New Schedulability Tests for Real-Time task sets scheduled by Deadline Monotonic on Multiprocessors

Marko Bertogna, Michele Cirinei and Giuseppe Lipari

Scuola Superiore Sant'Anna, Pisa, Italy
E-mail: marko@sssup.it, cirinei@gandalf.sssup.it, lipari@sssup.it

Abstract. In this paper, we address the problem of schedulability analysis of a set of real-time periodic (or sporadic) tasks on multiprocessor hardware platforms, under fixed priority global scheduling. In a multiprocessor system with M processors, a global scheduler consists of a single queue of ready tasks for all processors, and the scheduler selects the first M tasks to execute on the M processors. We allow preemption and migration of tasks between processors.
This paper presents two different contributions. First, we derive a sufficient schedulability test for periodic and sporadic task system scheduled with fixed priority when priorities are assigned according to Deadline Monotonic. This test is efficient when dealing with heavy tasks (i.e. tasks with high utilization). Then, we develop an independent analysis for preperiod deadline systems. This leads to a new schedulability test with density and utilization bounds that are tighter than the existing ones.

1 Introduction

Recently, multicore hardware platforms (i.e. with more than one processor on a single chip) are gaining momentum both in the high-end processor market and in the embedded systems market. There are many reasons for this widespread popularity, the most important being the technological constraints that make it impossible to design and implement faster single-processor chips at reasonable costs.

However, the current state-of-practice programming methodologies have not yet shifted toward parallel computing. This is particularly unfortunate in real-time systems. As a matter of fact, the real-time scheduling theory for multiprocessor systems is not yet well studied as the corresponding single-processor scheduling theory. In particular, many negative results are known for real-time scheduling on multi-processors.

Recently, a number of research papers have addressed the problem of schedulability analysis of real-time task sets on a multi-processor platform when global scheduling is considered. In a multiprocessor system with m processors, global scheduling consists in having one single queue of ready tasks for all processors and a scheduler selects the first m tasks from the queue to be executed on the processors. Preemption and migration are allowed, i.e. a task may be interrupted by higher priority tasks at any time and it may resume execution on a different processor. A totally different approach is static partitioning of tasks to processors, where, before execution, tasks are statically allocated to processors and cannot migrate. On each processor, a single processor scheduler is run.

In comparison to static partitioning, global scheduling suffers the cost of migration. This cost is mostly due to the cache: when moving a task from one processor to another, chances are that the task must reload the cache of the second processor. This cost, which might result excessive in traditional multiprocessor platforms, is greatly reduced in multicore chips, as the processors share part of the cache, and one processor can access the cache contents of another processor at little additional cost.

Moreover, global scheduling is particularly useful in case of open dynamic systems, where tasks may dynamically enter and leave the system. In fact, with static partitioning, every time a task enters the system, it must be allocated to a processor, and optimal allocation is an NP-Hard problem. Therefore, admission control and allocation become difficult and time consuming. Also, when a task leaves the system, there may be the need for re-allocation and load balancing, and this reintroduces migration overhead.

On the other hand, under global scheduling a task is not allocated to a processor. Therefore, when a task wants to enter the system, the only remaining problem is admission control, i.e. to understand if the task can be admitted into the system without jeopardizing the guarantee on the already admitted real-time tasks. This test is commonly referred to as *schedulability test*. In this paper we propose schedulability tests based on utilization and density bounds, which are polynomial in the number of tasks.

When considering global scheduling, schedulers can be roughly divided in three groups depending on the priority that a task has during its execution. If the priority of a task cannot change throughout the whole task lifetime, the scheduling algorithm has ”fixed task priority”. If the priority can change only at job boundaries, as with EDF, then the algorithm has ”fixed job priority”. The above classes are often referred to as ”priority driven”. Finally, if the priority can change also during the job execution, as for the P-fair class of algorithms described in [1], then the algorithm has ”(fully) dynamic priority”. Algorithms from the latter class can have a higher utilization bound, reaching the number of processors when deadlines are equal to periods. On the other side, they have a higher number of preemptions and migrations and a more difficult implementation. For these reasons, it may be more favorable to use a priority driven scheduler that has all the mentioned advantages related to the global scheduling.

This work will analyze the first group of algorithms, which assigns statically the priority to each task and that is often briefly called *fixed priority*. One of the most used priority assignment in this class is Rate Monotonic (RM), that assigns priorities proportional to the inverse of the periods. RM has been proved to be optimal in the uniprocessor case, in the sense that if a task set can be scheduled with fixed priority with a particular priority assignment, then it is also schedulable with RM. If a system can have deadlines less than periods, then the Deadline Monotonic priority assignment (DM) is optimal on a single processor. Efficient schedulability tests are known in the uniprocessor case for both DM and RM. When considering systems with more than one CPU, the above optimality is lost. This is mainly due to the ”Dhall effect” [2], that takes place when scheduling on the same platform tasks with high utilization and tasks with low utilization. To overcome this effect reaching a higher utilization bound, there are proposed solutions that give maximum priority to the heaviest tasks and schedule the remaining ones with RM or DM.

1.1 Our contribution

This paper presents various contributions. First, we discuss two recent solutions to the multiprocessor schedulability analysis using deadline-monotonic algorithm, one proposed by Andersson, Baruah and Jonsson [3], which will be denoted by ABJ, and the other one proposed by Baker [4], which will be denoted by BAK. We prove that neither test dominates the other.

Using a technique similar to the one used in [5] for the EDF case, we then propose a schedulability test that, bounding the interference imposed on a task, is able to successfully guarantee a larger portion of schedulable task sets, especially in presence of *heavy tasks* (i.e. tasks whose utilization is greater than 0.5).

In order to derive a result that generalizes ABJ, as well as a utilization bound proposed by Baker, we develop a new scheduling analysis that leads to tighter density and utilization bounds. With these new results, we can evaluate the performances of DM-based hybrid scheduling algorithms, which are solutions that treat in a different way the tasks with high and with low utilization, overcoming Dhall's effect and reaching higher utilization bounds.

The paper is organized as follows. In Section 2 we introduce the terminology and notation. In Section 3, the two main existing results on the schedulability analysis with RM and DM are summarized and compared. In Section 4, we present our first test, which improves over the test proposed in [4] bounding the interference that can be imposed on a task. In Section 5, we propose a new scheduling analysis that leads to tighter density and utilization bounds. Using this result, we then characterize the performances of previously proposed hybrid algorithms based on RM and DM. Finally, in Section 6 we present our conclusions.

2 System model

We consider a set τ of periodic or sporadic tasks to be scheduled on m identical processors. A task τ_k is a sequence of jobs J_k^j, each one with an arrival time r_k^j and a finishing time f_k^j. Each task $\tau_k = (C_k, D_k, T_k) \in \tau$ is characterized by a worst-case computation time C_k, a period or minimum interarrival time T_k, and a relative deadline D_k. Goal of the scheduling algorithm is to guarantee that each job will complete before its absolute deadline $d_k^j = r_k^j + D_k$.

For convenience, tasks are numbered in decreasing priority order. We denote with *constrained deadline* (resp. *implicit deadline*) the systems with $D_k \leq T_k$ (resp. $D_k = T_k$). We define the *utilization* of a task τ_k as $U_k = \frac{C_k}{T_k}$. We also define the *density* of a task τ_k as $\lambda_k = \frac{C_k}{D_k}$, which represents the "worst-case" request of a task in a generic time interval. Let U_{max} (resp. λ_{max}) be the largest utilization (resp. the largest density) among all tasks.

The *workload* $W_k(a,b)$ of task τ_k is the sum of all intervals in which τ_k is executing in interval $[a,b]$. The *load* $L_k(a,b)$ of a task τ_k in $[a,b]$ is the workload divided by the length of the interval: $L_k(a,b) = \frac{W_k(a,b)}{b-a}$.

The *competing (work)load* of a task τ_k is the sum of the (work)loads of all tasks τ_i, with $i < k$.

The *interference* $I_k(a,b)$ on a task τ_k over an interval $[a,b]$ is the cumulative length of all intervals in which the task is ready to execute but it cannot execute due to higher priority jobs. We also define the *interference* $I_{i,k}(a,b)$ of a task τ_i on a task τ_k over an interval $[a,b]$ as the cumulative length of all intervals in which τ_k is ready to execute, τ_i is executing while τ_k is not. Notice that by definition: $I_{i,k}(a,b) \leq I_k(a,b), \forall i,k,a,b$.

We finally define the *interfering load* on a task τ_k over an interval $[a,b]$, as the interference $I_k(a,b)$ divided by the length of the interval, i.e.: $L_k^{int}(a,b) = \frac{I_k(a,b)}{b-a}$. Similarly, we define the *interfering load* of a task τ_i on a task τ_k in an interval $[a,b]$, as: $L_{i,k}^{int}(a,b) = \frac{I_{i,k}(a,b)}{b-a}$.

3 Summary of existing results

The schedulability problem for RM or DM has been widely studied in the uniprocessor case. Only recently the multiprocessor case has been analyzed in more detail. In particular, there are two previously proposed works that derive schedulability tests with polynomial time complexity, one proposed by Andersson, Baruah, Jonsson [3] and the other by Baker [4]. We will refer to both results with the first letters of the authors.

3.1 The ABJ test

The following result has been presented in [3,6] and is valid only for systems with deadlines equal to periods.

Theorem 1 (ABJ). *A task set with $U_{tot} \leq \frac{m^2}{3m-2}$ and $U_{max} \leq \frac{m}{3m-2}$ is schedulable with Rate Monotonic (RM) upon m processors.*

The test is very simple but is only applicable to task sets composed by tasks with limited utilization. In the same paper is proposed a slightly modified version of RM that allows to reach a utilization bound of $\frac{m^2}{3m-2}$ with no restriction on the utilization of a task. We will better describe this algorithm and the related bound in the last part of this paper.

3.2 The BAK test

With a different approach, Baker derived in [4,7] another test that is valid also for prepe-riod deadline systems with unrestricted task utilization. The idea is based on the consideration that if a job J_k^j of task τ_k misses its deadline d_k^j, it means that the competing load of task τ_k in interval $[r_k^j, d_k^j]$ is at least $m(1-\lambda_k)$. The situation is depicted in Figure 1. Therefore, if it would be possible to show, for every job J_k^j, that the higher priority tasks cannot generate so much competing workload in interval $[r_k^j, d_k^j]$, then the schedulability would be guaranteed.

Unfortunately, checking the condition directly in $[r_k^j, d_k^j]$ is not simple without over-estimating the contribution of each task. To find a better estimation, Baker proposes to *enlarge* the interval to the largest possible interval such that the competing load is still greater than $m(1-\lambda_k)$. This new interval is called *busy window*. By deriving an upper bound on the combined load produced in the busy window, the final result is obtained.

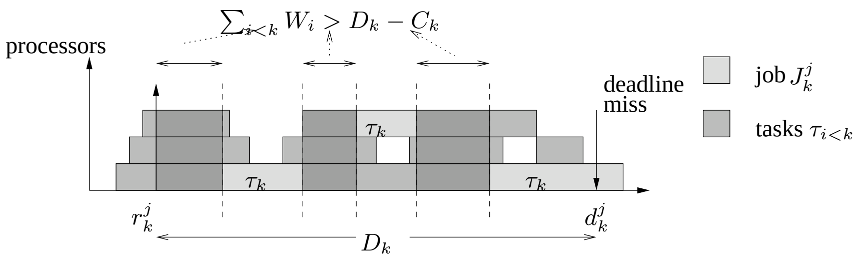

Fig. 1. Problem window.

Theorem 2 (BAK). *A task set τ composed by n tasks is schedulable with DM on a SMP with m processors if*

$$\forall \tau_k : \sum_{i=1}^{k-1} \beta_i \le m(1-\lambda_k), \quad \textit{where} \quad \beta_i = \begin{cases} U_i(1 + \frac{T_i - C_i}{D_k}) & \textit{if} \quad \lambda_k \ge U_i \\ U_i(1 + \frac{T_i - C_i}{D_k}) + \frac{C_i - \lambda_k T_i}{D_k} & \textit{if} \quad \lambda_k < U_i \end{cases}$$

It consists of $n-1$ inequalities, one for each task, excluded the task with shortest deadline that doesn't have any interfering task.

Even if the BAK test is valid also for preperiod deadline systems with unrestricted task utilization, we show with a simple example that it doesn't dominate ABJ. Consider a platform with $m = 2$ processors and a task set $\tau = \{\tau_1 = (4,9); \tau_2 = (4,9); \tau_3 = (1,10)\}$. Since the largest utilization is $\frac{4}{9} \le \frac{m}{3m-2} = 0.5$, then ABJ is applicable. The total utilization is less than $\frac{m^2}{3m-2} = 1$ and ABJ is passed. On the other side, it is easy to verify that the BAK test fails for $k = 3$. Then, BAK is not more general than ABJ.

Note that this also proves that the utilization bound proposed in [4] for implicit deadline systems cannot follow from the general test. Anyway, we will prove along this paper that the bound is indeed correct, and our indipendent analysis will show that a similar result is valid also for preperiod deadline systems.

4 The BCL test

A previous work [5] modified Baker's analysis for the EDF case, deriving a test that better behaves with tasks that have a high utilization. In this section, we briefly report some of the results derived in that work adapting them, when needed, to the fixed priority case. We will consider systems with constrained deadline. When a proof is not reported, it is identical to the EDF case and can be found in the cited paper.

4.1 Interference time

The interference (resp. interfering load) that a task τ_i causes on another task τ_k in an interval $[a,b]$ is never greater than the workload (resp. load) of τ_i in the same interval:

$$\forall i,k,a,b: \quad I_{i,k}(a,b) \le W_i(a,b) \le b-a, \quad \text{and} \quad L_{i,k}^{int}(a,b) \le L_i(a,b) \le 1.$$

Moreover, note that the competing workload of a task τ_k in a generic interval cannot be less than the interference I_k in the same interval. Considering again Figure 1 and the definition of interference, we have that if a task τ_k misses a deadline in d_k^j, the interference on task τ_k in $[r_k^k, d_k^j]$ must be greater than $(D_k - C_k)$.

Since the fixed priority global scheduling algorithm is work-conserving, we have that in the time instants in which a job is ready but not executing, each processor must be occupied by a job of a higher priority task. Then, the followig results is valid.

Lemma 1. *The interference (resp.interfering load) that a task τ_k can suffer in interval $[a, b]$ is the sum of the interferences (resp. interfering loads) of all higher priority tasks in the same interval, divided by the number of processors: $I_k(a, b) = \frac{\sum_{i<k} I_{i,k}(a,b)}{m}$ (resp.: $L_k^{int}(a, b) = \frac{\sum_{i<k} L_{i,k}^{int}(a,b)}{m}$).*

For constrained deadline systems scheduled with fixed priority, we then have that when a deadline d_k^j is missed: $\sum_{i<k} I_{i,k}(r_k^j, d_k^j) > m(D_k - C_k)$.

Therefore, for a job to meet its deadline, the competing interference on the task must be less than or equal to $m(D_k - C_k)$. For a task to be schedulable, the condition must hold for all its jobs.

We define the worst-case interference for task τ_k as: $\overline{I}_k = \max_j(I_k(r_k^j, d_k^j)) = I_k(r_k^{j*}, d_k^{j*})$, where $j*$ is the job instance in which the total interference is maximal. To simplify the notation, we define: $\overline{I}_{i,k} = I_{i,k}(r_k^{j*}, d_k^{j*})$. With the above notation we can easily extend a necessary and sufficient test derived in [5] for the EDF case.

Theorem 3. *A task set with constrained deadlines is schedulable with fixed priority iff, for each task τ_k, one of the following is true:*

1) $\sum_{i<k} \min\left(\overline{I}_{i,k}, D_k - C_k\right) < m(D_k - C_k)$
2) $\sum_{i<k} \min\left(\overline{I}_{i,k}, D_k - C_k\right) = m(D_k - C_k)$ *and* $\exists h < k : 0 < \overline{I}_{h,k} \leq D_k - C_k$

Proof. In [5], a corresponding theorem is proved for EDF. The difference is only in the tasks that have to be considered in the sum. In the EDF case, the sum is extended to all tasks (excluded τ_k). With fixed priority, the sum can be limited to the first $k - 1$ tasks, because the interference on task τ_k of each task $\tau_{i\geq k}$ is null.

To better understand the key idea behind Theorem 3, consider again the situation depicted in Figure 1. It is clear that, if the interference that a task τ_i can impose on task τ_k in window $[r_k^j, d_k^j]$ is greater than $D_k - C_k$, it is sufficient to consider only the portion $D_k - C_k$ in the sum to verify the schedulability of task τ_k.

4.2 Combined workload

The schedulability test of Theorem 3 requires the interferences $\overline{I}_{i,k}$. Since we are not able to compute these values without a simulation of the system, we will use an upper bound, deriving only a sufficient condition. We know that an upper bound on the interference $I_{i,k}(r_k^j, d_k^j)$ is the workload $W_i(r_k^j, d_k^j)$. The workload of each interfering task $\tau_{i<k}$ is maximized when the last job is released at instant $(d_k^j - C_i)$, and the job has just the time to complete its execution requirements before the deadline of task τ_k. The situation is depicted in Figure 2 and detailed in [4]. In such a situation, let

$N_i = \left(\left\lfloor \frac{D_k - C_i}{T_i} \right\rfloor + 1\right)$ be the number of requests r_i^h that τ_i makes in $[r_k^j, d_k^j]$. An upper bound on the workload of τ_i in a generic interval $[r_k^j, d_k^j]$ is then:

$$W_i(r_k^j, d_k^j) \leq N_i C_i + \varepsilon_i(r_k^j, d_k^j)$$

where $\varepsilon_i(r_k^j, d_k^j)$ is the execution time that the first job of τ_i having execution in $[r_k^j, d_k^j]$ spends inside the considered interval. This is often called *carry-in* of task τ_i in interval $[r_k^j, d_k^j]$. Note that an upper bound on the carry-in is the following:

$$\varepsilon_i(r_k^j, d_k^j) \leq \min\left(C_i, (D_k - N_i T_i + D_i - C_i)_0\right),$$

where we used $(x)_0$ as a short notation for $\max(0, x)$. Obviously, the carry-in of a task cannot exceed the worst-case computation time C_i of the task. Moreover, since the first missed deadline is later at d_k^j, the finishing time of the first job of τ_i cannot be later than its deadline. From Figure 2, it follows that the carry-in cannot be greater than $(D_k - N_i T_i + D_i - C_i)$, when this term is positive, proving the bound.

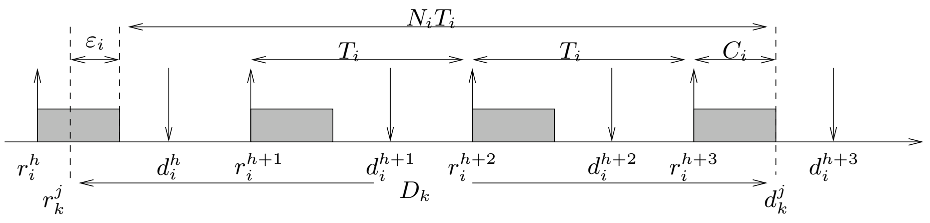

Fig. 2. Carry-in of a task $\tau_{i<k}$ in $[r_k^j, d_k^j]$.

Denoting with β_i an upper bound on the load of task τ_i in interval $[r_k^j, d_k^j]$, we can then write:

$$\beta_i = \frac{N_i C_i + \min\left(C_i, (D_k - N_i T_i + D_i - C_i)_0\right)}{D_k}. \quad (1)$$

Expressing Theorem 3 using the load instead of the workload and modifying it with the derived bound on the load of a task, we get the following sufficient condition.

Theorem 4 (BCL). *A task set with constrained deadlines is schedulable with fixed priority if, for each task τ_k, one of the followings is true:*

1) $\sum_{i<k} \min\left(\beta_i, 1 - \lambda_k\right) < m(1 - \lambda_k)$
2) $\sum_{i<k} \min\left(\beta_i, 1 - \lambda_k\right) = m(1 - \lambda_k)$ *and* $\exists i \neq k : \; 0 < \beta_i \leq 1 - \lambda_k$.

where β_i is expressed by Equation (1).

One of the main differences between this test and the results presented in [4] lies in term $(1 - \lambda_k)$ in the minimum. This term directly derives from term $D_k - C_k$ in Theorem 3. The underlying idea is that when considering the interference of a heavy task τ_i over another task τ_k, we do not want to overestimate its contribution to the total interference. If we consider its entire load, when we sum it together with the load of the other tasks on all m processors, its contribution could be much higher than $\frac{D_k - C_k}{D_k}$ and we could end up overestimating the total interference. Therefore, we must consider only the fraction of its workload that can actually interfere with task τ_k. This fraction is bounded by $1 - \lambda_k$.

5 Density-based test

In this section we will develop an indipendent analysis of the multiprocessor schedulability problem when using a fixed priority scheduler with Deadline Monotonic priority assignment. This will allow to derive a new density based test that represents the corresponding version for DM of a utilization based test derived in [8] for implicit deadline systems scheduled with EDF, and extended for preperiod deadlines in [5].

We will then show that this test generalizes the ABJ test and allows to characterize hybrid algorithms based on DM that have better performances when scheduling heavy and light tasks on the same platform.

In [4], a similar bound for implicit deadline systems is presented but not correctly proved. The task set we introduced in Section 3 can be used to show that it cannot follow from the general BAK test. However, a corollary derived from our test when deadlines are equal to periods will show that the bound is indeed correct.

Lemma 2. *In a constrained deadline system scheduled with fixed priority, if a task τ_k misses a deadline d_k^j, then*

$$\sum_{i \leq k} L_i(r_k^j, d_k^j) > m(1 - \lambda_k) + \lambda_k$$

The proof is detailed in [4] and follows from Figure 1.

For the next results, we assume the constrained deadline model is used.

Lemma 3. *If the load $L_i(I)$ of a task τ_i in an interval $I = [a, b]$, with $|I| \geq D_i$, is greater than $2\lambda_i$, then τ_i has two and only two releases, r_i^1 and r_i^2, in I.*

Proof. First we prove that the interval should at least include two releases of task τ_i.

Suppose there exists a task τ_i with $L_i(I) > 2\lambda_i$ and less than two releases inside interval I. The load of τ_i inside that interval is due to at most two instances. Since $D_i \leq |I|$, such load is: $L_i(I) \leq \frac{2C_i}{|I|} \leq \frac{2C_i}{D_i} = 2\lambda_i$, contradicting the hypothesis.

We say that a job J_i^l is "entirely contained" inside an interval, if its arrival time, r_i^l, and the arrival time of the next released job of the same task, r_i^{l+1}, are both contained inside the considered interval. Let ξ be the number of jobs of τ_i entirely contained in I. We showed that $\xi \geq 1$. Now we prove that $\xi < 2$.

Note that task τ_i produces the maximal load in I when (see Figure 3):

(i) The first job of τ_i executing in I is released at instant $(a - D_i + C_i)$ and executes for C_i units at the beginning of the interval.
(ii) The last job of τ_i executing in I is released at instant $(b - C_i)$ and executes for C_i units at the end of the interval.

Suppose there exists a task τ_i for which $\xi \geq 2$ and $L_i(I) > 2\lambda_i$. The load of τ_i under conditions (i) and (ii) is:

$$L_i(I) = \frac{2C_i + \xi C_i}{\xi T_i + 2C_i + T_i - D_i} \leq \frac{2C_i + \xi C_i}{\xi T_i + 2C_i} = \frac{(2 + \xi)U_i}{\xi + 2U_i}$$

The above expression can be greater than $2U_i$ only when $\xi < 2$. So when $\xi \geq 2$ it will be $L_i(I) \leq 2U_i \leq 2\lambda_i$, contradicting the hypothesis. Therefore it should be $\xi = 1$, as the situation depicted in Figure 4.

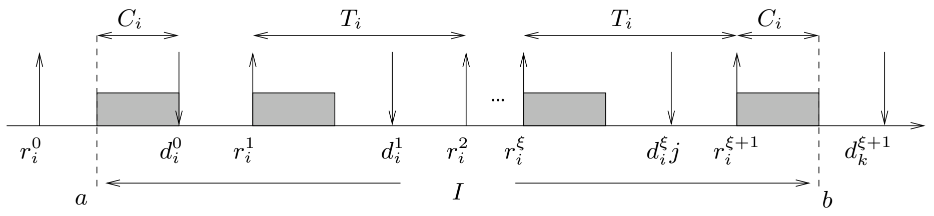

Fig. 3. Densest possible packing of jobs for task τ_i.

From now on, we will call J_i^0 the first job of task τ_i that executes in interval I. The following jobs are numbered accordingly.

Lemma 4. *If the load $L_i(I)$ of a task τ_i in an interval $I = [a, b]$, with $|I| \geq D_i$, is greater than $2\lambda_i$, then:*

$$U_i < \frac{1}{4} \tag{2}$$

$$\sum_{h \leq i} L_h([r_i^0, a]) > m(1 - \lambda_i) + \lambda_i \tag{3}$$

$$D_i > \frac{2}{3}|I| \tag{4}$$

$$|[r_i^0, a]| > \frac{D_i}{2} + C_i \tag{5}$$

Proof. Lemma 3 guarantees that there is one and only one job of τ_i entirely contained in $[a, b]$, as in Figure 4. The load of τ_i in this situation is $L_i(I) < \frac{3C_i}{T_i + 2C_i} = \frac{3U_i}{1+2U_i}$. Since $L_i(I) > 2\lambda_i \geq 2U_i$, then $U_i < \frac{1}{4}$, proving Equation (2).

Let $x = [r_i^0, a]$, $y = [a, r_i^1]$ and $z = [r_i^1, b]$.

It is: $L_i(y + z) > 2\lambda_i$ and $L_i(x + y) \leq \lambda_i$. Moreover:

$$\begin{aligned}
L_i(z) &\leq \frac{2C_i}{T_i + C_i} = \frac{2U_i}{1 + U_i} < 2U_i \leq 2\lambda_i \\
L_i(y) &= \frac{L_i(y + z)(|y| + |z|) - L_i(z)|z|}{|y|} > 2\lambda_i + \frac{|z|(2\lambda_i - L_i(z))}{|y|} \geq 2\lambda_i \\
L_i(x) &= \frac{L_i(x + y)(|x| + |y|) - L_i(y)|y|}{|x|} \leq \lambda_i + \frac{|y|(\lambda_i - L_i(y))}{|y|} < \lambda_i
\end{aligned} \tag{6}$$

Since job J_i^0 didn't yet complete its execution at instant a, it means that when in interval x task τ_i is not executing, all m processors are executing higher priority tasks and $L_i(x) + L_i^{int}(x) = 1$. From Equation (6) we have: $L_i^{int}(x) > 1 - \lambda_i$.

Using Lemma 1, Equation (3) is proved by: $\sum_{h \leq i} L_h(x) \geq \sum_{h<i} L_{h,i}^{int}(x) + L_i(x) = mL_i^{int}(x) + L_i(x) = mL_i^{int}(x) + (1 - L_i^{int}(x)) > m(1 - \lambda_i) + \lambda_i$.

Now, let $w = [a, d_i^0]$. Note that an upper bound on $L_i(y + z)$ is $\frac{3C_i}{T_i + C_i + (T_i - D_i) + |w|}$.

Since this quantity should be greater than $2\lambda_i$, we have: $|w| < \frac{5}{2}D_i - 2T_i - C_i$.

Then: $|I| = T_i + C_i + (T_i - D_i) + |w| < \frac{3}{2}D_i$, proving Equation (4).

And: $|[r_i^0, a]| = D_i - |w| > 2T_i - \frac{3}{2}D_i + C_i \geq \frac{D_i}{2} + C_i$, proving Equation (5).

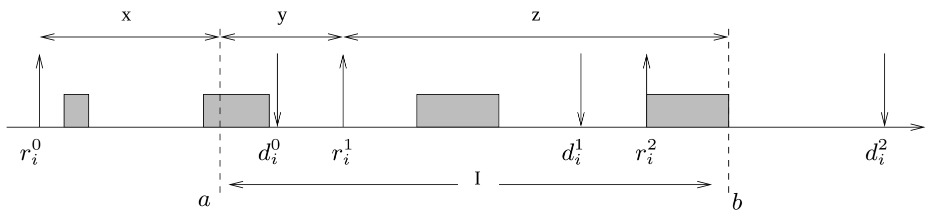

Fig. 4. A situation with $L_i(I) > 2\lambda_i$.

Theorem 5. *A set of periodic or sporadic tasks with constrained deadlines is schedulable with Deadline Monotonic priority assignment on $m \geq 2$ processors if:*

$$\lambda_{tot} = \sum_{i=1}^{n} \frac{C_i}{D_i} \leq \frac{m}{2}(1 - \lambda_{max}) + \lambda_{max} \qquad (7)$$

Proof. The proof is by contradiction. Using deadline monotonic priorities, suppose there exists a task set τ with $\lambda_{tot} \leq \frac{m}{2}(1-\lambda_{\max})+\lambda_{\max}$, which misses a deadline. Let d_k^j be the first missed deadline. Task τ_k is interfered by tasks in the set $T = \{\tau_1, \ldots, \tau_{k-1}\}$.

Case 1: $\forall \tau_i \in T : L_i(r_k^j, d_k^j) \leq 2\lambda_i$.
Since $\sum_{i<k} L_{i,k}^{int}(r_k^j, d_k^j) > m(1 - \lambda_k)$, we get:

$$\lambda_{tot} \geq \sum_{i<k} \lambda_i + \lambda_k \geq \sum_{i<k} \frac{L_i(r_k^j, d_k^j)}{2} + \lambda_k \geq \sum_{i<k} \frac{L_{i,k}^{int}(r_k^j, d_k^j)}{2} + \lambda_k$$
$$> \frac{m}{2}(1 - \lambda_k) + \lambda_k \geq \frac{m}{2}(1 - \lambda_{\max}) + \lambda_{\max}.$$

Case 2: There is at least one task $\tau_i \in T : L_i(r_k^j, d_k^j) > 2\lambda_i$.
Let $H = \{\tau_i \in T : L_i(r_k^j, d_k^j) > 2\lambda_i\}$. Since $D_i \leq D_k$, Lemma 3 guarantees that for every task $\tau_i \in H$ there is one and only one job entirely contained in $[r_k^j, d_k^j]$, as in Figure 4. Let τ_h be the task in H with the earliest released job that interferes τ_k in $[r_k^j, d_k^j]$. So it is: $r_h^0 \leq r_i^0, \forall \tau_i \in H$. Then no task in H can have only one job entirely contained in $[r_h^0, d_k^j]$. For Lemma 3 we then have:

$$\forall \tau_i \in H : L_i(r_h^0, d_k^j) \leq 2\lambda_i \qquad (8)$$

Let $a = [r_h^0, r_k^j]$, $b = [r_k^j, d_k^j]$ and $(a + b) = [r_h^0, d_k^j]$.
Since task τ_k can execute only when task τ_h is not interfered, and job J_h^0 has remaining execution time at instant r_k^j, then $L_k(a) \leq L_h(a)$. Equation (6) (with $I = [r_k^j, d_k^j]$) gives $L_h(a) < \lambda_h$. So we have:

$$L_k(a) < \lambda_h \qquad (9)$$

Case 2.1: $\forall \tau_i \in T : L_i(r_h^0, d_k^j) \leq 2\lambda_i$.
Equation (3) with $I = [r_k^j, d_k^j]$ and $L_h(I) > 2\lambda_h$ gives:
$\sum_{i \leq k} L_i(a) \geq \sum_{i \leq h} L_i(a) > m(1 - \lambda_h) + \lambda_h$.

Using the above relation together with Lemma 2 we get:

$$\sum_{i\le k} L_i(a+b) = \sum_{i\le k} \frac{L_i(a)|a| + L_i(b)|b|}{|a+b|}$$
$$> \frac{(m(1-\lambda_h)+\lambda_h)|a| + (m(1-\lambda_k)+\lambda_k)|b|}{|a|+|b|} \qquad (10)$$

Consider separately the case $\lambda_h \le \lambda_k$ and $\lambda_h > \lambda_k$.
- Case $\lambda_h \le \lambda_k$:
Equation (10) gives: $\sum_{i\le k} L_i(a+b) > m(1-\lambda_k) + \lambda_k$.
Equation (9) gives: $L_k(a) < \lambda_h \le \lambda_k$. Remember that $L_k(b) < \lambda_k$, because deadline d_k^j is missed. So, $L_k(a+b) = \frac{L_k(a)|a|+L_k(b)|b|}{|a+b|} < \frac{\lambda_k|a|+\lambda_k|b|}{|a+b|} < \lambda_k$.
Since $\forall \tau_i \in T : L_i(a+b) \le 2\lambda_i$, we get:

$$\lambda_{tot} \ge \sum_{i<k} \lambda_i + \lambda_k > \sum_{i\le k} \frac{L_i(a+b)}{2} + \frac{\lambda_k}{2} > \frac{m}{2}(1-\lambda_k)+\lambda_k \ge \frac{m}{2}(1-\lambda_{\max})+\lambda_{\max}.$$

- Case $\lambda_h > \lambda_k$:
From Equation (10) with $(\lambda_h - \lambda_k) > 0$, $m \ge 2$ and $\frac{|b|}{|a|+|b|} > \frac{T_h+C_h}{2T_h+C_h} = \frac{1+U_h}{2+U_h} > \frac{1}{2}$:

$$\sum_{i\le k} L_i(a+b) > m(1-\lambda_h)+\lambda_k + \frac{(\lambda_h-\lambda_k)|a| + m(\lambda_h-\lambda_k)|b|}{|a|+|b|}$$
$$> m(1-\lambda_h)+\lambda_h + (\lambda_h-\lambda_k)\frac{|b|}{|a|+|b|}$$
$$> m(1-\lambda_h)+\lambda_h + \frac{(\lambda_h-\lambda_k)}{2}$$

Since $|a| < D_h \le D_k$, then at most two jobs of τ_k can execute in interval $(a+b)$.
So we have: $L_k(a+b) \le \frac{2C_k}{2T_h+C_h} < \frac{C_k}{T_h} \le \frac{C_k}{D_h} < \frac{C_k}{\frac{2}{3}D_k} = \frac{3}{2}\lambda_k$, where we used Equation (4) with $|I| = |b| = D_k$.
Moreover: $L_h(a+b) \le \frac{3C_h}{2T_h+C_h} = \frac{3U_h}{2+U_h} < \frac{3}{2}U_h \le \frac{3}{2}\lambda_h$.
Since $\forall \tau_i \in T : L_i(a+b) \le 2\lambda_i$, it is:

$$\lambda_{tot} > \sum_{i<k, i\ne h} \frac{L_i(a+b)}{2} + \lambda_h + \lambda_k > \sum_{i\le k} \frac{L_i(a+b)}{2} + \frac{\lambda_h+\lambda_k}{4}$$
$$> \frac{m}{2}(1-\lambda_h)+\lambda_h \ge \frac{m}{2}(1-\lambda_{\max})+\lambda_{\max}$$

contradicting the hypothesis.
Case 2.2: There is at least one task $\tau_g \in T : L_g(r_h^0, d_k^j) > 2\lambda_g$.
Lemma 3 guarantees that τ_g has one and only one job entirely contained in $(a+b)$.
Let $a' = [r_g^0, r_h^0]$. We will now prove that: $\forall \tau_i \in T : L_i(a'+a+b) \le 2\lambda_i$.
Suppose there is a task $\tau_f \in T$ for which $L_f(a'+a+b) > 2\lambda_f$. Applying repeatedly Equation (4) with $I = (a'+a+b)$, $I = (a+b)$ and $I = b$, we get:
$D_f > \frac{2}{3}(2T_g+C_g) \ge \frac{4}{3}D_g > \frac{4}{3}\frac{2}{3}(2T_h+C_h) \ge \frac{16}{9}D_h > \frac{16}{9}\frac{2}{3}D_k > D_k$.

Therefore, τ_f cannot be in T, proving the assertion.
Equation (4) with $I = (a+b)$ gives: $D_g > \frac{2}{3}(2T_h + C_h) > T_h \geq D_h$, showing that τ_g has lower priority than τ_h. This means that, since at instant r_k^j job J_h^0 still has to complete: $L_g(a) \leq L_h(a)$. Equation (6) with $I = b$ gives $L_h(a) < \lambda_h$. Then:

$$L_g(a) < \lambda_h \tag{11}$$

Note that for Equation (8) $\tau_g \notin H$, so: $L_g(b) < 2\lambda_g$. Since $L_g(a+b) > 2\lambda_g$, then: $L_g(a) > 2\lambda_g$. Combining the latter relation with Equation (11), we get:

$$\lambda_g < \frac{\lambda_h}{2} \tag{12}$$

Equation (3) with $I = [r_h^0, d_k^j]$ gives: $\sum_{i \leq k} L_i(a') \geq \sum_{i \leq g} L_i(a') > m(1-\lambda_g) + \lambda_g$. For what we said for Case 2.1, it is also: $\sum_{i \leq k} L_i(a) > m(1-\lambda_h) + \lambda_h$. Using the above relations together with Lemma 2 we get:

$$\sum_{i \leq k} L_i(a'+a+b) > \frac{(m(1-\lambda_g)+\lambda_g)|a'| + (m(1-\lambda_h)+\lambda_h)|a| + (m(1-\lambda_k)+\lambda_k)|b|}{|a'|+|a|+|b|} \tag{13}$$

We consider separately the case $\lambda_h \leq \lambda_k$ and $\lambda_h > \lambda_k$.
- Case $\lambda_h \leq \lambda_k$:
Equations (12) and (13) give: $\sum_{i \leq k} L_i(a'+a+b) > m(1-\lambda_k) + \lambda_k$.
Being $D_g \leq D_k$, task τ_k can execute only when task τ_g is not interfered. Considering that job J_g^0 has remaining execution at instant r_h^0, then: $L_k(a') \leq L_g(a')$. Equation (6) for $I = [r_h^0, d_k^j]$ gives $L_g(a') < \lambda_g$. So we have: $L_k(a') < \lambda_g < \lambda_k$. Moreover, for Equation (9) it is: $L_k(a) < \lambda_h \leq \lambda_k$. Using $L_k(b) < \lambda_k$, we get: $L_k(a'+a+b) = \frac{L_k(a')|a'| + L_k(a)|a| + L_k(b)|b|}{|a'+a+b|} < \frac{\lambda_k|a'| + \lambda_k|a| + \lambda_k|b|}{|a'|+|a|+|b||} = \lambda_k$.
Since $\forall \tau_i \in T : L_i(a'+a+b) \leq 2\lambda_i$:

$$\lambda_{tot} \geq \sum_{i<k} \lambda_i + \lambda_k > \sum_{i \leq k} \frac{L_i(a'+a+b)}{2} + \frac{\lambda_k}{2} > \frac{m}{2}(1-\lambda_k) + \lambda_k \geq \frac{m}{2}(1-\lambda_{\max}) + \lambda_{\max}.$$

- Case $\lambda_h > \lambda_k$:
Equation (4) with $I = (a+b)$ gives $D_g > \frac{2}{3}(2T_h + C_h)$. Since $D_k \geq D_g$, we get $T_h < \frac{3}{4}D_k - \frac{C_h}{2}$, and:

$$|a'+a+b| < D_g + (2T_h + C_h) < D_k + (2(\frac{3}{4}D_k - \frac{C_h}{2}) + C_h) = \frac{5}{2}D_k \tag{14}$$

This means that at most three jobs of task τ_k can execute in interval $(a'+a+b)$. If $r_g^0 \geq r_k^{j-1}$, then $L_k(a'+a+b) \leq \frac{2C_k}{|a'+a+b|} < \frac{2C_k}{2T_h + C_h} < \frac{C_k}{T_h} < \frac{3}{2}U_k \leq \frac{3}{2}\lambda_k$, where we used $T_h > \frac{2}{3}D_k$. If $r_g^0 < r_k^{j-1}$, then $L_k(a'+a+b) < \frac{3C_k}{2T_k} < \frac{3}{2}U_k \leq \frac{3}{2}\lambda_k$.
In both cases: $L_k(a'+a+b) \leq \frac{3}{2}\lambda_k$.
Similarly: $L_g(a'+a+b) \leq \frac{3C_g}{2T_g + C_g} = \frac{3U_g}{2+U_g} < \frac{3}{2}U_g \leq \frac{3}{2}\lambda_g$.

Since $\forall \tau_i \in T : L_i(a' + a + b) \leq 2\lambda_i$:

$$\begin{aligned}\lambda_{tot} &> \sum_{i<k, i\neq h,g} \frac{L_i(a' + a + b)}{2} + \lambda_h + \lambda_g + \lambda_k \\ &> \sum_{i\leq k} \frac{L_i(a' + a + b)}{2} + \lambda_h + \frac{\lambda_g}{4} + \frac{\lambda_k}{4} - \frac{L_h(a' + a + b)}{2}\end{aligned}$$

For Equation (12): $(\lambda_h - \lambda_g) > 0$. Using Equation (13), with $(\lambda_h - \lambda_k) > 0$ and $m \geq 2$:

$$\begin{aligned}\sum_{i\leq k} L_i(a'+a+b) &> m(1-\lambda_h) + \frac{m(\lambda_h-\lambda_g)|a'| + m(\lambda_h-\lambda_k)|b| + \lambda_g|a'| + \lambda_h|a| + \lambda_k|b|}{|a'+a+b|} \\ &> m(1-\lambda_h) + \lambda_h + \frac{(\lambda_h-\lambda_g)|a'| + (\lambda_h-\lambda_k)|b|}{|a'+a+b|}\end{aligned}$$

Since for Equation (14) it is $\frac{|b|}{|a'+a+b|} > \frac{D_k}{\frac{5}{2}D_k} = \frac{2}{5}$, then:

$$\lambda_{tot} > \frac{m}{2}(1 - \lambda_h) + \lambda_h + \frac{(\lambda_h - \lambda_g)|a'|}{2|a' + a + b|} + \frac{7}{10}\lambda_h + \frac{\lambda_g}{4} - \frac{L_h(a' + a + b)}{2}$$

To derive an upper bound for $L_h(a' + a + b)$, note that, using Equation (4), it is: $T_h \geq D_h > \frac{2}{3}D_k \geq \frac{2}{3}D_g > \frac{2}{3}|a'|$. Then at most two jobs of τ_h can execute in a'.
- If $|a'| \leq T_h$, at most one job of τ_h can execute in a'. Using twice Equation (4):

$$L_h(a'+a+b) \leq \frac{4C_h}{|a'+a+b|} < \frac{4C_h}{2T_g + C_g} < \frac{4C_h}{2D_g} < \frac{4C_h}{2\frac{2}{3}(2T_h + C_h)} < \frac{3}{2}U_h \leq \frac{3}{2}\lambda_h$$

Equation (4) and (5), with $I = [r_h^0, d_k^j]$, give:
$\frac{|a'|}{|a'+a+b|} > \frac{\frac{D_g}{2} + C_g}{\frac{D_g}{2} + C_g + |a+b|} > \frac{\frac{1}{2}\frac{2}{3}|a+b| + C_g}{\frac{1}{2}\frac{2}{3}|a+b| + C_g + |a+b|} > \frac{1}{4}$. Then:

$$\begin{aligned}\lambda_{tot} &> \frac{m}{2}(1 - \lambda_h) + \lambda_h + \frac{(\lambda_h - \lambda_g)}{8} + \frac{7}{10}\lambda_h + \frac{\lambda_g}{4} - \frac{3}{4}\lambda_h \\ &> \frac{m}{2}(1 - \lambda_h) + \lambda_h \geq \frac{m}{2}(1 - \lambda_{\max}) + \lambda_{\max}.\end{aligned}$$

- If $|a'| > T_h$, then two jobs of τ_h can execute in a', and it is:

$$L_h(a' + a + b) \leq \frac{5C_h}{|a'| + |a + b|} < \frac{5C_h}{3T_h + C_h} < \frac{5}{3}U_h \leq \frac{5}{3}\lambda_h$$

Equation (2) for $I = [r_k^j, d_k^j]$ gives $C_h < \frac{T_h}{4}$.
So we have: $\frac{|a'|}{|a'+a+b|} > \frac{T_h}{T_h + 2T_h + C_h} > \frac{T_h}{3T_h + \frac{T_h}{4}} = \frac{4}{13}$. And:

$$\begin{aligned}\lambda_{tot} &> \frac{m}{2}(1 - \lambda_h) + \lambda_h + \frac{2(\lambda_h - \lambda_g)}{13} + \frac{7}{10}\lambda_h + \frac{\lambda_g}{4} - \frac{5}{6}\lambda_h \\ &> \frac{m}{2}(1 - \lambda_h) + \lambda_h \geq \frac{m}{2}(1 - \lambda_{\max}) + \lambda_{\max}.\end{aligned}$$

When deadlines are equal to periods, a utilization based test immediately follows.

Corollary 1. *A set of periodic or sporadic tasks with deadline equal to period is schedulable with Rate Monotonic priority assignment on $m \geq 2$ processors if:*

$$U_{tot} \leq \frac{m}{2}(1 - U_{max}) + U_{max}$$

The above result is more general than ABJ. This is easy to see taking $U_{\max} = \frac{m}{3m-2}$. Then a task set is schedulable when: $U_{tot} \leq \frac{m}{2}(1 - \frac{m}{3m-2}) + \frac{m}{3m-2} = \frac{m^2}{3m-2}$, as ABJ.

Theorem 5 is useful not only with RM and DM, but also with ”hybrid” algorithms. Hybrid algorithms are modified versions of classic scheduling algorithms that can reach a higher utilization bound, dealing separately with heavy and light tasks (see [3,4,9,10]). This is done in order to overcome Dhall’s effect [2], an effect that limits the scheduling performances of the classic algorithms when tasks with high and low utilization have to be scheduled on the same platform. Consider Algorithm RM-US[U_{th}] (Rate Monotonic with Utilization Separation U_{th}) that assigns maximum priority up to the heaviest $m-1$ tasks having utilization greater than the threshold U_{th} and schedules the remaining tasks with priorities according to RM.

```
ALGORITHM RM-US[$U_{th}$](tasks ordered by decreasing utilization):
– For($i=0, i<m, i=i+1$){If ($U_i > U_{th}$) {give $\tau_i$ maximum priority}; else break}
– Schedule the remaining tasks with priorities according to RM.
```

Andersson et al. showed that, when the threshold is $\frac{m}{3m-2}$, such algorithm can schedule any periodic task set with total utilization $U_{tot} \leq \frac{m^2}{3m-2}$.

Using Corollary 1, we can generalize the analysis to any utilization threshold. Being $H < m$ the number of tasks with $U_i > U_{th}$, the minimum total utilization needed for a task set to be schedulable is at least the minimum total utilization needed by an algorithm that schedules each one of the H ”heavy” tasks on a dedicated processor and the others on the remaining $m - H$ processors with RM [3,10]. Applying Corollary 1, the light tasks can be scheduled on $m - H$ processors if their total utilization is at most $\frac{m-H}{2}(1 - U_{th}) + U_{th}$. Therefore, the total utilization that still guarantees the schedulability with RM-US[U_{th}] is: $HU_{th} + \frac{m-H}{2}(1 - U_{th}) + U_{th}$. The maximum with respect to U_{th} of the minimum with respect to H of this expression is reached when $U_{th} = \frac{1}{3}$, which represents the utilization threshold that guarantees the highest utilization bound for the RM-US class of algorithms, relatively to our schedulability algorithm. Using this value inside the previous expression we have the following.

Corollary 2. *A periodic or sporadic task set with deadlines equal to periods is schedulable with RM-US[$\frac{1}{3}$] on m processors if $U_{tot} \leq \frac{m+1}{3}$*

Note that the bound of Corollary 2 is better than the bound derived in [3] for RM-US[$\frac{m}{3m-2}$], when $m \geq 2$.

Since Theorem 5 is valid also for preperiod deadline systems, we can as well characterize a similar algorithm that uses as separation value the density of each task. We call this algorithm DM-DS[λ_{th}] (Deadline Monotonic with Density Separation λ_{th}), where λ_{th} is the density threshold that discriminate between maximum priority ”heavy” task, and ”light” task to be scheduled with DM.

```
ALGORITHM DM-DS[$\lambda_{th}$](tasks ordered by decreasing density):
– For($i=0, i<m, i=i+1$){If ($\lambda_i > \lambda_{th}$) {give $\tau_i$ maximum priority}; else break;}
– Schedule the remaining tasks with priorities according to DM.
```

Proceding as before, but using Theorem 5 instead of Corollary 1, we can derive that the density threshold that guarantees the highest density bound using the test of Theorem 5 is $\lambda_{th} = \frac{1}{3}$, and state the following result.

Corollary 3. *A periodic or sporadic task set with preperiod deadlines is schedulable with DM-DS[$\frac{1}{3}$] on m processors if $\lambda_{tot} \leq \frac{m+1}{3}$*

6 Conclusions

In this paper we presented two schedulability tests to be used with the fixed priority scheduling algorithm. The first test is very efficient in presence of heavy tasks and it is the correspondent version of a similar test appeared in [5] for the EDF case. The second test behaves well with a high number of light tasks, given that the total density of the task set doesn't exceed a derived bound. It represents the DM version of an EDF schedulability test presented in [8] and extended for constrained deadline systems in [5]. We showed that our result allows to generalize and improve the existing related bounds, as well as to characterize the performances of previously proposed hybrid algorithms. An advantage of our result is that it is valid also for sporadic task systems with unbounded task utilization and with preperiod deadlines. No density bounds for constrained deadline systems have been previously proposed.

References

1. Baruah, S., Cohen, N., Plaxton, C., Varvel, D.: Proportionate progress: A notion of fairness in resource allocation. Algorithmica **6** (1996)
2. Dhall, S.K., Liu, C.L.: On a real-time scheduling problem. Operations Research **26** (1978)
3. Andersson, B., Baruah, S., Jonsson, J.: Static-priority scheduling on multiprocessors. In IEEE, ed.: Proceedings of the IEEE Real-Time Systems Symposium. (2001)
4. Baker, T.: Multiprocessor EDF and deadline monotonic schedulability analysis. In: Proceedings of the 24th IEEE International Real-Time Systems Symposium, RTSS'03. (2003)
5. Bertogna, M., Cirinei, M., Lipari, G.: Improved schedulability analysis of EDF on multiprocessor platforms. In: Proceedings of the IEEE Euromicro Conference on Real Time Systems, Mallorca, Spain, IEEE (2005)
6. Andersson, B.: Static-priority scheduling on multiprocessors. PhD thesis, Department of Computer Engineering, Chalmer University of Technology, Goteborg, Sweden (2003)
7. Baker, T.: An analysis of deadline-monotonic schedulability on a multiprocessor. FSU computer science technical report, Department of Computer Science, Florida State University, Tallahassee, Florida (2003) available at http://www.cs.fsu.edu/research/reports.
8. Goossens, J., Funk, S., Baruah, S.: Priority-driven scheduling of periodic task systems on multiprocessors. Real-Time Systems **25**(2-3) (2003) 187–205
9. Baruah, S.K.: Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors. IEEE Trans. Computers **53**(6) (2004) 781–784
10. Srinivasan, A., Baruah, S.K.: Deadline-based scheduling of periodic task systems on multiprocessors. Inf. Process. Lett. **84**(2) (2002) 93–98