The Space of Rate Monotonic Schedulability

Enrico Bini
Scuola Superiore S. Anna
Pisa, Italy
e.bini@sssup.it

Giorgio C. Buttazzo
University of Pavia, Italy
INFM Research Unit
buttazzo@unipv.it

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 are used for on-line guarantee of dynamic systems, where tasks can be 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 under the Rate Monotonic priority assignment. Using this approach, we derive a new schedulability test which can be tuned through a parameter to balance complexity vs. 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 the Rate Monotonic analysis.

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 [8], Liu and Layland proved that 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 periodic task set under RM.

Their result refers to the following task model. 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 k^{th} 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 $t = 0$ (that is, $\Phi_i = 0$ for all $i = 1, \ldots, n$), relative deadlines are equal to periods (that is, $d_{ik} = k\,T_i$) and tasks are independent (that is, 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

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

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.

In [7], Lehoczky, Sha, and Ding performed a statistical study and showed that for task sets with randomly generated parameters the LL-test is able to guarantee schedulability up to a processor utilization of about 88%. Exact schedulability tests for RM yielding to necessary and sufficient conditions have been independently derived in [5, 7, 1]. Using the method proposed in [1], 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 worst-case response time R_i of a task can be computed using the following iterative formula:

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

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 pseudo-polynomial, thus it is not suited to be used for on-line admission control, when the number of tasks is large. In [13], Sha, Rajkumar and Lehozcky 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. In [1], Audsley et al. generalized the response time analysis including resource constraints, and in [3], Burns, Davis, and Punnekats extended it to take fault-tolerant constraints into account. In [14], Sjödin and Hansson 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 pseudo-polynomial.

In the last years, other authors [11, 10, 2] 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. [2] improves the acceptance ratio up to a limit of $\sqrt{2}$ for large n, compared with the Liu and Layland one. According to this method, a set of periodic tasks is schedulable by RM if

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

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

In [6, 4] 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. A similar approach was proposed by Park et al. [12] and, in the case of a graph structured task, by Liu and Hu [9].

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 vs. acceptance ratio, so that it can be used on line 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 on-line 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. A qualitative behavior of the proposed test is illustrated in Figure 1.

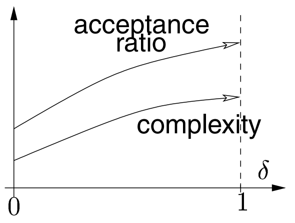


Figure 1. Qualitative properties of the tunable guarantee test.

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 results presented in [6, 12, 4].

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. Finally, Section 7 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 tasks, where each task τ_i is characterized by an initial activation time Φ_i (phase), a worst-case computation time C_i, and a period T_i. Each task τ_i consists of an infinite sequence of jobs, where τ_{ik} denotes the k^{th} 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, 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:

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

For the sake of simplicity, in this paper we assume that relative deadlines are equal to periods ($D_i = T_i$), however the analysis can easily be extended to consider $D_i \le T_i$. Finally, $U_i = C_i/T_i$ denotes the utilization factor of task τ_i.

As proved by Liu and Layland in [8], the worst-case scenario for a periodic task set scheduled by RM occurs when all the tasks are simultaneously activated at the same time, so without loss of generality we assume $\Phi_i = 0$ for all the tasks.

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 RM schedulable tasks sets. In particular, region $\mathbb{M}_n$ is defined as follows:

$$\mathbb{M}_n(T_1,\ldots,T_n) = \{(C_1,\ldots,C_n) \in \mathbf{R}_+^n : \Gamma_n \text{ is schedulable by RM}\}. \tag{4}$$

We note that periods T_i are considered as parameters, whereas C_i are free variables. Hence, we obtain a constraint on the C_i variables, which is a function of the periods T_i. The space where 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, Sha and Ding in [7], through the following theorem:

Theorem 1 (Theorem 2 in [7]) *Given a periodic task set* $\Gamma_n = \{\tau_1,\ldots,\tau_n\}$,

1. τ_i *can be scheduled for all tasks phasings using 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 $\mathcal{S}_i = \{rT_j : j = 1\ldots i, r = 1\ldots \left\lfloor \frac{T_i}{T_j} \right\rfloor\}$.

2. *The entire task set is schedulable for all tasks phasings using RM if and only if:*

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

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 $\wedge$):

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

Table 1. A 3-task set example of $\mathcal{S}_i$ when $T_1 = 3$, $T_2 = 8$ and $T_3 = 20$.

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

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

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

The last result provides a first analytical formulation of the $\mathbb{M}_n$ feasibility region, which is more formally expressed by the following theorem.

Theorem 2 *The region of the schedulable tasks sets $\mathbb{M}_n$, as defined by equation (4), is given by:*

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

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

Proof. It directly follows from Theorem 1 and equation (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:

$$\left\{\begin{array}{l}
C_1 \le 3 \qquad 3 \in \mathcal{S}_1 \\[1ex]
\left\Vert\begin{array}{lll}
C_1 + C_2 \le 3 & 3 \in \mathcal{S}_2 & \\
2C_1 + C_2 \le 6 & 6 \in \mathcal{S}_2 & \text{plane } \alpha \text{ in fig. 2} \\
3C_1 + C_2 \le 8 & 8 \in \mathcal{S}_2 & \text{plane } \beta \text{ in fig. 2}
\end{array}\right. \\[1ex]
\left\Vert\begin{array}{lll}
C_1 + C_2 + C_3 \le 3 & 3 \in \mathcal{S}_3 & \\
2C_1 + C_2 + C_3 \le 6 & 6 \in \mathcal{S}_3 & \\
3C_1 + C_2 + C_3 \le 8 & 8 \in \mathcal{S}_3 & \\
3C_1 + 2C_2 + C_3 \le 9 & 9 \in \mathcal{S}_3 & \\
4C_1 + 2C_2 + C_3 \le 12 & 12 \in \mathcal{S}_3 & \\
5C_1 + 2C_2 + C_3 \le 15 & 15 \in \mathcal{S}_3 & \text{plane } \eta \\
6C_1 + 2C_2 + C_3 \le 16 & 16 \in \mathcal{S}_3 & \text{plane } \theta \\
6C_1 + 3C_2 + C_3 \le 18 & 18 \in \mathcal{S}_3 & \text{plane } \xi \\
7C_1 + 3C_2 + C_3 \le 20 & 20 \in \mathcal{S}_3 & \text{plane } \pi
\end{array}\right.
\end{array}\right. \tag{5}$$

where the $||$ symbol denotes the logical OR among the equations, whereas the $\{$ symbol denotes the logical AND. Figure 2 shows a graphical representation of the $\mathbb{M}_3(3, 8, 20)$ region in the C-space.

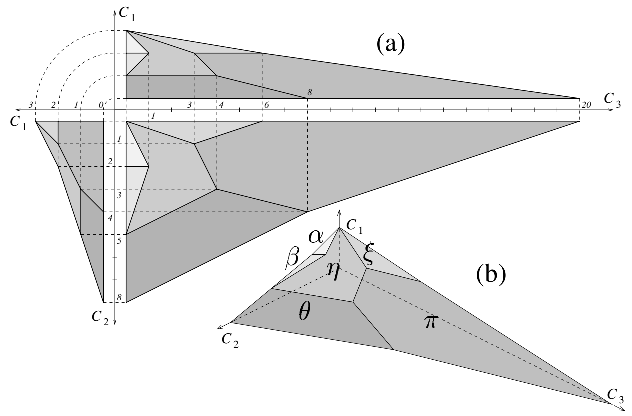


Figure 2. A view of $\mathbb{M}_3(3, 8, 20)$ in the C-space: (a) the projection view, (b) the isometric view.

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: there can be equations in the list which are not shown in the picture because OR-ed with a more relaxed one.

Observation 2 In Figure 2 we can distinguish 6 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 $\Gamma_n = \{(T_1, C_1), \ldots, (T_n, C_n)\}$ into a point in the C-space and then checking whether it belongs to $\mathbb{M}_n(T_1, \ldots, T_n)$.

Observation 4 The idea of building a tunable test works fine with this formulation. In fact, every subregion $\mathbb{B} \subseteq \mathbb{M}_n$, obtained by eliminating some of the equations to be OR-ed, is a subset of $\mathbb{M}_n$, and hence the feasibility check in $\mathbb{B}$ 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 the practical application of Theorem 2 for any guarantee tests.

Observation 6 The mathematical expression 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 $\left\lceil \frac{t}{T_j} \right\rceil \frac{T_j}{t}$ is equal to 1 for all j and t, the $\mathbb{M}_n$ region becomes $\sum C_i/T_i \leq 1$, which is the EDF schedulability condition. This is just the mathematical translation of the commonly known behavior of RM when all the periods are in the same harmonic chain [6].

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. in [12]. 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 $\mathcal{S}_i$.

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

4. The space of RM schedulability

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 equation (4), is given by*

$$\mathbb{M}_n(T_1, \ldots, T_n) = \{(C_1, \ldots, C_n) \in \mathbf{R}_+^n : \bigwedge_{i=1\ldots n} \bigvee_{t \in \mathcal{P}_{i-1}(T_i)} \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t\}$$

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

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

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 $\mathcal{P}_{i-1}(T_i)$, instead of $\mathcal{S}_i$. This may seem a little change, but it is not. For example, Figure 3 shows the difference between the sets $\mathcal{S}_3$ and $\mathcal{P}_2(T_3)$ for the task set reported in Table 1.

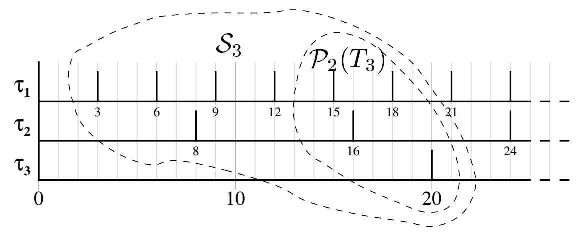
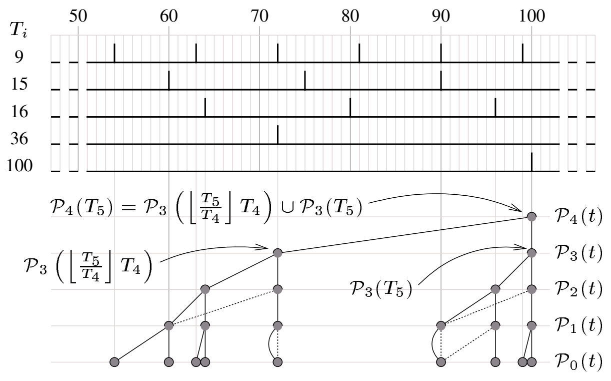


Figure 3. Comparison between $\mathcal{S}_3$ and $\mathcal{P}_2(T_3)$ for the task set shown in Table 1.

With the introduction of the set $\mathcal{P}_{i-1}(T_i)$, 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. As clear from Figure 3, it can easily be shown that $\mathcal{P}_{i-1}(T_i) \subseteq \mathcal{S}_i$. This allows to dramatically reduce and bound the time needed for the 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.



T_i
50
60
70
80
90
100
9
15
16
36
100
$\mathcal{P}_4(T_5) = \mathcal{P}_3\left(\left\lfloor \frac{T_5}{T_4} \right\rfloor T_4\right) \cup \mathcal{P}_3(T_5)$
$\mathcal{P}_3\left(\left\lfloor \frac{T_5}{T_4} \right\rfloor T_4\right)$
$\mathcal{P}_3(T_5)$
$\mathcal{P}_4(t)$
$\mathcal{P}_3(t)$
$\mathcal{P}_2(t)$
$\mathcal{P}_1(t)$
$\mathcal{P}_0(t)$

Figure 4. An example of $\mathcal{P}_i(t)$.

Figure 4 shows all the recurrent calls of $\mathcal{P}_4(T_5)$ in the case of $T_1 = 9, T_2 = 15, T_3 = 16, T_4 = 36$ and $T_5 = 100$. 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:*

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

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 $C_i + W_{i-1}(T_i) \leq T_i$.

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:*

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

By Definition 1, the set $\mathsf{Busy}(\Gamma_i, b)$ is the union of open intervals, hence the set $[0, b] \setminus \mathsf{Busy}(\Gamma_i, b)$ has always a maximum and so the last idle instant $\psi_i(b)$ is well defined[1]. This formalism is needed because the point $\psi_i(b)$ is useful for simplifying the computation of $W_i(b)$ and to express the RM schedulability condition. 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 $\Gamma_i = \{\tau_1, \ldots, \tau_i\}$,*

$$W_i(b) = \min_{t \in [0,b]} \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j + (b - t). \qquad (7)$$

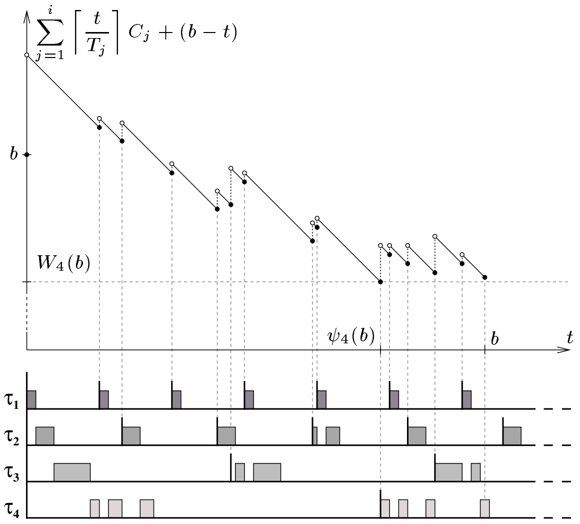


Figure 5. Lemma 2 interpretation.

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

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

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

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

or, equivalently,

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

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 Figure 5, which shows the upper bounding function[2] 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 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 RM, let $\psi_i(b)$ be the last idle instant in $[0, b]$ as defined in def. 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 proved.

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 $\lfloor b/T_{i+1} \rfloor^{\text{th}}$ 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:

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

[1]The symbol \ denotes the set difference operator.
[2]Black dots indicate the value of the function in the discontinuities.

For the inductive hypothesis, we have that: $\psi_i\left(\lfloor b/T_{i+1}\rfloor T_{i+1}\right) \in \mathcal{P}_i\left(\lfloor b/T_{i+1}\rfloor T_{i+1}\right)$ 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 RM and the set $\mathcal{P}_i(b)$ as defined in equation (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 Lemmæ 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 the Rate Monotonic algorithm;
2. $\displaystyle\bigwedge_{i=1\ldots n} \bigvee_{t \in \mathcal{P}_{i-1}(T_i)} \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j \le t.$

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:

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

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

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

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

$$\bigvee_{t \in \mathcal{P}_{i-1}(T_i)} \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j \le t$$

because $\lceil t/T_i \rceil = 1$ for all $t \in \mathcal{P}_{i-1}(T_i)$.

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

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

as required by theorem 3. □

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 RM schedulability condition in Theorem 3 can be equivalently expressed as

$$\forall i = 1 \ldots n \quad C_i + W_{i-1}(T_i) \le T_i$$

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

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

and $W_0(T_1) = 0$ by extension.

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

```
boolean RMTest($\Gamma_n$) {
  int $i$;

  for ($i$=1;$i \le n$;$i$++)
    if ($C_i$ + WorkLoad($i-1$,$T_i$)> $T_i$)
      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 $c = \lceil b/T_i \rceil$):

$$W_i(b) = \min\left\{ \min_{t \in \mathcal{P}_{i-1}(fT_i)} \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j + (b - t), \; \min_{t \in \mathcal{P}_{i-1}(b)} \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j + (b - t), \right\} \qquad (9)$$

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 equation (9) can be written as

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

from the result of Lemma 4. Similarly, the second element in equation (9) can be written as:

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

Hence, equation (9) can be expressed in a recurrent form as follows:

$$W_i(b) = \min\{b - f(T_i - C_i) + W_{i-1}(fT_i), c\, C_i + W_{i-1}(b)\}. \tag{10}$$

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 Figure 4), 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 last$\psi$[BIG_ENOUGH];
double lastWorkLoad[BIG_ENOUGH];

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

  if ($i \le 0$ ) return 0 ;
  if ($b \le$ last$\psi$ [$i$ ]) /* if WorkLoad($i, b$) already computed */
    return lastWorkLoad[$i$ ]; /* don't go further */
  $f = \lfloor b/T_i \rfloor$ ; $c = \lceil b/T_i \rceil$ ;
  branch0 = $b - f\,(T_i - C_i)$ + WorkLoad ($i-1$ , $f\,T_{i-1}$ );
  branch1 = $c\,C_{n-1}$ + WorkLoad ($i-1$ , $b$ );
  last$\psi$[$i$ ] = $b$ ;
  lastWorkLoad[$i$ ] = min(branch0, branch1 );
  return lastWorkLoad [$i$ ];
}
```

We note that the i^{th} 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 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:

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

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

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

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

$$\delta_1 \le \delta_2 \iff W_i^{(\delta_1)}(b) \ge W_i^{(\delta_2)}(b)$$
$$W_i^{(1)}(b) = W_i(b).$$

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 last$\psi$[BIG_ENOUGH];
double lastWorkLoad[BIG_ENOUGH];

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

  if ($i \le 0$ )
    return 0 ;
  if ($b \le$ last$\psi$ [$i$ ]) /* if WorkLoad($i, b$) already computed */
    return lastWorkLoad [$i$ ]; /* don't go further */
  $f = \lfloor b/T_i \rfloor$ ; $c = \lceil b/T_i \rceil$ ;
  branch0 = $b - f\,(T_i - C_i)$ + WorkLoad ($i-1$ , $f\,T_{i-1}$ );
  if ($T_i \le b\,\delta$ )
    branch1 = $c\,C_{n-1}$ + WorkLoad ($i-1$ , $b$ );
  else
    branch1 = branch0 ; /* one branch is cut ! */
  last$\psi$[$i$ ] = $b$ ;
  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. 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 testing universe of the task sets for which there exists a feasible schedule, if $S^* \subseteq U$ is the subset of all the RM schedulable task sets, and $S^T \subseteq S^*$ is the subset of task sets schedulable by a generic sufficient test T, then the acceptance ratio $\mathsf{acceptance}(T)$ of a test T is computed as follows:

$$\mathsf{acceptance}(T) = \frac{\#S^T}{\#S^*}$$

where $\#$ indicates "cardinality of". From this definition it follows that $\mathsf{acceptance}(T) \leq 1$ for all T, and $\mathsf{acceptance}(T) = 1$ for the RTA 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 $\mathit{steps}(T, \Gamma)$ 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 $\mathsf{steps}(T)$, defined by the following cumulative distribution function:

$$F_{\mathsf{steps}(T)}(x) = P\{\mathit{steps}(T, \Gamma) \leq x\} \qquad \Gamma \in U.$$

This model is useful because the maximum and the average number of iteration steps can easily be extracted from the probability density, which is

$$f_{\mathsf{steps}(T)} = \frac{\mathrm{d}F_{\mathsf{steps}(T)}}{\mathrm{d}x}.$$

In our experiments, simulations have been performed by generating 10^8 task sets, each composed by 8 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). Figure 6 illustrates the results of a first experiment, which compares three necessary and sufficient tests: the classical Response Time Analysis (RTA) of Audsley et al. [1], the Response Time analysis Improved (RTI) by Sjödin et al. [14], and the Hyperplanes Exact Test (HET) presented in this paper. The figure plots the probability density achieved for the three tests.

We notice that the HET test 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.

Figure 7 shows the result of another experiment, in which we evaluated the dependency of the average number of steps from 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) [1], the Response Time analysis Improved (RTI) [14], the Liu and Layland test (LL) [8], and the Hyperbolic Bound (HB) [2]. 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 8 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 $\mathsf{steps}(T)$ and $\mathsf{acceptance}(T)$ 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 Figure 8.

As we can see, the performance of the δ-HET test cov-

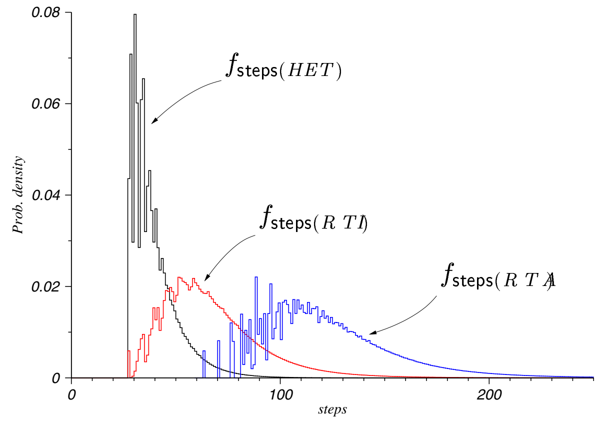


Figure 6. Probability density of $\mathsf{steps}(RTA)$, $\mathsf{steps}(RTI)$ and $\mathsf{steps}(HET)$.

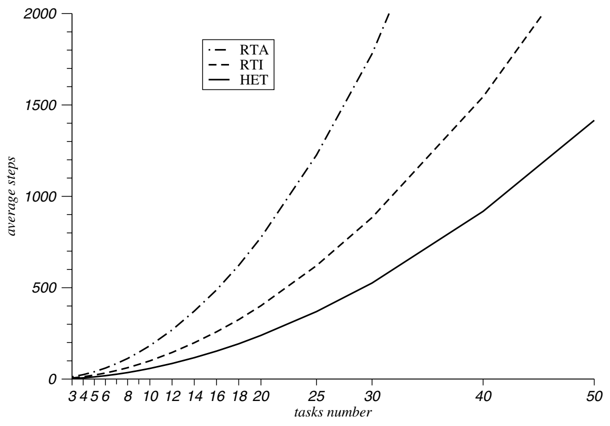


Figure 7. Average number of steps as a function of the number of tasks.

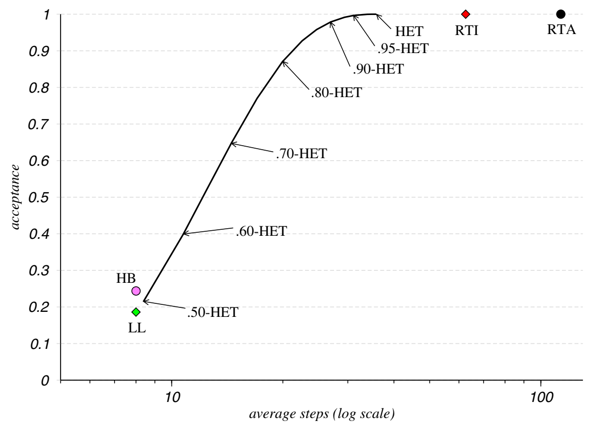


Figure 8. Comparison of guarantee tests.

ers 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. 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 on-line 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 Figure 2).

References

[1] N. C. Audsley, A. Burns, K. Tindell, and A. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, December 1993.

[2] E. Bini, G. C. Buttazzo, and G. Buttazzo. A hyperbolic bound for the rate monotonic algorithm. In *IEEE Proc. of the* 13th *Euromicro Conf. on Real-Time Systems*, June 2001.

[3] A. Burns, R. Davis, and S. Punnekkat. Feasibility analysis of fault-tolerant real-time task sets. In *IEEE Proceedings of the Euromicro Workshop on Real-Time Systems*, pages 29–33, June 1996.

[4] D. Chen, A. K. Mok, and T.-W. Kuo. Utilization bound revisited. In *Proceedings of the* 6th *Real-Time Computing Systems and Applications*, pages 295–302, 1999.

[5] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

[6] T.-W. Kuo and A. K. Mok. Load adjustment in adaptive real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, December 1991.

[7] J. P. Lehoczky, L. Sha, and Y. Ding. The rate-monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 166–172, 1989.

[8] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):40–61, 1973.

[9] H. Liu and X. Hu. Efficient performance estimation for general real-time task systems. In *IEEE/ACM International Conference on Computer Aided Design*, pages 464–471, 2001.

[10] J. W. S. Liu. *Real-Time Systems*. Prentice Hall, 2000.

[11] Y. Oh and S. H. Son. Allocating fixed-priority periodic tasks on multiprocessor systems. *Real-Time Systems*, 9:207–239, 1995.

[12] D. Park, S. Natarajan, and M. J. Kim. A generalized utilization bound test for fixed-priority real-time scheduling. In *Proceedings of the* 2nd *International Workshop on Real-Time Systems and Applications*, pages 73–76, October 1995.

[13] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, 1990.

[14] M. Sjödin and H. Hansson. Improved response-time analysis calculations. In *Proceedings of the* 19th *IEEE Real-Time Systems Symposium*, pages 399–409, December 1998.