Multiprocessor EDF and Deadline Monotonic Schedulability
Analysis
FSU Computer Science Technical Report TR-030401

Theodore P. Baker

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

7 February 2003*

Abstract

Schedulability tests are presented for preemptive earliest-deadline-first and deadline-monotonic scheduling of periodic real-time tasks on a single-queue m-server system, in which the deadline of a task may be less than or equal to the task period. These results subsume and generalize several known utilization-based multiprocessor schedulability tests, and are derived via an independent proof.

1 Introduction

This paper derives simple sufficient conditions for schedulability of systems of periodic or sporadic tasks in a multiprocessor preemptive scheduling environment.

In 1973 Liu and Layland[13] proved that systems of independent periodic tasks for which the relative deadline of each task is equal to its period will be scheduled to meet all deadlines by an preemptive earliest-deadline-first (EDF) scheduling policy so long as total processing demand does not exceed 100 percent of the system capacity. Besides showing that EDF scheduling is optimal for such task systems, this utilization bound provides a simple and effective a priori test for EDF schedulability. In the same paper Liu and Layland showed that if one is restricted to a fixed priority per task the optimal priority assignment is rate monotonic (RM), where tasks with shorter periods get higher priority. Liu and Layland showed further that a set of n periodic tasks is guaranteed to meet deadlines on a single processor under RM scheduling if the system utilization is no greater than n(2^1/n - 1). This test for RM scheduling and the 100% test for EDF scheduling have proven to be very useful tests for schedulability on a single processor system.

It is well known that the Liu and Layland results break down on multiprocessor systems[14]. Dhall and Liu[9] gave examples of task systems for which RM and EDF scheduling can fail at very low processor utilizations, essentially leaving all but one processor idle nearly all of the time. Reasoning from such examples, it is tempting to conjecture that there is unlikely to be a useful utilization bound test for EDF or RM scheduling, and even that these are not good real-time scheduling policies for multiprocessor systems. However, neither conclusion is actually justified.

The ill behaving examples have two kinds of tasks:

*Revision date : 2003/03/2615 : 30 : 20.
tasks with a high ratio of compute time to deadline, and tasks with a low ratio of compute time to deadline. It is the mixing of these two kinds of tasks that causes the problem. A policy that segregates the heavier tasks from the lighter tasks, on disjoint sets of CPU’s, would have no problem with this example. Examination of further examples leads one to conjecture that such a segregated scheduling policy would not miss any deadlines until a very high level of CPU utilization is achieved, and even permits the use of simple utilization-based schedulability tests.

In 1997, Phillips, Stein, Torng, and Wein[16] studied the competitive performance of on-line multiprocessor scheduling algorithms, including EDF and fixed priority scheduling, against optimal (but infeasible) clairvoyant algorithms. Among other things, they showed that if a set of tasks is feasible (schedulable by any means) on m processors of some given speed then the same task set is schedulable by preemptive EDF scheduling on m processors that are faster by a factor of \((2 - \frac{1}{m})\). Based on this paper, several overlapping teams of authors have produced a series of schedulability tests for multiprocessor EDF and RM scheduling[2, 6, 7, 8, 18].

We have approached the problem in a somewhat different and more direct way, which allows for tasks to have preperiod deadlines. This led to more general schedulability conditions, of which the above cited schedulability tests are special cases. The rest of this paper presents the derivation of these more general multiprocessor EDF and deadline monotonic (DM) schedulability conditions, and their relationship to the above cited prior work.

This conference paper is a condensation and summary of two technical reports, one of which deals with EDF scheduling[4] and the other of which deals with deadline monotonic scheduling[3]. To fit the conference page limits, it refers to those reports (available via HTTP) for most of the details of the proofs. The preliminaries apply equally to both EDF and RM scheduling. When the two cases diverge, the EDF case is treated first, and in slightly more detail.

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 \(\tau_i\) has minimum interrelease time (called period for short) \(T_i\), 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 \(t_2\).

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”.

Assume that the jobs of a set of periodic tasks are scheduled on \(m\) processors preemptively according to an EDF or DM scheduling 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 deadlines (absolute job deadlines for EDF, and relative task deadlines for DM) earlier than or equal to the jobs that are not executing.

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 is to analyze what happens when a deadline is missed. Consider a first failure of scheduling for a given task set, i.e., a sequence of job release times and compute times consistent with the interrelease and worst-case compute 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 \( \tau_k \) be the task of a job that misses its deadline at this first point. Let \( t \) be the release time of this job of \( \tau_k \). We call \( \tau_k \) a problem task, the job of \( \tau_k \) released at time \( t \) a problem job, and the time interval \([t, t + d_k]\) a problem window.

**Definition 1 (demand)** The demand of a time interval is the total amount of computation that would need to be completed within the window for all the deadlines within the interval to be met.

**Definition 2 (load)** The load of an interval \([t, t + \Delta]\) is \( W/\Delta \), where \( W \) is the demand of the interval.

If we can find a lower bound on the load of a problem window that is necessary for a job to miss its deadline, and we can show 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 load of a problem window can be established using the following well known argument, which is also the basis of [16]:

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, \( d_k - c_k \).

This situation is illustrated for \( m = 3 \) processors in Figure 1. The diagonally shaded rectangles indicate times during which \( \tau_k \) executes. The dotted rectangles indicate times during which all \( m \) processors must be busy executing other jobs in the demand for this interval.

**Lemma 3 (lower bound on load)** If \( W/d_k \) is the load of the interval \([t, t + d_k]\), where \( t + d_k \) is a missed deadline of \( \tau_k \), 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 \( \tau_k \) 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 with deadlines on or before \( t + d_k \). The sum of the lengths of all the intervals in the problem window for which all \( m \) processors are executing other jobs belonging to the demand of the interval must be at least \( d_k - x \). Summing up the latter demand and the execution of \( \tau_k \) itself, we have

\[
W \geq m(d_k - x) + x > md_k - (m - 1)c_k
\]

If we divide both sides of the inequality by \( d_k \), the lemma follows. □

### 4 Bounding Carry-In

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 \( \beta \geq W/\Delta \) it will follow from Lemma 3 that the condition \( \beta \leq m - (m - 1)c_k/d_k \) is sufficient to guarantee schedulability. The upper bound \( \beta \) on \( W/\Delta \) is the sum of individual upper bounds \( \beta_i \) on the load \( W_i/\Delta \) due to each individual task in the window. It then follows that

\[
\frac{W}{\Delta} = \sum_{i=1}^{n} \frac{W_i}{\Delta} \leq \sum_{i=1}^{n} \beta_i
\]
While our first interest is in a problem window, it turns out that one can obtain a tighter schedulability condition by considering a well chosen downward extension \([t, t + \Delta]\) of a problem window, which we call a window of interest.

For any task \(\tau_i\) that can execute in a window of interest, we divide the window into three parts, which we call the head, the body, and the tail of the window with respect to \(\tau_i\), as shown in Figure 2. The contribution \(W_i\) of \(\tau_i\) 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.

The head is the initial segment of the window up to the earliest possible release time (if any) of \(\tau_i\) 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 \(\tau_i\) 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 \(\tau_i\). The rest of the window is the body and tail, which are formally defined closer to where they are used, in Section 5.

Figure 2 shows a window with a carried-in job. The release time of the carried-in job is \(t' = t - \phi\), where \(\phi\) is the offset of the release time from the beginning of the window. If the minimum interrelease time constraint prevents any releases of \(\tau_i\) within the window, the head comprises the entire window. Otherwise, the head is an initial segment of the window. If there is no carried-in job, the head is said to be null.

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

1. It constrains the time of the first release of \(\tau_i\) (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 \(W_i\).

\[\text{Definition 4 (carry-in)}\] The carry-in of \(\tau_i\) at time \(t\) is the residual compute time of the last job of task \(\tau_i\) released before \(t\), if any, and is denoted by the symbol \(\epsilon\).

\begin{figure}[htb]
\centering
\includegraphics[width=0.5\textwidth]{carry-in.png}
\caption{Figure 3. Carry-in depends on competing demand.}
\end{figure}

The carry-in of a job depends on the competing demand. The larger the value of \(\phi\) 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 \(\epsilon\). We make this reasoning more precise in Lemmas 5 and 9.

\[\text{Lemma 5 (carry-in bound)}\] If \(t'\) is the last release time of \(\tau_i\) 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 \(\tau_i\), then

1. If the carry-in \(\epsilon\) of task \(\tau_i\) at time \(t\) is nonzero, then \(\epsilon = c_i - (\phi - y)\).
2. The load of the interval \([t', t]\) is at least \((m - 1)(\phi - c_i + \epsilon) / \phi + 1)\).

\[\text{Proof:}\] Suppose \(\tau_i\) has nonzero carry-in. Let \(x\) be the amount of time that \(\tau_i\) executes in the interval \([t', t' + \phi]\). For example, see Figure 3. By definition, \(\epsilon = c_i - x\). Since the job of \(\tau_i\) does not complete in the interval, whenever \(\tau_i\) is not executing during the interval all \(m\) processors must be executing other jobs that can preempt that job of \(\tau_i\). This has two consequences:

1. \(x = \phi - y\), and so \(\epsilon = c_i - (\phi - y)\)
2. The load of the interval \([t', t' + \phi]\) is at least \((mx + (\phi - y)) / \phi\).
From the first observation above, we have $y = \phi - c_i + \epsilon$. Putting these two facts together gives
\[
\frac{m y + (\phi - y)}{\phi} = (m-1)\frac{y}{\phi} + 1 = (m-1)\frac{\phi - c_i + \epsilon}{\phi} + 1
\]
\[
\square
\]
Since the size of the carry-in, $\epsilon$, 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 $\epsilon$ closely depends on being able to restrict the window of interest. Previous analysis of single-processor schedulability (e.g., [13, 3, 10, 11]) 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 $\tau_k$ misses a deadline and the last time before $t$ at which there are no pending jobs that can preempt $\tau_k$. By definition, no demand that can compete with $\tau_k$ 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 if its combined load is at least $m(1-\lambda) + \lambda$. 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 (busy window)** Any problem interval for task $\tau_k$ 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 $\tau_k$. 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. $\square$

**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 a busy window for $\tau_k$ contains a problem window for $\tau_k$, and so $\Delta \geq d_k$.

**Lemma 9 (λ-busy carry-in bound)** Let $[t, t + \Delta)$ be a λ-busy window. Let $r$ be the last release time of $\tau_i$, where $i \neq k$, before time $t$. If $\phi \geq d_i$ the carry-in of $\tau_i$ at $t$ is zero. If the carry-in of $\tau_i$ 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. $\square$

## 5 EDF Schedulability

We want to find a close upper bound on the contribution $W_t$ of each task $\tau_i$ to the demand in a particular window of time. We have bounded the contribution to $W_t$ of the head of the window. We are now ready to derive a bound on the whole of $W_t$, including the contributions of head, body, and tail for the EDF case.

The tail of a window with respect to a task $\tau_i$ is the final segment, beginning with the release time of
the carried-out job of $\tau_i$ in the window (if any). The carried-out job has a release time within the window and its next release time is beyond the window. That is, if the release time of the carried-out job is $t''$, $t'' < t + N < t'' + T_i$. If there is no such job, then the tail of the window is null. We use the symbol $\delta$ to denote the length of the tail, as shown in Figure 2.

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 $\tau_i$ (if any).

Let $n$ be the number of jobs of $\tau_i$ 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 (EDF demand)** For any busy window $[t, t + N]$ of task $\tau_k$ (i.e., the maximal $\lambda$-busy downward extension of a problem window) and any task $\tau_i$, the EDF demand $W_i$ of $\tau_i$ in the busy window is no greater than

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

where $\phi = nT_i + d_i - \Delta$, $n = \lfloor (\Delta - d_i)/T_i \rfloor + 1$ if $\Delta \geq d_i$, and $n = 0$ otherwise.

**Proof**: We will identify a worst-case situation, where $W_i$ achieves the largest possible value for a given value of $\Delta$. 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 out by looking only at the case where $\Delta \geq d_i$, then go back and consider later the case where $\Delta < d_i$.

Looking at Figure 4, 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 4 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 deadline coincide with the end of the window. That is, the maximum contribution to $W$ from the body and tail is achieved when $\delta = d_i$. In this case there is a tail of length $d_i$ and the number of complete executions of $\tau_i$ in the body and tail is $n = \lfloor (\Delta - d_i)/T_i \rfloor + 1$.

From Lemma 9, we can see that the contribution $\epsilon$ of the head to $W_i$ is a nonincreasing function of $\phi$. Therefore, $\epsilon$ is maximized when $\phi$ is as small as possible. However, reducing $\phi$ increases the size of the head, and may reduce the contribution to $W_i$ of the body and tail.

Looking at Figure 4, we see that the length of the head, $T_i - \phi$, cannot be larger than $\Delta - ((n - 1)T_i + d_i)$ without pushing all of the final execution of $\tau_i$ outside the window. Reducing $\phi$ below $nT_i + d_i - \Delta$ results in at most a linear increase in the contribution of the head, accompanied by a decrease of $c_i$ in the contribution of the body and tail. Therefore the value of $W_i$ is maximized for $\phi = nT_i + d_i - \Delta$.

We have shown that

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

It is now time to consider the case where $\Delta < d_i$. There can be no body or tail contribution, since it is impossible for a job of $\tau_i$ to have both release time and deadline within the window. If $W_i$ is nonzero, the only contribution can come from a carried-in job. Lemma 9 guarantees that this contribution is at most $\max\{0, c_i - \phi \lambda\}$. For $n = \lfloor (\Delta - d_i)/T_i \rfloor + 1 = 0$ we have

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

**Lemma 11 (upper bound on EDF load)** For any busy window $[t, t + N]$ with respect to $\tau_k$ the EDF load $W_i/\Delta$ due to $\tau_i$ is at most $\beta_i$, where

$$\beta_i = \begin{cases} \frac{\phi}{\lambda} \left(1 + \frac{T_i - d_i}{\lambda} \right) & \text{if } \lambda \geq \frac{\phi}{\lambda} \\ \frac{\phi}{\lambda} \left(1 + \frac{T_i - d_i}{\lambda} \right) + \frac{\phi}{\lambda} \frac{\lambda}{\lambda + d_i} & \text{if } \lambda < \frac{\phi}{\lambda} \end{cases}$$

© 2003 T.P. Baker. All rights reserved.
Proof:

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

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

Let $\alpha$ be the function defined by the expression on the right of the inequality above, i.e.,

$$\alpha(\Delta) = \frac{nc_i + \max\{0, c_i - \phi \lambda\}}{\Delta}$$

There are two cases:

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

We have $c_i - \lambda \phi \leq 0$, and $\phi \geq c_i/\lambda$. Since we also know that $\phi < T_i$, we have $\lambda \geq \frac{\phi}{T_i}$. From the definition of $n$, we have

$$n = \left[\frac{\Delta - d_i}{T_i}\right] + 1 \leq \frac{\Delta - d_i}{T_i} + 1 = \frac{\Delta - d_i + T_i}{T_i}$$

$$\alpha(\Delta) = \frac{nc_i}{\Delta} \leq \frac{c_i}{T_i}(1 + \frac{T_i - d_i}{\Delta})$$

$$\leq \frac{c_i}{T_i}(1 + \frac{T_i - d_i}{d_k}) = \beta_i$$

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

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

$$\alpha(\Delta) = \frac{nc_i + c_i - \lambda \phi}{\Delta}$$

$$= \frac{n(c_i - \lambda T_i) + c_i - \lambda (d_i - \Delta)}{\Delta}$$

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, $\lambda < \frac{d_i}{T_i}$.

From the definition of $n$, it follows that

$$n \leq \frac{\Delta - d_i}{T_i} + 1 = \frac{\Delta - d_i + T_i}{T_i}$$

$$\alpha(\Delta) = \frac{n(c_i - \lambda T_i) + c_i - \lambda (d_i - \Delta)}{\Delta}$$

$$\leq \frac{\Delta - d_i + T_i (c_i - \lambda T_i) + c_i - \lambda (d_i - \Delta)}{\Delta}$$

$$\leq \frac{c_i}{T_i}(1 + \frac{T_i - d_i}{d_k}) + \frac{c_i - \lambda T_i}{\Delta}$$

$$\leq \frac{c_i}{T_i}(1 + \frac{T_i - d_i}{d_k}) + \beta_i$$

**Case 2.2:** $c_i - \lambda T_i \leq 0$. That is, $\lambda \geq \frac{d_i}{T_i}$.

From the definition of $n$, it follows that

$$n > \frac{\Delta - d_i}{T_i}$$

$$\alpha(\Delta) = \frac{n(c_i - \lambda T_i) + c_i - \lambda (d_i - \Delta)}{\Delta}$$

$$\leq \frac{\Delta - d_i + T_i (c_i - \lambda T_i) + c_i - \lambda (d_i - \Delta)}{\Delta}$$

$$\leq \frac{c_i}{T_i}(1 + \frac{T_i - d_i}{d_k})$$

$$\leq \frac{c_i}{T_i}(1 + \frac{T_i - d_i}{d_k}) = \beta_i$$

\hfill \Box

Using the above lemmas, we can prove the following theorem, which provides a sufficient condition for schedulability.

**Theorem 12 (EDF schedulability test)** A set of periodic tasks $T_1, \ldots, T_n$ is schedulable on $m$
processors using preemptive EDF scheduling if, for every task \( \tau_k \),

\[
\sum_{i=1}^{n} \min\{1, \beta_i\} \leq m(1 - \frac{c_k}{d_k}) + \frac{c_k}{d_k}
\]

where \( \beta \) 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 (1).

Let \( \tau_k \) be the first task to miss a deadline and \( [t, t + \Delta] \) be a busy window for \( \tau_k \), as in Lemma 7. Since \( [t, t + \Delta] \) is \( \frac{c_k}{d_k} \)-busy we have \( W/\Delta > m(1 - \frac{c_k}{d_k}) + \frac{c_k}{d_k} \). By Lemma 11, \( W_i/\Delta \leq \beta_i \), for \( i = 0, \ldots, n \). Since \( t + \Delta \) is the first missed deadline, we know that \( W_i/\Delta \leq 1 \). It follows that

\[
\sum_{i=1}^{n} \min\{1, \beta_i\} \geq \frac{W}{\Delta} > m(1 - \frac{c_k}{d_k}) + \frac{c_k}{d_k}
\]

The above is a contradiction of (1). \( \square \)

The schedulability test above must be checked individually for each task \( \tau_k \). 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 EDF test)** A set of periodic tasks \( \tau_1, \ldots, \tau_n \) is schedulable on \( m \) processors using preemptive EDF scheduling if

\[
\sum_{i=1}^{n} \min\{1, \frac{c_i}{T_i}(1 + \frac{T_i - d_i}{d_{\min}})\} \leq m(1 - \lambda) + \lambda
\]

where \( \lambda = \max\{\frac{c_i}{d_i} \mid i = 1, \ldots, n\} \) and \( d_{\min} = \min\{d_k \mid i = 1, \ldots, n\} \).

**Sketch of proof:**

Corollary 13 is proved by repeating the proof of Theorem 12, adapted to fit the definitions of \( \lambda \) and \( d_{\min} \). \( \square \)

Goessens, Funk, and Baruah\[8\] showed the following:

**Corollary 14 (Goessens, Funk, Baruah\[8\])** A set of periodic tasks \( \tau_1, \ldots, \tau_n \), all with deadline equal to period, is guaranteed to be schedulable on \( m \) processors using preemptive EDF scheduling if

\[
\sum_{i=1}^{n} \frac{c_i}{T_i} \leq m(1 - \lambda) + \lambda
\]

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

Their proof is derived from a theorem in \[7\], on scheduling for uniform multiprocessors, which in turn is based on \[16\]. This can be shown independently as special case of Corollary 13, by replacing \( d_i \) by \( T_i \).

The above cited theorem of Goossens, Funk, and Baruah is a generalization of a result of Srinivasan and Baruah\[18\], who 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. \( \sum_{i=1}^{n} \frac{c_i}{T_i} \leq \frac{m^2}{m^2-1} \)
2. \( \frac{c_i}{T_i} \leq \frac{m}{2(m-1)} \), for \( 1 \leq i \leq n \).

They then proved the following theorem.

**Theorem 15 (Srinivasan, Baruah\[18\])** Any periodic task system that is light on \( m \) processors is scheduled to meet all deadlines on \( m \) processors by EDF.

The above result is a special case of Corollary 14, taking \( \lambda = m/(2m - 1) \).

6 DM Schedulability

The analysis of deadline monotonic schedulability is similar to that given above for the EDF case, with a few critical differences.

**Lemma 16 (DM demand)** For any busy window \([t, t + \Delta]\) for \( \tau_k \) and any task \( \tau_i \), the DM demand \( W_i \) of \( \tau_i \) in the busy window is no greater than \( n c_i - \max\{0, c_i - \phi\lambda\} \), where \( n = [(\Delta - \delta_i)/T_i] + 1 \), \( \phi = n T_i + \delta_i - \Delta \), \( \delta_i = c_i \) if \( i < k \), and \( \delta_i = d_i \) if \( i = k \).

**Sketch of proof:**

The full analysis is given in \[5\]. We first consider the case where \( i < k \), and then consider the special
case where \( i = k \). Looking at Figure 5, one can see that \( W_i \) is maximized for \( i < k \) when \( \delta_i = c_i \) and \( \phi = nT_i + c_i - \Delta \).

For the case \( i = k \) it is not possible to have \( \delta_i = c_i \). Since \( \tau_k \) is the problem job, it must have a deadline at the end of the busy window. Instead of the situation in Figure 5, for \( \tau_k \) the densest packing of jobs is as shown in Figure 6. That is, the difference for this case is that the length \( \delta_i \) of the tail is \( d_i \) instead of \( c_i \).

The number of periods of \( \tau_k \), the solution to the busy window in both is \( n = [(\Delta - \delta_i)/T_i] + 1 \), and the maximum contribution of the head is \( \varepsilon = \max \{ 0, c_i - \phi \lambda \} \). All differences are accounted for by the fact that \( \delta_i = d_i \) instead of \( c_i \).

**Lemma 17 (upper bound on DM load)** For any busy window \([t, t + \Delta]\) with respect to task \( \tau_k \) the DM load \( W_i / \Delta \) due to \( \tau_i, i < k \), is at most

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

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

The above lemma leads to the following DM schedulability test.

**Theorem 18 (DM schedulability test)** A set of periodic tasks is schedulable on \( m \) processors using preemptive deadline-monotonic scheduling if, for every task \( \tau_k \),

\[
\sum_{i=1}^{k-1} \beta_i \leq m(1 - \frac{c_k}{d_k})
\]

where \( \beta_i \) is as defined in Lemma 17.

The proof is given in [5]. It is similar to that of Theorem 12, but using the appropriate lemmas for DM scheduling.

**Corollary 19 (simplified DM test)** A set of periodic tasks \( \tau_1, \ldots, \tau_n \) is schedulable on \( m \) processors using preemptive DM scheduling if

\[
\sum_{i=1}^{n} \frac{c_i}{T_i} (1 + \frac{T_i - \delta_i}{d_k}) \leq m(1 - \lambda) + \lambda
\]

where \( \lambda = \max \{ \frac{c_i}{T_i} | i = 1, \ldots, n \} \), \( \delta_i = c_i \) for \( i < k \), and \( \delta_k = d_k \).

Corollary 19 is proved by repeating the proof of Theorem 18, adapted to fit the definition of \( \lambda \).

If we assume the deadline of each task is equal to its period the schedulability condition of Corollary 19 for deadline monotone scheduling becomes a lower bound on the minimum achievable utilization for rate monotone scheduling.

**Corollary 20 (RM 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

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

where \( \lambda = \max \{ \frac{c_i}{T_i} | i = 1, \ldots, n \} \).

The proof, which is given in [5], is similar to that of Theorem 18.

Analogously to Funk, Gossens, and Baruah\[7\], Andersson, Baruah, and Jonsson[2] 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:

© 2003 T.P. Baker. All rights reserved.
1. $\sum_{i=1}^{n} \frac{c_i}{T_i} \leq \frac{m^2}{3m-2}$

2. $\frac{n}{m} \leq \frac{m}{3m-2}$, for $1 \leq i \leq n.$

Then they proved the following theorem.

**Theorem 21 (Andersson, Baruah, Jonsson[2])**
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 20. If we take $\lambda = m/(3m - 2)$, it follows that the system of tasks is schedulable to meet deadlines if

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

Baruah and Goossens[6] proved the following similar result.

**Corollary 22 (Baruah, Goossens[6])** A set of tasks, all with deadline equal to period, is guaranteed to be schedulable on $m$ processors using RM scheduling if $\frac{n}{m} \leq 1/3$ for $i = 1, \ldots, n$ and $\sum_{i=1}^{n} \frac{c_i}{T_i} \leq 3m$.

This is a slightly weakened special case of our Corollary 20. For $\lambda = 1/3$, it follows that the system of tasks is schedulable to meet deadlines if

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

### 7 Ramifications

The theorems and corollaries above 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 the simpler forms, one computes $\lambda$ and then checks the schedulability condition once for the entire task set. With the more general forms, 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.

The schedulability tests of Theorems 12 and 18 allow preperiod deadlines, but are more complicated than the corresponding utilization bound tests. It is natural to wonder whether this extra complexity gains anything over the well known technique of “padding” execution times and then using the utilization bound test. By padding the execution times we mean that if a task $r_i$ has execution time $c_i$ and deadline $d_i < T_i$, we replace it by $r_i'$, where $c_i' = c_i + T_i - d_i$ and $d_i' = T_i' = T_i$.

With deadline monotonic scheduling, the original task $r_i$ can be scheduled to meet its deadline if the following condition holds for $r_i'$:

$$\frac{c_i'}{T_i'} + \sum_{i<k} \frac{c_i}{T_i} \leq \frac{m}{2} (1 - \lambda') + \lambda'$$

where $\lambda = \max_{i=1}^{n} \left( \frac{c_i'}{T_i'}, \lambda \right)$.

There are cases where this test is less accurate than that of Theorem 18. Suppose we have three proces-
sors and three tasks, with periods \( T_1 = T_2 = T_3 = 1 \),
deadlines \( d_1 = d_2 = 1, d_3 = 1/2 \), and compute
times \( c_1 = c_2 = c_3 = 1/4 \). Obviously, the tasks
are schedulable. However, if we apply the padded
utilization test to \( \tau_3 \), we have \( c_3 = 1/4 + 1/2 = 3/4 \),
and \( \lambda' = 3/4 \). The test fails, as follows:

\[
\sum_{i=1}^{2} \frac{1}{4} + \frac{3}{4} = \frac{10}{8} > \frac{3}{2}(1 - 3/4) + 3/4 = \frac{9}{8}
\]

On the other hand this set of tasks passes the test of
Theorem 18, as follows:

\[
\sum_{i=1}^{2} \frac{11}{32} + \frac{1}{2} = \frac{27}{32} < 3(1 - 1/4) = \frac{72}{32}
\]

A similar padding technique can be applied for
EDF, but again it is sometimes less accurate than
Theorem 12.

Of course, these schedulability tests are only sufficient
conditions for schedulability. They are like the
Liu and Layland \( n(2^{l/n} - 1) \) utilization bound, both
in being very conservative and in still having practical
value.

Though the utilization tests are not tight in the sense of
being necessary conditions for schedulability,
Goossens, Funk, and Baruah[8] showed that the
EDF bound is tight in the sense that there is no
utilization bound \( \bar{U} > m(1 - \lambda) + \lambda + \epsilon \),
where \( \epsilon > 0 \) and
\( \lambda = \max\{c_i/T_i \mid i = 0, \ldots, n\} \), for which \( \bar{U} \leq U \)
guarantees EDF schedulability. Since the EDF
utilization bound is tight it is natural to wonder whether
the same is true of the RM bound. The following theorem
shows that it is not tight.

**Theorem 23 (looseness of RM utilization bound)**

There exist task sets that are not feasible with pre-
emptive RM scheduling on \( m \) processors and have
utilization arbitrarily close to \( \lambda + m \ln(\frac{2}{1+\lambda}) \),
where \( \lambda \) 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 \( \lambda \).

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:

\[
T_{(i-1)*m+j} = p_i \quad \text{for} \quad 1 \leq i \leq k, 1 \leq j \leq m
\]

\[
c_{(i-1)*m+j} = p_{i+1} - p_i \quad \text{for} \quad 1 \leq i \leq k, 1 \leq j \leq m
\]

\[
T_n = p_{k+1}
\]

\[
c_n = T_n - 2 \sum_{i=1}^{k} (p_{i+1} - p_i)
\]

\[
= T_n - 2p_{k+1} - \sum_{i=2}^{k} p_i + 2 \sum_{i=2}^{k} p_i + 2p_1
\]

\[
= 2p_1 - T_n
\]

These constraints guarantee that task \( \tau_n \) barely
has time to complete if all \( n \) tasks are released to-
gether 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 \( \tau_n \).

If \( \frac{T_n}{\tau_n} = \lambda \), we have

\[
\lambda T_n = c_n = 2p_1 - T_n
\]

\[
T_n = \frac{2p_1}{1 + \lambda} = p_{k+1}
\]

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

\[
U = \lambda + \sum_{i=1}^{k} \frac{p_{i+1} - p_i}{p_i}
\]

\[
= \lambda + m \left( \sum_{i=1}^{k} \frac{p_{i+1}}{p_i} - k \right)
\]

The partial derivatives of \( U \) with respect to \( p_i \) are

\[
\frac{\partial U}{\partial p_1} = m\left( \frac{1}{\lambda} \right)
\]

\[
\frac{\partial U}{\partial p_i} = \frac{1}{\lambda} \quad \text{for} \quad 1 < i < k
\]

\[
\frac{\partial U}{\partial p_k} = \frac{2p_1}{(1 + \lambda)p_k^2}
\]

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

\[
\frac{p_{k+1}}{p_k} = \frac{2}{(1 + \lambda) p_k} = \frac{p_{k+1}}{p_k} \\
\frac{p_{i+1}}{p_i} = \frac{p_i}{p_{i-1}} \quad \text{for } 1 < i \leq k
\]

Let \( x = \frac{p_{k+1}}{p_k} = \cdots = \frac{p_i}{p_{i-1}} \). It follows that

\[
x^k = \prod_{i=1}^{k} \frac{p_{k+1}}{p_k} = \frac{p_{k+1}}{p_k} = \frac{2}{(1 + \lambda)}
\]

\[
x = \left( \frac{2}{1 + \lambda} \right)^\frac{1}{k}
\]

\[
U = \lambda + m(kx - k) = \lambda + mk \left( \left( \frac{2}{1 + \lambda} \right)^\frac{1}{k} - 1 \right)
\]

L'Hôpital's Rule can be applied to find the limit of the above expression for large \( k \), which is \( \lambda + m \ln \left( \frac{2}{1 + \lambda} \right) \). \( \square \)

We conjecture that the upper bound on the minimum achievable RM utilization achieved by the example above may be tight.

Srinivasan and Baruah[18] and Andersson, Baruah, and Jonsson[2] showed how to relax the restriction that \( c_i/T_i \leq \lambda \) in the utilization tests, for situations where there are a few high-utilization tasks. They propose EDF and RM versions of a hybrid scheduling policy called EDF/US[\( \lambda \)], where \( \lambda = m/(2m - 2) \) for EDF and \( \lambda = m/(3m - 2) \) for RM.

EDF/RM-US[\( \lambda \)]:

(heavy task rule) If \( c_i/T_i > \lambda \) then schedule \( \tau_i \)'s jobs at maximum priority.

(light task rule) If \( c_i/T_i \leq \lambda \) then schedule \( \tau_i \)'s jobs according to their normal EDF or RM priorities.

They then proved two theorems, which we paraphrase and combine as follows:

**Theorem 24 (SB[18] & ABJ[2])** Algorithm EDF/RM-US[\( \lambda \)] correctly schedules on \( m \) processors any periodic task system with total utilization \( U \leq m \lambda \).

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 the algorithm can do no worse than scheduling each of the heavy tasks on its own processor, and then scheduling the remainder (which must be light on the remaining processors) using the regular EDF or RM algorithm.

The above result can be generalized slightly, as follows:

**Theorem 25** Algorithm EDF/RM-US[\( \lambda \)] correctly schedules on \( m \) processors any periodic task system such that only \( k \) tasks \((0 \leq k \leq m)\) have utilization greater than \( \lambda \) and the utilization of the remaining tasks is at most \((m - k)(1 - \lambda) + \lambda m \) for EDF and at most \(((m - k)/2)(1 - \lambda) + \lambda m \) for RM.

**Proof:** As argued by Srinivasan and Baruah, the performance of this algorithm cannot be worse than an algorithm that dedicates one processor to each of the heavy tasks, and uses EDF or RM to schedule the remaining tasks on the remaining processors. The utilization bound theorem then guarantees the remaining tasks can be scheduled on the remaining processors. \( \square \)

If there is a need to support preperiod deadlines, this idea can be taken further, by changing the “heavy task rule” to single out for special treatment a few tasks that fail the test conditions of one of our schedulability tests that allows preperiod deadlines, and run the rest of the tasks using EDF or DM scheduling.

### 8 Conclusion and Future Work

We have demonstrated simple schedulability tests for EDF and DM scheduling on a homogeneous multiprocessor system with preperiod deadlines. These can be applied statically, or applied dynamically as an admission test. Besides extending and generalizing previously known utilization-based tests for EDF and RM multiprocessor schedulability by supporting pre-period deadlines, we also provide a distinct and independent proof technique.

In future work, we plan to look at how the utilization bounds presented here for dynamic processor assignment bear on the question of whether to use static or dynamic processor assignment[1, 15, 12]. We have
some experience dating back to 1991[17]) with an implementation of a fixed-priority multiprocessor kernel with dynamic migration of tasks. However, that experience is now out of date, due to advances in processor architectures that have increased the penalty for moving a task between processors. That penalty has not been taken into account in the present. A more complete analysis will require consideration of the task migration penalty.

References


