Real-Time Systems, 30, 129–154, 2005

Measuring the Performance of Schedulability Tests*

ENRICO BINI e.bini@sssup.it
Scuola Superiore S. Anna, Pisa, Italy

GIORGIO C. BUTTAZZO buttazzo@unipv.it
University of Pavia, Pavia, Italy

Abstract. The high computational complexity required for performing an exact schedulability analysis of fixed priority systems has led the research community to investigate new feasibility tests which are less complex than exact tests, but still provide a reasonable performance in terms of acceptance ratio. The performance of a test is typically evaluated by generating a huge number of synthetic task sets and then computing the fraction of those that pass the test with respect to the total number of feasible ones. The resulting ratio, however, depends on the metrics used for evaluating the performance and on the method for generating random task parameters. In particular, an important factor that affects the overall result of the simulation is the probability density function of the random variables used to generate the task set parameters.

In this paper we discuss and compare three different metrics that can be used for evaluating the performance of schedulability tests. Then, we investigate how the random generation procedure can bias the simulation results of some specific scheduling algorithm. Finally, we present an efficient method for generating task sets with uniform distribution in a given space, and show how some intuitive solutions typically used for task set generation can bias the simulation results.

Keywords: RM: Rate Monotonic; EDF: Earliest Deadline First; FP: Fixed Priorities; HB: Hyperbolic Bound

1. Introduction

Fixed priority scheduling is the most used approach for implementing real-time applications. When the system consists mostly of periodic activities, tasks are usually scheduled by the Rate Monotonic (RM) algorithm, which assigns priorities proportionally to the task activation rates (Liu and Layland, 1973). Although other scheduling algorithms, like Earliest Deadline First (EDF) (Liu and Layland, 1973) and Least Laxity First (LLF) (Mok, 1983), have been proved to be more effective than RM in exploiting the available computational resources, RM is still the most used algorithm in industrial systems, mainly for its efficient implementation, simplicity, and intuitive meaning. In particular, the superior behavior of EDF over RM has been shown under different circumstances (Buttazzo, 2003), including overload conditions, number of preemptions, jitter, and aperiodic service. Nevertheless, the simpler implementation of RM on current commercial operating systems, along with a number of misconceptions on the two algorithms, seem to be the major causes that prevent the use of dynamic priority schemes in practical applications.

*This work has been partially supported by the European Union, under contract IST-004527, and by the Italian Ministry of University Research (MIUR), under contract 2003094275.

The exact schedulability analysis of fixed priority systems, however, requires high computational complexity, even in the simple case in which task relative deadlines are equal to periods (Joseph and Pandya, 1986; Lehoczky et al., 1995; Audsley et al., 1993). Methods for speeding up the analysis of task sets have been recently proposed (Manabe and Aoyagi, 1995; Sjödin and Hansson, 1998; Bini and Buttazzo, 2004b), but the complexity of the approach remains pseudo-polynomial in the worst case. This problem becomes especially relevant for those real-time applications that require an on-line admission control and run on small microprocessors with low processing power.

This fact has led the research community to investigate new feasibility tests that are less complex than exact tests, but still provide a reasonable performance in terms of acceptance ratio. For example, Bini and Buttazzo proposed a sufficient guarantee test, the Hyperbolic Bound (Bini et al., 2003), that has the same $O(n)$ complexity of the classical utilization bound proposed by Liu and Layland (1973), but better performance. To increase schedulability, Han and Tyan (1997) suggested to modify the task set with smaller, but harmonic, periods using an algorithm of a $O(n^2 \log n)$ complexity. Other sufficient feasibility tests have been proposed in Burchard et al. (1995) and Lauzac et al. (2003), where additional information on period relations is used to improve schedulability within a polynomial time complexity.

A tunable test, called the Hyperplane δ-Exact Test, has recently been proposed (Bini and Buttazzo, 2004b) to balance performance versus complexity using a single parameter. If $\delta = 1$, the test is exact and hence it has a pseudo-polynomial complexity; if $\delta < 1$ the test becomes only sufficient and its complexity and performance decrease with δ.

When dealing with approximate tests, the problem of evaluating their performance with respect to the exact case becomes an important issue. The effectiveness of a guarantee test for a real-time scheduling algorithm is measured by computing the number of accepted task sets with respect to the total number of feasible ones. Such a ratio can be referred to as the *acceptance ratio*. The higher the ratio, the better the test. When the ratio is equal to one, then the guarantee test results to be necessary and sufficient for the schedulability of the task set.

Computing the acceptance ratio using a rigorous approach is not always possible, except for very simple cases. In all the other cases, the performance of a guarantee test has to be evaluated through extensive simulations, where a huge number of synthetic task sets need to be generated using random parameters. Using this approach, both the evaluation metrics and the way task parameters are generated significantly affect the overall performance of the test. This potential biasing factor may lead to wrong design choices if the hypotheses used in the simulation phase differ from the actual working conditions. Unfortunately, to our best knowledge, no much work has been done in the literature to help understanding these problematic issues.

First, Lehoczky et al. (1989) introduced the breakdown utilization as a means for measuring the performance of the Rate Monotonic scheduling algorithm. Then, other authors (Park et al., 1995; Chen et al., 2003; Lee et al., 2004) provided methods for computing the utilization upper bound, which can indirectly measure the effectiveness of RM. In a previous work, Bini and Buttazzo (2004a) showed the limitations of the former approaches, also proposing a standard metric for evaluating the performance of schedulability tests.

In this paper we introduce a more general framework where the three approaches are discussed and compared through theoretical results and experimental evidence. The dependency between simulation results and task generation routines is discussed, showing how some intuitive solutions used for generating synthetic task sets can bias the results of the simulation. A key contribution of this work is an efficient method for generating task sets with uniform distribution in the space of utilizations.

The rest of the paper is organized as follows. Section 2 introduces the metrics commonly used in the literature for evaluating the performance of feasibility tests, and presents a new evaluation criterion that better describes the properties of a test. Section 3 presents an efficient method for generating random task set parameters which does not bias the simulation results. Section 4 compares the introduced metrics and illustrates some experimental results carried out by simulation. Finally, Section 5 states our conclusions.

1.1. Notation and Assumptions

The notation used throughout the paper is reported in Table 1. To make a consistent comparison among the feasibility tests discussed in this work, the basic assumptions made on the task set are the same as those typically considered in the real-time literature where the original tests have been proposed. That is, each periodic task τ_i consists of an infinite sequence of jobs $\tau_{i,k}$ ($k = 1, 2, \ldots$), where the first job $\tau_{i,1}$ is released at time $r_{i,1} = \Phi_i$ (the task phase) and the generic kth job $\tau_{i,k}$ is released at time $r_{i,k} = \Phi_i + (k-1)T_i$. The worst-case scenario occurs for simultaneous task activations (i.e., $\Phi_i = 0, i = 1, \ldots, n$). Relative deadlines are smaller than or equal to periods ($D_i \leq T_i$). Tasks are independent (that is, they do not have resource constraints, nor precedence relations) and are fully preemptive. Context switch time is neglected. Notice that, although such a simple task model can be fruitfully enriched to capture more realistic situations, the objective of this paper in not to derive a new feasibility test, but to clarify some issues related to the performance of existing tests, most of which have been derived under the simplifying assumptions reported above.

For this reason these simplistic assumptions do not restrict the applicability of the results, but provide a more general framework which allows comparing different results found for different application models.

Table 1. Notation.

Notation	Meaning
$\tau_i = (C_i, T_i, D_i)$	The ith periodic task
C_i	Computation time of task τ_i
T_i	Period of task τ_i
D_i	Relative deadline of task τ_i
$U_i = C_i/T_i$	Utilization of task τ_i
$\Gamma = \{\tau_1, \ldots, \tau_n\}$	A set of n periodic tasks
$U = \sum_1^n U_i$	Utilization of the task set
$\mathbb{A}, \mathbb{B}, \mathbb{X} = \{\Gamma_1, \ldots, \Gamma_p\}$	A group of task sets (domain)

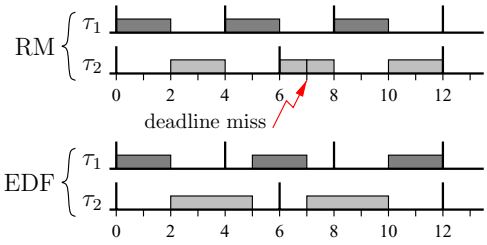

Figure 1. Task set schedulable by EDF, but not by RM.

2. Metrics

It is well known that the RM scheduling algorithm cannot schedule all the feasible task sets. For example, the task set shown in Figure 1 has a total utilization of 1, thus it is schedulable by EDF, but not by RM.

The problem of measuring the difference, in terms of schedulability, between these two algorithms has often attracted the interest of the real-time research community. Hence, a number of methods have been proposed to evaluate the performance of a feasibility test for RM. The most common approaches available in the real-time literature are based on the definition of the following concepts:

– Breakdown utilization (Lehoczky et al., 1989);

– Utilization upper bound (Park et al., 1995; Chen et al., 2003; Lee et al., 2004).

Unfortunately, both techniques present some drawback that will be analyzed in detail in Sections 2.1 and 2.2. To overcome these problems, a new performance evaluation criterion, the *Optimality Degree* (**OD**), will be introduced in Section 2.3.

2.1. Breakdown Utilization

The concept of breakdown utilization was first introduced by Lehoczky et al. (1989) in their seminal work aimed at providing exact characterization and average case behavior of the RM scheduling algorithm.

Definition 1 (Section 3 in Lehoczky et al., 1989). A task set is generated randomly, and the computation times are scaled to the point at which a deadline is first missed. The corresponding task set utilization is the breakdown utilization U_n^*.

In such a scaling operation, computation times are increased if the randomly generated task set is schedulable, whereas they are decreased if the tasks set is not schedulable. This scaling stops at the boundary which separates the RM-schedulable and RM-unschedulable

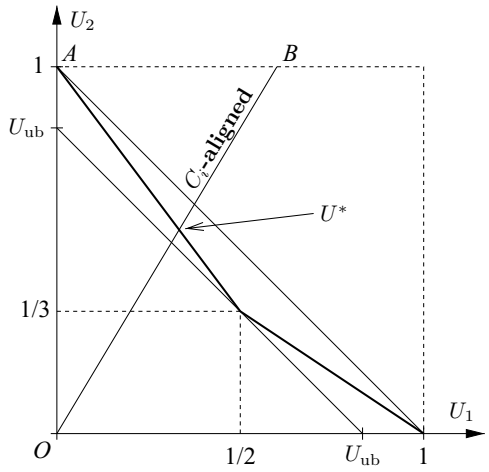

Figure 2. Interpretation of the breakdown utilization.

task sets. Hence, all task sets with the same period configuration that can be obtained by scaling the computation times have the same breakdown utilization, because they all hit the boundary in the same point. We denote these task sets as *a group of* C_i***-aligned*** *task sets*. This concept is illustrated in Figure 2.

For example, if we assume the computation times to be uniformly distributed in $[0, T_i]$ (i.e., the U_i uniform in [0, 1]), the number of task sets belonging to the same C_i**-aligned** group is proportional to the length of the $\overline{OB}$ segment in Figure 2.

Note that a group of C_i**-aligned** task sets contains more sets when the U_i are similar to each other, because in this case the segment $\overline{OB}$ is longer. The utilization disparity in a set of tasks can be expressed through the following parameter, called the U*-difference*:

$$\delta = \frac{\max_i\{U_i\} - \min_i\{U_i\}}{\sum_{i=1}^{n} U_i}. \tag{1}$$

If $\delta = 0$, all the task utilization factors are the same, whereas $\delta = 1$ denotes the maximum degree of difference.

We first notice that all the task sets belonging to the same C_i**-aligned** group have the same value of δ, since positive scaling factors may be brought outside the min, max and sum operators in Equation (1), and then simplified. We then show some relationship between a C_i**-aligned** group and its value of U*-difference* δ.

As it can be argued from Figure 2, when a group of C_i**-aligned** task sets is characterized by a small value of δ, the segment $\overline{OB}$ is longer (that is, the number of task sets in the hypercube $(U_1, \ldots, U_n) \in [0, 1]^n$ belonging to the same group is larger).

In the case of two tasks, the relationship between δ and $|\overline{OB}|$ can be explicitly derived. Without any loss of generality, we assume $U_1 \leq U_2$. Since δ does not vary within the same C_i**-aligned** group, the relation can be computed by fixing any point on the $\overline{OB}$ segment.

Thus, to simplify the computation, we select the set with $U_2 = 1$. For this group we have $U_1 = |\overline{AB}|$ (please refer to Figure 2 to visualize the segment $\overline{AB}$). Hence:

$$\delta = \frac{\max_i\{U_i\} - \min_i\{U_i\}}{\sum_{i=1}^{n} U_i} = \frac{U_2 - U_1}{U_2 + U_1} = \frac{1 - |\overline{AB}|}{1 + |\overline{AB}|}$$

$$|\overline{AB}| = \frac{1-\delta}{1+\delta}$$

then:

$$|\overline{OB}| = \sqrt{1 + |\overline{AB}|^2} = \frac{\sqrt{2(1+\delta^2)}}{1+\delta}. \tag{2}$$

Note that if $\delta = 1$, then $|\overline{OB}| = 1$, whereas if $\delta = 0$, then $|\overline{OB}| = \sqrt{2}$.

In the general case of n tasks, the relationship between δ and $|\overline{OB}|$ cannot be explicitly found. However, we can provide an intuition in order to understand it. The longest $\overline{OB}$ segment is equal to the diagonal of a unitary n-dimensional cube, whose length is $\sqrt{\sum_1^n 1^2} = \sqrt{\sum_1^n 1} = \sqrt{n}$, which grows with n. The result illustrated above can be summarized in the following observation.

Remark 1. When uniformly generating the utilizations $(U_1, \ldots, U_n)$ in $[0, 1]^n$, a group of C_i**-aligned** task sets contains more sets if it has a smaller value of *U-difference* δ. This phenomenon becomes more considerable (by a factor $\sqrt{n}$) as the number of tasks grows.

These geometrical considerations are very relevant when evaluating the performance of the scheduling algorithm. In fact, among all task sets with the same total utilization, RM is more effective on those sets characterized by tasks with very different utilization (i.e., $\delta \to 1$). The intuition behind this statement is also supported by the following facts:

– Liu and Layland proved in 1973 that the worst-case configuration of the utilization factors occurs when they are all equal to each other;

– using the Hyperbolic Bound (Bini et al., 2003), which guarantees feasibility if $\prod_{i=1}^{n}(1 + U_i) \leq 2$, it has been shown that the schedulability under RM is enhanced when tasks have very different utilizations U_i.

Hence, we can state that:

Remark 2 . For a given total utilization factor $\overline{U}$, the feasibility of a task set under the RM scheduling algorithm improves when task utilizations have a large difference, that is when δ is close to 1.

To further support this remark we have run a simulation experiment in which we generated 10^6 task sets consisting of $n = 8$ tasks, whose periods are uniformly distributed in

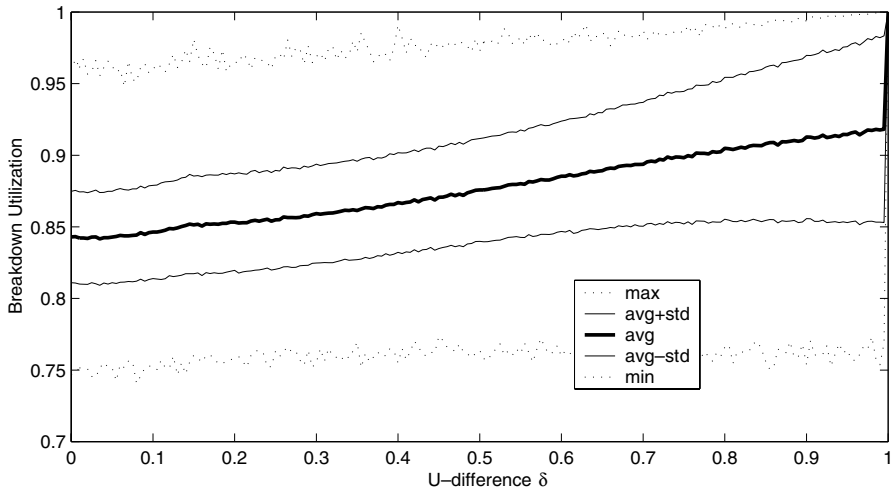

Figure 3. The breakdown utilization as a function of δ.

[1, 1000]. Computation times are generated so that the set has a specific value of δ. We then considered the breakdown utilization $U_n^*(\delta)$ as a random variable. For every value of δ, Figure 3 illustrates the results of the simulation, reporting (from top to bottom) $\max\{U_n^*(\delta)\}$, $E[U_n^*(\delta)] + \sigma_{U_n^*(\delta)}$, $E[U_n^*(\delta)]$, $E[U_n^*(\delta)] - \sigma_{U_n^*(\delta)}$ and $\min\{U_n^*(\delta)\}$.[1] Notice that all values of $\min\{U_n^*(\delta)\}$ are always above the Liu and Layland utilization bound, which, for $n = 8$, is 0.7241. As predicted by Remark 2, the breakdown utilization increases as δ grows.

Summarizing the observations discussed above, we can conclude that when task sets are randomly generated so that task utilizations have a uniform distribution in the hypercube $(U_1, \ldots, U_n) \in [0, 1]^n$, the number of sets with a low value of δ will be dominant. Hence the average breakdown utilization will be biased by those task sets with a small *U-difference* δ, for which RM is more critical. Moreover such a biasing increases as n grows (see Remark 1). Hence, we can state the following remark.

Remark 3. The use of breakdown utilization as a metric for evaluating the performance of the schedulability algorithms, penalizes RM more than EDF.

The formal proof of the last observation is out of the scope of this paper. However, the intuitive justification we provided so far is confirmed in Section 4 by additional experimental results (see Figure 13).

In order to characterize the average case behavior of the RM scheduling algorithm, Lehoczky et al. treated task periods and execution times as random variables, and then studied the asymptotic behavior of the breakdown utilization. Their result is expressed by the following theorem.

Theorem 1 (from Section 4 in Lehoczky et al., 1989)**.** *Given task periods uniformly generated in the interval* $[1, B]$, $B \geq 1$, *then for* $n \to \infty$ *the breakdown utilization* U_n^*

converges to the following values:

– *if* $B = 1$:

$$U_n^* = 1 \tag{3}$$

– *if* $1 < B \leq 2$*:*

$$U_n^* \to \frac{\log_e B}{B-1} \tag{4}$$

– *if* $B \geq 2$:

$$U_n^* \to \frac{\log_e B}{\frac{B}{\lfloor B \rfloor} + \sum_{i=2}^{\lfloor B \rfloor - 1} \frac{1}{i}} \tag{5}$$

and the rate of convergence is $O(\sqrt{n})$.

Then, the authors derive a probabilistic schedulability bound for characterizing the average case behavior of RM.

Remark 4 (Example 2 in Lehoczky et al., 1989). For randomly generated task sets consisting of a large number of tasks (i.e. $n \to \infty$) whose periods are drawn in a uniform distribution (i.e. hyp. of Th. 1 hold) with largest period ratio ranging from 50 to 100 (i.e. $50 \leq B \leq 100$), 88% is a good approximation to threshold of the schedulability (i.e. the breakdown utilization) for the rate monotonic algorithm.

The major problem in using this results is that it is based on specific hypotheses about periods and computation times distributions. Such hypotheses are needed to cut some degrees of freedom on task set parameters and provide a "*single-value*" result, but they may not hold for a specific real-time application.

In the next sections, we will overcome this limitation by making the metrics dependent on a specific task set domain.

2.2. Utilization Upper Bound

The utilization upper bound U_{ub} is a well known parameter, used by many authors (Liu and Layland, 1973; Park et al., 1995; Chen et al., 2003; Lee et al., 2004) to provide a schedulability test for the RM algorithm. Following the approach proposed by Chen et al. (2003), the utilization upper bound is defined as a function of a given domain $\mathbb{D}$ of task sets.

Definition 2. The utilization upper bound $U_{\text{ub}}(\mathbb{D})$ is the maximum utilization such that if a task set is in the domain $\mathbb{D}$ and satisfies $\sum U_i \leq U_{\text{ub}}$ then the task set is schedulable. More formally:

$$U_{\text{ub}}(\mathbb{D}) = \max \left\{ U_{\text{b}} : \left(\Gamma \in \mathbb{D} \ \wedge \ \sum_{\tau_i \in \Gamma} U_i \leq U_{\text{b}} \right) \Rightarrow \Gamma \text{ is schedulable} \right\}. \tag{6}$$

From the definition above it follows that:

$$\mathbb{D}_1 \subseteq \mathbb{D}_2 \Rightarrow U_{\text{ub}}(\mathbb{D}_1) \geq U_{\text{ub}}(\mathbb{D}_2). \tag{7}$$

Using this definition, the previous Liu and Layland utilization bound can be expressed as a particular case of U_{ub}. Let $\mathbb{T}_n$ be the domain of all sets of n tasks. Then we can write that:

$$U_{\text{ub}}(\mathbb{T}_n) = n(\sqrt[n]{2} - 1) \tag{8}$$

which is also referred to as *utilization least upper bound*, because $\mathbb{T}_n$ is the largest possible domain, and hence $U_{\text{ub}}(\mathbb{T}_n) = U_{\text{lub}}$ is the lowest possible utilization upper bound (see Eq. (7)).

The typical choice for the domain of the task sets is given by fixing task periods and deadlines. So the domain $\mathbb{M}_n(T_1, \ldots, T_n, D_1, \ldots, D_n)$, also simply denoted by $\mathbb{M}_n$, is the domain consisting of all task sets having periods $T_1, \ldots, T_n$ and deadlines $D_1, \ldots, D_n$ (so only the computation times $C_1, \ldots, C_n$ are varying).

For such a domain, the following result has been found (Chen et al., 2003) when $D_i = T_i$ and $\frac{T_n}{T_1} \leq 2$:

$$U_{\text{ub}}\left(\mathbb{M}_n \cap \frac{T_n}{T_1} \leq 2 \cap D_i = T_i\right) = 2\prod_{i=1}^{n-1}\frac{1}{r_i} + \sum_{i=1}^{n-1} r_i - n \tag{9}$$

where $r_i = \frac{T_{i+1}}{T_i}$ for $i = 1, \ldots, n-1$.

When $n = 2$ and $r = \frac{T_2}{T_1}$, it was proved (Chen et al., 2003) that:

$$U_{\text{ub}}(\mathbb{M}_2 \cap D_i = T_i) = 1 - \frac{(r - \lfloor r \rfloor)(\lceil r \rceil - r)}{r}. \tag{10}$$

When no restrictions apply on periods and deadlines, the utilization upper bound $U_{\text{ub}}(\mathbb{M}_n)$, shortly denoted by U_{ub} from now and on, can be found by solving n linear programming problems (Park et al., 1995; Lee et al., 2004). The ith optimization problem (for i from 1 to n) comes from the schedulability constraint of the task τ_i. It is formulated as follows:

$$\begin{aligned} &\min_{(C_1,\ldots,C_i)} \ \sum_{j=1}^{i} \frac{C_j}{T_j} \\ &\text{subject to} \ \begin{cases} C_i + \displaystyle\sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t & \forall t \in \mathcal{P}_{i-1}(D_i) \\ C_j \geq 0 & \forall j = 1, \ldots, i \end{cases} \end{aligned} \tag{11}$$

where the reduced set of schedulability points $\mathcal{P}_{i-1}(D_i)$, found in Manabe and Aoyagi (1995) and Bini and Buttazzo (2004b), is used instead of the largest one firstly introduced in Lehoczky et al. (1989). If we label the solution of the ith problem (11) as $U_{\text{ub}}^{(i)}$, the

utilization upper bound is given by:

$$U_{\text{ub}} = \min_{i=1,\ldots,n} U_{\text{ub}}^{(i)} \tag{12}$$

Once the utilization upper bound U_{ub} is found, the guarantee test for RM can simply be performed as follows:

$$\sum_{i=1}^{n} U_i \leq U_{\text{ub}}. \tag{13}$$

The utilization bound defined above is tight, meaning that for every value $U > U_{\text{ub}}$ there exists a task set, having utilization U, which is not schedulable by RM. However, a common misconception is to believe that the test provided by Eq. (13) is a necessary and sufficient condition for the RM schedulability, meaning that "*all the task sets having total utilization $U > U_{\text{ub}}$ are not schedulable*". Unfortunately, this is not true and the RM schedulability is far more complex.

In fact, the following lemma holds:

Lemma 1. *Given a utilization upper bound $U_{\text{ub}} < 1$, there always exists a task set having total utilization $U > U_{\text{ub}}$ which is schedulable by RM.*

For the sake of simplicity, the proof will be shown for $D_i = T_i$, but the lemma holds when $0 < D_i < T_i$ as well.

Proof: Trivially, every task set having $U_n = 1$ and $U_1 = \cdots = U_{n-1} = 0$ proves the lemma, because it is indeed RM-schedulable and its utilization is 1. However, one might object that, since $n - 1$ utilizations are equal to 0, this set is equivalent to a task set of a single task, for which $U_{\text{ub}} = 1$, so invalidating the hypothesis for applying the lemma. Seeking an example with a non-trivial task set requires a little extra effort. Lehoczky et al. (1989) showed that a task set is schedulable if:

$$\forall i = 1, \ldots, n \quad U_i + \sum_{j=1}^{i-1} \left\lceil \frac{T_i}{T_j} \right\rceil \frac{T_j}{T_i} U_j \leq 1. \tag{14}$$

Let us consider a task set having $U_1 = U_2 = \cdots = U_{n-1} = \epsilon$ and consequently $U_n = U - (n-1)\epsilon$. By imposing the sufficient condition (14) for the first $n - 1$ tasks we obtain:

$$\forall i = 1, \ldots, n-1 \quad \epsilon + \sum_{j=1}^{i-1} \left\lceil \frac{T_i}{T_j} \right\rceil \frac{T_j}{T_i} \epsilon \leq 1$$

$$\forall i = 1, \ldots, n-1 \quad \epsilon \leq \frac{1}{1 + \sum_{j=1}^{i-1} \left\lceil \frac{T_i}{T_j} \right\rceil \frac{T_j}{T_i}}$$

and then:

$$\epsilon \leq \min_{i=1,\ldots,n-1} \frac{1}{1 + \sum_{j=1}^{i-1} \left\lceil \frac{T_i}{T_j} \right\rceil \frac{T_j}{T_i}}. \tag{15}$$

So every choice of ϵ satisfying Eq. (15) is capable of scheduling the first $n-1$ tasks. To schedule the τ_n as well, we need that:

$$U - (n-1)\epsilon + \sum_{j=1}^{n} \left\lceil \frac{T_i}{T_j} \right\rceil \frac{T_j}{T_i} \epsilon \leq 1$$

then:

$$\epsilon \leq \frac{1 - U}{\sum_{j=1}^{n} \left\lceil \frac{T_i}{T_j} \right\rceil \frac{T_j}{T_i} - (n-1)}. \tag{16}$$

So, to have both Eqs. (15) and (16) true, we select ϵ such that:

$$0 < \epsilon \leq \min \left\{ \min_{i=1,\ldots,n-1} \frac{1}{1 + \sum_{j=1}^{i-1} \left\lceil \frac{T_i}{T_j} \right\rceil \frac{T_j}{T_i}}, \frac{1 - U}{\sum_{j=1}^{n-1} \left\lceil \frac{T_i}{T_j} \right\rceil \frac{T_j}{T_i} - (n-1)} \right\}. \tag{17}$$

This choice is always possible because the right side of Eq. (17) is strictly positive, due to the fact that $U < 1$. Hence the lemma follows. □

The previous lemma showes that the condition (13) is not necessary.

Nevertheless, U_{ub} is very helpful to understand the goodness of a given period configuration, because it tends to the EDF utilization bound (which is 1) as RM tends to schedule all the feasible task sets.

In the next section we introduce a new metric that keeps the good properties of the utilization upper bound U_{ub}, still able to provide a measure of all the schedulable task sets.

2.3. OD: Optimality Degree

As stated before, the reason for proposing a new metric is to provide a measure of the real number of task sets which are schedulable by a given scheduling algorithm. As for the definition of the utilization upper bound, the concept of Optimality Degree (**OD**) is defined as a function of a given domain $\mathbb{D}$. This a very important property because, if some task set specification is known from the design phase (e.g., some periods are fixed, or constrained in an interval), it is possible to evaluate the performance of the test on that specific class of task sets, by expressing the known information by means of the domain $\mathbb{D}$.

Definition 3. The Optimality Degree $\mathbf{OD}_{\mathsf{A}}(\mathbb{D})$ of an algorithm A on the domain of task sets $\mathbb{D}$ is:

$$\mathbf{OD}_{\mathsf{A}}(\mathbb{D}) = \frac{|\mathsf{sched}_{\mathrm{A}}(\mathbb{D})|}{|\mathsf{sched}_{\mathrm{Opt}}(\mathbb{D})|} \tag{18}$$

where

$$\mathsf{sched}_{\mathrm{A}}(\mathbb{D}) = \{\Gamma \in \mathbb{D} : \Gamma \text{ is schedulable by } \mathsf{A}\}, \tag{19}$$

and Opt is any optimal scheduling algorithm.

From this definition it follows that:

– $0 \leq \mathbf{OD}_{\mathsf{A}}(\mathbb{D}) \leq 1$, for any scheduling algorithm A and for any domain $\mathbb{D}$;

– for any optimal algorithm Opt, $\mathbf{OD}_{\mathsf{Opt}}(\mathbb{D}) = 1$ for all domains $\mathbb{D}$.

Definition 3 is very general and can be applied to every scheduling algorithm. In this section we will focus on $\mathbf{OD}_{\mathsf{RM}}(\mathbb{D})$ and we will consider EDF as a reference for optimality, since our goal is to measure the difference between RM and EDF in terms of schedulable task sets. In fact, by using the definition of $\mathbb{T}_n$, as the domain of all possible sets of n tasks (see Section 2.2), we can measure the goodness of the RM algorithm by evaluating $\mathbf{OD}_{\mathsf{RM}}(\mathbb{T}_n)$.

However, the domain $\mathbb{T}_n$ is hard to be characterized, since the concept of *"all the possible task sets"* is too fuzzy. As a consequence, $\mathbf{OD}_{\mathsf{RM}}(\mathbb{T}_n)$ can only be computed by simulation, assuming the task set parameters as random variables with some probability density function (p.d.f.). In order to make the $\mathbf{OD}_{\mathsf{RM}}(\mathbb{T}_n)$ metric less fuzzy and to find some meaningful result, we need to reduce the degrees of freedom of domain $\mathbb{T}_n$.

An important classification of the task sets is based on its utilization U. Hence, we could be interested in knowing how many task sets are schedulable among those belonging to a domain $\mathbb{D}$ with a given utilization U.

Notice that by using the utilization upper bound we can only say that a task set is schedulable if $U \leq U_{\mathrm{ub}}(\mathbb{D})$, but there is still uncertainty among those with $U > U_{\mathrm{ub}}(\mathbb{D})$. Instead by using the Optimality Degree the number of schedulable task sets can be measured by:

$$\mathbf{OD}_{\mathsf{RM}}(\mathbb{D}, U) = \mathbf{OD}_{\mathsf{RM}}\left(\mathbb{D} \cap \sum_{i=1}^{n} U_i = U\right) \tag{20}$$

The relation between $\mathbf{OD}_{\mathsf{RM}}(\mathbb{D}, U)$ and $U_{\mathrm{ub}}(\mathbb{D})$ is expressed more formally in the following Theorem.

Theorem 2. *Given a domain of task sets* $\mathbb{D}$, *the following relationships hold*:

$$\mathbf{OD}_{\mathsf{RM}}(\mathbb{D}, U) = 1 \quad \forall U \in [0, U_{\mathrm{ub}}(\mathbb{D})] \tag{21}$$

$$0 < \mathbf{OD}_{\mathsf{RM}}(\mathbb{D}, U) < 1 \quad \forall U \in (U_{\mathrm{ub}}(\mathbb{D}), 1) \tag{22}$$

Proof: From Definition 2 it follows that all the task sets having utilization $U \leq U_{\mathrm{ub}}(\mathbb{D})$ are schedulable by RM, thus Eq. (21) holds.

From Definition 2, since $U_{\mathrm{ub}}(\mathbb{D})$ is the maximum utilization bound there must be some task set in $\mathbb{D}$ which is not schedulable, hence:

$$\mathbf{OD}_{\mathsf{RM}}(\mathbb{D}, U) < 1.$$

From Lemma 1 it follows that for every value of utilization $U < 1$ it is always possible to build a task set which is schedulable by RM. Hence:

$$\mathbf{OD}_{\mathsf{RM}}(\mathbb{D}, U) > 0$$

which proves the theorem. □

It is worth observing that $\mathbf{OD}_{\mathsf{RM}}(\mathbb{D}, U)$ does not suffer the weakness of $U_{\mathrm{ub}}(\mathbb{D})$, because it is capable of measuring the number of schedulable task sets even when the total utilization exceeds $U_{\mathrm{ub}}(\mathbb{D})$.

To derive a numeric value from $\mathbf{OD}_{\mathsf{RM}}(\mathbb{D}, U)$, we need to model the utilization as a random variable with a p.d.f. $f_U(u)$ and then compute the expectation of $\mathbf{OD}_{\mathsf{RM}}(\mathbb{D}, U)$.

Definition 4. We define the Numerical Optimality Degree (abbreviated by **NOD**) as the expectation of $\mathbf{OD}_{\mathsf{RM}}(\mathbb{D}, U)$, which is:

$$\mathbf{NOD}_{\mathsf{RM}}(\mathbb{D}) = E[\mathbf{OD}_{\mathsf{RM}}(\mathbb{D}, U)] = \int_0^1 \mathbf{OD}_{\mathsf{RM}}(\mathbb{D}, u)\ f_U(u)\mathrm{d}u. \tag{23}$$

As for the average 88% bound derived by Lehoczky et al., in order to achieve a numerical result we have to pay the price of modelling a system quantity (the utilization U) by a random variable. If we assume the utilization uniform in [0, 1] Eq. (23) becomes:

$$\mathbf{NOD}_{\mathsf{RM}}(\mathbb{D}) = \int_0^1 \mathbf{OD}_{\mathsf{RM}}(\mathbb{D}, u)\ \mathrm{d}u \tag{24}$$

which finalizes our search. In fact, this value well represents the fraction of the feasible task sets which are schedulable by RM in the following hypotheses:

– the task sets belong to $\mathbb{D}$;

– the utilization of the task sets is a uniform random variable in [0, 1].

Moreover, as expected, we have that:

$$\int_0^1 \mathbf{OD}_{\mathsf{RM}}(\mathbb{D}, u)\, \mathrm{d}u = \int_0^{U_{\mathrm{ub}}(\mathbb{D})} \mathbf{OD}_{\mathsf{RM}}(\mathbb{D}, u)\, \mathrm{d}u + \int_{U_{\mathrm{ub}}(\mathbb{D})}^1 \mathbf{OD}_{\mathsf{RM}}(\mathbb{D}, u)\, \mathrm{d}u =$$
$$\int_0^{U_{\mathrm{ub}}(\mathbb{D})} 1\, \mathrm{d}u + \int_{U_{\mathrm{ub}}(\mathbb{D})}^1 \mathbf{OD}_{\mathsf{RM}}(\mathbb{D}, u)\, \mathrm{d}u = U_{\mathrm{ub}}(\mathbb{D}) + \int_{U_{\mathrm{ub}}(\mathbb{D})}^1 \mathbf{OD}_{\mathsf{RM}}(\mathbb{D}, u)\, \mathrm{d}u$$

and then:

$$\int_0^1 \mathbf{OD}_{\mathsf{RM}}(\mathbb{D}, u)\, \mathrm{d}u \geq U_{\mathrm{ub}}(\mathbb{D}) \tag{25}$$

which shows that **NOD** is less pessimistic than U_{ub}.

3. Synthetic Task Set Generation

The typical way to measure the performance of a guarantee test is to randomly generate a huge number of synthetic task sets and then verify which percentage of feasible sets pass the test. However, the way task parameters are generated may significantly affect the result and bias the judgement about the schedulability tests. In this section we analyze some common techniques often used to randomly generate task set parameters and we highlight their positive and negative aspects.

3.1. Generating Periods

Under fixed priority scheduling, task periods significantly affect the schedulability of a task set. For example, it is well known that, when

$$\forall i = 1, \ldots, n-1 \qquad T_i \text{ evenly divides } T_{i+1}$$

and relative deadlines are equal to periods, then any task set with total utilization less than or equal to 1 can be feasibly scheduled by RM (Kuo and Mok, 1991), independently of the computation times.

Whereas task execution times can have great variations, due to the effect of several low-level architecture mechanisms or to the complex structure of a task, task periods are more deterministic, since are defined by the user and then enforced by the operating system. As a consequence, the assumption of treating task periods as random variables with uniform distribution in a given interval may not reflect the characteristics of real applications and could be inappropriate for evaluating a schedulability test. For example, using a uniform p.d.f., possible harmonic relations existing among the periods cannot be replicated.

On the other hand, at an early stage of evaluating a guarantee test without any a priori knowledge about the environment where it is going to be used, assuming some probability density for the period is something we cannot avoid.

3.2. Computation Times Generation

Once task periods have been selected, running a guarantee test in the whole space of task set configurations requires computation times to be generated with a given distribution. In the absence of specific knowledge of application task, a common approach is to assume the computation time C_i uniform in $[0, T_i]$. Notice that, since computation times C_i and utilizations U_i differ by a scaling factor of T_i, this is equivalent of assuming each U_i to be uniform in [0, 1].

Considering the dependency of some schedulability test from the total processor utilization, a desirable feature of the generation algorithm is the ability to create synthetic task sets with a given utilization factor $\bar{U}$. Hence, individual task utilizations U_i should be generated with a uniform distribution in $[0, \bar{U}]$, subject to the constraint $\sum U_i = \bar{U}$. Implementing such an algorithm, however, hides some pitfalls. In the next paragraphs we will describe some "common sense" algorithms, discuss their problems, and then propose a new efficient method.

3.3. The UScaling Algorithm

A first intuitive approach, referred to as the UScaling algorithm, is to generate the U_i's in $[0, \bar{U}]$ and then scale them by a factor $\frac{\bar{U}}{\sum_1^n U_i}$, so that the total processor utilization is exactly $\bar{U}$. The UScaling algorithm has an $O(n)$ complexity and, using Matlab syntax,[2] can be coded as shown in Figure 4.

Unfortunately, the algorithm illustrated above incurs in the same problem discussed in Section 2.1 for the scaling of C_i**-aligned** task sets, whose effect was to bias the breakdown utilization. Here, the consequence of the scaling operation is that task sets having similar U_i's (those with δ close to 0) are generated with higher probability.

Figure 5 illustrates the values of 5000 utilization tuples, generated by the UScaling algorithm with $n = 3$ and $\bar{U} = 1$. As expected, the generated values are more dense around the point where all the U_i are equal to $\bar{U}/n$. As argued in Remark 2, these task sets are penalized by RM, hence generating the utilizations by the UScaling algorithm is pessimistic for RM.

3.4. The UFitting Algorithm

A second algorithm for generating task sets with a given utilization $\bar{U}$ consists in making U_1 uniform in $[0, \bar{U}]$, U_2 uniform in $[0, \bar{U} - U_1]$, U_3 uniform in $[0, \bar{U} - U_1 - U_2]$, and so

```
function vectU = UScaling(n, Ū)
vectU = rand(1,n);
vectU = vectU.*Ū./sum(vectU);
```

Figure 4. Matlab code for the UScaling algorithm.

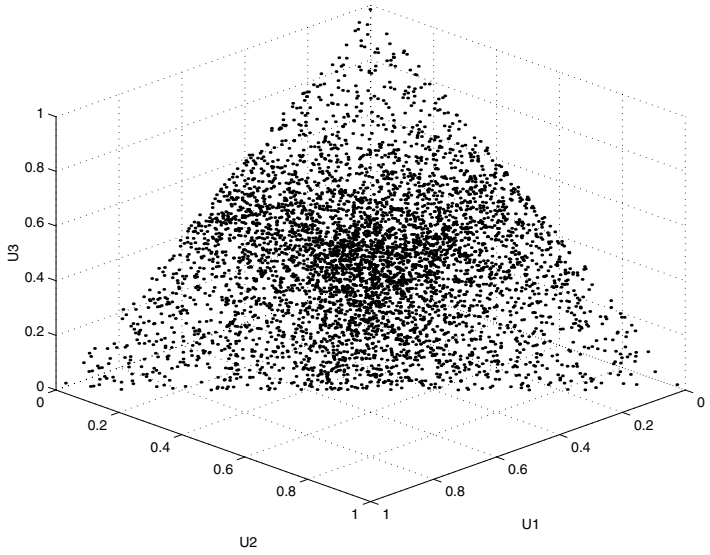

Figure 5. Result of the UScaling algorithm.

```
function vectU = UFitting(n, Ū)
upLimit = Ū;
for i=1:n−1,
  vectU(i) = rand∗upLimit;
  upLimit = upLimit−vectU(i);
end
vectU(n) = upLimit;
```

Figure 6. Matlab code for the UFitting algorithm.

on, until U_n is deterministically set to the value $U_n = \bar{U} - \sum_{i=1}^{n-1} U_i$. This method, referred to as UFitting, is described by the code illustrated in Figure 6.

The UFitting algorithm has an $O(n)$ complexity, but it has the major disadvantage of being asymmetrical, meaning that the U_1 has a different distribution than U_2, and so forth.

Moreover, as depicted in Figure 7, the achieved distribution is again not uniform, and task sets having different values of U_i's (those with δ close to 1) are generated with higher probability. Hence, for the same reasons stated in Remark 2, generating the utilizations by the UFitting algorithm favors RM with respect to EDF.

3.5. The UUniform Algorithm

The problems previously encountered can be solved through the method depicted in Figure 8, referred to as the UUniform algorithm.

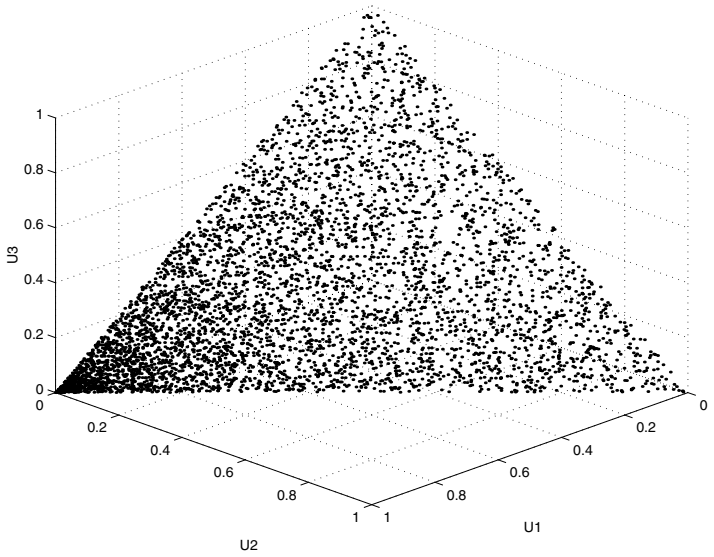

Figure 7. Result of the UFitting algorithm.

```
function vectU = UUniform(n, U)
while 1
  vectU = U.*rand(1,n-1);
  if sum(vectU) <= U          % boundary condition
     break
  end
end
vectU(n) = U-sum(vectU);
```

Figure 8. Matlab code for the UUniform algorithm.

As depicted in Figure 9, the UUniform algorithm generates task utilizations with uniform distribution.

The problem with this algorithm, however, is that it has to run until the boundary condition is verified once (see the code of UUniform). As proved by Bini et al. (2003) the probability of such an event is $1/(n-1)!$, hence the average number of iterations needed to generate a single tuple is $(n-1)!$, which makes the algorithm unpractical.

3.6. The UUniFast Algorithm

To efficiently generate task sets with uniform distribution and with $O(n)$ complexity, we introduce a new algorithm, referred to as the UUniFast algorithm. It is built on the consideration that the p.d.f. of the sum of independent random variables is given by the convolution

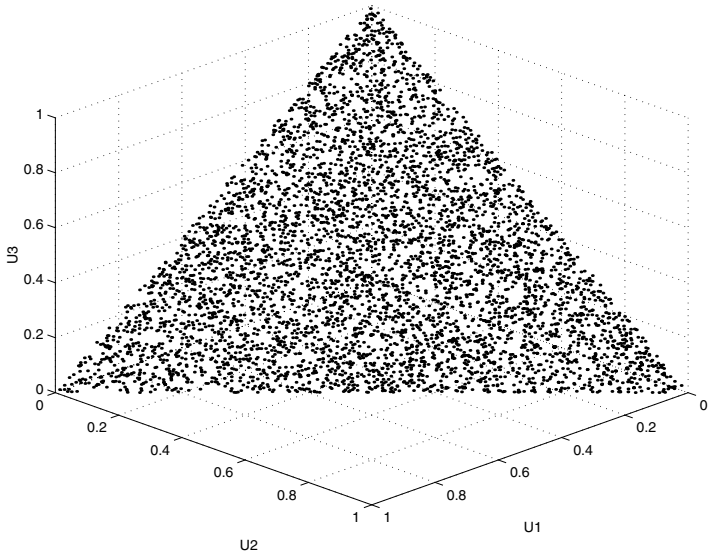

Figure 9. Result of the UUniform algorithm.

of their p.d.f.'s. Given that the i random variables are uniformly distributed in [0, 1] (so their p.d.f.'s are 1 in [0, 1] and 0 everywhere else) the convolution is a piecewise polynomy of the $(i-1)$th degree. In our fortunate case, the sum of the variables must be less than or equal to a value $b \leq 1$, thus the p.d.f. of the sum of the uniform variables, constrained to be less than or equal to b, is:

$$f_i(u) = \begin{cases} \dfrac{1}{b}u^{i-1} & \text{if } u \in [0, b] \\ 0 & \text{otherwise} \end{cases} \tag{26}$$

Hence, the cumulative distribution function (c.d.f.) is:

$$F_i(u) = \begin{cases} 0 & \text{if } u \leq 0 \\ \left(\dfrac{u}{b}\right)^i & \text{if } 0 < u \leq b \\ 1 & \text{if } u > b \end{cases} \tag{27}$$

Using this c.d.f.'s, the UUniFast algorithm first generates a value of the sum of $n-1$ variables. Then it sets the first utilization equal to the difference between $\bar{U}$ and the just generated value. So it keeps generating the random variable "sum of i uniform variables" and computing the single utilization U_i as the difference with the previous sum. The algorithm can be precisely described by the code reported in Figure 10.

As we can see from the code, the complexity of the UUniFast algorithm is $O(n)$ and the generated utilization tuples are characterized by a uniform distribution (the result is the same as that illustrated in Figure 9).

```
function vectU = UUniFast(n, Ū)
sumU = Ū;
  for i=1:n−1,
  nextSumU = sumU.∗rand^(1/(n−i));
  vectU(i) = sumU − nextSumU;
  sumU = nextSumU;
end
vectU(n) = USum;
```

Figure 10. Matlab code for the UUniFast algorithm.

```
function vectU = UUniSort(n, Ū)
v = [0, rand(1,n−1), Ū];
sumU = sort(v);
vectU = sumU(2:n+1)−sumU(1:n);
```

Figure 11. Matlab code for the UUniSort algorithm.

3.7. The UUniSort Algorithm

A more elegant version of the algorithm for achieving a uniform distribution is to generate $n-1$ uniform values in $[0, \bar{U}]$, add 0 and $\bar{U}$ to the vector, sort it, and then set the utilizations equal to the difference of two adjacent values (Marzario, 2004). This algorithm, referred to as UUniSort, is coded as shown in Figure 11.

Note that the UUniSort algorithm, although more elegant than UUniFast, has an $O(n \log(n))$ complexity.

3.8. Comparing The Algorithms

Figures 5 and 7 qualitatively show the uneven distribution of the generated task sets in the utilization space for the UScaling and the UFitting algorithm, respectively. A quantitative measure of the resulting distribution can be provided by computing the *U-difference* parameter defined in Eq. (1).

Figure 12 shows the probability density function of the *U-difference*, when $n = 8$ and $2 \cdot 10^5$ random task sets are generated.

As expected, the utilizations generated by the UScaling algorithm are characterized by values of *U-difference* close to zero, meaning that the utilization values tend to be similar to each other. On the other hand, the utilizations generated by the UFitting algorithm differ much more, as proved by the long tail of the related *U-difference* density function.

4. Metrics Comparison

In this section we present a set of simulation experiments aimed at comparing the three metrics discussed in the paper. Although the numerical results provided here are not intended

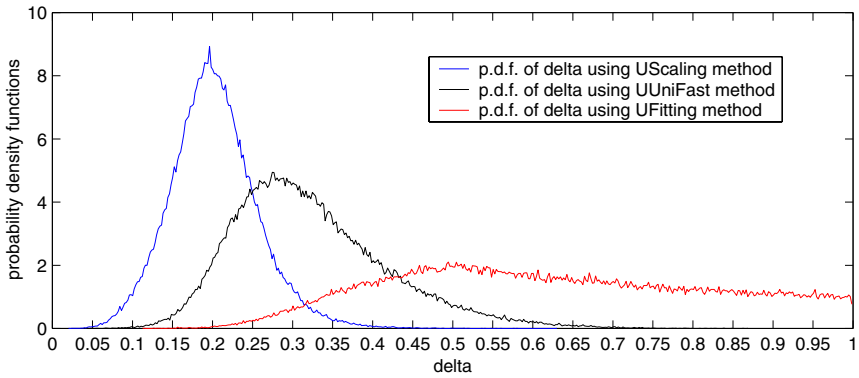

Figure 12. Value of δ for different generation methods.

to be an absolute measure of the RM schedulability, the objective of this work is to show that RM is capable of scheduling many more task sets than commonly believed.

Comparing the three metrics is not trivial, because they have different definitions and require different simulations. More specifically, while the utilization upper bound U_{ub} depends only on periods and deadlines, both the breakdown utilization U^* and the optimality degree **OD** also depend on the computation times (i.e., on the utilizations). Hence, the algorithm selected for the random generation of the computation times/utilizations may affect the results.

Two groups of experiments are presented: the first group is intended to check the influence of the random parameters generation routines on the considered metrics; the second group is aimed at testing how period relations affect the metrics.

4.1. Effects of the Generation Algorithm

The first two experiments show the influence of the generation algorithm on the metrics. We remind that U_{ub} does not depend on the computation times of the tasks set. For this reason, this subsection only compares the breakdown utilization U_n^* with the Optimality Degree $\mathbf{OD}_{\mathsf{RM}}(\mathbb{D}, U)$. The simulation was carried out by fixing the periods and deadlines, and then generating the computation times. Note that this choice is not restrictive, because the distribution of U_n^* and $\mathbf{OD}_{\mathsf{RM}}(\mathbb{D}, U)$ scales with U_{ub}. Task periods were set to the values shown in Table 2.

In Table 2 all the $U_{\text{ub}}^{(i)}$ are reported. The computation times where the optimal solution of the problem in Eq. (11) occurs is reported as well. As you can notice, the schedulability of the task τ_i is not influenced in any way by all the lower priority tasks. This fact is represented in the table by the symbol "-". For the specific period selection, U_{ub} is given by the minimum among the numbers in the fourth column, which is $\frac{9}{10} = 0.9$.

Considering the Definition 1, we expect the breakdown utilization to be always greater than the U_{ub}. This fact is confirmed by the simulation results reported in Figure 13. In this experiment, $2 \cdot 10^5$ tuples have been generated for each method described in Section 3. The

Table 2. Task set parameters.

				where $U_{\text{ub}}^{(i)}$ occurs					
i	T_i	D_i	$U_{\text{ub}}^{(i)}$	C_1	C_2	C_3	C_4	C_5	C_6
1	3	3	1	3	–	–	–	–	–
2	8	8	$\frac{11}{12} = 0.9167$	2	2	–	–	–	–
3	20	20	$\frac{9}{10} = 0.9$	0	4	8	–	–	–
4	42	42	$\frac{201}{210} = 0.9571$	1	0	2	22	–	–
5	120	120	$\frac{201}{210} = 0.9571$	0	0	0	36	12	–
6	300	300	$\frac{9}{10} = 0.9$	0	0	0	0	60	120

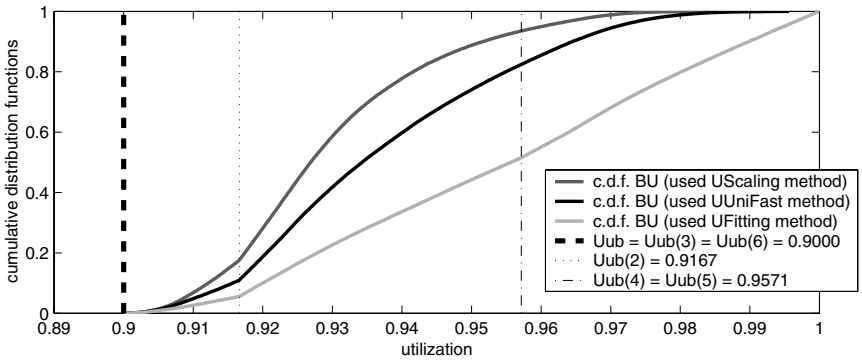

Figure 13. c.d.f. of U^* for different algorithms.

plots in the figure clearly show the biasing factor introduced by the different generation algorithms. In fact, the breakdown utilization has a larger c.d.f. when tasks are generated with the UScaling algorithm, meaning that RM is penalized. The opposite effect occurs when tasks are generated with UFitting, which favors RM more than under the uniform distribution produced by UUniFast. As synthetic values for the c.d.f.'s we can use the expectation of the three breakdown utilization random variables obtained by the three different methods, that is

$$E[U^*|_{\text{UScaling}}] = 0.9296,$$
$$E[U^*|_{\text{UUniFast}}] = 0.9372,$$
$$E[U^*|_{\text{UFitting}}] = 0.9545.$$

However, as extensively discussed in Section 2.1, the breakdown utilization is not a fair metric for evaluating the difference between RM and EDF in terms of schedulability. For this purpose, the **Optimality Degree** (**OD**) has been introduced in Section 2.3. Figure 14 reports the OD values as a function of the total utilization and for different generation methods.

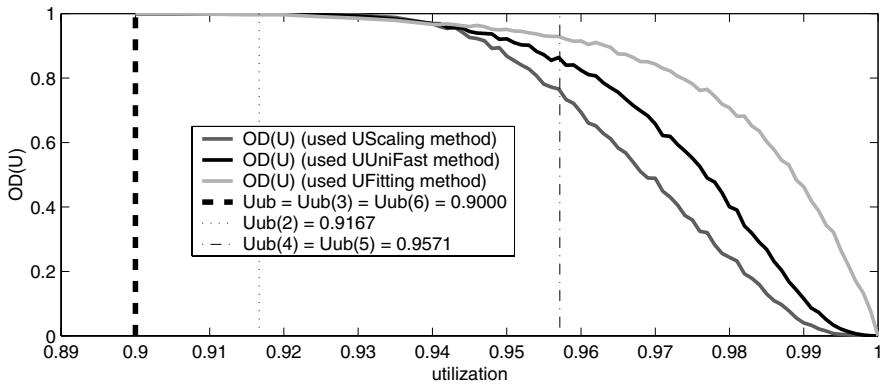

Figure 14. OD for different algorithms.

In this experiment, $3{\cdot}10^5$ task sets are generated, 5000 for each value of the utilization. The insight we derive is consistent with the previous experiment: the UScaling algorithm penalizes RM more than UUniFast, whereas UFitting favors RM by generating easier task sets.

Assuming the total utilization is uniform in [0, 1], the **NOD** parameter (see Definition 4 and Eq. (24)) is computed for the three methods:

$$\begin{aligned}\mathbf{NOD}|_{\text{UScaling}} &= 0.9679\\ \mathbf{NOD}|_{\text{UUniFast}} &= 0.9739\\ \mathbf{NOD}|_{\text{UFitting}} &= 0.9837\end{aligned}$$

showing a much better behavior of RM than expected with the breakdown utilization. The result of the experiments are summarized in Table 3.

The metric we consider more reliable is $\mathbf{NOD}|_{\text{UUniFast}}$, because it is related to the real percentage of feasible task sets and it refers to task sets uniformly distributed in the utilization space. As a consequence, in the next simulations, the UUniFast algorithm is adopted for generating the utilizations U_i.

Table 3. Schedulability results for RM.

Metric	Value
$U_{\text{ub}}(\mathbb{M}_n)$	0.9
$E[U^*\|_{\text{UScaling}}]$	0.9296
$E[U^*\|_{\text{UUniFast}}]$	0.9372
$E[U^*\|_{\text{UFitting}}]$	0.9545
$\mathbf{NOD}\|_{\text{UScaling}}$	0.9679
$\mathbf{NOD}\|_{\text{UUniFast}}$	0.9739
$\mathbf{NOD}\|_{\text{UFitting}}$	0.9837

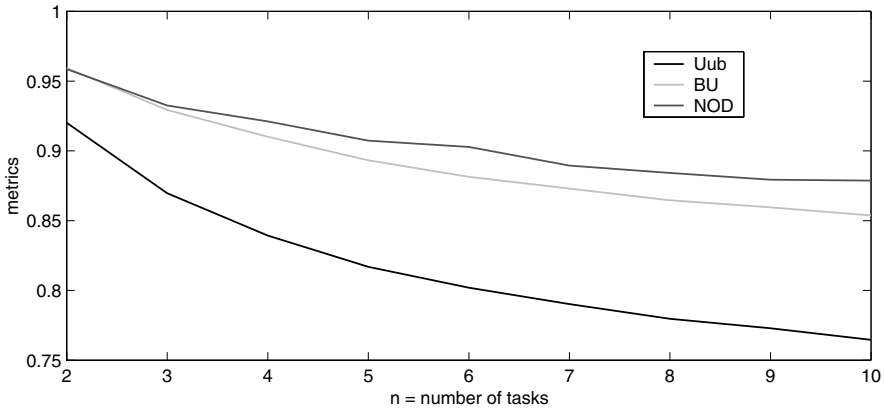

Figure 15. Metrics as a function of n.

4.2. Effects of the Task Set Parameters

In the two experiments described in this section task periods are uniformly generated in $[1, B]$ and then utilizations are generated by the algorithm UUniFast. The objective of the experiments is to compare the metrics $E[U_{\text{ub}}]$, $E[U^*]$ and $E[\mathbf{NOD}]$. For the sake of simplicity, we will omit the expectation operator $E[\cdot]$.

The objective of the first experiment is to study the dependency of the metrics on the number n of tasks. To do that, we set $B = 100$ and generated a total number of $2 \cdot 10^4$ task sets. As expected, all the metrics report a decrease in the schedulability under RM and they are ordered in a way similar to the first experiment.

The second experiment of this section aims at considering the dependency of the metrics on the task periods. To do so, we set $n = 4$ and let B vary in the interval $[1, 10^4]$. We

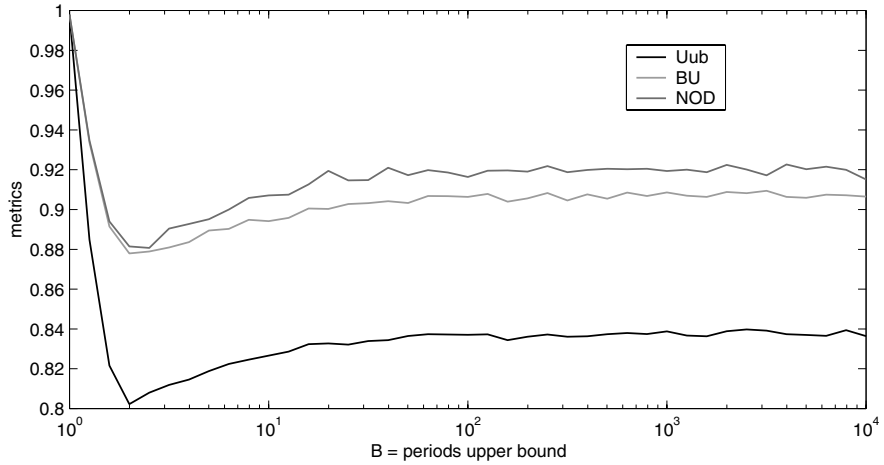

Figure 16. Metrics as a function of periods.

generated $5 \cdot 10^4$ task sets. The result is reported in Figure 16. When $B = 1$ the periods are all the same value. The value of 1 is then what we expected.

In addition, all the three metrics seems to have a minimum for $B = 2$. This fact is also confirmed by the asymptotical study of the breakdown utilization (reported in Theorem 1) and by many different proofs in the real-time literature (Liu and Layland, 1973; Burchard et al., 1995; Chen et al., 2003; Bini et al., 2003) which show this phenomenon.

5. Conclusions

The motivation for writing this paper has been to analyze in great detail how the methodology used for evaluating the performance of fixed priority scheduling algorithms affects the results. In particular, we have considered two metrics commonly used in the literature and showed that both the *breakdown utilization* and the *utilization upper bound* can be unfair in judging the performance of the Rate/Deadline Monotonic scheduling algorithms. We also illustrated that significant biasing factors can be introduced by the routines used for generating random task sets.

The main result achieved from this study is that current metrics intrinsically evaluate the behavior of RM in pessimistic scenarios, which are more critical for fixed priority assignments than for dynamic systems. The use of unbiased metrics, such as the *Optimality Degree*, shows that the penalty payed in terms of schedulability by adopting fixed priority scheduling is less than commonly believed.

Notes

1. $E[X]$ denotes the expectation of the random variable X, and σ_X denotes its standard deviation.
2. We remind that in Matlab arrays can be assigned using the same syntax used for variables.

References

Audsley, N. C., Burns, A., Richardson, M., Tindell, K. W., and Wellings: A. J. 1993. Applying new scheduling theory to static priority pre-emptive scheduling.*Software Engineering Journal* 8(5): 284–292.

Bini, E., and Buttazzo, G. C. 2004a. Biasing effects in schedulability measures. In: *Proceedings of the* 16th *Euromicro Conference on Real-Time Systems*. Catania, Italy, pp. 196–203.

Bini, E., and Buttazzo, G. C. 2004b. Schedulability analysis of periodic fixed priority systems. *IEEE Transactions on Computers* 53(11): 1462–1473.

Bini, E., Buttazzo, G. C., and Buttazzo, G. M. 2003. Rate monotonic scheduling: The hyperbolic bound. *IEEE Transactions on Computers* 52(7): 933–942.

Burchard, A., Liebeherr, J., Oh, Y., and Son, S. H. 1995. New strategies for assigning real-time tasks to multiprocessor systems. *IEEE Transactions on Computers* 44(12): 1429–1442.

Buttazzo, G. C. 2003. Rate monotonic vs. EDF: Judgment day. In: *Proceedings of the EMSOFT*. Philadelphia, PA USA, pp. 67–83.

Chen, D., Mok, A. K., and Kuo, T.-W. 2003. Utilization bound revisited. *IEEE Transaction on Computers* 52(3): 351–361.

Han, C.-C., and Tyan, H.-y. 1997. A better polynomial-time schedulability test for real-time fixed-priority scheduling algorithm. In: *Proceedings of the* 18th *IEEE Real-Time Systems Symposium*. San Francisco, CA USA.

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

Kuo, T.-W., and Mok, A. K. 1991. Load adjustment in adaptive real-time systems'. In: *Proceedings of the* 12th *IEEE Real-Time Systems Symposium*. San Antonio, TX USA, pp. 160–170.

Lauzac, S., Melhem, R., and Mossé, D. 2003. An improved rate-monotonic admission control and its applcations. *IEEE Transactions on Computers* 52(3): 337–350.

Lee, C.-G., Sha, L., and Peddi, A. 2004. Enhanced utilization bounds for QoS management. *IEEE Transactions on Computers* 53(2): 187–200.

Lehoczky, J. P., Sha, L., and Ding, Y. 1989. The rate-monotonic scheduling algorithm: Exact characterization and average case behavior. In: *Proceedings of the* 10th *IEEE Real-Time Systems Symposium*. Santa Monica, CA USA, pp. 166–171.

Lehoczky, J. P., Sha, L., and Strosnider, J. K. 1995. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environment. *IEEE Transactions on Computers* 44(1): 73–91.

Liu, C. L., and Layland, J. W. 1973. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM* 20(1): 46–61.

Manabe, Y., and Aoyagi, S. 1995. A feasibility decision algorithm for rate monotonic scheduling of periodic real-time tasks. In: *Proeedings. of the* 1st *Real-Time Technology and Applications Symposium*. pp. 297–303.

Marzario, L. 2004. Personal communication.

Mok, A. K. 1983. Fundamental design problems of distributed systems for the hard-real-time environment. Ph.D. thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Boston, MA USA.

Park, D.-W., Natarajan, S., Kanevsky, A., and Kim, M. J. 1995. A generalized utilization bound test for fixed-priority real-time scheduling'. In: *Proceedings of the* 2nd *International Workshop on Real-Time Systems and Applications*. Tokyo, Japan, pp. 73–77.

Sjödin, M., and Hansson, H. 1998. Improved response-time analysis calculations. In: *Proceedings of the* 19th *IEEE Real-Time Systems Symposium*. Madrid, Spain, pp. 399–408.

Enrico Bini received the Ph.D. in Computer Engineering from Scuola Superiore Sant'Anna in Pisa, in October 2004. In 2000 he received the Laurea degree in Computer Engineering from "Università di Pisa" and, one year later, he obtained the "Diploma di Licenza" from the Scuola Superiore Sant'Anna. In 1999 he studied at Technische Universiteit Delft, in the Nederlands, by the Erasmus student exchange program. In 2001 he worked at Ericsson Lab Italy in Roma. In 2003 he was a visiting student at University of North Carolina at Chapel Hill, collaborating with prof. Sanjoy Baruah. His research interests cover scheduling algorithms, real-time operating systems, embedded systems design and linear programming.

Giorgio Buttazzo is an Associate Professor of Computer Engineering at the University of Pavia, Italy. He graduated in Electronic Engineering at the University of Pisa in 1985, received a Master in Computer Science at the University of Pennsylvania in 1987, and a Ph.D. in Computer Engineering at the Scuola Superiore S. Anna of Pisa in 1991. During 1987, he worked on active perception and real-time control at the G.R.A.S.P.