Schedulability Analysis of Periodic Fixed Priority Systems

Enrico Bini and Giorgio C. Buttazzo, *Member*, *IEEE*

Abstract—Feasibility analysis of fixed priority systems has been widely studied in the real-time literature and several acceptance tests have been proposed to guarantee a set of periodic tasks. They can be divided in two main classes: polynomial time tests and exact tests. Polynomial time tests can efficiently be used for on-line guarantee of real-time applications, where tasks are activated at runtime. These tests introduce a negligible overhead, when executed upon a new task arrival, however provide only a sufficient schedulability condition, which may cause a poor processor utilization. On the other hand, exact tests, which are based on response time analysis, provide a necessary and sufficient schedulability condition, but are too complex to be executed on line for large task sets. As a consequence, for large task sets, they are often executed off line. This paper proposes a novel approach for analyzing the schedulability of periodic task sets on a single processor under an arbitrary fixed priority assignment. Using this approach, we derive a new schedulability test which can be tuned through a parameter to balance complexity versus acceptance ratio, so that it can be used on line to better exploit the processor, based on the available computational power. Extensive simulations show that our test, when used in its exact form, is significantly faster than the current response time analysis methods. Moreover the proposed approach, for its elegance and compactness, offers an explanation of some known phenomena of fixed priority scheduling and could be helpful for further work on schedulability analysis.

Index Terms—Real-time systems and embedded, systems analysis and design, fixed priority scheduling.

1 Introduction

Fixed priority scheduling is widely used in modern real-time systems, since it can be easily implemented on top of commercial kernels that provide a limited number of priority levels. One of the most common fixed priority assignment follows the Rate Monotonic (RM) algorithm, according to which tasks' priorities are ordered based on tasks' activation rates, so that the task with the shortest period is assigned the highest priority. In general, however, there can be particular situations in which task priorities, although fixed, do not necessarily follow the Rate Monotonic assignment.

Liu and Layland [1] proved that in a uniprocessor system RM is optimal among all fixed priority schemes, meaning that, if a task set is not schedulable by RM, then it cannot be scheduled by any other fixed priority assignment. In the same paper, the authors also derived a simple guarantee test to verify the feasibility of a set of n periodic tasks under RM. Each periodic task τ_i consists of an infinite sequence of jobs τ_{ik} $(k = 1, 2, \ldots)$, where the first job τ_{i1} is released at time $r_{i1} = \Phi_i$ (the task phase) and the generic kth job τ_{ik} is released at time $r_{ik} = \Phi_i + (k-1)\,T_i$, where T_i is the task period. Each job is characterized by a worst-case execution time C_i, a relative deadline D_i and an absolute deadline $d_{ik} = r_{ik} + D_i$. The ratio $U_i = C_i/T_i$ is called the *utilization factor* of task τ_i and represents the fraction of processor time used by that task. Finally, the value

$$U_p = \sum_{i=1}^{n} U_i$$

is called the *total processor utilization factor* and represents the fraction of processor time used by the periodic task set. Clearly, if $U_p > 1$ no feasible schedule exists for the task set.

The schedulability condition for RM is derived for a set Γ_n of n periodic tasks under the assumptions that all tasks start simultaneously at time $t = 0$, relative deadlines are equal to periods and tasks are independent, meaning that they do not have resource constraints, nor precedence relations. Under such assumptions, a set of n periodic tasks is schedulable by the RM algorithm if

$$\sum_{i=1}^{n} U_i \leq n\,(2^{1/n} - 1). \tag{1}$$

Throughout the paper, we will refer to the previous schedulability condition as the LL-test. We recall that

$$\lim_{n\to\infty} n\,(2^{1/n} - 1) = \ln 2 \simeq 0.69.$$

After this first result, a lot of work has been done to improve the schedulability bound of the RM algorithm or relax some restrictive assumption on the task set.

Lehoczky et al. [2] performed a statistical study and showed that for task sets with randomly generated parameters the LL-test is able to accept schedulable task sets with an average breakdown utilization of about 88 percent. Exact schedulability tests for RM yielding to necessary and sufficient conditions have been independently derived [3], [2], [4], [5]. Using the method proposed by Audsley et al. [4], a periodic task set is schedulable with the RM algorithm if and only if the worst-case response time of each task is less than or equal to its deadline. The

- *E. Bini is with the Scuola Superiore S.Anna, Piazza Martiri della Libertà 33, 56127 Pisa, Italy. E-mail: e.bini@sssup.it.*
- *G.C. Buttazzo is with the Università di Pavia, Via Ferrata 1, 27100 Pavia, Italy. E-mail: buttazzo@unipv.it.*

Manuscript received 28 Apr. 2003; revised 29 Jan. 2004; accepted 9 Mar. 2004.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0027-0403.

worst-case response time R_i of a task can be computed using the following iterative formula:

$$\begin{cases} R_i^{(0)} = C_i \\ R_i^{(k)} = C_i + \sum_{j:D_j<D_i} \left\lceil \frac{R_i^{(k-1)}}{T_j} \right\rceil C_j, \end{cases} \quad (2)$$

where the worst-case response time of task τ_i is given by the smallest value of $R_i^{(k)}$ such that $R_i^{(k)} = R_i^{(k-1)}$. It is worth noting, however, that the complexity of the exact test is pseudopolynomial, thus it may be unsuited for online admission control, especially in those real-time applications consisting of large task sets running on slow microprocessors. Leung and Whitehead [6] considered the case of deadlines smaller than periods and proved that the Deadline Monotonic priority assignment is optimal. Sha et al. [7] extended the Rate Monotonic analysis in the presence of resource constraints, where access to resources is performed using concurrency control protocols, such as the Priority Inheritance Protocol and the Priority Ceiling Protocol. Audsley et al. [4] generalized the response time analysis including resource constraints. Burns et al. [8] extended it to take fault-tolerant constraints into account. Sjödin and Hansson [9] provided some methods for reducing the number of iterations in computing the tasks response times; however, the worst-case complexity of their test is still pseudopolynomial.

In the last years, other authors [10], [11], [12] proposed novel approaches for deriving polynomial time tests with better acceptance ratio than the LL-test. For example, the Hyperbolic Bound (HB) proposed by Bini et al. [12] improves the acceptance ratio by a factor of $\sqrt{2}$ for large n, compared with the Liu and Layland test. According to HB method, a set of periodic tasks is schedulable by RM if

$$\prod_{i=1}^{n}(U_i + 1) \leq 2. \quad (3)$$

The authors also extended the test in the presence of resource constraints and aperiodic servers.

The Liu and Layland utilization bound has been improved by considering some additional information on the task set, such as the number of harmonic chains and the value of the periods [13], [14]. A similar approach was proposed by Park et al. [15] and, in the case of a graph structured task, by Liu and Hu [16].

In this paper, we propose a novel and more general approach for analyzing the schedulability of periodic task sets under the Rate Monotonic priority assignment. Using this approach, we derived a new schedulability test, called δ-HET (Hyperplanes δ-Exact Test), which can be tuned through a parameter to balance complexity versus acceptance ratio, so that it can be used online to better exploit the processor, based on the available computational power. So, for example, if the processor is powerful enough, and there is sufficient time for online guarantee, our test can be tuned to behave like the response time test, but with less execution overhead. On the other hand, when the processor utilization is high and the overhead of the guarantee test must be contained, our test can be set to run in less time, with decreased performance.

Extensive simulations show that our test, when used in its exact form, is significantly faster than the current response time analysis methods, and performs much better than polynomial tests (such LL or HB) when used in its reduced form.

As a last remark, we like to notice that the main result of this paper, expressed by Theorem 3, is very general and provides a new view of the some results present in the literature [13], [15], [14].

The rest of the paper is organized as follows: Section 2 describes the task model and states our notation. Section 3 explains our approach in detail. Section 4 illustrates the proof of the main result of the paper. Section 5 introduces the tunable guarantee test, called the δ-HET test. Section 6 compares the δ-HET test with others similar tests proposed in the literature. Section 7 disusses some extensions that allow to apply the proposed approach to more realistic cases. Finally, Section 8 presents our conclusions and future work.

2 Task Model and Notation

In this section, we introduce the task model and the notation we will use throughout the paper. We consider a set $\Gamma_n = \{\tau_1, \ldots, \tau_n\}$ of n periodic (or sporadic) tasks, where each task $\tau_i = (\Phi_i, C_i, T_i, D_i)$ is characterized by an initial activation time Φ_i (phase), a worst-case computation time C_i, a period (or a minimum interarrival time) T_i, and a relative deadline D_i not greater than T_i. We denote a fixed-priority assignment as FP and, without loss of generality, we assume the tasks in Γ_n are ordered by decreasing priority, so that τ_1 is the highest priority task. If priorities are assigned based on the RM algorithm, then we have FP=RM. Each task τ_i consists of an infinite sequence of jobs, where τ_{ik} denotes the kth job of task τ_i. In particular, r_{ik} and f_{ik} denote the release time and the finishing time of τ_{ik}, respectively. Each job has a relative deadline $D_i \leq T_i$, thus, for the periodicity assumption, the release time r_{ik} and the absolute deadline d_{ik} of job τ_{ik} can be computed as follows:

$$r_{ik} = \Phi_i + (k-1)T_i, \qquad d_{ik} = r_{ik} + D_i.$$

Finally, $U_i = C_i/T_i$ denotes the utilization factor of task τ_i.

As proven by Liu and Layland [1], the worst-case scenario for a periodic task set scheduled by RM occurs when all the tasks are simultaneously activated at the same time. Hence, without loss of generality, we assume $\Phi_i = 0$ for all the tasks and then we will denote the task $\tau_i = (0, C_i, T_i, D_i)$ simply by (C_i, T_i, D_i). It is not difficult to see that under arbitrary fixed priorities (non-RM) this scenario of simultaneous activation is still the worst. Moreover, it has been proven that such a condition represents the worst-case scenario also in the presence of shared resources [7].

To simplify the presentation of the proposed approach, we initially assume that tasks cannot be blocked and are fully preemptive. The presence of blocking times and the extension to nonpreemptive scheduling will be considered in Section 7.

In our formulation, a task set is viewed as a point in a specific space of the task set parameters, hence the feasibility test will be expressed as a check that verifies whether a point belongs to a region $\mathbb{M}_n$ of the FP schedulable tasks sets. In particular, region $\mathbb{M}_n$ is defined as follows:

$$\mathbb{M}_n(T_1,\dots T_n, D_1,\dots D_n) = \{(C_1,\dots,C_n) \in R_+^n : \Gamma_n \text{ is schedulable by FP}\}. \quad (4)$$

We note that periods T_i and deadlines D_i are considered as parameters, whereas C_i are the free variables. Hence, we obtain a constraint on the C_i variables, which is a function of all the T_i and the D_i. The space in which every coordinate is represented by a task computation time C_i, is called the C-space. In the following section, we express region $\mathbb{M}_n$ in a convenient form which simplifies the feasibility check.

3 Expressing $\mathbb{M}_n$

A first attempt to analytically characterize the $\mathbb{M}_n$ region in the C-space was indirectly done by Lehoczky et al. [2]. In their work, deadlines are assumed equal to periods, whereas, in this paper, from Section 4 on, we consider the more general case of $D_i \le T_i$. They proved the following theorem [2]:

Theorem 1 (Theorem 2 in [2]). *Given a periodic task set* $\Gamma_n = \{\tau_1,\dots,\tau_n\}$,

1. *τ_i is feasibly schedulable (for any task phasing) by the RM algorithm if and only if:*

$$L_i = \min_{t\in\mathcal{S}_i} \frac{\sum_{j=1}^{i}\left\lceil \frac{t}{T_j}\right\rceil C_j}{t} \le 1,$$

where $\mathcal{S}_i = \{rT_j : j = 1\dots i, r = 1\dots \left\lfloor \frac{T_i}{T_j}\right\rfloor\}$.

2. *The entire task set is feasibly schedulable (for any task phasing) by RM if and only if:*

$$\max_{i=1\dots n} L_i \le 1.$$

Manipulating this result, we can restate the theorem in a more expressive form (in the next mathematical passages we will widely use the logical OR operator $\vee$ and the logical AND operator $\wedge$):

$$\max_{i=1\dots n}\min_{t\in\mathcal{S}_i}\frac{\sum_{j=1}^{i}\left\lceil \frac{t}{T_j}\right\rceil C_j}{t} \le 1 \Longleftrightarrow$$
$$\Longleftrightarrow \bigwedge_{i=1\dots n}\min_{t\in\mathcal{S}_i}\frac{\sum_{j=1}^{i}\left\lceil \frac{t}{T_j}\right\rceil C_j}{t} \le 1 \Longleftrightarrow$$
$$\Longleftrightarrow \bigwedge_{i=1\dots n}\bigvee_{t\in\mathcal{S}_i}\frac{\sum_{j=1}^{i}\left\lceil \frac{t}{T_j}\right\rceil C_j}{t} \le 1.$$

The last result provides a first analytical formulation of the $\mathbb{M}_n$ feasibility region in the important subcase of Rate Monotonic priorities. It is expressed by the following theorem:

TABLE 1
A Three-Task Set Example of $\mathcal{S}_i$
When $T_1 = 3$, $T_2 = 8$, and $T_3 = 20$

i	T_i	$\mathcal{S}_i$
1	3	$\{3\}$
2	8	$\{3, 6, 8\}$
3	20	$\{3, 6, 8, 9, 12, 15, 16, 18, 20\}$

Theorem 2. *When deadlines are equal to periods, the region of the schedulable task sets $\mathbb{M}_n$, as defined by (4), is given by:*

$$\mathbb{M}_n(T_1,\dots,T_n,D_1,\dots,D_n)|_{D_i=T_i} = \{(C_1,\dots,C_n)\in\mathbf{R}_+^n :$$
$$\bigwedge_{i=1\dots n}\ \bigvee_{t\in\mathcal{S}_i}\ \sum_{j=1}^{i}\left\lceil\frac{t}{T_j}\right\rceil C_j \le t\},$$

where $\mathcal{S}_i = \{r\,T_j : j = 1\dots i, r = 1\dots \left\lfloor\frac{T_i}{T_j}\right\rfloor\}$.

Proof. It directly follows from Theorem 1 and (4) which defines $\mathbb{M}_n$. □

To understand this result, consider a simple example composed by three tasks. Table 1 reports the periods T_i and the sets $\mathcal{S}_i$.

The equations we get by expanding Theorem 2 are:

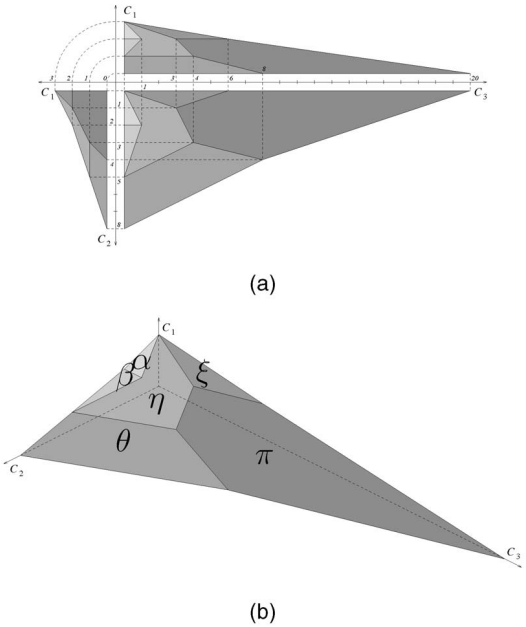
$$\left\{\begin{array}{llll}
C_1 \le 3 & 3\in\mathcal{S}_1 & & \\
\| \ C_1+C_2 \le 3 & 3\in\mathcal{S}_2 & & \\
\| \ 2C_1+C_2 \le 6 & 6\in\mathcal{S}_2 & \text{plane } \alpha \text{ in Fig. 1} & \\
\| \ 3C_1+C_2 \le 8 & 8\in\mathcal{S}_2 & \text{plane } \beta \text{ in Fig. 1} & \\
\| \ C_1+C_2+C_3 \le 3 & 3\in\mathcal{S}_3 & & \\
\| \ 2C_1+C_2+C_3 \le 6 & 6\in\mathcal{S}_3 & & \\
\| \ 3C_1+C_2+C_3 \le 8 & 8\in\mathcal{S}_3 & & \\
\| \ 3C_1+2C_2+C_3 \le 9 & 9\in\mathcal{S}_3 & & \\
\| \ 4C_1+2C_2+C_3 \le 12 & 12\in\mathcal{S}_3 & & \\
\| \ 5C_1+2C_2+C_3 \le 15 & 15\in\mathcal{S}_3 & \text{plane } \eta & \\
\| \ 6C_1+2C_2+C_3 \le 16 & 16\in\mathcal{S}_3 & \text{plane } \theta & \\
\| \ 6C_1+3C_2+C_3 \le 18 & 18\in\mathcal{S}_3 & \text{plane } \xi & \\
\| \ 7C_1+3C_2+C_3 \le 20 & 20\in\mathcal{S}_3 & \text{plane } \pi, &
\end{array}\right. \quad (5)$$

where the $\|$ symbol denotes the logical OR among the equations in the array, whereas the $\{$ symbol denotes the logical AND.[1] Fig. 1 shows a graphical representation of the $\mathbb{M}_3(3, 8, 20)$ region in the C-space.

It is now worth making the following considerations:

- **Observation 1**. The $\mathbb{M}_n$ region is delimited by planes (hyperplanes in higher dimensions). Every plane equation must be contained in the equation list (5). The opposite is not true; in fact, there can be equations in the list that are not shown in the picture because ORed with a more relaxed one.
- **Observation 2**. In Fig. 1, we can distinguish six different planes, meaning that in the equation list (5) there are 7 useless equations. Clearly, the situation can change for different values of the periods.
- **Observation 3**. A necessary and sufficient test can be derived by translating the task set $\Gamma_n = \{(T_1, C_1), \dots, (T_n, C_n)\}$ into a point in the C-space and then checking whether it belongs to $\mathbb{M}_n(T_1,\dots,T_n)$.

1. The notation ("$\|$","{") is used instead of the standard ("$\vee$","$\wedge$") for a better readability.



(a)

(b)

Fig. 1. A view of $\mathbb{M}_3(3, 8, 20)$ in the C-space: (a) the projection view, (b) the isometric view.

- **Observation 4**. Such a formulation is appropriate for making a tunable schedulability test. In fact, every subregion $\mathbb{H} \subseteq \mathbb{M}_n$, obtained by eliminating some of the equations to be OR-ed, is a subset of $\mathbb{M}_n$, and hence the feasibility check in $\mathbb{H}$ will be less complex (i.e., less equations to be checked) and "less necessary" (i.e., smaller region) than the one based on $\mathbb{M}_n$.
- **Observation 5**. For large task sets, the number of equations to be checked is huge and is equal to the sum of the number of elements in all $\mathcal{S}_i$. When the ratio T_n/T_1 is large, the number of equations is so high that prevents a practical application of Theorem 2 for any guarantee test.
- **Observation 6**. The sum reported in Theorem 2 can also be written as

$$\sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil \frac{T_j}{t} U_j \leq 1.$$

  We note that coefficients $\left\lceil \frac{t}{T_j} \right\rceil \frac{T_j}{t}$ are always greater than or equal to 1. Coefficients close to 1 delimit large regions. In particular, if there exists a t such that $\left\lceil \frac{t}{T_j} \right\rceil \frac{T_j}{t}$ is equal to 1 for all j, the $\mathbb{M}_n$ region becomes $\sum C_i/T_i \leq 1$. This is just the mathematical translation of the commonly known behavior of RM when all the periods are in the same harmonic chain [13].
- **Observation 7**. Operation research algorithms could also be applied on the $\mathbb{M}_n$ region to find the maximum achievable utilization bound. This approach has been followed by Park et al. [15]. However, this method involves a higher number of equations, so it is either slow or not very accurate.

Among the considerations above, the most negative seems to be Observation 5, which says that the high number of equations prevents a practical use of the method. However, as stated in Observation 2, many equations in (5) are useless, so the idea is to reduce the number of equations by eliminating the redundant elements in $\mathcal{S}_i$.

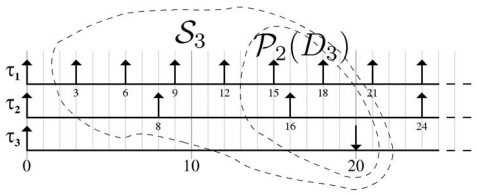


Fig. 2. Comparison between $\mathcal{S}_3$ and $\mathcal{P}_2(D_3)$ for the task set shown in Table 1.

Before entering into a detailed discussion of such a reduction process, it is worth noting that such a reduction has been so effective as to make the test not only applicable in practice, but even better than all other tests proposed in the literature. The reduction has been condensed in the next theorem, which is the key contribution of the paper. The theorem and its proof are reported in the next section.

4 Feasibility Analysis in the C-Space

The following theorem significantly reduces the number of equations that are needed to delimit the $\mathbb{M}_n$ region.

Theorem 3. *The region of the schedulable task sets $\mathbb{M}_n$, as defined by (4), is given by*

$$\mathbb{M}_n(T_1, \ldots, T_{n-1}, D_1, \ldots, D_n) = \{(C_1, \ldots, C_n) \in R_+^n : \bigwedge_{i=1 \ldots n} \bigvee_{t \in \mathcal{P}_{i-1}(D_i)} C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t\},$$

where $\mathcal{P}_i(t)$ is defined by the following recurrent expression:

$$\begin{cases} \mathcal{P}_0(t) = \{t\} \\ \mathcal{P}_i(t) = \mathcal{P}_{i-1}\left(\left\lfloor \frac{t}{T_i} \right\rfloor T_i \right) \cup \mathcal{P}_{i-1}(t). \end{cases} \quad (6)$$

Before proving the theorem, we first illustrate its application. Then, the formal proof will be given in a dedicated subsection.

The difference between this result and that of Theorem 2 is only the presence of the set $\mathcal{P}_{i-1}(D_i)$, instead of $\mathcal{S}_i$. This may seem a little change, but it is not. For example, Fig. 2 shows the difference between the sets $\mathcal{S}_3$ and $\mathcal{P}_2(D_3)$ for the task set reported in Table 1 (deadlines are assumed equal to the periods).

With the introduction of set $\mathcal{P}_{i-1}(D_i)$, the test to check whether a task set belongs to $\mathbb{M}_n$ becomes not only reasonable, but better than every necessary and sufficient test proposed in the literature. Notice that $\mathcal{P}_{i-1}(T_i) \subseteq \mathcal{S}_i$ (remember that $\mathcal{S}_i$ is defined when $D_i = T_i$). This proposition can be formally proved by induction on i and it is clarified by Fig. 2. This allows to dramatically reduce and bound the time needed for the schedulability test.

Due to the double recurrent form of its definition, the "worst-case" cardinality of a generic $\mathcal{P}_i(t)$ set is 2^i. We intentionally say "worst-case" cardinality because if the two sets to be joined overlap, the cardinality, of course, reduces.

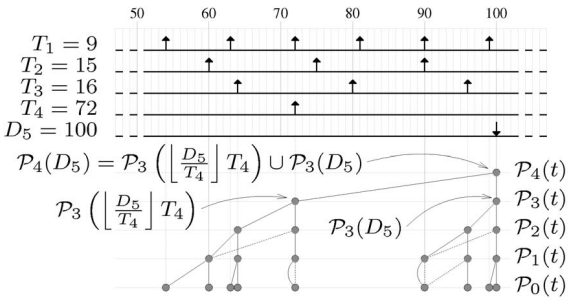


Fig. 3. An example of $\mathcal{P}_i(t)$.

We note that set $\mathcal{P}_{i-1}(D_i)$ is only a function of D_i and of all T_j for j from 1 to $i-1$ and, more in general, $\mathcal{P}_i(t)$ is a function of time t and of all the periods $T_1, \ldots, T_i$. This remark explains why the region in the C-space is not a function of the lowest priority period T_n. Fig. 3 shows all the recurrent calls of $\mathcal{P}_4(D_5)$ in the case of $T_1 = 9$, $T_2 = 15$, $T_3 = 16$, $T_4 = 36$, and $D_5 = 100$. In this figure, we can clearly see how the $\mathcal{P}_j(t)$ definition works. Every set $\mathcal{P}_j(t)$ is represented by a big grey dot. When $j \neq 0$, each set is the union of two sets, and the union relationship is represented by a line connecting two sets. A dashed line means that the union does not contribute with new points. Such a case happens, for example, when $\left\lfloor \frac{t}{T_j} \right\rfloor T_j = t$.

4.1 Proof of Theorem 3

To prove Theorem 3, we need the following definitions:

Definition 1. *A job τ_{ik} is said to be active at time t if $r_{ik} < t < f_{ik}$.*

Definition 2. *The processor is i-**busy** at time t if there exists a job of a task in Γ_i active in t. More formally, the following function represents the subset of points in $[0, b]$ where the processor is i-busy:*

$$Busy(\Gamma_i, b) = \{t \in [0, b] : \exists \tau_{jk} \text{ such that } \tau_{jk} \text{ is active at } t, \tau_j \in \Gamma_i\}.$$

Definition 3. *The worst-case workload $W_i(b)$ of the i highest priority tasks in $[0, b]$ is the total time the processor is i-busy in $[0, b]$. By extension, $W_0(b) = 0$ for all b.*

Note that, using the concept of workload, the schedulability condition of τ_i can be expressed by

$$C_i + W_{i-1}(D_i) \leq D_i. \tag{7}$$

Definition 4. *Given the subset Γ_i of the i highest priority tasks, we define $\psi_i(b)$ to be the last instant in $[0, b]$ in which the processor is not i-busy, that is:*

$$\psi_i(b) = \max\{t \in [0, b] \ \wedge \ t \notin Busy(\Gamma_i, b)\}.$$

By Definition 1, the set $Busy(\Gamma_i, b)$ is the union of open intervals, hence the set $[0, b] \setminus Busy(\Gamma_i, b)$ has always a maximum and so the last idle instant $\psi_i(b)$ is well defined.[2]

This formalism is needed because the point $\psi_i(b)$ is useful for simplifying the computation of $W_i(b)$ and to express the FP schedulability condition (7). The following lemma provides a method to easily compute the workload in $[0, b]$ through the last idle instant $\psi_i(b)$.

Lemma 1. *Given a subset $\Gamma_i = \{\tau_1, \ldots, \tau_i\}$ of the i highest priority tasks, the workload $W_i(b)$ can be written as*

$$W_i(b) = \sum_{j=1}^{i} \left\lceil \frac{\psi_i(b)}{T_j} \right\rceil C_j + (b - \psi_i(b)).$$

Proof. From the definition of $\psi_i(b)$, we note that no task instance in Γ_i is active at $\psi_i(b)$ and all the released jobs have been completed at that time. Hence,

$$W_i(\psi_i(b)) = \sum_{j=1}^{i} \left\lceil \frac{\psi_i(b)}{T_j} \right\rceil C_j.$$

Moreover, because the processor is always i-busy in $[\psi_i(b), b]$, the workload of the i highest priority tasks in such an interval is $(b - \psi_i(b))$. Hence, the lemma follows. □

Lemma 2. *For any schedulable task subset $\Gamma_i = \{\tau_1, \ldots, \tau_i\}$,*

$$W_i(b) = \min_{t \in [0,b]} \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j + (b - t). \tag{8}$$

Proof. We first observe that $\sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j$ is the processor demand in $[0, t]$, which is the time required by the tasks to be executed in $[0, t]$. So, it must be that

$$\forall t \quad W_i(t) \leq \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j$$

because the processor has no other job to be executed.

Moreover, since in a feasible schedule the workload in $[t, b]$(i.e., $W_i(b) - W_i(t)$) is smaller than the length of the interval, we have that

$$\forall t \in [0, b] \quad W_i(b) - W_i(t) \leq (b - t).$$

or, equivalently,

$$\forall t \in [0, b] \quad W_i(b) \leq \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j + (b - t). \tag{9}$$

Now, since $\psi_i(b) \in [0, b]$, for Lemma 1 there exists a value in $[0, b]$ for which the equality holds, so the lemma follows: □

The meaning of Lemma 2 is illustrated in Fig. 4, which shows the upper bounding function[3] of the workload for a specific set of four periodic tasks. Notice that the minimum of such a function is $W_4(b)$ and it falls in $\psi_4(b)$.

In other words, the workload $W_i(b)$ in $[0, b]$ can be upper estimated by the computation of $\sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j + (b - t)$ in any $t \in [0, b]$. This is not directly useful because this estimation is

2. The symbol $\setminus$ denotes the set difference operator.

3. Black dots indicate the value of the function in the discontinuities.

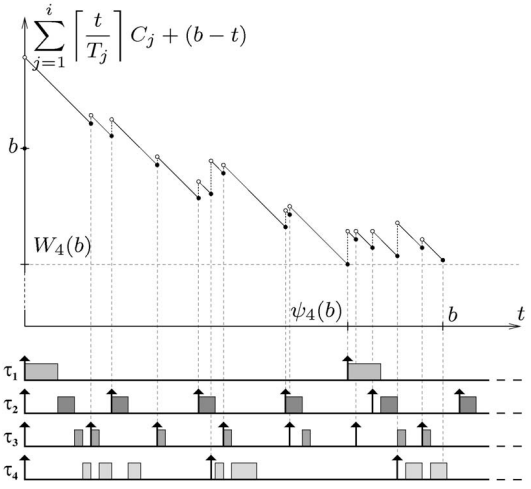


Fig. 4. Lemma 2 interpretation.

rough, but the workload can be exactly calculated by the function $\sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j + (b-t)$ once we know the last idle instant $\psi_i(b)$. Unfortunately the complexity moves from the workload estimation to the $\psi_i(b)$ search. So we now restrict the set of possible values of $\psi_i(b)$ by the following lemma.

Lemma 3. *Given a task subset $\Gamma_i = \{\tau_1, \tau_2, \ldots, \tau_i\}$ schedulable by FP, let $\psi_i(b)$ be the last idle instant in $[0, b]$ as defined in Definition 4, and let $\mathcal{P}_i(b)$ be the set of points defined by the following recurrent expression:*

$$\begin{cases} \mathcal{P}_0(b) = \{b\} \\ \mathcal{P}_i(b) = \mathcal{P}_{i-1}\left(\left\lfloor \frac{b}{T_i} \right\rfloor T_i \right) \cup \mathcal{P}_{i-1}(b). \end{cases}$$

Then,

$$\psi_i(b) \in \mathcal{P}_i(b).$$

Proof. We demonstrate the lemma by induction on i.

Initial Step. If $i = 1$, we have to prove that, for a schedulable task τ_1, $\psi_1(b) \in \mathcal{P}_1(b)$ for all b. We observe that, in this case,

$$\mathcal{P}_1(b) = \mathcal{P}_0\left(\left\lfloor \frac{b}{T_1} \right\rfloor T_1 \right) \cup \mathcal{P}_0(b) = \left\{ \left\lfloor \frac{b}{T_1} \right\rfloor T_1, b \right\}.$$

Since τ_1 is schedulable, then the last idle instant $\psi_1(b)$ can only be:

1. at $\lfloor b/T_1 \rfloor T_1$, if the last instance in $[0, b]$ of τ_1 is active at b;
2. at b, otherwise.

Both values are in $\mathcal{P}_1(b)$ and the initial step is proven.

Inductive Step. If $\psi_i(b) \in \mathcal{P}_i(b)$ for all b, we have to prove that, given a schedulable task subset $\Gamma_{i+1} = \{\tau_1, \ldots, \tau_{i+1}\}$, then $\psi_{i+1}(b) \in \mathcal{P}_{i+1}(b)$ for all b.

Let us consider the time interval $[\lfloor b/T_{i+1} \rfloor T_{i+1}, b]$. In this interval, two things can happen:

1. the processor is $(i+1)$-busy in the whole interval;
2. there exists an instant of time at which the processor is not $(i+1)$-busy.

In the first case, $\psi_{i+1}(b) = \psi_{i+1}(\lfloor b/T_{i+1} \rfloor T_{i+1})$ because in $[\lfloor b/T_{i+1} \rfloor T_{i+1}, b]$ the processor always runs a task in Γ_{i+1}. Moreover, it must be

$$\psi_{i+1}(\lfloor b/T_{i+1} \rfloor T_{i+1}) = \psi_i(\lfloor b/T_{i+1} \rfloor T_{i+1});$$

otherwise, the last instance of τ_{i+1} in $[0, \lfloor b/T_{i+1} \rfloor T_{i+1}]$ would miss its deadline at $\lfloor b/T_{i+1} \rfloor T_{i+1}$, contradicting the hypothesis of τ_{i+1} schedulability.

In the second case, let $x \in [\lfloor b/T_{i+1} \rfloor T_{i+1}, b]$ be an instant of time where no tasks in Γ_{i+1} are active. Since at time x the $\lfloor b/T_{i+1} + 1 \rfloor$th job of τ_{i+1} is terminated, τ_{i+1} is never active in $[x, b]$. This implies that $\psi_{i+1}(b) = \psi_i(b)$.

Merging the two cases, we get:

$$\psi_{i+1}(b) = \psi_i\left(\left\lfloor \frac{b}{T_{i+1}} \right\rfloor T_{i+1} \right) \bigvee \psi_{i+1}(b) = \psi_i(b).$$

For the inductive hypothesis, we have that: $\psi_i(\lfloor b/T_{i+1} \rfloor\ T_{i+1}) \in \mathcal{P}_i(\lfloor b/T_{i+1} \rfloor\ T_{i+1})$ and $\psi_i(b) \in \mathcal{P}_i(b)$. Hence,

$$\psi_{i+1}(b) \in \mathcal{P}_i\left(\left\lfloor \frac{b}{T_{i+1}} \right\rfloor T_{i+1} \right) \bigcup \mathcal{P}_i(b)$$

and finally, for the $\mathcal{P}_{i+1}(b)$ definition:

$$\psi_{i+1}(b) \in \mathcal{P}_{i+1}(b),$$

which proves the inductive step and the lemma. □

Lemma 4. *Given a task subset $\Gamma_i = \{\tau_1, \ldots, \tau_i\}$ schedulable by FP and the set $\mathcal{P}_i(b)$ as defined in (6), the workload $W_i(b)$ is*

$$W_i(b) = \min_{t \in \mathcal{P}_i(b)} \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j + (b-t).$$

Proof. By observing that $\mathcal{P}_i(b) \subseteq [0, b]$, the lemma directly follows from Lemmae 1, 2, and 3. □

We are now ready to prove the theorem.

Proof of Theorem 3. We have to prove the equivalence between the following two sentences:

1. for all $i = 1 \ldots n$, τ_i is schedulable by a fixed priority algorithm;
2. $\bigwedge_{i=1 \ldots n} \bigvee_{t \in \mathcal{P}_{i-1}(D_i)} \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j \le t.$

Without loss of generality, we can assume that, for the schedulability of task τ_i, all tasks $\tau_1, \ldots, \tau_{i-1}$ are schedulable. In this case, the schedulability condition of the single task τ_i can be written as:

$$C_i + W_{i-1}(D_i) \le D_i,$$

where $W_{i-1}(D_i)$ is the workload of the first $i-1$ tasks in the interval $[0, D_i]$. Using Lemma 4, such a condition of individual schedulability can be written as:

$$C_i + \min_{t \in \mathcal{P}_{i-1}(D_i)} \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j + (D_i - t) \leq D_i$$

$$C_i + \min_{t \in \mathcal{P}_{i-1}(D_i)} \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j - t \leq 0$$

$$\bigvee_{t \in \mathcal{P}_{i-1}(D_i)} C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t.$$

Hence, the schedulability condition for all the tasks is clearly given by

$$\bigwedge_{i=1 \ldots n} \bigvee_{t \in \mathcal{P}_{i-1}(D_i)} C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t$$

as required by Theorem 3. □

In the next paragraphs, two examples are illustrated to extend the applicability of this result: First, one considers a non-RM priority assignment and the second one deals with deadlines smaller than periods.

4.2 Example with Non-RM priorities

Let us consider the same tasks shown in Table 1. However, in this example, we invert the priorities of the tasks with period 8 and 20 (we still assume $D_i = T_i$). In particular, the resulting tasks set is ordered as follows:

i	T_i	D_i	$\mathcal{P}_{i-1}(D_i)$
1	3	3	$\{3\}$
2	20	20	$\{18, 20\}$
3	8	8	$\{0, 6, 8\}$

By expanding the equations given by Theorem 3, which provides a necessary and sufficient condition, we get the following inequalities, which describe the exact schedulability region:

$$\begin{cases} C_1 \leq 3 \\ \| \\ 6C_1 + C_2 \leq 18 \\ 7C_1 + C_2 \leq 20 \\ \| \\ C_3 \leq 0 \\ 2C_1 + C_2 + C_3 \leq 6 \\ 3C_1 + C_2 + C_3 \leq 8 \end{cases}$$

Notice that the inequality $C_3 \leq 0$ would force C_3 to be zero. Such a strict constraint is not a problem because it is ORed with more relaxed ones.

As was done for the Rate Monotonic case, we can now draw the schedulability region in the C-space, formerly defined as $\mathbb{M}_n$. The result is depicted in Fig. 5.

It is worth noticing that this region is smaller than the RM one (see Fig. 1). This is a clear consequence of the RM optimality: A task set schedulable by a generic fixed priority assignment is also schedulable by Rate Monotonic.

4.3 Example with Deadlines Smaller than Periods

We now consider a task set with deadlines smaller than periods. Tasks are the same as those in Table 1, but deadlines D_2 and D_3 are decreased by one unit of time, leading to the following task set:

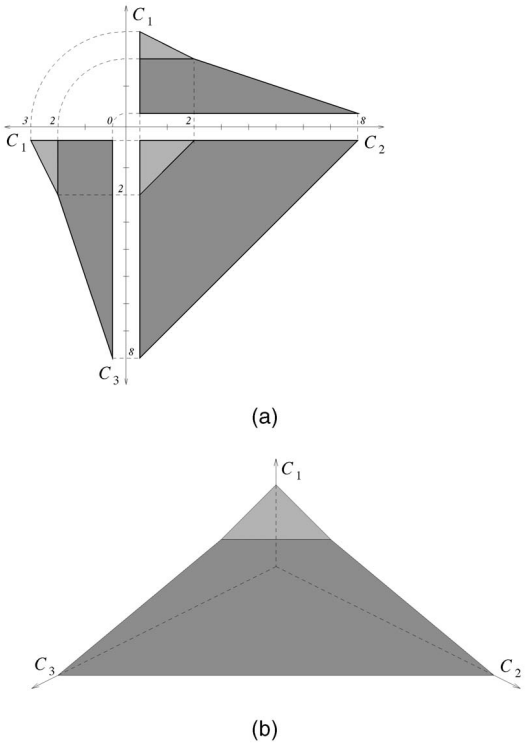


Fig. 5. The schedulability region when $\mathrm{FP} \neq \mathrm{RM}$: (a) the projection view and (b) the isometric view.

i	T_i	D_i	$\mathcal{P}_{i-1}(D_i)$
1	3	3	$\{3\}$
2	8	7	$\{6, 7\}$
3	20	19	$\{15, 16, 18, 19\}$

Applying Theorem 3, we get the following set of inequalities:

$$\begin{cases} C_1 \leq 3 \\ \| \\ 2C_1 + C_2 \leq 6 \\ 3C_1 + C_2 \leq 7 \\ \| \\ 5C_1 + 2C_2 + C_3 \leq 15 \\ 6C_1 + 2C_2 + C_3 \leq 16 \\ 6C_1 + 3C_2 + C_3 \leq 18 \\ 7C_1 + 3C_2 + C_3 \leq 19 \end{cases}$$

The resulting schedulability region $\mathbb{M}_n$ is shown in Fig. 6. We notice that the region achieved in this case is smaller than that shown in Fig. 1, confirming that shortening deadlines reduces schedulability.

5 The Hyperplanes δ-Exact Test

In this section, we first show how the necessary and sufficient test is derived and then we describe the method to make it tunable.

As we saw in the last section, the FP schedulability condition in Theorem 3 can be equivalently expressed as

$$\forall i = 1 \ldots n \quad C_i + W_{i-1}(D_i) \leq D_i,$$

where the workload $W_{i-1}(D_i)$ is given by Lemma 4:

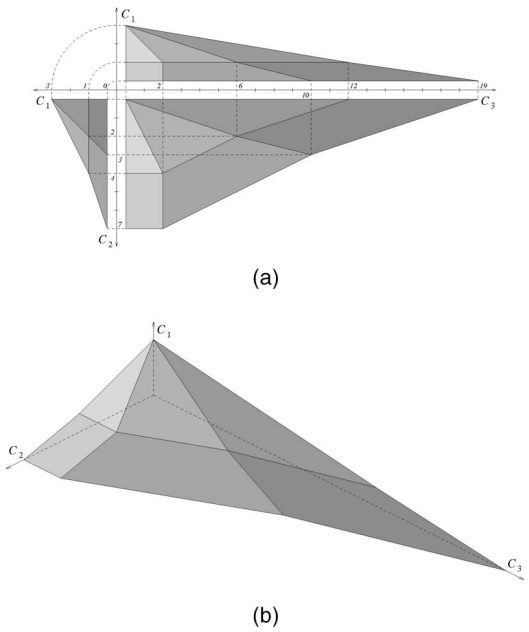


Fig. 6. The schedulability region when $D_i < T_i$: (a) the projection view and (b) the isometric view.

$$W_{i-1}(D_i) = \min_{t \in \mathcal{P}_{i-1}(D_i)} \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j + (D_i - t)$$

and $W_0(D_1) = 0$ by extension.

Using this formulation, the pseudo-C code of the necessary and sufficient test can be written as follows:

```
Boolean FPTest(Γ_n) {
int i;
for (i = 1; i ≤ n; i++)
  if (C_i + WorkLoad(i − 1, D_i) > D_i)
    return false;
  return true;
}
```

Now, we focus our attention on the function $WorkLoad(i, b)$, which is the workload of the i highest priority tasks in $[0, b]$, also called $W_i(b)$ previously. From Lemma 4, we know that

$$W_i(b) = \min_{t \in \mathcal{P}_i(b)} \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j + (b - t)$$

and then, from the definition of $\mathcal{P}_i(b)$, we can write (in the following expressions we define $f = \lfloor b/T_i \rfloor$ and $c = \lceil b/T_i \rceil$):

$$W_i(b) = \min \left\{ \min_{t \in \mathcal{P}_{i-1}(fT_i)} \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j + (b - t), \min_{t \in \mathcal{P}_{i-1}(b)} \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j + (b - t) \right\}, \quad (10)$$

where we split the set $\mathcal{P}_i(b)$ in the two subsets which compose it.

We now write these expressions in a more meaningful form. By noting that the ith element in the first sum is always equal to fC_i (due to the schedulability of τ_i), the first element of (10) can be written as

$$\begin{aligned} &\min_{t \in \mathcal{P}_{i-1}(fT_i)} \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j + (b - t) = \\ &= fC_i + \min \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j + (b - t) = \\ &= b + fC_i - fT_i + \min \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j + (fT_i - t) = \\ &= b - f(T_i - C_i) + W_{i-1}(fT_i) \end{aligned}$$

from the result of Lemma 4. Similarly, the second element in (10) can be written as:

$$\begin{aligned} &\min_{t \in \mathcal{P}_{i-1}(b)} \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j + (b - t) = \\ &= cC_i + \min \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j + (b - t) = \\ &= cC_i + W_{i-1}(b). \end{aligned}$$

Hence, (10) can be expressed in a recurrent form as follows:

$$W_i(b) = \min\{b - f(T_i - C_i) + W_{i-1}(fT_i), cC_i + W_{i-1}(b)\}. \quad (11)$$

Such a recurrent expression of $W_i(b)$ directly follows from the recurrent definition of $\mathcal{P}_i(b)$. As in the $\mathcal{P}_i(b)$ definition the two sets $\mathcal{P}_{i-1}(\lfloor b/T_i \rfloor)$ and $\mathcal{P}_{i-1}(b)$ could overlap (see Fig. 3), it can happen that, for particular values of j and t , two calls of $W_j(t)$ could return the same value. In this case, following both branches would be a waste of time. This can be avoided by keeping track of the execution flow through a global variable which can be used to prune all the useless branches. The resulting algorithm is shown below:

```
double lastψ[BIG_ENOUGH];
double lastWorkLoad[BIG_ENOUGH];

double WorkLoad(int i, double b) {
  int f, c;
  double branch0, branch1;

  if (i ≤ 0) return 0;
  if (b ≤ lastψ[i]) /* if WorkLoad(i, b) already computed */
    return lastWorkLoad[i]; /* don't go further */
  f = ⌊b/T_i⌋; c = ⌈b/T_i⌉;
  branch0 = b − f (T_i − C_i) + WorkLoad(i − 1, f T_i);
  branch1 = c C_i + WorkLoad(i − 1, b);
  lastψ[i] = b;
  lastWorkLoad[i] = min(branch0, branch1);
  return lastWorkLoad[i];
}
```

We note that the ith element of the array $lastWorkLoad$ keeps track of the call $WorkLoad(i, last\psi[i])$. The necessary and sufficient algorithm obtained in this way, called the Hyperplanes Exact Test (HET), is quite more efficient than the response time based algorithm. The performance comparison is presented in Section 6.

5.1 The δ-HET Test

The tunable test δ-HET is obtained by reducing the $\mathcal{P}_i(b)$ set as a function of an additional parameter δ, as follows:

$$\begin{cases} \mathcal{P}_0(b,\delta) = \{b\} & \forall \delta \\ \mathcal{P}_i(b,\delta) = \begin{cases} \mathcal{P}_{i-1}\left(\left\lfloor \frac{b}{T_i} \right\rfloor T_i, \delta\right) \cup \mathcal{P}_{i-1}(b,\delta) & \text{if } b\delta \geq T_i \\ \mathcal{P}_{i-1}\left(\left\lfloor \frac{b}{T_i} \right\rfloor T_i, \delta\right) & \text{otherwise.} \end{cases} \end{cases}$$

By this definition, it is easy to prove the following properties of the $\mathcal{P}_i(b,\delta)$ set:

$$\delta_1 \leq \delta_2 \Leftrightarrow \mathcal{P}_i(b,\delta_1) \subseteq \mathcal{P}_i(b,\delta_2)$$
$$\mathcal{P}_i(b,1) = \mathcal{P}_i(b).$$

Similar properties also hold for the workload expressed by Lemma 4. If we define

$$W_i^{(\delta)}(b) = \min_{t \in \mathcal{P}_i(b,\delta)} \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j + (b-t),$$

then we have that

$$\delta_1 \leq \delta_2 \Leftrightarrow W_i^{(\delta_1)}(b) \geq W_i^{(\delta_2)}(b)$$
$$W_i^{(1)}(b) = W_i(b).$$

Both properties follow directly from the definition of $W_i^{(\delta)}$. In particular, the first property derives form the observation that the minimum cannot decrease when computed on a smaller set.

The δ-HET test can then be derived by substituting $\mathcal{P}_i(b)$ with $\mathcal{P}_i(b,\delta)$. The resulting algorithm for computing the new workload function is illustrated below.

```
double lastψ[BIG_ENOUGH];
double lastWorkLoad[BIG_ENOUGH];

double WorkLoad(int i, double b, double δ) {
  int f, c;
  double branch0, branch1;

  if (i ≤ 0)
    return 0;
  if (b ≤ lastψ[i]) /* if WorkLoad(i,b) already computed */
    return lastWorkLoad[i]; /* don't go further */
  f = ⌊b/T_i⌋; c = ⌈b/T_i⌉;
  branch0 = b − f (T_i − C_i) + WorkLoad(i − 1, f T_i);
  if (T_i ≤ b δ)
    branch1 = c C_i + WorkLoad(i − 1, b);
  else
    branch1 = branch0; /* one branch is cut! */
  lastψ[i]] = b;
  lastWorkLoad[i] = min(branch0, branch1);
  return lastWorkLoad[i];
}
```

6 Performance of δ-HET

In this section, we compare the δ-HET test with the common schedulability tests proposed in the literature for the RM algorithm, when deadlines are equal to periods. The performance of a test is evaluated in terms of both acceptance ratio and complexity. The acceptance ratio is measured by the number of accepted task sets with respect to those accepted by a necessary and sufficient test. In particular, if U denotes the set of task sets for which there exists a feasible schedule, let $S^* \subseteq U$ be those task sets which are schedulable with fixed priorities (and, hence, RM schedulable), and let $S^T \subseteq S^*$ be those task sets which are guaranteed by a sufficiency test T. Then the acceptance ratio $acceptanceRatio(T)$ of a test T is computed as follows:

$$acceptanceRatio(T) = \frac{|S^T|}{|S^*|},$$

where the operator $|\cdot|$ indicates "the number of elements." From this definition it follows that $acceptanceRatio(T) \leq 1$ for all T, and $acceptanceRatio(T) = 1$ for the response-time analysis and all the necessary and sufficient tests.

The complexity of a test is measured by counting the number of innermost loop iterations. Formally, we define $stepsNumber(T,\Gamma)$ as the number of innermost loop steps required to compute the guarantee test T on the tasks set Γ. Moreover, we model the computation time of a guarantee test T as a random variable $steps(T)$, defined by the following cumulative distribution function (c.d.f.):

$$F_{\text{steps}(T)}(x) = P\{stepsNumber(T,\Gamma) \leq x\} \qquad \Gamma \in U,$$

where $P\{\cdot\}$ denotes the probability of an event. This model is useful because the maximum and the average number of iteration steps can easily be extracted from the probability density function (p.d.f.), which is

$$f_{\text{steps}(T)} = \frac{dF_{\text{steps}(T)}}{x}.$$

In our experiments, simulations have been performed by generating 10^8 task sets, each composed by eight tasks. Task periods T_i were randomly extracted in $[1, 1000000]$ (with uniform distribution) and computation times C_i were computed as random variables in $[0, T_i]$ (also with uniform distribution). Fig. 7 illustrates the results of a first experiment, which compares three necessary and sufficient tests: the classical Response Time Analysis (RTA) of Audsley et al. [4], the Response Time analysis Improved (RTI) by Sjödin et al. [9], and the Hyperplanes Exact Test (HET) presented in

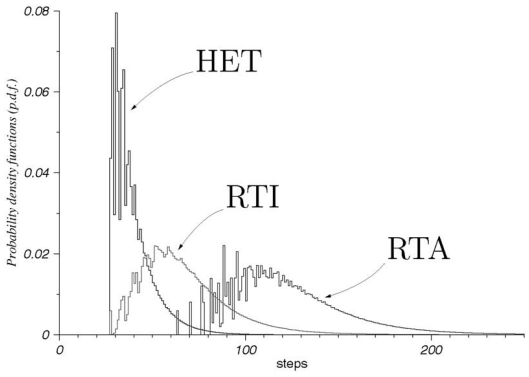


Fig. 7. p.d.f. of $steps(RTA)$, $steps(RTI)$, and $steps(HET)$.

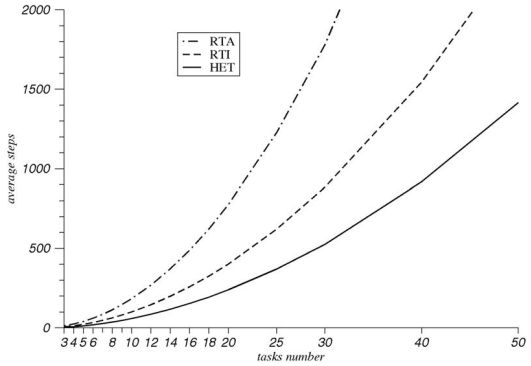


Fig. 8. Average number of steps as a function of the number of tasks.

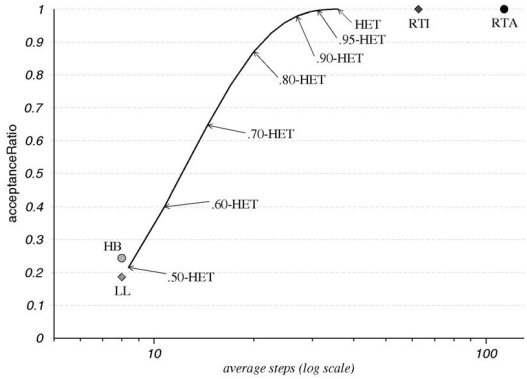


Fig. 9. Comparison of guarantee tests.

this paper. The figure plots the probability density achieved for the three tests.

We notice that the HET test, although pseudopolynomial, is significantly faster than the others, not only for the average case, but also, and especially, for its worst case. The appreciable "noise" of the probability density is not due to the low number of sets (10^8), but it comes from the intrinsic structure of the algorithm.

Fig. 8 shows the result of another experiment, in which we evaluated the dependency of the average number of steps on the number of tasks in the set. As we can clearly see, although for all tests the average number of steps increases exponentially, in the case of the HET algorithm the speed of growth is significantly smaller.

In a third experiment, we evaluated the performance of the δ-HET test with respect to several other tests proposed in the literature, such as the Response Time Analysis (RTA) [4], the Response Time analysis Improved (RTI) [9], the Liu and Layland test (LL) [1], and the Hyperbolic Bound (HB) [12]. The δ-HET has been executed for several values of $\delta \in [0.5, 1]$. The comparison is made in terms of both acceptance ratio and complexity, on a universe of 10^6 tasks sets consisting of eight tasks.

In particular, each test is characterized by two values (that is, the average number of steps and the acceptance ratio), thus it is represented as a point in a plane, having $steps(T)$ and $acceptanceRatio(T)$ as coordinates. In such a plane, the best area for a guarantee test is the one located around the upper-left corner, where the acceptance ratio is high and the complexity is low. The result of this experiment is shown in Fig. 9.

As we can see, the performance of the δ-HET test covers all the intermediate positions in the plane. In particular when $n = 8$, for $\delta = 0.5$, the δ-HET test has a performance similar to the one of the LL-test, whereas, for $\delta = 1$ (exact analysis), it is still significantly better than the RTI test.

7 Extensions

This section extends the proposed analysis to the case of shared resources and nonpreemptive scheduling. We then show some additional application of the proposed methodology.

7.1 Shared Resources

The blocking times due to resource sharing can be taken into account by introducing, for each task τ_i, a blocking term B_i to be added to the task computation time [7]. Hence, the schedulability condition (7) can be extended as follows:

$$C_i + B_i + W_{i-1}(D_i) \leq D_i. \tag{12}$$

Using the same reasoning, Theorem 3 can also be extended as follows, to take blocking times into account.

Theorem 4. *A task set Γ composed by tasks $\tau_i = (C_i, T_i, D_i, B_i)$ is schedulable if:*

$$\bigwedge_{i=1\ldots n} \bigvee_{t \in \mathcal{P}_{i-1}(D_i)} C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t - B_i. \tag{13}$$

Clearly, this condition is more stringent than the one expressed by Theorem 3. To better understand the impact of this result, we consider an example with the same task set as before, in which tasks τ_1 and τ_2 have blocking times $B_1 = B_2 = 1$.

i	T_i	D_i	B_i	$\mathcal{P}_{i-1}(D_i)$
1	3	3	1	$\{3\}$
2	8	7	1	$\{6, 7\}$
3	20	19	0	$\{15, 16, 18, 19\}$

Applying Theorem 3, the schedulability region in the C-space can be described by the following inequalities:

$$\begin{cases} C_1 \leq 2 \\ \| \; 2C_1 + C_2 \leq 5 \\ \| \; 3C_1 + C_2 \leq 6 \\ \| \; 5C_1 + 2C_2 + C_3 \leq 15 \\ \| \; 6C_1 + 2C_2 + C_3 \leq 16 \\ \| \; 6C_1 + 3C_2 + C_3 \leq 18 \\ \| \; 7C_1 + 3C_2 + C_3 \leq 19 \end{cases}$$

The shape of the region for the given example is illustrated in Fig. 10. Again, we notice that this region is smaller than that shown in Fig. 6, achieved in the absence of blocking times.

As a last remark, notice that the last theorem provides only a sufficient condition because when a blocking time is present, the worst-case scenario can be different than the synchronous (classical Liu and Layland) one.

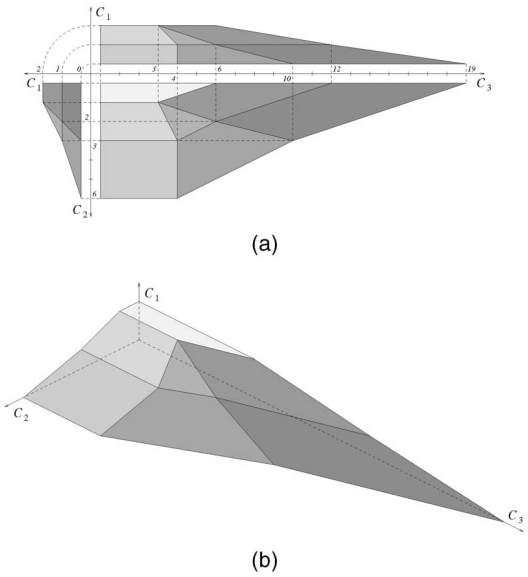


Fig. 10. The schedulability region in presence of shared resources: (a) the projection view and (b) the isometric view.

7.2 Nonpreemptive Scheduling

The proposed analysis can be immediately extended to nonpreemptive scheduling by observing that nonpreemptive scheduling is just a special case of resource sharing, where all tasks share a single resource for their entire execution. Hence, Theorem 4 can be applied in this case if blocking times are computed as

$$B_i = \max_{j>i} C_j \tag{14}$$

We now show how the schedulability region of task set in Table 1 is reduced in the nonpreemptive case. By applying Theorem 4 with (14), we get:

$$\begin{cases} C_1 \leq 3 - \max\{C_2, C_3\} \\ \| \; 2C_1 + C_2 \leq 6 - C_3 \\ \| \; 3C_1 + C_2 \leq 8 - C_3 \\ \| \; 5C_1 + 2C_2 + C_3 \leq 15 \\ \| \; 6C_1 + 2C_2 + C_3 \leq 16 \\ \| \; 6C_1 + 3C_2 + C_3 \leq 18 \\ \| \; 7C_1 + 3C_2 + C_3 \leq 20 \end{cases}$$

and then:

$$\begin{cases} C_1 + C_2 \leq 3 \\ C_1 + C_3 \leq 3 \\ \| \; 2C_1 + C_2 + C_3 \leq 6 \\ \| \; 3C_1 + C_2 + C_3 \leq 8 \\ \| \; 5C_1 + 2C_2 + C_3 \leq 15 \\ \| \; 6C_1 + 2C_2 + C_3 \leq 16 \\ \| \; 6C_1 + 3C_2 + C_3 \leq 18 \\ \| \; 7C_1 + 3C_2 + C_3 \leq 20 \end{cases}$$

Simplyfing the redundant equations, we finally get:

$$\begin{cases} C_1 + C_2 \leq 3 \\ C_1 + C_3 \leq 3 \end{cases} \tag{15}$$

Equation (15) summarize some known properties of the nonpreemptive scheduling algorithms. For example, it is well known that every single computation time C_i must be not greater than the smallest period. In the considered example, the smallest period (T_1) is 3 and the found condition deny all the C_i to be greater than 3.

8 Conclusions and Future Work

In this paper, we presented a novel approach for analyzing the schedulability of periodic task sets under the Rate Monotonic priority assignment. Such an approach allowed us to precisely describe the feasibility region in the space of task computation times (the C-space) and to derive a tunable guarantee test (δ-HET), by which we can balance acceptance ratio and complexity. Such a tunability property of the δ-HET test is important in those cases in which the performance of a polynomial time test is not sufficient for achieving high processor utilization, and the overhead introduced by exact tests is too high for an online admission control.

We believe that the proposed formulation opens a novel direction in the schedulability analysis of fixed priority systems, allowing further research in this domain. As a future work, we plan to investigate the case where task computation times are considered as random variables with known probability distribution. In this case, the probability to meet the deadlines of a task set can be computed by the integral of the C_i probability density on the $\mathbb{M}_n$ region.

Another interesting situation that can be addressed by the proposed method deals with the case in which not all the computation times are fixed, but we have some freedom to select the size of some tasks (for instance when using imprecise computation models). In this condition, the HET approach can be used to decide the best values of those free variables in order to improve schedulability. In fact, fixing a C_i value is equivalent to cutting the $\mathbb{M}_n$ region in the C-space at a certain coordinate (this can easily be visualized in Fig. 1).

References

[1] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *J. ACM*, vol. 20, no. 1, pp. 40-61, 1973.

[2] J.P. Lehoczky, L. Sha, and Y. Ding, "The Rate-Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," *Proc. 10th IEEE Real-Time Systems Symp.*, pp. 166-172, 1989.

[3] M. Joseph and P. Pandya, "Finding Response Times in a Real-Time System," *The Computer J., British Computer Soc.*, vol. 29, no. 5, pp. 390-395, Oct. 1986.

[4] N.C. Audsley, A. Burns, K. Tindell, and A. Wellings, "Applying New Scheduling Theory to Static Priority Preemptive Scheduling," *Software Eng. J.*, Dec. 1993.

[5] Y. Manabe and S. Aoyagi, "A Feasibility Decision Algorithm for Rate Monotonic Scheduling of Periodic Real-Time Tasks," *Proc. First Real-Time Technology and Applications Symp.*, pp. 297-303, May 1995.

[6] J.Y.-T. Leung and J. Whitehead, "On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks," *Performance Evaluation*, vol. 2, pp. 237-250, 1982.

[7] L. Sha, R. Rajkumar, and J.P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Trans. Computers*, vol. 39, no. 9, pp. 1175-1185, 1990.

[8] A. Burns, R. Davis, and S. Punnekkat, "Feasibility Analysis of Fault-Tolerant Real-Time Task Sets," *IEEE Proc. Euromicro Workshop Real-Time Systems*, pp. 29-33, June 1996.

[9] M. Sjödin and H. Hansson, "Improved Response-Time Analysis Calculations," *Proc. 19th IEEE Real-Time Systems Symp.,* pp. 399-409, Dec. 1998.
[10] Y. Oh and S.H. Son, "Allocating Fixed-Priority Periodic Tasks on Multiprocessor Systems," *Real-Time Systems,* vol. 9, pp. 207-239, 1995.
[11] J.W.S. Liu, *Real-Time Systems.* Prentice Hall, 2000.
[12] E. Bini, G.C. Buttazzo, and G.M. Buttazzo, "Rate Monotonic Scheduling: The Hyperbolic Bound," *IEEE Trans. Computers,* vol. 52, no. 7, pp. 933-942, July 2003.
[13] T.-W. Kuo and A.K. Mok, "Load Adjustment in Adaptive Real-Time Systems," *Proc. IEEE Real-Time Systems Symp.,* Dec. 1991.
[14] D. Chen, A.K. Mok, and T.-W. Kuo, "Utilization Bound Re-Visited," *Proc. Sixth Real-Time Computing Systems and Applications,* pp. 295-302, 1999.
[15] D. Park, S. Natarajan, and M.J. Kim, "A Generalized Utilization Bound Test for Fixed-Priority Real-Time Scheduling," *Proc. Second Int'l Workshop Real-Time Systems and Applications,* pp. 73-76, Oct. 1995.
[16] H. Liu and X. Hu, "Efficient Performance Estimation for General Real-Time Task Systems," *IEEE/ACM Int'l Conf. Computer Aided Design*, pp. 464-471, 2001.

Enrico Bini received the Laurea degree in computer engineering from the Università di Pisa, in 2000. He is a PhD student in computer engineering at Scuola Superiore Sant'Anna in Pisa (Italy). In the same year, he received the Licenza degree from the Scuola Superiore Sant'Anna. In 1999, he was a visiting student at Technische Universiteit Delft, in the Netherlands. In 2001, he was with Ericsson Lab Italy in Roma. In 2003, he was a visiting student at University of North Carolina at Chapel Hill, collaborating with Professor Sanjoy Baruah. His interests cover scheduling algorithms, real-time operating systems, embedded systems design, and optimization algorithms.

Giorgio C. Buttazzo received a degree in electronic engineering at the University of Pisa in 1985, received the masters degree in computer science at the University of Pennsylvania in 1987, and the PhD degree in computer engineering at the Scuola Superiore S. Anna of Pisa in 1991. He is an associate professor of computer engineering at the University of Pavia, Italy. During 1987, he worked on active perception and real-time control at the G.R.A.S.P. Laboratory of the University of Pennsylvania, in Philadelphia. From 1991 to 1998, he was an assistant professor at the Scuola Superiore S. Anna of Pisa, doing research on robot control systems and real-time scheduling. His main research interests include real-time operating systems, dynamic scheduling algorithms, quality of service control, multimedia systems, advanced robotics applications, and neural networks. He is a member of the IEEE and the IEEE Computer Society.