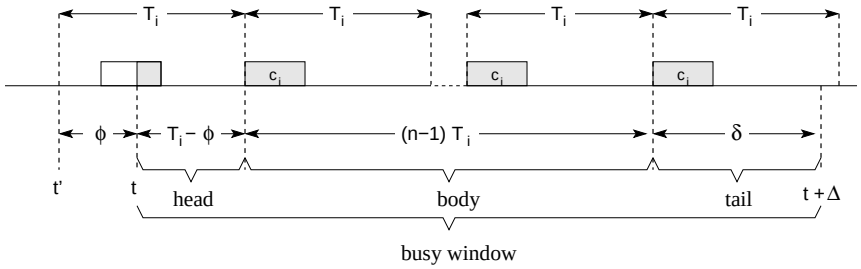

Figure 9: Dense packing of jobs.

By the deadline monotonic scheduling policy, τ_i can preempt τ_k if and only if $i < k$, *i.e.*, if and only if $d_i < d_k$. Since $c_i \leq d_i \leq d_k$ and $\Delta \geq d_k$, we have $\Delta \geq c_i$.

Looking at Figure 9, it is easy to see that the maximum possible contribution of the body and tail to W_i is achieved when successive jobs are released as close together as possible. Moreover, if one imagines shifting all the release times in Figure 9 earlier or later, as a block, one can see that the maximum is achieved when the last job is released just in time to have its complete execution fit within the window. That is, the maximum contribution to W from the body and tail is achieved when $\delta_i = c_i$. In this case there is a tail of length c_i and the number of complete executions of τ_i in the body and tail is $n = \lfloor (\Delta - c_i)/T_i \rfloor + 1$.

From Lemma 9, one can see that the contribution ϵ of the head to W_i is a nonincreasing function of ϕ. Therefore, ϵ is maximized when ϕ is as small as possible. However, reducing ϕ increases the size of the head, and so may reduce the contribution to W_i of the body and tail. This is a trade-off that we will analyze further.

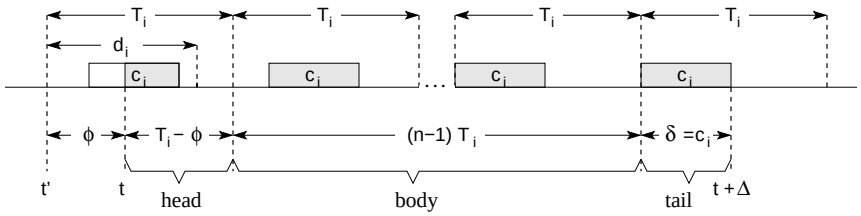

Figure 10: Densest possible packing of jobs.

Looking at Figure 10, we see that the length of the head, $T_i - \phi$, cannot be larger than $\Delta - ((n-1)T_i + c_i)$ without pushing at least part of the final execution of τ_i outside the window. Reducing ϕ below $nT_i + c_i - \Delta$ results in at most a linear increase in the contribution of the head, accompanied by and exactly linear decrease of c_i in the contribution of the body and tail. Therefore we expect the value of W_i to be maximized for $\phi = nT_i + c_i - \Delta$.

To make this reasoning more precise, consider the situation where $\delta = c_i$ and $\phi = nT_i + c_i - \Delta$, as shown in Figure 10. The value of W_i in this case is $\epsilon + nc_i$, and Lemma 9 tells us that $\epsilon \leq \max\{0, c_i - \phi\lambda\}$.

Suppose ϕ is increased by any positive value $\sigma < T_i - \phi$. All the release times are shifted earlier by σ. This

does not change the contribution to W_i of any of the jobs, except for possibly the carried-in job, which may have some of its demand shifted out of the window. Obviously, increasing ϕ cannot increase the value of W_i.

Now suppose ϕ is reduced by any positive value $\sigma \leq \phi$. All the release times are shifted later by σ. This shifts the the last job released in the window, resulting in a reduction of W_i by σ. At the same, time ϵ may increase by at most $\sigma\lambda$. The net change in ϵ is $\sigma(\lambda - 1) \leq 0$. Therefore, there is no way that reducing ϕ below $\phi = nT_i + c_i - \Delta$ can increase the value of W_i.

We have shown that the value of W_i is maximized for $i < k$ when $\delta_i = c_i$ and $\phi = nT_i + c_i - \Delta$.

$$W_i \quad \leq \quad nc_i + \epsilon \leq nc_i + \max\{0, c_i - \phi\lambda\}$$

Now, we consider the special case where $i = k$. For the case $i < k$, we saw that the demand of τ_i in the body and tail is maximized when $\delta_i = c_i$. However, that is not possible for τ_k. Since τ_k is the problem job, it must have a deadline at the end of the busy window. That is, instead of the situation in Figure 10, for τ_k the densest packing of jobs is as shown in Figure 11.

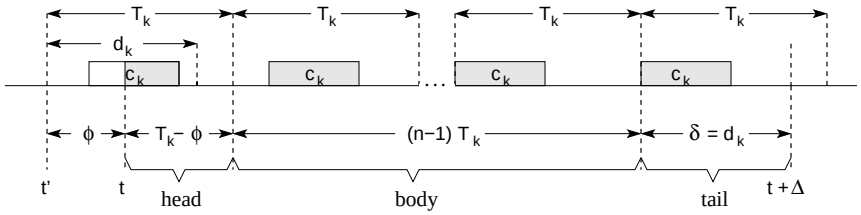

Figure 11: Densest possible packing of jobs for τ_k.

The only difference between this case and the one analyzed above is that the length δ_i of the tail is d_i instead of c_i. Making the corresponding adjustments to the analysis, we see that for the number of periods of τ_k spanning the busy window we still have $n = \lfloor(\Delta - \delta_i)/T_i\rfloor + 1$, and the maximum contribution of the head is still $\epsilon = \max\{0, c_i - \phi\lambda\}$. All differences are accounted for by the fact that $\delta_i = d_i$ instead of c_i. □

Lemma 11 (upper bound on load) *For any busy window $[t, t+\Delta)$ with respect to task τ_k the load W_i/Δ due to τ_i, $i < k$, is at most*

$$\beta_i = \begin{cases} \frac{c_i}{T_i}(1 + \frac{T_i - \delta_i}{d_k}) & \text{if } \lambda \geq \frac{c_i}{T_i} \\ \frac{c_i}{T_i}(1 + \frac{T_i - \delta_i}{d_k}) + \frac{c_i - \lambda T_i}{d_k} & \text{if } \lambda < \frac{c_i}{T_i} \end{cases}$$

where $\delta_i = c_i$ for $i < k$, and $\delta_k = d_k$.

Proof:

The objective of the proof is to find an upper bound for W_i/Δ that is independent of Δ. Lemma 10 says that

$$\frac{W_i}{\Delta} \quad \leq \quad \frac{nc_i + \max\{0, c_i - \phi\lambda\}}{\Delta}$$

Let $\alpha(\Delta)$ be the function defined by the expression on the right of the inequality above. We will analyze α to find the maximum with respect to Δ, for $\Delta \geq d_k$. This analysis of α involves consideration of two cases, depending on whether $\max\{0, c_i - \phi\lambda\} = 0$.

Case 1: $\max\{0, c_i - \phi\lambda\} = 0$.

We have $c_i - \lambda\phi \le 0$, and since $\phi < T_i$, we have $\lambda \ge \frac{c_i}{T_i}$. From the definition of n, we have

$$\begin{aligned} n &\le \frac{\Delta - \delta_i}{T_i} + 1 = \frac{\Delta - \delta_i + T_i}{T_i} \\ \alpha(\Delta) &= \frac{nc_i}{\Delta} \le \frac{c_i}{T_i}\frac{\Delta - \delta_i + T_i}{\Delta} \\ &\le \frac{c_i}{T_i}(1 + \frac{T_i - \delta_i}{\Delta}) \end{aligned}$$

Since $\Delta \ge d_k$ and $T_i - \delta_i \ge 0$, it follows that

$$\alpha(\Delta) \le \frac{c_i}{T_i}(1 + \frac{T_i - \delta_i}{d_k}) = \beta_i \quad (1)$$

Case 2: $\max\{0, c_i - \phi\lambda\} \ne 0$.

We have $c_i - \lambda\phi > 0$. Since $\phi = nT_i + \delta_i - \Delta$,

$$\begin{aligned} \alpha(\Delta) &= \frac{nc_i + c_i - \lambda\phi}{\Delta} \\ &= \frac{nc_i + c_i - \lambda(nT_i + \delta_i - \Delta)}{\Delta} \\ &= \frac{n(c_i - \lambda T_i) + c_i - \lambda(\delta_i - \Delta)}{\Delta} \end{aligned}$$

We have two subcases, depending on the sign of $c_i - \lambda T_i$.

Case 2.1: $c_i - \lambda T_i > 0$. That is, $\lambda < \frac{c_i}{T_i}$.

From the definition of n, it follows that

$$\begin{aligned} n &\le \frac{\Delta - \delta_i}{T_i} + 1 = \frac{\Delta - \delta_i + T_i}{T_i} \\ \alpha(\Delta) &= \frac{n(c_i - \lambda T_i) + c_i - \lambda(\delta_i - \Delta)}{\Delta} \\ &\le \frac{\frac{\Delta - \delta_i + T_i}{T_i}(c_i - \lambda T_i) + c_i - \lambda(\delta_i - \Delta)}{\Delta} \\ &\le \frac{(\Delta - \delta_i + T_i)(c_i - \lambda T_i) + c_i T_i - \lambda(\delta_i - \Delta)T_i}{\Delta T_i} \\ &\le \frac{c_i\Delta - c_i\delta_i + c_i T_i - \Delta\lambda T_i + \delta_i\lambda T_i - \lambda T_i^2 + c_i T_i - \delta_i\lambda T_i + \Delta\lambda T_i}{\Delta T_i} \\ &\le \frac{c_i\Delta - c_i\delta_i + c_i T_i - \lambda T_i^2 + c_i T_i}{\Delta T_i} \\ &\le \frac{c_i}{T_i}(1 + \frac{T_i - \delta_i}{\Delta}) + \frac{c_i - \lambda T_i}{\Delta} \end{aligned}$$

Since $\Delta \ge d_k$, $T_i - \delta_i \ge 0$, and $c_i - \lambda\delta_i > 0$, we have

$$\alpha(\Delta) \le \frac{c_i}{T_i}(1 + \frac{T_i - \delta_i}{d_k}) + \frac{c_i - \lambda T_i}{d_k} = \beta_i \quad (2)$$

Case 2.2: $c_i - \lambda T_i \le 0$. That is, $\lambda \ge \frac{c_i}{T_i}$.

From the definition of n, it follows that

$$\begin{aligned} n &> \frac{\Delta - \delta_i}{T_i} \\ \alpha(\Delta) &= \frac{n(c_i - \lambda T_i) + c_i - \lambda(\delta_i - \Delta)}{\Delta} \\ &< \frac{\frac{\Delta-\delta_i}{T_i}(c_i - \lambda T_i) + c_i - \lambda(\delta_i - \Delta)}{\Delta} \\ &< \frac{(\Delta - \delta_i)(c_i - \lambda T_i) + c_i T_i - \lambda(\delta_i - \Delta)T_i}{\Delta T_i} \\ &< \frac{c_i\Delta - c_i\delta_i - \Delta\lambda T_i + \delta_i\lambda T_i - \lambda T_i^2 + c_i T_i - \delta_i\lambda T_i + \Delta\lambda T_i}{\Delta T_i} \\ &< \frac{c_i\Delta - c_i\delta_i + c_i T_i}{\Delta T_i} \\ &< \frac{c_i}{T_i}(1 + \frac{T_i - \delta_i}{\Delta}) \end{aligned}$$

Since $\Delta \geq d_k$, we have

$$\alpha(\Delta) < \frac{c_i}{T_i}(1 + \frac{T_i - \delta_i}{d_k}) = \beta_i \tag{3}$$

□

5 Schedulability Condition

Using the above analysis, we can now prove the following theorem, which provides a sufficient condition for deadline monotonic schedulability:

Theorem 12 (schedulability test) *A set of periodic tasks is schedulable on m processors using preemptive deadline-monotonic scheduling if, for every task τ_k,*

$$\sum_{i=1}^{k-1} \beta_i \leq m(1 - \frac{c_k}{d_k}) \tag{4}$$

where β_i is as defined in Lemma 11.

Proof: The proof is by contradiction. Suppose some task misses a deadline. We will show that this leads to a contradiction of (4).

Let τ_k be the first task to miss a deadline and $[t, t + \Delta)$ be a busy window for τ_k, as in Lemma 7. Since $[t, t+\Delta)$ is λ-busy for $\lambda = \frac{c_k}{d_k}$, we have $W/\Delta > m(1-\lambda)+\lambda$. By Lemma 11, $W_i/\Delta \leq \beta_i$ for $i = 0, \ldots, k$. Observe that

$$\beta_k = \frac{c_k}{T_k}(1 + \frac{T_k - d_k}{d_k}) = \frac{c_k}{d_k}$$

It follows that

$$\beta_k + \sum_{i=1}^{k-1} \beta_i \geq \frac{W_k}{\Delta} + \sum_{i=1}^{k-1} \frac{W_i}{\Delta} = \frac{W}{\Delta} > m(1-\lambda) + \lambda$$

Since $\beta_k = \frac{c_k}{d_k} = \lambda$, we can subtract λ from both sides of the inequality, and obtain a contradiction of (4).□

The schedulability test above must be checked individually for each task τ_k. If we are willing to sacrifice some precision, there is a simpler test that only needs to be checked once for the entire system of tasks.

Corollary 13 (simplified test) *A set of periodic tasks $\tau_1, \ldots, \tau_n$ is schedulable on m processors using preemptive DM scheduling if*

$$\sum_{i=1}^{n} \frac{c_i}{T_i}(1 + \frac{T_i - \delta_i}{d_k}) \quad \leq \quad m(1-\lambda) + \lambda \tag{5}$$

where $\lambda = \max\{\frac{c_i}{d_i} \mid i = 1, \ldots, n\}$, $\delta_i = c_i$ for $i < k$, and $\delta_k = d_k$.

Proof:

Corollary 13 is proved by repeating the proof of Theorem 12, adapted to fit the new definition of λ.

Let τ_k be the first task to miss a deadline and let $[t, t+\Delta)$ be the busy window whose existence is guaranteed by Lemma 7. Since $[t, t+\Delta)$ is $\frac{c_k}{d_k}$-busy, we have

$$\frac{W}{\Delta} > m(1 - \frac{c_k}{d_k}) + \frac{c_k}{d_k} = m - (m-1)\frac{c_k}{d_k}$$

Since $\frac{c_k}{d_k} \leq \lambda$ and $m - 1 \geq 0$, we have

$$\frac{W}{\Delta} > m - (m-1)\frac{c_k}{d_k} \geq m - (m-1)\lambda = m(1-\lambda) + \lambda$$

By Lemma 11, $W_i/\Delta \leq \beta_i$, for $i = 0, \ldots, n$. Since $\lambda \geq \frac{c_i}{d_i} \geq \frac{c_i}{T_i}$, only the first case of the definition of β_i applies, *i.e.*, $\beta_i = \frac{c_i}{T_i}(1 + \frac{T_i - \delta_i}{d_k})$. It follows that

$$\sum_{i=1}^{k} \frac{c_i}{T_i}(1 + \frac{T_i - \delta_i}{d_k}) \quad = \quad \sum_{i=1}^{k} \beta_i \geq \frac{W}{\Delta} > m(1-\lambda) + \lambda$$

Since $k \leq n$, the above contradicts (5).□

If we assume the deadline of each task is equal to its period the schedulability condition of Lemma 13 for deadline monotone scheduling becomes a lower bound on the minimum achievable utilization bound for rate monotone scheduling on a multiprocessor.

Corollary 14 (utilization bound) *A set of periodic tasks, all with deadline equal to period, is guaranteed to be schedulable on m processors, $m \geq 2$, using preemptive rate monotonic scheduling if*

$$\sum_{i=1}^{n} \frac{c_i}{T_i} \leq \frac{m}{2}(1-\lambda) + \lambda \tag{6}$$

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

Proof: Suppose some task misses a deadline. We will show that this leads to a contradiction of (6). Let τ_k be the first task to miss a deadline and let $[t, t+\Delta)$ be the busy window whose existence is guaranteed by Lemma 7. Since $[t, t+\Delta)$ is $\frac{c_k}{d_k}$-busy, we have

$$\frac{W}{\Delta} > m(1 - \frac{c_k}{d_k}) + \frac{c_k}{d_k} \tag{7}$$

By the deadline monotonic property, we have $d_i \leq d_k$, and since we are assuming $d_i = T_i$, we have

$$\begin{aligned}
\frac{c_i}{T_i}(1 + \frac{T_i - c_i}{d_i}) &\geq \frac{c_i}{T_i}(1 + \frac{T_i - c_i}{d_k}) = \beta_i \\
\frac{c_i}{T_i}(1 + \frac{T_i - c_i}{T_i}) &\geq \beta_i \\
\frac{c_i}{T_i}(2 - \frac{c_i}{T_i}) &\geq \beta_i \\
2\frac{c_i}{T_i} &\geq \beta_i
\end{aligned}$$

Summing up the loads due to $\tau_1, \ldots, \tau_k$, and applying (7) we have

$$\begin{aligned}
\frac{c_k}{T_k} + \sum_{i=1}^{k-1} 2\frac{c_i}{T_i} &\geq \frac{c_k}{T_k} + \sum_{i=1}^{k-1} \beta_i \\
&\geq \frac{W_k}{\Delta} + \sum_{i=1}^{k-1} \frac{W_i}{\Delta} \\
&\geq \frac{W}{\Delta} \\
&> m(1 - \frac{c_k}{d_k}) + \frac{c_k}{d_k}
\end{aligned}$$

Subtracting $\frac{c_k}{T_k}$ from both sides, we have

$$\begin{aligned}
\sum_{i=1}^{k-1} 2\frac{c_i}{T_i} &> m(1 - \frac{c_k}{d_k}) \\
\sum_{i=1}^{k-1} \frac{c_i}{T_i} &> \frac{m}{2}(1 - \frac{c_k}{d_k})
\end{aligned}$$

Adding back $\frac{c_k}{T_k}$ to both sides, we have

$$\sum_{i=1}^{k} \frac{c_i}{T_i} > \frac{m}{2}(1 - \frac{c_k}{d_k}) + \frac{c_k}{d_k} = \frac{m}{2} - \frac{c_k}{d_k}(\frac{m}{2} - 1)$$

Since $\frac{m}{2} - 1 \geq 0$ and $\frac{c_k}{d_k} \leq \lambda$, we have

$$\frac{m}{2} - \frac{c_k}{d_k}(\frac{m}{2} - 1) \geq \frac{m}{2} - \lambda(\frac{m}{2} - 1) = \frac{m}{2}(1 - \lambda) + \lambda$$

Since $k \leq n$, the above contradicts (6). □

Theorem 12 and Corollaries 13 and 14 are intended for use as schedulability tests. They can be applied directly to prove that a task set will meet deadlines with DM or RM scheduling, either before run time for a fixed set of tasks or during run time as an admission test for a system with a dynamic set of tasks. With Corollaries 13 and 14, one computes $\lambda = \max_{i=1}^{n} \frac{c_i}{d_i}$ and then checks the schedulability condition for this value of λ. With Theorem 12, one checks the schedulability condition for each task. In the latter case the specific value(s) of k for which the test fails provide some indication of where the problem lies.

6 Untightness of Utilization Bound

These schedulability tests are only *sufficient* conditions for schedulability. They are very conservative. In particular, the reasoning used to obtain the lower bound on demand in Lemma 3 does not take into account the

relations between the periods of the tasks. For example, consider the following set of four tasks, to be scheduled on $m = 3$ processors:

i	c_i	$T_i (= d_i)$
1	1	2
2	1	2
3	1	3
4	5	6

Because the period of task τ_3 is relatively prime to the period of tasks τ_1 and τ_2, release times (and deadlines) of these tasks can coincide at most once in every 6 time units. It follows that one processor is available for τ_4 to execute for at least 5 out of every 6 time units, and so every task will be able to make its deadline. However, if we apply Corollary 14 to this set we see that it does not pass the test:

$$\begin{aligned} \sum_{i=1}^{n} \frac{c_i}{T_i} &= \frac{3}{6} + \frac{3}{6} + \frac{2}{6} + \frac{5}{6} = \frac{13}{6} \\ &> \frac{13}{12} = \frac{3}{2}(1 - \frac{5}{6}) + \frac{5}{6} = \frac{m}{2}(1 - \lambda) + \lambda \end{aligned}$$

To shed some light on how much room for improvement there is in the lower bound on the minimum achievable utilization given by Corollary 14, we present the worst-behaved task set that we have been able to find, and then analyze it.

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 $\lambda + m\ln(\frac{2}{1+\lambda})$, where λ is the maximum single-task utilization.*

Proof: The task set and analysis are derived from Liu and Layland[13]. The difference is that here there are m processors instead of one, and the utilization of the longest-period task is bounded by a λ.

The task set contains $n = km + 1$ tasks where k is an arbitrary integer greater than or equal to 1. The task execution times and periods are defined in terms of a set of parameters $p_1, \ldots, p_{k+1}$ as follows:

$$\begin{aligned} T_{(i-1)*m+j} &= p_i \text{ for } 1 \le i \le k, 1 \le j \le m \\ c_{(i-1)*m+j} &= p_{i+1} - p_i \text{ for } 1 \le i \le k, 1 \le j \le m \\ T_n &= p_{k+1} \\ c_n &= T_n - 2\sum_{i=1}^{k}(p_{i+1} - p_i) \\ &= T_n - 2p_{k+1} - 2\sum_{i=2}^{k} p_i + 2\sum_{i=2}^{k} p_i + 2p_1 \\ &= 2p_1 - T_n \end{aligned}$$

These constraints guarantee that task τ_n barely has time to complete if all n tasks are released together at time zero. The RM schedule will have all m processors busy executing tasks $\tau_1, \ldots, \tau_{n-1}$ for $\sum_{i=1}^{k} 2(p_{i+1} - p_i) = T_n - c_n$ out of the T_n available time units, leaving exactly c_n units to complete τ_n.

If $\frac{c_n}{T_n} = \lambda$, we have

$$\begin{aligned} \lambda T_n &= c_n = 2p_1 - T_n \\ T_n &= \frac{2p_1}{1+\lambda} = p_{k+1} \end{aligned}$$

We will choose $p_1, \ldots, p_{k+1}$ to minimize the total utilization, which is

$$\begin{aligned} U &= \lambda + \sum_{i=1}^{k} m\frac{p_{i+1} - p_i}{p_i} \\ &= \lambda + m\left((\sum_{i=1}^{k} \frac{p_{i+1}}{p_i}) - k \right) \end{aligned}$$

The partial derivatives of U with respect to p_i are

$$\begin{aligned} \frac{\partial U}{\partial p_1} &= m(\frac{2}{(1+\lambda)p_k} - \frac{p_2}{p_1^2}) \\ \frac{\partial U}{\partial p_i} &= m(\frac{1}{p_{i-1}} - \frac{p_{i+1}}{p_i^2}) \text{ for } 1 < i < k \\ \frac{\partial U}{\partial p_k} &= m(\frac{1}{p_{k-1}} - \frac{2p_1}{(1+\lambda)p_k^2}) \end{aligned}$$

Since the second partial derivatives are all positive, a unique global minimum exists when all the first partial derivatives are zero. Solving the equations above for zero, we get

$$\begin{aligned} \frac{p_2}{p_1} &= \frac{2}{(1+\lambda)p_k} = \frac{p_{k+1}}{p_k} \\ \frac{p_{i+1}}{p_i} &= \frac{p_i}{p_{i-1}} \text{ for } 1 < i \leq k \end{aligned}$$

Let $x = \frac{p_{k+1}}{p_k} = \cdots = \frac{p_2}{p_1}$. It follows that

$$\begin{aligned} x^k &= \prod_{i=1}^{k} \frac{p_{i+1}}{p_i} = \frac{p_{k+1}}{p_1} = \frac{2}{(1+\lambda)} \\ x &= (\frac{2}{1+\lambda})^{\frac{1}{k}} \\ U &= \lambda + m(kx - k) = \lambda + mk\left((\frac{2}{1+\lambda})^{\frac{1}{k}} - 1 \right) \end{aligned}$$

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

$$\begin{aligned} \lim_{k\to\infty} U &= \lim_{n\to\infty} \lambda + mk(\frac{2}{1+\lambda}^{\frac{1}{k}} - 1) \\ &= \lambda + m \lim_{k\to\infty} k(\frac{2}{1+\lambda}^{\frac{1}{k}} - 1) \\ &= \lambda + m \lim_{k\to\infty} \frac{\frac{2}{1+\lambda}^{\frac{1}{k}} - 1}{\frac{1}{k}} \end{aligned}$$

$$
\begin{aligned}
&= \lambda + m \lim_{k\to\infty} \frac{\frac{d}{dn}(\frac{2}{1+\lambda}^{\frac{1}{k}} - 1)}{\frac{d}{dn}\frac{1}{k}} \\
&= \lambda + m \lim_{k\to\infty} \frac{\frac{2}{1+\lambda}^{\frac{1}{k}} \ln(\frac{2}{1+\lambda})\frac{-1}{k^2}}{\frac{-1}{k^2}} \\
&= \lambda + m \lim_{k\to\infty} \frac{2}{1+\lambda}^{\frac{1}{k}} \ln(\frac{2}{1+\lambda}) \\
&= \lambda + m \ln(\frac{2}{1+\lambda}) \lim_{k\to\infty} \frac{2}{1+\lambda}^{\frac{1}{k}} \\
&= \lambda + m \ln(\frac{2}{1+\lambda})
\end{aligned}
$$

□

The task set above is constructed so as to achieved the minimum utilization among a specific family of task sets that are barely schedulable on m processors, following Liu and Layland's worst-case analysis for RM scheduling on a single-processor. However, at the time of this writing we have not been able to show that this family captures the worst case for multiprocessors, i.e., that there is no task set outside this family that fully utilizes the processors and achieves a lower utilization.

We have spent some time looking at examples of task sets, including some where the critical zone property fails, and not yet found one where the achievable utilization is lower than $\lambda + m \ln(\frac{2}{1+\lambda})$. Therefore we still are led to conjecture that this may be the actual minimum achievable utilization.

Figure 12 shows how much of a gap there is between the lower bound on the minimum achievable RM utilization given by Corollary **??** and the upper bound given by Theorem 15.

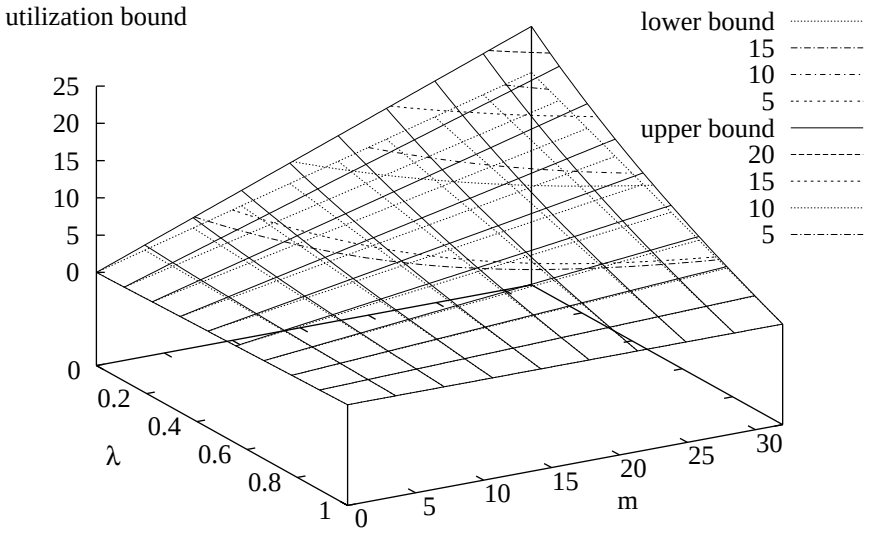

Figure 12: Comparison of upper and lower bounds on minimum achievable utilization.

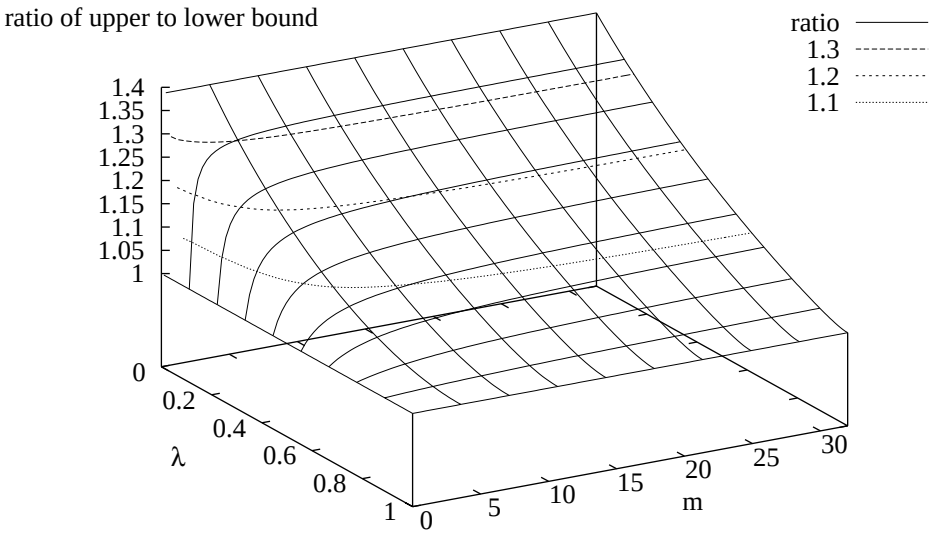

Figure 13: Ratio of upper bound to lower bound on minimum achievable utilization.

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 our Corollary 14. If we take $\lambda = m/(3m-2)$, 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[5] prove 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 our Corollary 14. If we take $\lambda = 1/3$, it follows that the system of tasks is schedulable to meet deadlines if

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

Our results generalize and extend the above cited results in the following ways:

1. Theorem 12 can be applied to tasks with pre-period deadlines. This is important for systems were some tasks have bounded jitter requirements.
2. Theorem 12 can be applied to any set of periodic 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 $\frac{c_k}{d_k} \leq 1/4$ the system is guaranteed by Corollary 14 to be schedulable up to utilization $\frac{3}{4}m+1/4$, as compared to Baruah and Goossens' $m^2/(3m-2)$.
4. If the total utilization is lower than $\frac{m}{2}(1-\lambda)+\lambda$ Corollary 14 can accommodate a few tasks with utilization higher than $m/(3m-2)$.

Andersson, Baruah, and Jonsson[1] proposed the following hybrid scheduling algorithm, which they call RM-US$[m/(3m-2)]$:

(heavy task rule) If $\frac{c_i}{T_i} > m/(3m-2)$ then schedule τ_i's jobs at maximum priority.

(light task rule) If $\frac{c_i}{T_i} \leq m/(3m-2)$ then schedule τ_i's jobs according to their normal rate monotonic priorities.

They then proved the following theorem:

Theorem 18 (Andersson, Baruah and Jonsson) *Algorithm RM-US$[m/(3m-2)]$ correctly schedules on m processors any periodic task system whose utilization is at most* $m^2/(3m-2)$.

The proof is based on the observation that the upper bound on total utilization guarantees that 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.

Theorem 12 and Corollary 14 suggest similar hybrid scheduling algorithms. For example, Algorithm RM-US$[m/(3m-2)]$ can be generalized as follows:

Algorithm RM-US$[\lambda]$

(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.

Theorem 19 *Algorithm RM-US[λ] correctly schedules on m processors any periodic task system such that only k tasks ($0 \leq k \leq m$) have utilization greater than λ and the utilization of the remaining tasks is at most $m(1-\lambda)+\lambda$.*

Proof: As argued by Baruah and Goossens, the performance of this algorithm cannot be worse than an algorithm that dedicates one processor to each of the heavy tasks, and uses RM to schedule the remaining tasks on the remaining processors. Corollary 14 then guarantees the remaining tasks can be scheduled on the remaining processors. □

8 Conclusion

We have demonstrated an efficiently computable schedulability test for DM and RM scheduling on a homogeneous multiprocessor system, which allows preperiod deadlines. This improves on previously known multiprocessor RM schedulability conditions by relaxing the assumption that deadline equal period.

For the case where periods equal deadlines this test reduces to a simple lower bound on the minimum achievable utilization. That is, a system of independent periodic or aperiodic tasks can be scheduled by RM to meet all deadlines if the total utilization is at most $\frac{m}{2}(1-\lambda)+\lambda$, where λ 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, we derived an upper bound of $\lambda + m\ln(\frac{2}{1+\lambda})$, which we conjecture may eventually be proven to be a tight lower bound.

References

[1] B. Andersson, S. Baruah, J. Jonsson, "Static-priority scheduling on multiprocessors", *Proceedings of the IEEE Real-Time Systems Symposium*, London, England (December 2001).

[2] B. Andersson, J. Jonsson, "Fixed-priority preemptive multiprocessor scheduling: to partition or not to partition", *Proceedings of the International Conference on Real-Time Computing Systems*, Cheju Island, South Korea (December 2000).

[3] "An Analysis of EDF Schedulability on a Symmetric Multiprocessor", technical report TR-030202, Florida State University Department of Computer Science, Tallahassee, Florida (February 2003).

[4] T.P. Baker, "Stack-based scheduling of real-time processes", *The Real-Time Systems Journal 3,1* (March 1991) 67-100. (Reprinted in *Advances in Real-Time Systems*, IEEE Computer Society Press (1993) 64-96).

[5] S. Baruah, Joel Goossens, "Rate-monotonic scheduling on uniform multiprocessors", UNC-CS TR02-025, University of North Carolina Department of Computer Science (May 2002).

[6] S.K. Dhall, C.L. Liu, "On a real-time scheduling problem", *Operations Research 26 (1)* (1998) 127-140.

[7] T.M. Ghazalie and T.P. Baker, "Aperiodic servers in a deadline scheduling environment", *the Real-Time Systems Journal 9,1* (July 1995) 31-68.

[8] R. Ha, "Validating timing constraints in multiprocessor and distributed systems", Ph.D. thesis, technical report UIUCDCS-R-95-1907, Department of Computer Science, University of Illinois at Urbana-Champaign (1995).

[9] R. Ha, J.W.S. Liu, "Validating timing constraints in multiprocessor and distributed real-time systems", technical report UIUCDCS-R-93-1833 , Department of Computer Science, University of Illinois at Urbana-Champaign, (October 1993).

[10] R. Ha, J.W.S. Liu, "Validating timing constraints in multiprocessor and distributed real-time systems", *Proceedings of the 14th IEEE International Conference on Distributed Computing Systems*, IEEE Computer Society Press (June 1994).

[11] Klein, M., Ralya, T., Pollak, B., Obenza, R. , González Harbour, M., *A practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems* Kluwer (August 1993).

[12] Lehoczky, J.P., Sha, L., Ding, Y., "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior", *Proceedings of the IEEE Real-Time System Symposium* (1989) 166-171.

[13] C.L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment", JACM 20.1 (January 1973) 46-61.

[14] J. W.S. Liu, *Real-Time Systems*, Prentice-Hall (2000) 71.

[15] C.A. Phillips, C. Stein, E. Torng, J Wein, "Optimal time-critical scheduling via resource augmentation", *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing* (El Paso, Texas, 1997) 140-149.

