EDF scheduling on multiprocessor platforms: some (perhaps) counterintuitive observations

Joël Goossens

Department of Computer Science
Université Libre de Bruxelles
Brussels, Belgium

Shelby Funk and Sanjoy Baruah

Department of Computer Science
University of North Carolina
Chapel Hill, North Carolina 27599–3175

Abstract

The earliest-deadline-first scheduling of hard real-time systems upon multiprocessor platforms is considered. Several results that run somewhat counter to intuition are presented. With respect to interprocessor migrations, it is shown that what seem like intuitive heuristics for decreasing the number of such migrations may in fact severely exacerbate the problem. With respect to scheduling periodic task systems, it is demonstrated that uniprocessor results which permit the efficient identification of worst-case system behavior do not extend to multiprocessors.

1 Introduction

In **hard-real-time** (HRT) systems, there are certain basic units of work, known as **jobs**, which must be executed in a timely manner. In one popular model of hard-real-time systems, each job is assumed to be characterized by three parameters – an *arrival time*, an *execution requirement*, and a *deadline*, with the interpretation that the job must be executed for an amount equal to its execution requirement between its arrival time and its deadline.

Given the specifications of a HRT system, *feasibility analysis* is the process of determining whether the system can be executed in such a manner that all jobs do indeed complete by their deadlines. If a HRT is deemed feasible, then a *run-time scheduling algorithm* is responsible for determining, during the execution of the system, which jobs[s] should execute at each instant in time such that all job deadlines are indeed met.

A HRT system is said to be *preemptive* if it is permitted that any executing job may be interrupted at any instant in time, and its execution resumed later, with no additional cost or penalty. The *earliest deadline first scheduling algorithm* (EDF) is a very popular run-time scheduling algorithm for scheduling preemptive HRT systems. There are several reasons for this:

- EDF is known to be an **optimal** scheduling algorithm in uniprocessor systems: any preemptive HRT system feasible on a uniprocessor will meet all deadlines if scheduled using EDF.
- Very efficient implementations of EDF have been designed (see, e.g., [11]).
- It has been shown that when a set of jobs is scheduled using EDF, then the total number of *preemptions* experienced can be bounded from above.

With respect to *multiprocessor* systems, it has been shown [2, 8] that no on-line scheduling algorithm can be optimal. Nevertheless, EDF remains a predictable and resource-efficient algorithm to use in multiprocessor systems – we have recently [3, 7] formally justified this assertion, by obtaining conditions which permit us to ensure that a task system can be scheduled to meet all deadlines using EDF upon a given multiprocessor platform, provided we know it to be feasible upon some other multiprocessor platform. In addition, the other two advantages of the algorithm (efficient implementations, and bounded preemptions) remain true; and furthermore, it can be shown that in EDF-scheduled multiprocessor systems the total number of *interprocessor migrations* of individual jobs can also be bounded from above.

Encouraged by this evidence demonstrating that EDF tends to behave favorably in multiprocessor systems, we have been working on further enhancing our understanding of multiprocessor EDF scheduling. In doing so we have identified several aspects of EDF behavior upon multiprocessor platforms that are somewhat unexpected and counterintuitive. In addition to being, in our opinion, interesting in their own right, such anomalous behavior makes it difficult to extend uniprocessor results to the multiprocessor case. We believe that these problems must be studied and understood thoroughly, if we are to indeed develop a complete theory of multiprocessor scheduling that is as useful and comprehensive as uniprocessor scheduling theory is today. In this document, we will discuss two particular problems with multiprocessor scheduling:

- When executing EDF upon a multiprocessor platform in which all processors are not of the same speed (such multiprocessors are called *uniform* multiprocessors – Section 2), how do we decide which job we should execute on which processor at each instant in time, in order that the overall number of interprocessor migrations may be minimized?

- How do we determine whether a given *periodic task system* [10] can be scheduled by EDF upon a multiprocessor platform comprised of several identical processors, such that all jobs meet their deadlines?

With respect to the first of these questions, we show that the intuitive solution – the implementation of EDF that we expected would minimize the number of interpreocessor migrations – turns out to not always be the correct one. With respect to the second question, we have discovered, somewhat unexpectedly, that techniques that work for uniprocessors (the concept of a *critical instance* of job arrivals; the fact that *synchronous job arrivals* represent the worst case) do not generalize to this multiprocessor problem. Hence the problem of determining a necessary and sufficient test for determining whether a given periodic task system will be successfully scheduled by EDF upon a particular multiprocessor platform remains open.

2 Definitions

2.1 Job model

We will assume that a hard-real-time system may be modelled as an arbitrary collection of individual **jobs**. Each **job** $J = (r, c, d)$ is characterized by an arrival time r, an execution requirement c, and a deadline d, with the interpretation that this job needs to execute for c units over the interval $[r, d)$.

Periodic tasks. Periodic task systems are a particular kind of hard-real-time system. The periodic task model has proven very useful for the modelling and analysis of real-time computer application systems. In this model, each recurring real-time process is modelled as a **periodic task**, and is characterized by four parameters – an **execution requirement**, a **deadline delay**, an **offset** and a **period**. Each such periodic task generates an infinite sequence of jobs, which need to be executed by the system. A periodic task $\tau = (e, d, o, p)$ with execution-requirement parameter e, a deadline delay d, an offset o and period parameter p generates an infinite sequence of jobs $J_k = (o + k \cdot p, e, o + k \cdot p + d)$, $k = 0, 1, 2, \ldots$ (i.e., it generates a job at each instant $o + k \cdot p$, which needs to execute for e units by a deadline of $o + k \cdot p + d$, for all non-negative integers k). Interesting sub-cases are **synchronous systems**, where all tasks are started at the same time (otherwise the system is said to be **asynchronous**, or **offset free** if the offsets—i.e., the times at which the first requests occur—are not fixed by the problem but may be chosen by the scheduler [5, 4]); **implicit deadline** systems, where each deadline coincides with the period (i.e., each request must simply be completed before the next request of the same task occurs)[1]; **constrained deadline** systems, where the deadlines are not greater than the periods and **arbitrary deadline** systems, where no constraint exists between the deadline and the period (notice that if the deadline is greater than the period, many requests of a single task may be simultaneously active, even if the system is feasible, i.e., all deadlines are met). The *utilization* of a periodic task is defined to be the ratio of its execution requirement to its period: the utilization u of task $\tau = (e, d, o, p)$ equals $\frac{e}{p}$.

2.2 Multiprocessors

In **multiprocessor** platforms there are several processors available upon which jobs may execute. In this paper, we will be studying the scheduling of hard-real-time systems on multiprocessor platforms. We assume that *job migration is permitted:* that is, a job that has been preempted on a particular processor may resume execution on a different processor. Nevertheless, we would like to limit the number of such interprocessor migrations in the schedules that we generate. We will also assume that *job parallelism is forbidden:* each job may execute on at most one processor at any given instant in time.

In this paper, we will study two kinds of multiprocessor machines. In **identical** multiprocessors, all the processors are identical in the sense that they have the same computing power. By contrast, each processor in a **uniform** multiprocessor is characterized by its own computing capacity, with the interpretation that a job that executes on a processor of computing capacity s for t time units completes $s \times t$ units of execution. (Observe that identical multiprocessors are a

[1] With respect to *synchronous, implicit-deadline* systems, we note that such tasks are often represented by an ordered pair of just two parameters: $\tau = (e, p)$, with the first parameter representing the execution requirement and the second, the period.

special case of uniform multiprocessors, in which the computing capacities of all the processors are equal.)

With respect to uniform multiprocessors, it is necessary to define some additional notation:

Definition 1 Let π denote an m-processor multiprocessor platform with processor speeds $s_1, \ldots, s_m$, where $s_i \geq s_{i+1}$ for $i = 1, \ldots, m-1$.

$m(\pi)$**:** denotes the number of processors in π: $m(\pi) \stackrel{\text{def}}{=} m$.

$s_i(\pi)$**:** denotes the speed of the i^{th} fastest processor of π: $s_i(\pi) \stackrel{\text{def}}{=} s_i$.

$S(\pi)$**:** denotes π's cumulative processing power: $S(\pi) \stackrel{\text{def}}{=} \sum_{i=1}^{m(\pi)} s_i(\pi)$.

$\lambda(\pi)$**:** We define an additional parameter $\lambda(\pi)$ as follows:

$$\lambda(\pi) \stackrel{\text{def}}{=} \max_{1 \leq k < m(\pi)} \frac{\sum_{i=k+1}^{m(\pi)} s_i(\pi)}{s_k(\pi)} . \qquad (1)$$

■

The parameter $\lambda(\pi)$ needs some explanation. Intuitively, it measures the degree of "identicalness" of π — the closer π is to being an identical multiprocessor (one in which all processors are the same), the larger the value of $\lambda(\pi)$. This value has an upper bound of $(m(\pi) - 1)$, which occurs when π consists of $m(\pi)$ identical processors. At the other extreme, $\lambda(\pi)$ is arbitrarily small when the processors have very different speeds. For example, $\lambda(\pi) < \epsilon$ for the $m(\pi)$-processor platform where $s_{i+1}(\pi) \leq \frac{\epsilon}{m(\pi)} \cdot s_i(\pi)$ for $i = 1, \ldots, m(\pi) - 1$.

Some further definitions:

- A hard-real-time system I is said to be **feasible** upon a (uniprocessor or multiprocessor) platform π if there exists some schedule of I upon π in which every job in I meets its deadline.
- If A is an algorithm which schedules I on platform π to meet all its deadlines, then we say that I is **A-schedulable** upon π.

3 EDF upon uniform multiprocessors

In [3], we studied the EDF scheduling of HRT systems upon uniform multiprocessors. EDF was originally defined for *uni*processors, and while the uniprocessor definition extends in a straightforward manner to identical multiprocessors, defining EDF for *uniform* multiprocssors involved more effort. We now briefly present the defintion we adopted in [3].

EDF on uniform processors. Recall that the earliest deadline first scheduling algorithm (EDF) chooses for execution at each instant in time the currently active job[s] that have the smallest deadlines. In [3], we assumed that EDF is implemented upon uniform multiprocessor systems according to the following rules:

1. No processor is idled while there is an active job awaiting execution.
2. When fewer than m jobs are active, they are required to execute upon the fastest processors while the slowest are idled.
3. Higher priority jobs are executed on faster processors. More formally, if the j'th-slowest processor is executing job J_g at time t under our EDF implementation, it must be the case that the deadline of J_g is not greater than the deadlines of jobs (if any)executing on the $(j+1)$'th-, $(j+2)$'th-, $\ldots$,, m'th-slowest processors.

With this definition of EDF, we obtained the following results concerning the EDF-scheduling of HRT systems upon uniform multiprocessors:

Theorem 1 ([3]) Let I denote an instance of jobs that is feasible on a uniform multiprocessor platform π. Let π' denote another uniform multiprocessor platform. If the following condition is satisfied by platforms π and π':

$$S(\pi') \geq \lambda(\pi') \cdot s_1(\pi) + S(\pi) \qquad (2)$$

then I will meet all deadlines when scheduled using the EDF algorithm executing on π'.

■

Theorem 2 ([3]) Consider a set $\{\tau_1, \ldots, \tau_n\}$ of implicit-deadline periodic tasks (i.e., each task has its deadline parameter equal to its period) indexed according to non-increasing utilization (i.e., $u_i \geq u_{i+1}$ for all i, $1 \leq i < n$, where $u_i \stackrel{\text{def}}{=} \frac{e_i}{p_i}$). Let $U_n \stackrel{\text{def}}{=} \sum_{j=1}^{n} u_j$. Periodic task system τ will meet all deadlines when scheduled on π using EDF, if the following condition holds

$$S(\pi) \quad \geq \lambda(\pi) \cdot u_1 + U_n. \qquad (3)$$

■

One criticism of the results in [3] concerns our choice of definition for EDF. While there should be little controversy about the choice of *which* jobs to execute at each instant in time — by very definition of EDF, these are necessarily the earliest-deadline jobs — the decision on which of the selected jobs to execute on which *processor* is not quite so obvious.

Recall that one of the claimed advantages of EDF on multiprocessors was the bound upon the number of preemptions and interprocessor migrations — upon identical multiprocessors, this number is bounded from above at the number of jobs. With uniform multiprocessors and the above working definition of EDF, this bound no longer holds — indeed, the number of preemptions and interprocessor migrations may approach the product of the number of jobs and the number of processors, as the following example illustrates.

Example 3 Consider a uniform multiprocessor platform comprised of m processors, with the j'th processor having computing capacity $(m - j + 1)$, for $j = 1, 2, \ldots, m$. Consider an HRT instance of n jobs (where $n > m$), with attributes as follows:

- Jobs $J_1, \ldots, J_m$ all arrive at time-instant zero; job J_k arrives at time-instant $(k - m)$, for $k = m + 1, \ldots, n$.
- Job J_k has execution requirement $(\sum_{\ell=m-k+1}^{m} \ell)$ for $k = 1, \ldots, m$; job J_k has execution requirement $(\sum_{\ell=1}^{m} \ell)$ for $k = m + 1, \ldots, n$.
- Job J_k has deadline $k \cdot m$ for all k, $1 \leq k \leq n$.

EDF would assign the jobs priorities according to indices: job J_k would have greater priority than job J_{k+1} for all k. Hence, the j'th fastest processor would begin executing job J_j at time-instant 0; at time-instant 1, job J_1, which has an execution requirement of m units, would complete execution on the fastest processor and the job on each processor would migrate to the next-fastest processor while job J_{m+1} would begin execution on the slowest processor; at time-instant 2, job J_2 would complete execution on the fastest processor and the job on each processor would migrate to the next-fastest processor while job J_{m+2} would begin execution on the slowest processor; and so on. This scenario repeats at each successive time-instant: the job on the fastest processor completes execution, causing a cascade of migrations and the introduction of a new job on the slowest processor. As $n \to \infty$, the number of interprocessor migrations therefore approaches $(n \times (m - 1))$ — the product of the number of jobs and the number of processors minus one; alternatively, each job ends up undergoing an average of $m - 1$ migrations.

■

Thus, while we can still derive an upper bound upon the number of preemptions and interprocessor migrations, this bound is quite a bit higher than in the case of uniprocessors or identical multiprocessors.

In an attempt to be able to have fewer interpocessor migrations, we considered different possible implementations of EDF upon uniform multiprocessors: these implementations differ from one other in that they migrate jobs under different conditions. These different possible implementations are informally described below, and are illustrated in Figure 1 by considering the scheduling of a job set, indexed according to EDF priority, upon a 4-processor uniform multiprocessor with processor-speeds are $s_1 = 4$, $s_2 = 3$, $s_3 = 2$ and $s_4 = 1$.

- The **greedy EDF** scheduling algorithm, or EDF-G, is the version of EDF we considered above. EDF-G always assigns earlier-deadline jobs to faster processors. In this case, jobs will migrate under two conditions. Firstly, when a new job J arrives with higher priority than some currently-executing job, jobs with lower priority will migrate to slower processors allowing J to begin execution on the appropriate processor. Secondly, if a job J completes execution and there are lower priority jobs executing, these jobs will migrate to faster processors. If all the processors were in use and there were jobs awaiting execution when J completed execution, then the highest priority job awaiting execution will be allowed to execute on the slowest processor. Note that a job arrival or completion may result in a cascade of migrations. This can be seen at $t = 3$ in Figure 1 (a), and in Example 3 above.

- The **migration-balancing EDF** scheduling algorithm, or EDF-MB, always ensures that the idling processors are the slowest processors. Migrations only occur when a job J completes execution on a processor P and there are jobs executing on processors slower than P. When a new job arrives and there is at least one idling processor, the job will be assigned to the fastest idling processor. If there are no idling processors, the newly arriving job will preempt the lowest-priority executing job if it has higher priority — otherwise, it will wait for a processor to become available. When a job completes execution, jobs executing on slower processors will migrate according to priority. The highest priority job running on a processor slower than p will migrate to the newly available processor. If this job was not executing on the slowest processor, then the highest priority job execut-

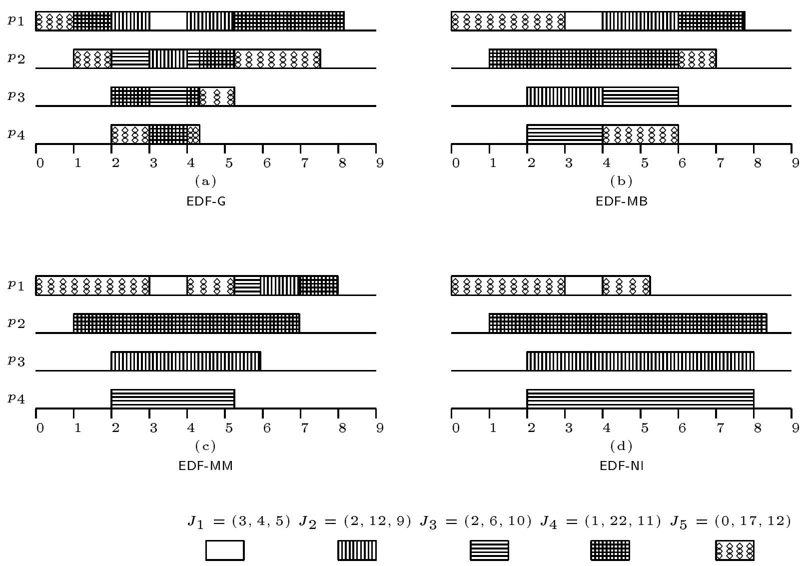

Figure 1: Different types of EDF on uniform multiprocessors.

ing on a slower processor will migrate and so on until the job executing on the slowest processor migrates to the newly available processor. This is illustrated at $t = 4$ in Figure 1 (b).

- The **migration-minimizing EDF** scheduling algorithm, or EDF-MM, also ensures that the idling processors are the slowest ones. EDF-MM uses the same algorithm as EDF-MB when a new job arrives, causing no migrations. When an active job completes execution on processor P and there are jobs awaiting execution, the highest priority waiting job will begin execution on P. If no jobs are waiting and there exists at least one active processor slower than P, the job that was executing on the slowest active processor will migrate to processor P. Thus, a job completion will cause at most one migration. This algorithm is illustrated in Figure 1 (c).
- The **non-idling EDF** scheduling algorithm, or EDF-NI, never migrates jobs to take advantage of fast idling processors. This algorithm is illustrated in Figure 1 (d).

As the names *migration balancing* and *migration minimizing* indicate, these algorithms are designed to reduce the number of migrations that occur in a typical schedule. Indeed, while EDF-G may initiate migrations upon job arrivals *and* completions, EDF-MB and EDF-MM only initiate migrations upon job completions. Our initial expectation was that any HRT system scheduled using these variants of EDF — EDF-MB, EDF-MM, and EDF-NI — would always result in fewer preemptions than if it were scheduled using the EDF-G implementation (or "traditional" EDF); our goal was to determine whether this decrease in the number of preemptions was worth the expected loss in resource-utilization that results from the greater restrictions placed upon the system. Somewhat to our surprise, however, this supposition proved false: as the following theorem shows, what seem like intuitive heuristics for decreasing the number of interprocessor migrations may in fact severely exacerbate the problem.

Theorem 4 When scheduling a given HRT system upon a particular uniform multiprocessor platform, the number of migrations in the resulting schedule if EDF-G is the EDF-scheduling algorithm used is unrelated to the the number of migrations in the resulting schedule if EDF-MB or EDF-MM is the EDF-scheduling algorithm used.

Proof. The crucial observation is that a single migration occurring upon job arrival may prevent several migrations from occurring later in time. Below, we illustrate just such a situation where EDF-G actually results in fewer migrations than EDF-MB or EDF-MM (note that, since this illustration involves only two processors, EDF-MB and EDF-MM produce the same schedule).

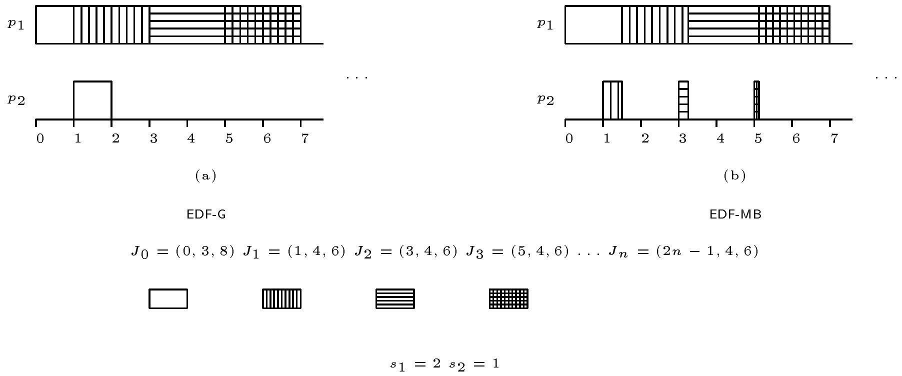

Figure 2: EDF-Gmay result in fewer migrations.

Figure 2 illustrates that a single migration may prevent future migrations from occurring. When EDF-G is used J_0 is preempted at $t = 1$, which allows $J_1, \ldots, J_n$ to execute completely on processor p_1. When EDF-MB is used, J_0 completes its execution on p_1 after J_1 arrives even though J_1 has higher priority. Job J_1 migrates to p_1 once J_0 completes, but its completion time is delayed due to the $\frac{1}{2}$ unit of execution on p_2. This, in turn, forces J_2 to begin execution on p_2 and then migrate to p_1, and so on through J_n. Thus, the single migration that occurs when using EDF-G prevents the n migrations that occur when using EDF-MB. ■

4 EDF-feasibility analysis

In this section, we consider the scheduling of HRT systems that are comprised entirely of periodic tasks, upon identical multiprocessor platforms. In *uniprocessor* platforms, such systems have been thoroughly studied, and EDF-scheduling of such systems is very well understood. However, the results reported here indicate that many of the important uniprocessor results do not extend to the identical multiprocessor case.

4.1 The synchronous case

Recall that a synchronous periodic task system is one in which all periodic tasks are assumed to have the same offset parameter (without loss of generality, assumed equal to zero). That is, all tasks are assumed to have their first jobs arrive at time-instant zero. For uniprocessor platforms, synchronous systems are very popular in the literature for a least three reasons:

1. They occur quite often in applications.

2. The synchronous case constitutes the worst case from a schedulability point of view — if a synchronous system is schedulable, then the periodic task system obtained from this synchronous system by assigning arbitrary offsets to the different periodic tasks is guaranteed to also be schedulable.

3. There are often much smaller feasibility intervals (i.e., a finite interval such that it is sure that no deadline will ever be missed iff, when we only keep the requests made in this interval, all deadlines for them in this interval are met) in the synchronous case than in the asynchronous one. Indeed, while the length of the feasibility interval for asynchronous systems is approximately twice the least common multiple of the periods of the tasks (and thus of exponential size) [9], it is of pseudo-polynomial length for *bounded-density* synchronous systems [1].

For multiprocessors platforms, we believe that the point 1 remains valid, however, we shall see that points 2 and 3 do not hold in the multiprocessor case. We shall first examine the point 2 below; the point 3 will be examined specifically in section 4.2.

For uniprocessor platforms using EDF we know that the synchronous case is the worst case. More formally:

Theorem 5 ([10]) Let $S = \big[\tau_i = (e_i, d_i, p_i) \mid i = 1, \ldots, n\big]$ denote a periodic task system with arbitrary deadlines. If S is EDF-schedulable on a uniprocessor platform in the synchronous case, S is EDF-schedulable on the same platform in all asynchronous situations.

In summary, Theorem 5 says that in uniprocessor systems the synchronous case is the worst case. The significance of this property is the following: to determine whether a periodic task system is EDF-schedulable for all possible offset values, it suffices to

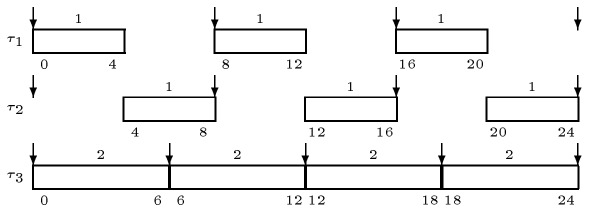

Figure 3: The synchronous schedule (the EDF schedule repeats after time $t = 24$).

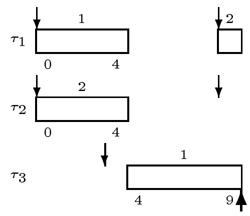

Figure 4: The first request of τ_3 misses its deadline at time $t = 9$.

test the schedulability of just the synchronous case. We shall see that this property does not hold for multiprocessor platforms.

Theorem 6 For identical multiprocessor platforms using EDF and implicit deadline periodic task sets, the synchronous case is not necessarily the worst case.

Proof. Consider the following platform composed of two (identical) unit-speed processors and the following periodic task set $\left[\tau_1 = (e_1 = 4, d_1 = p_1 = 8), \tau_2 = (e_2 = 4, d_2 = p_2 = 8), \tau_3 = (e_3 = 6, d_3 = p_3 = 6)\right]$. The system is EDF-schedulable in the synchronous case as exhibited by Figure 3. But the system is not EDF-schedulable in the asynchronous situation given by $o_1 = o_2 = 0$ and $o_3 = 3$ as exhibited in Figure 4. In Figures 3 through 6, $\downarrow$ represents a task request, $\bigcirc$ a deadline (we omit the representation of the deadline when considering implicit deadline systems) $\uparrow$ a deadline failure and $\overset{c}{\underset{a\ \ b}{\square}}$ an execution during the interval $[a, b)$ on the processor c. (Execution in the interval $[a, a+1)$ on processor c is denoted $\overset{c}{\underset{a}{\square}}$.) ■

It may be noticed that the example illustrated in Figure 3 shows another interesting phenomenon: in case of tied deadlines, the method chosen to break ties **is** significant in the multiprocessor case (this is not the case in uniprocessors — see, e.g., [6]). Indeed, in the schedule of Figure 3, at time $t = 18$, the three tasks are active and their deadlines coincide, if the scheduler assigns the processors to τ_1 and τ_2 (and consequently suspends τ_3) τ_3 will miss its deadline.

Since implicit-deadline periodic task systems are a special case of arbitrary-deadline periodic task systems, we immediately obtain the following corollary:

Corollary 7 For uniform multiprocessor platforms using EDF and arbitrary-deadline periodic task sets, the synchronous case is not necessarily the worst case.

Proof. The example used in the proof of Theorem 6 is also an example for the general case of uniform multiprocessor platforms with arbitrary-deadline periodic task sets. ■

4.2 The first busy period

Liu and Layland [10] noticed that, for the synchronous uniprocessor case, it is not necessary to look at a full hyper-period (i.e., the interval $[0, P)$ where $P \stackrel{\text{def}}{=} \text{lcm}\{p_i \mid 1 \leq i \leq n\}$) to detect deadline failures: the interval $[0, L)$, where L is the first instant after time-instant zero at which EDF would idle the processor, is a feasibility interval. If $U = \sum_{i=1}^{n} \frac{e_i}{p_i} < 1$, we always have $L < P$; but if $U = 1$ there is no idle slot after the system start time. In previous work [6] we introduced the notion of an *idle point*.

Definition 2 The time t is an idle point of the schedule of a system if all requests occurring strictly before t have completed their execution before or at time t.

Time-instant zero is clearly an idle point in the EDF-schedule of any synchronous periodic task system.

Theorem 8 ([6]) When EDF is used to schedule a synchronous set of tasks with arbitrary deadlines on a single processor and there is a deadline failure at time t, there is no processor idle point in the interval $(0, t)$.

Consequently the interval $[0, L)$ is a feasibility interval for synchronous arbitrary deadline systems, where L is the position of the first idle point after time-instant zero. We shall see that the property given by Theorem 8 does not hold for multiprocessor platforms. We consider first the case of constrained deadline systems.

Theorem 9 For identical multiprocessor platform using EDF and constrained deadline synchronous periodic task sets, $[0, L)$ is not a feasibility interval, where L corresponds to the first idle point after the origin in the schedule.

Proof. Consider the platform composed of two (identical) unit-speed processors and the task set:

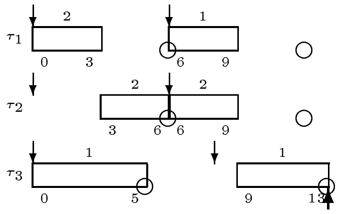

Figure 5: The second request of τ_3 misses its deadline at time $t = 12$.

$\left[\tau_1 = (e_1 = 3, d_1 = p_1 = 6), \tau_2 = (e_2 = 3, d_2 = p_2 = 6), \tau_3 = (e_3 = 5, d_3 = 5, p_3 = 8)\right]$. Figure 5 gives the schedule using EDF on the platform. The first deadline miss occurs at time $t = 13$ while $L = 6$. ■

Notice that the example considered in the proof of Theorem 9 does not cover the case of implicit deadlines. We shall see that for implicit–deadline systems the same phenomeon exists.

Theorem 10 For identical multiprocessor platform using EDF and implicit deadline synchronous periodic task sets, $[0, L)$ is not a feasibility interval, where L corresponds to the first idle point after the origin in the schedule.

Proof. Consider the platform composed of two (identical) unit-speed processors and the task set: $\left[\tau_1 = (e_1 = 3, d_1 = p_1 = 6), \tau_2 = (e_2 = 2, d_2 = p_2 = 7), \tau_3 = (e_3 = 5, d_3 = p_3 = 5)\right]$. Figure 6 gives the schedule using EDF on the platform. The first deadline miss occurs at time $t = 85$ while the first busy period ends at time $t = 5$. ■

5 Conclusions

In this paper we have considered the problem of scheduling hard real-time systems using EDF upon multiprocessor platforms. Upon uniform multiprocessors, we have shown that there may occur, in general, many more interprocessor migrations than is the case upon uniprocessors and identical multiprocessors, and that what seem like intuitive heuristics for decreasing the number of such migrations may in fact severely *increase* the number of migrations. With respect to scheduling periodic task systems upon identical multiprocessors, we observed two interesting facts: first, that the synchronous case is not necessarily the worst case (as is true upon uniprocessors), and next, that the first busy period is not a feasibility interval (once again, in contrast to the uniprocessor case). These results lead us to conclude that scheduling upon multiprocessor platforms is not an obvious extension of our knowledge concerning the uniprocessor case, and much more research is required to cover specifically this topic.

References

[1] Baruah, S., Howell, R., and Rosier, L. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems: The International Journal of Time-Critical Computing 2* (1990), 301–324.

[2] Dertouzos, M., and Mok, A. K. Multiprocessor scheduling in a hard real-time environment. *IEEE Transactions on Software Engineering 15*, 12 (1989), 1497–1506.

[3] Funk, S., Goossens, J., and Baruah, S. On–line scheduling on uniform multiprocessors. In *Proceedings of the 22nd IEEE Real-Time System Symposium* (London, England, December 2001).

[4] Goossens, J. *Scheduling of Hard Real-Time Periodic Systems with Various Kinds of Deadline and Offset Constraints.* PhD thesis, Université Libre de Bruxelles, Belgium, 1999.

[5] Goossens, J., and Devillers, R. The non-optimality of the monotonic priority assignments for hard real-time offset free systems. *Real-Time Systems: The International Journal of Time-Critical Computing 13*, 2 (1997), 107–126.

[6] Goossens, J., and Devillers, R. Feasibility intervals for the deadline driven scheduler with arbitrary deadlines. In *Proceedings of the International Conference on Real-time Computing Systems and Applications* (Hong Kong, December 1999), IEEE Computer Society Press, pp. 54–61.

[7] Goossens, J., Funk, S., and Baruah, S. Priority–driven scheduling of periodic task systems on uniform multiprocessors. *Real Time Systems*. To appear.

[8] Hong, K., and Leung, J. On-line scheduling of real-time tasks. In *Proceedings of the Real-Time Systems Symposium* (Huntsville, Alabama, December 1988), IEEE, pp. 244–250.

[9] Leung, J., and Merrill, M. A note on the preemptive scheduling of periodic, real-time tasks. *Information Processing Letters 11* (1980), 115–118.

[10] Liu, C., and Layland, J. Scheduling algorithms for multiprogramming in a hard real-time

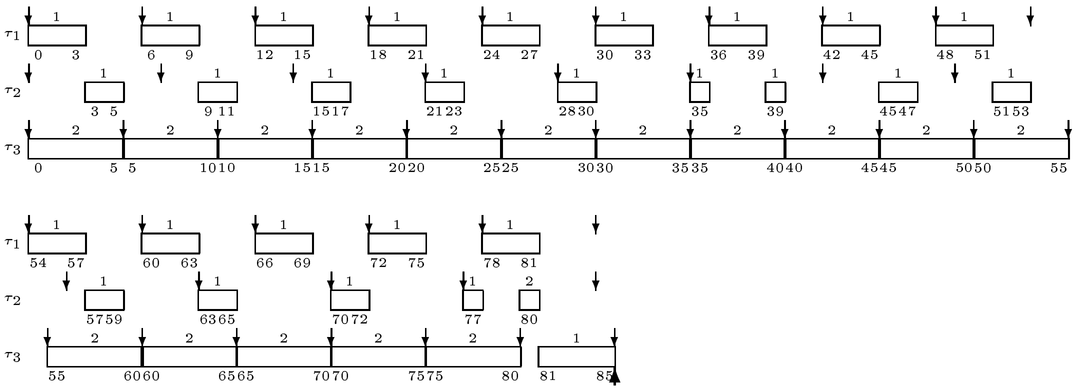

Figure 6: The 18th request of τ_3 misses its deadline at time $t = 85$.

environment. *Journal of the ACM 20*, 1 (1973), 46–61.

[11] MOK, A. Task management techniques for enforcing ED scheduling on a periodic task set. In *Proc. 5th IEEE Workshop on Real-Time Software and Operating Systems* (Washington D.C., May 1988), pp. 42–46.