An Analysis of EDF Schedulability on a Multiprocessor

FSU Computer Science Technical Report TR-030202

Theodore P. Baker

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

7 February 2003*

Abstract

A schedulability test is presented for preemptive deadline scheduling of periodic or sporadic real-time tasks on a single-queue m-server system. The new condition does not require that the task deadline be equal to the task period. This generalizes and refines a utilization-based schedulability test of Goossens, Funk, and Baruah, and also provides an independent and different proof.

1 Introduction

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

It has long been known that preemptive earliest-deadline-first (EDF) scheduling is optimal for single-processor batch-oriented systems, in the sense that a set of independent jobs with deadlines is feasible if and only if it is EDF-schedulable. In 1973 Liu and Layland[15] extended this result to systems of independent periodic tasks for which the relative deadline of each task is equal to its period, proving that so long as the total processing demand does not exceed 100 percent of the system capacity EDF will not permit any missed deadlines. Besides showing that that EDF scheduling is optimal for such task systems, this utilization bound provides a simple and effective *a priori* test for EDF schedulability.

It also well known that the optimality of EDF scheduling breaks down on multiprocessor systems[16]. For example, consider a batch of jobs $J_1, \ldots, J_{m+1}$ with deadlines $d_1 = mx, \ldots, d_m = mx, d_{m+1} = mx + 1$, and compute times $c_1 = 1, \ldots, c_m = 1, c_{m+1} = mx + 1$, where x is an arbitrary constant, all released at time zero. If job J_{m+1} is allowed to monopolize one processor, all the tasks can be completed by their deadlines using m processors, as shown in Figure 1. However, an EDF scheduler for m processors would schedule jobs $J_1 \ldots J_m$ ahead of J_{m+1}, causing J_{m+1} to miss its deadline, as shown in Figure 2.

The example above generalizes immediately to a set of periodic tasks. Each batch job is replaced by a periodic task with period T_i and relative deadline d_i equal to the deadline of the corresponding batch job. If we do this

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

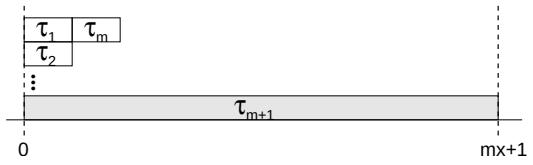

Figure 1: All jobs can be scheduled if J_{m+1} is given its own processor.

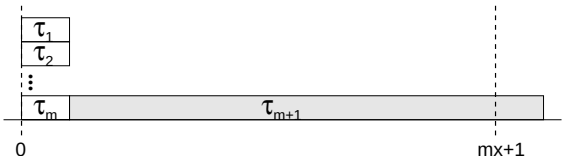

Figure 2: J_{m+1} misses deadline with EDF scheduling.

the total utilization U of the task set is

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

U can be made arbitrarily close to one by choosing x sufficiently large. That is, all but one of the m processors will be idle almost 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 scheduling, and even that EDF is not a good real-time scheduling policy for multiprocessor systems. However, neither conclusion is actually justified.

In the example there are two kinds of tasks: "heavy" ones, with high ratio of compute time to deadline, and "light" ones, with low ratio of compute time to deadline. It is the mixing of those two kinds of tasks that causes a problem for EDF. A modified EDF policy, that segregates the heavy tasks from the light 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 EDF 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 that above, Srinivasan and Baruah[18] examined the deadline-based 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 $m/(2m-1)$ can be scheduled successfully on m processors if the total utilization is at most $m^2/(2m-1)$. They then proposed a modified EDF scheduling algorithm that gives higher priority to tasks with utilizations above $m/(2m-1)$, which is able to successfully schedule *any* set of independent periodic tasks with total utilization up to $m^2/(2m-1)$. Srinivasan and Baruah derived these results indirectly, using a theorem relating EDF schedulability on sets of processors of different speeds by Phillips, Stein, Torng, and Wein[17]. Goossens, Funk, and Baruah [8] generalized these results, showing that a system of independent periodic tasks can be scheduled successfully on m processors by EDF scheduling if the total utilization is at most $m(1-u_{max})+u_{max}$, where u_{max} is the maximum utilization of any individual task.

We have approached the problem in a different and more direct way. This leads to a different and more general schedulability condition, of which the above cited results turn out to be special cases. The rest of this paper presents the derivation of this more general multiprocessor EDF schedulability condition, working through the

stages by which we discovered it, followed by a discussion of its applications and an explanation of how it relates to 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 (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.

Note that what we call a periodic task here is sometimes called a sporadic task. In this regard we follow Jane Liu[16], 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 preemptively according to an EDF 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 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. We will consider a first failure of scheduling for a given task set. That is, consider 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 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 a *problem task*, the job of τ_k released at time t a *problem job*, and the time interval $[t, t + d_k)$ a *problem window*.

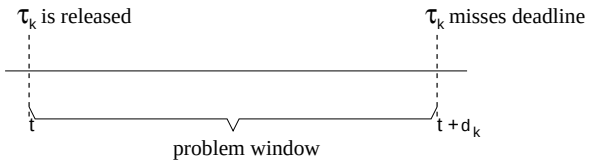

Figure 3: A problem window.

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/Δ*, 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 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 load of a problem window can be established using the following well known argument, which is also used by [17]:

> 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, $d_k - c_k$.

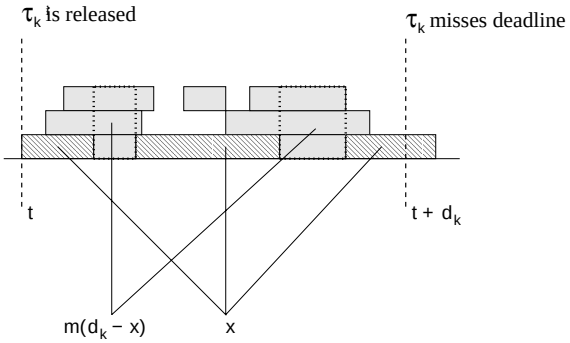

Figure 4: All processors must be busy whenever τ_k is not executing.

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 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 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 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 τ_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.□

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.*[17].

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 $\beta \leq m - (m-1)c_k/d_k$ is sufficient

to guarantee schedulability. The upper bound β on W/Δ is the sum of individual upper bounds β_i on the *load contribution* W_i/Δ for each individual task in the window. It then follows that

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

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, which we call a *window of interest*, or just the *window* for short.

For any task τ_i that can execute in the 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 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 $t' = t - \phi$, $t < t' + T_i < t + T_i$, $0 \le \phi < T_i$. 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.

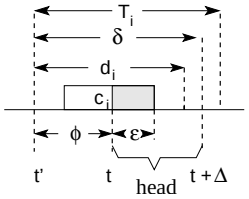

Figure 5: Window with head only ($\phi + \Delta \le T_i$).

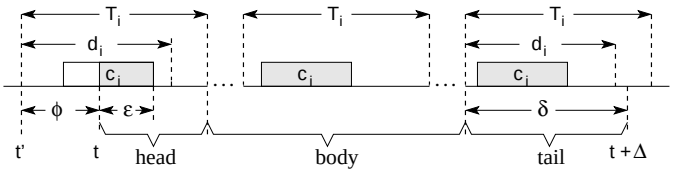

Figure 6: Window with head, body, and tail.

Figures 5 and 6 show windows with carried-in jobs. The release time of the carried-in job is $t' = t - \phi$, where ϕ is the offset of the release time from the beginning of the window. If the minimum interrelease time 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.

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

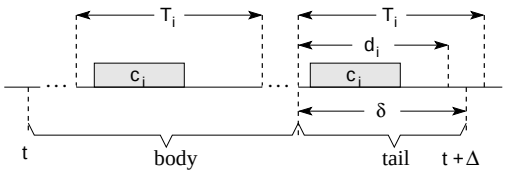

Figure 7: Window with body and tail only ($\phi = 0$).

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 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 execution 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.

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 load of the interval $[t', t)$ is at least $(m-1)(\phi - c_i + \epsilon)/\phi + 1$.*

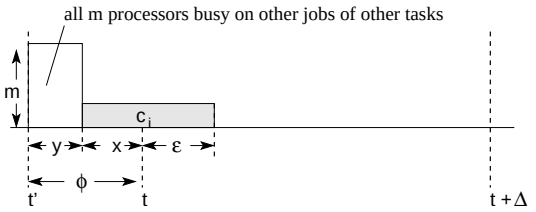

Figure 8: Carry-in depends on competing demand.

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. By definition, $\epsilon = c_i - x$. 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 load of the interval $[t', t' + \phi)$ 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., [15, 3, 10, 14]) 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 compete with τ_k is carried into the busy interval. By modifying the definition of busy interval slightly, we can also apply it here.

Definition 6 (busy intervals) *A time interval is* λ-busy *if its combined load is at least* $m(1-\lambda)+\lambda$ *and there are no missed deadlines prior to the end of the interval. 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* 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 a 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. Let* $t - \phi$ *be the last release time of* τ_i*, where* $i \neq k$*, 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 its next release time is 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, let $n = \lfloor (\Delta - d_i)/T_i \rfloor + 1$ if $\Delta \geq d_i$ and $n = 0$ otherwise. The demand W_i of τ_i in the busy window is no greater than*

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

where $\phi = nT_i + d_i - \Delta$.

Proof:

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, and narrow down the set of possibilities by stages, only the worst case is left.

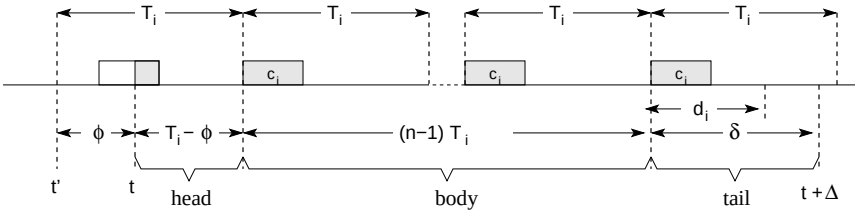

Figure 9: Dense packing of jobs.

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 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 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 τ_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 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 may reduce the contribution to W_i of the body and tail. This is a trade-off that we will analyze further.

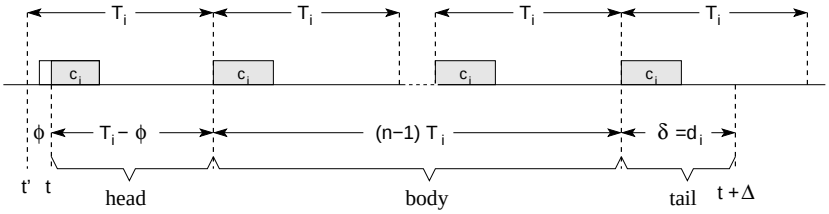

Figure 10: Densest possible packing of jobs.

Looking at Figure 10, we see that the length of the head, $T_i - \phi$, cannot be larger than $\Delta - ((n-1)T_i + d_i)$ without pushing all of the final execution of τ_i outside the window. Reducing ϕ 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 we expect the value of W_i to be maximized for $\phi = nT_i + d_i - \Delta$.

To make this reasoning more precise, consider the situation where $\delta = d_i$ and $\phi = nT_i + d_i - \Delta$, 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. So, 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 deadline of the last job released in the window, so that the deadline is outside the window and the job no longer contributes anything to W_i, resulting in a sudden reduction of W_i by c_i. At the same, time ϵ may increase by at most $\sigma\lambda$. This increase in ϵ is necessarily less than c_i. Therefore, there is no way that reducing ϕ below $\phi = nT_i + d_i - \Delta$ can increase the value of W_i.

We have shown that if $\Delta \geq d_i$ the value of W_i is maximized when $\delta = d_i$ and $\phi = nT_i + d_i - \Delta$, and this maximum is no greater than

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

It is now time to consider the special case where $\Delta < d_i$. In this case there can be no body or tail contribution, since it is impossible for a job of τ_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 $n = \lfloor \frac{\Delta - d_i}{T_i} \rfloor + 1 = 0$ 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 load) *For any λ-busy window $[t, t+\Delta)$ with respect to τ_k the load W_i/Δ due to τ_i is at most β_i, where*

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

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

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

Let α 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, depending on whether $\max\{0, c_i - \phi\lambda\} = 0$.

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

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

$$\begin{aligned} n = \lfloor \frac{\Delta - d_i}{T_i} \rfloor + 1 &\leq \frac{\Delta - d_i}{T_i} + 1 = \frac{\Delta - d_i + T_i}{T_i} \\ \alpha(\Delta) &= \frac{nc_i}{\Delta} \\ &\leq \frac{c_i}{T_i}\frac{\Delta - d_i + T_i}{\Delta} \\ &\leq \frac{c_i}{T_i}(1 + \frac{T_i - d_i}{\Delta}) \end{aligned}$$

Since $\Delta \geq d_k$, it follows that

$$\alpha(\Delta) \leq \frac{c_i}{T_i}(1 + \frac{T_i - d_i}{d_k}) = \beta_i \tag{2}$$

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

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

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

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

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

Since $\Delta \geq d_k$, we have

$$\alpha(\Delta) \leq \frac{c_i}{T_i}(1 + \frac{T_i - d_i}{d_k}) + \frac{c_i - \lambda T_i}{d_k} = \beta_i \tag{3}$$

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

From the definition of n, it follows that

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

Since $\Delta \geq d_k$, we have

$$\alpha(\Delta) < \frac{c_i}{T_i}(1 + \frac{T_i - d_i}{d_k}) = \beta_i \tag{4}$$

Observe that inequalities (2) and (4) cover the cases where $\lambda \geq \frac{c_i}{T_i}$, and that inequality (3) covers the case where $\lambda < \frac{c_i}{T_i}$ in the definition of β_i. □

5 Schedulability Condition

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

Theorem 12 (EDF schedulability test) *A set of periodic tasks $\tau_1, \ldots, \tau_n$ is schedulable on m processors using preemptive EDF scheduling if, for every task τ_k,*

$$\sum_{i=1}^{n} \min\{1, \beta_i\} \leq m(1 - \frac{c_k}{d_k}) + \frac{c_k}{d_k} \tag{5}$$

where β 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 (5).

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 $i = 0, \ldots, n$. 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 (5).□

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 EDF scheduling if*

$$\sum_{i=1}^{n} \min\{1, \frac{c_i}{T_i}(1 + \frac{T_i - d_i}{d_{min}})\} \quad \leq \quad m(1 - \lambda) + \lambda \tag{6}$$

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

Proof:

Corollary 13 is proved by 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. By Lemma 7, we know that $[t, t + \Delta)$ is $\frac{c_k}{d_k}$-busy, and so

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

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.*, $\beta_i = \frac{c_i}{T_i}(1 + \frac{T_i - d_i}{d_k})$. It follows that

$$\begin{aligned} \sum_{i=1}^{n} \frac{c_i}{T_i}(1 + \frac{T_i - d_i}{d_{min}}) &= \sum_{i=1}^{n} \min\{1, \beta_i\} \\ &\geq \frac{W}{\Delta} \\ &> m(1-\lambda) + \lambda \end{aligned}$$

The above contradicts (6).□

6 Relation to Prior Work

Srinivasan and Baruah[18] 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. $\sum_{i=1}^{n} \frac{c_i}{T_i} \leq \frac{m^2}{2m-1}$
2. $\frac{c_i}{T_i} \leq \frac{m}{2m-1}$, for $1 \leq i \leq n$.

They then proved the following theorem:

Theorem 14 (Srinivasan and Baruah[18]) *Any periodic task system that is light on m processors is scheduled to meet all deadlines on m processors by EDF.*

Goossens, Funk, and Baruah[8] refined the above work, showing the following:

Theorem 15 (Goossens, Funk, and 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 $\lambda = \max\{c_i/T_i \mid i = 1, \ldots, n\}$.

Their proof is derived from a theorem in [7], on scheduling for uniform multiprocessors, which in turn is based on [17]. They then showed that the above utilization bound is tight, in the sense that there is no utilization bound $\hat{U} > m(1-\lambda) + \lambda + \epsilon$, where $\epsilon > 0$ and $\lambda = \max\{c_i/T_i \mid i = 0, \ldots, n\}$, for which $U \leq \hat{U}$ guarantees EDF schedulability.

We have an independent proof of the above utilization bounds as a special case of Corollary 13, by replacing d_i by T_i. We have gone beyond the above results of [18] and [8] in the following ways:

1. We have provided an independent direct analysis of EDF schedulability, that does not depend on the Phillips *et al.* speedup result[17].
2. Theorem 12 and Corollary 13 can be applied to tasks with pre-period deadlines. This is important for systems were some tasks have bounded jitter requirements.

Srinivasan and Baruah[18] proposed adapting their utilization-bound schedulability condition to situations where there are a few high-utilization tasks in the form of the following priority scheduling algorithm, which they call EDF-US$[m/(2m-1)]$:

(heavy task rule) If $c_i/T_i > m/(2m-1)$ then schedule τ_i's jobs at maximum priority (*e.g.*, deadline = 0).

(light task rule) If $c_i/T_i \leq m/(2m-1)$ then schedule τ_i's jobs according to their normal deadlines.

They then proved the following theorem:

Theorem 16 (Srinivasan and Baruah[18]) *Algorithm EDF-US$[m/(2m-1)]$ correctly schedules on m processors any periodic task system whose utilization is at most $m^2/(2m-1)$.*

Their proof is based on the observation that their upper bound on total utilization guarantees that the number of heavy tasks cannot exceed m. The essence of the argument is that Algorithm EDF-US$[m/(2m-1)]$ 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 EDF.

Theorem 12 and Theorem 15 suggest the above hybrid scheduling algorithm can be generalized further. For example, Algorithm EDF-US$[m/(2m-1)]$ can be generalized as follows:

Algorithm EDF-US$[\lambda]$

(heavy task rule) If $c_i/T_i > \lambda$ then schedule τ_i's jobs at maximum priority (*e.g.*, deadline = 0).

(light task rule) If $c_i/T_i \leq \lambda$ then schedule τ_i's jobs according to their normal deadlines.

Theorem 17 *Algorithm EDF-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 λ and the utilization of the remaining tasks is at most $(m-k) - ((m-k)-1)\lambda$.*

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 to schedule the remaining tasks on the remaining processors. Theorem 15 then guarantees the remaining tasks can be scheduled on the remaining processors. □

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 Theorem 12 or Corollary 13, and run the rest of the tasks using EDF scheduling.

7 Conclusion

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

The notion of λ-busy interval, used to derive an EDF schedulability condition here, has broader applications. In[4] we use the same approach to improve on the utilization bound result of Andersson, Baruah, and Jonsson[1] for rate monotone scheduling on multiprocessors.

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] J. Anderson, A. Srinivasan, "Early-release fair scheduling", *Proceedings of the 12th Euromicro Conference on Real-Time Systems* (June 2001) 34-43.

[3] 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).

[4] T.P. Baker, "An analysis of deadline-monotonic scheduling on a multiprocessor", FSU Department of Computer Science technical report (2003). 67-100. (Reprinted in *Advances in Real-Time Systems*, IEEE Computer Society Press (1993) 64-96).

[5] S. Baruah, N. Cohen, C.G. Plaxton, D. Varvel, "Proportionate progress: a notion of fairness in resource allocation", *Algorithmica 15* (1996) 600-625.

[6] S. Baruah, J. Gherke, C.G. Plaxton, "Fast scheduling of periodic tasks on multiple resources", *Proceedings of the 9th International Parallel Processing Symposium* (April 1995) 280-288.

[7] S. Funk, J. Goossens, S. Baruah, "On-line scheduling on uniform multiprocessors", *Proceedings of the IEEE Real-Time Systems Syposium*, IEEE Computer Society Press (December 2001).

[8] J. Goossens, S. Funk, S. Baruah, "Priority-driven scheduling of periodic task systems on multiprocessors", technical report UNC-CS TR01-024, University of North Carolina Computer Science Department; extended version to appear in *Real Time Systems*, Kluwer.

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

[10] 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.

[11] 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).

[12] 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).

[13] 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).

[14] 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.

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

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

[17] 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.

[18] A. Srinivasan, S. Baruah, "Deadline-based scheduling of periodic task systems on multiprocessors", *Information Processing Letters 84* (2002) 93-98.

