Biasing Effects in Schedulability Measures*

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

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

Abstract

The performance of a schedulabilty 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.

1. Introduction

Rate Monotonic (RM) scheduling [13] is the most used approach for implementing periodic real-time applications in fixed priority systems. Although, the superior behavior of EDF over RM has been shown under different circumstances [5], the simpler implementation of RM seems to be the major cause that prevents the use of dynamic priority schemes in practical applications.

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 [8, 12, 1]. Methods for improving the computation of task response times have been recently proposed [14, 16, 2], but the complexity of the approach remains pseudo-polynomial in the worst case.

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 [3], that has the same $O(n)$ complexity of the classical utilization bound proposed by Liu and Layland [13], but better performance. Other feasibility tests [4, 7, 9] exploit additional information on period relations to verify schedulability in a polynomial time complexity. A tunable test, called the Hyperplane δ-Exact Test, has recently been proposed [2] to balance performance versus complexity using a single parameter.

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*.

Computing the acceptance ratio using a mathematical 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. So both the evaluation metric and the way task parameters are generated significantly affect the overall performance of the test.

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 routine bias the simulation 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 for task set generation can bias the simulation results.

2. Metrics

It is well known that the RM scheduling algorithm cannot schedule all the feasible task sets. Several methods have been proposed to compare the effectiveness of the RM scheduling algorithm with the optimal EDF. The most common approaches are based on the concept of *breakdown utilization* [11] and *utilization upper bound* [15, 6, 10]. As we discuss in Sections 2.1 and 2.2, both techniques present some drawback, thus in Section 2.3, we present a new performance evaluation criterion, the *Optimality Degree* (**OD**), which prevents those problems.

*Enrico Bini was visiting University of North Carolina at Chapel Hill when working on this paper.

2.1. Breakdown utilization

The concept of breakdown utilization was first introduced by Lehoczky, Sha, and Ding, in their seminal work [11] aimed at providing exact characterization and average case behavior of the RM scheduling algorithm.

Definition 1 (Section 3 in [11]) *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 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 1.

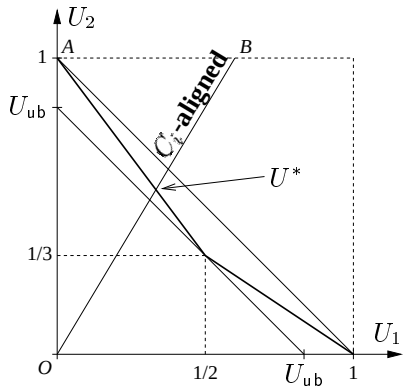

Figure 1. Interpretation of the breakdown utilization.

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 1.

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*:

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

As it can be argued from Figure 1, 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 $\left|\overline{OB}\right|$ 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 $U_1 = \left|\overline{AB}\right|$. Hence:

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

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

then:

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

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

In the general case of n tasks, finding the exact $\left|\overline{OB}\right|$ is much harder. However, 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 [13] that the worst-case configuration of the utilization factors occurs when they are all equal to each other;
- using the Hyperbolic Bound [3], 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.

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 Fig. 4).

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 [11]) *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 [11]) *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 $U_{\mathtt{ub}}$ is a well known parameter, used by many authors [13, 15, 6, 10] to provide a schedulability test for the RM algorithm. Following the approach proposed by Chen et al. [6], the utilization upper bound is defined as a function of a given domain $\mathbb{D}$ of task sets.

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

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

From the definition above it follows that:

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

According to this definition, Liu and Layland [13] found the upper bound $U_{\mathtt{ub}}$ for all possible sets of n tasks. If $\mathbb{T}_n$ is the domain of all sets of n tasks, they proved that:

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

which is also referred to as *utilization least upper bound*, because $\mathbb{T}_n$ is the largest possible domain, and hence $U_{\mathtt{ub}}(\mathbb{T}_n) = U_{\mathtt{lub}}$ 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 [6] when $D_i = T_i$ and $\frac{T_n}{T_1} \leq 2$:

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

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 [6] that:

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

When no restrictions apply on periods and deadlines, the utilization upper bound $U_{\mathtt{ub}}(\mathbb{M}_n)$, shortly denoted by $U_{\mathtt{ub}}$ from now and on, can be found by solving n linear programming problems [15, 10]. The i^{th} optimization problem

(for i from 1 to n) comes from the schedulability constraint of the task τ_i. It is formulated as follows:

$$\min_{(C_1,\ldots,C_i)} \quad \sum_{j=1}^{i} \frac{C_j}{T_j}$$

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

where the reduced set of schedulability points $\mathcal{P}_{i-1}(D_i)$, found in [14, 2], is used instead of the largest one firstly introduced in [11]. If we label the solution of the i^{th} problem (11) as $U_{\mathsf{ub}}^{(i)}$, the utilization upper bound is given by:

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

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

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

The utilization bound defined above is tight, meaning that for every value $U > U_{\mathsf{ub}}$ 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 Equation (13) is a necessary and sufficient condition for the RM schedulability, meaning that *"all the task sets having total utilization $U > U_{\mathsf{ub}}$ are not schedulable"*. Unfortunately, this is not true and the RM schedulability is far more complex. 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:*

$$\mathbf{OD}_{\mathsf{A}}(\mathbb{D}) = \frac{|\mathsf{sched}_{\mathsf{A}}(\mathbb{D})|}{|\mathsf{sched}_{\mathsf{Opt}}(\mathbb{D})|} \tag{14}$$

where

$$\mathsf{sched}_{\mathsf{A}}(\mathbb{D}) = \{\Gamma \in \mathbb{D} : \Gamma \textit{ is schedulable by } \mathsf{A}\}, \tag{15}$$

and Opt is any optimal scheduling algorithm.

From this definition it follows that:

- for any scheduling algorithm A and for any domain $\mathbb{D}$, $0 \leq \mathbf{OD}_{\mathsf{A}}(\mathbb{D}) \leq 1$;
- for any optimal algorithm Opt, $\mathbf{OD}_{\mathsf{Opt}}(\mathbb{D}) = 1$ for all domains $\mathbb{D}$.

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 $U \leq U_{\mathsf{ub}}(\mathbb{D})$, but there is still uncertainty among those with $U > U_{\mathsf{ub}}(\mathbb{D})$. Instead by using the Optimality Degree the number of schedulable task sets can be measured by:

$$\mathbf{OD}_{\mathsf{RM}}(\mathbb{D}, U) = \mathbf{OD}_{\mathsf{RM}}\left(\mathbb{D} \cap \sum_{i=1}^{n} U_i = U\right) \tag{16}$$

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

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 (NOD)** as the expectation of $\mathbf{OD}_{\mathsf{RM}}(\mathbb{D}, U)$, which is:*

$$\mathbf{NOD}_{\mathsf{RM}}(\mathbb{D}) = \int_0^1 \mathbf{OD}_{\mathsf{RM}}(\mathbb{D}, u)\ f_U(u)\mathrm{d}u. \tag{17}$$

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 $[0, 1]$ Equation (17) becomes:

$$\mathbf{NOD}_{\mathsf{RM}}(\mathbb{D}) = \int_0^1 \mathbf{OD}_{\mathsf{RM}}(\mathbb{D}, u)\ \mathrm{d}u \qquad (18)$$

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:

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

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

and then:

$$\int_0^1 \mathbf{OD}_{\mathsf{RM}}(\mathbb{D}, u)\ \mathrm{d}u \geq U_{\mathsf{ub}}(\mathbb{D}) \qquad (19)$$

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

3. Synthetic task sets 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.

In a synthetic task set, task periods are usually generated as random variables with a uniform distribution in a given interval. Although this choice may not reflect the characteristics of a real application, assuming some probability density for the periods cannot be avoided.

Once task periods T_i have been selected, running a guarantee test in the whole space of task set configurations requires computation times C_i to be generated with a uniform distribution 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 $\overline{U}$. Hence, individual task utilizations U_i should be generated with a uniform distribution in $[0, \overline{U}]$, subject to the constraint $\sum U_i = \overline{U}$. 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.

A first intuitive approach, referred to as the UScaling algorithm, is to generate the U_i's in $[0, \overline{U}]$ and then scale them by a factor $\frac{\overline{U}}{\sum_1^n U_i}$, so that the total processor utilization is exactly $\overline{U}$. The UScaling algorithm has an $O(n)$ complexity and, using Matlab syntax[1], can be so coded:

function vectU = UScaling(n, $\overline{U}$)
vectU = **rand**(1,n);
vectU = vectU.$*\overline{U}$./**sum**(vectU);

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 2 illustrates the values of 5000 utilization tuples, generated by the UScaling algorithm with $n = 3$ and $\overline{U} = 1$. As expected, the generated values are more dense

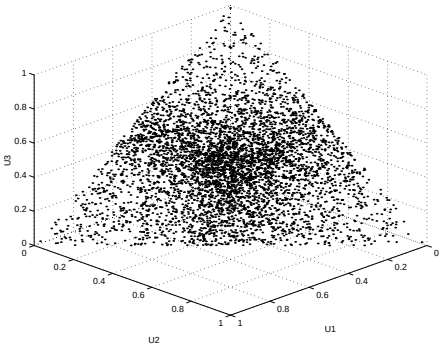

Figure 2. Result of the UScaling **algorithm.**

around the point where all the U_i are equal to $\frac{\overline{U}}{n}$. As argued in Remark 2, these task sets are penalized by RM, hence generating the utilizations by the UScaling algorithm is pessimistic for RM.

A second algorithm for generating task sets with a given utilization $\overline{U}$ consists in making U_1 uniform in $[0, \overline{U}]$, U_2 uniform in $[0, \overline{U} - U_1]$, U_3 uniform in $[0, \overline{U} - U_1 - U_2]$, and so on, until U_n is deterministically set to the value $U_n = \overline{U} - \sum_{i=1}^{n-1} U_i$. This method, referred to as UFitting, can be described by the following code:

function vectU = UFitting(n, $\overline{U}$)
upLimit = $\overline{U}$;
for i=1:n−1,
  vectU(i) = **rand**$*$upLimit;
  upLimit = upLimit−vectU(i);
end
vectU(n) = upLimit;

[1] Note that in Matlab arrays can be assigned as variables.

The UFitting algorithm has an $O(n)$ complexity, but 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 3, 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

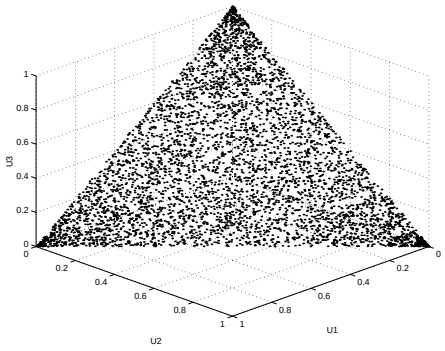

Figure 3. Result of the UFitting algorithm.

Remark 2, generating the utilizations by the UFitting algorithm favors RM with respect to EDF.

The problems described above can be solved though the following method, referred to as the UUniform algorithm:

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

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 in [3] the probability of such an event is $\frac{1}{(n-1)!}$, hence the average number of iterations needed to generate a single tuple is $(n-1)!$, which makes the algorithm unpractical.

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 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 $(i-1)^{\text{th}}$ 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:

$$f_i(u) = \begin{cases} \frac{1}{b}u^{i-1} & \text{if } u \in [0,b] \\ 0 & \text{otherwise} \end{cases} \quad (20)$$

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

$$F_i(u) = \begin{cases} 0 & \text{if } u \leq 0 \\ \left(\frac{u}{b}\right)^i & \text{if } 0 < u \leq b \\ 1 & \text{if } u > b \end{cases} \quad (21)$$

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 $\overline{U}$ 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 following code:

```
function vectU = UUniFast(n, U_bar)
sumU = U_bar;
for i=1:n-1,
  nextSumU = sumU.*rand^(1/(n-i));
  vectU(i) = sumU - nextSumU;
  sumU = nextSumU;
end
vectU(n) = USum;
```

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.

4. Simulation results

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 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 1.

			where $U_{\mathsf{ub}}^{(i)}$ occurs					
i	T_i, D_i	$U_{\mathsf{ub}}^{(i)}$	C_1	C_2	C_3	C_4	C_5	C_6
1	3	1	3	-	-	-	-	-
2	8	0.917	2	2	-	-	-	-
3	20	**0.9**	0	4	8	-	-	-
4	42	0.957	1	0	2	22	-	-
5	120	0.957	0	0	0	36	12	-
6	300	**0.9**	0	0	0	0	60	120

Table 1. Task set parameters.

In Table 1 all the $U_{\mathsf{ub}}^{(i)}$ are reported. The computation times where the optimal solution of the problem (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 third 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

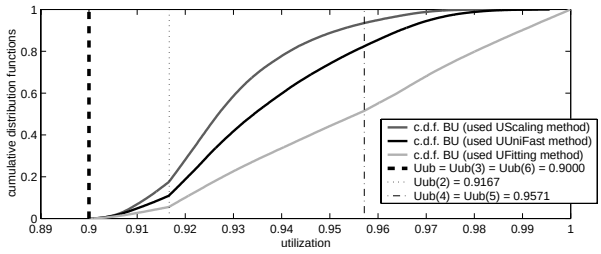

Figure 4. c.d.f. of U^* for different algorithms.

confirmed by the simulation results reported in Figure 4. In this experiment, $2 \cdot 10^5$ tuples have been generated for each method described in Section 3. The 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, reported in Table 2.

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 5 reports the OD values as a function of the total utilization and for different generation methods.

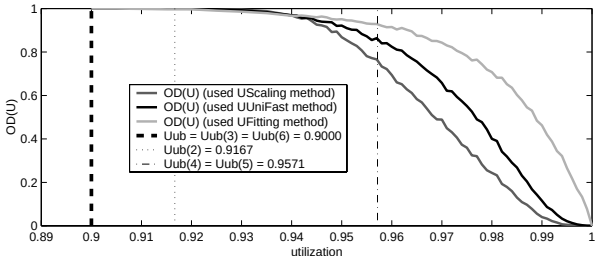

Figure 5. OD for different algorithms.

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 Equation (18)) is computed and reported in Table 2, showing a much better behavior with respect to the breakdown utilization.

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

Table 2. Schedulability results for RM.

The metric we consider more reliable is $\mathbf{NOD}|_{\mathsf{UUniFast}}$, 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.

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 $E[U_{\mathsf{ub}}]$, $E[U^*]$ 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 $2 \cdot 10^4$

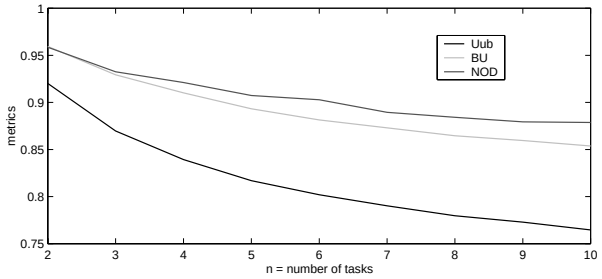

Figure 6. Metrics as a function of n.

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 generated $5 \cdot 10^4$ task sets. The result is reported in Figure 7. When $B = 1$ the periods are all the same value. The

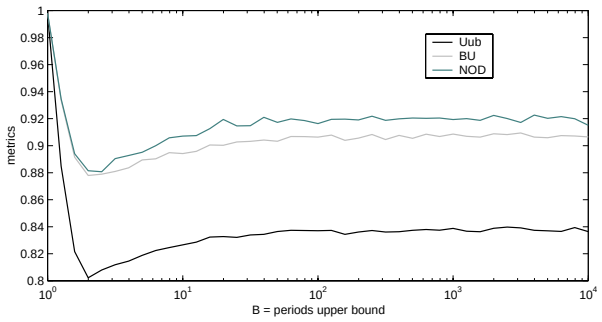

Figure 7. Metrics as a function of periods.

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 [13, 4, 6, 3] 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.

References

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

[2] E. Bini and G. C. Buttazzo. The space of rate monotonic schedulability. In *Proc. of the* 23^{rd} *IEEE Real-Time Systems Symposium*, 2002.

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

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

[5] G. C. Buttazzo. Rate monotonic vs. EDF: Judgment day. In *Proc. of the* 3^{rd} *International Conference on Embedded Software*, 2003.

[6] D. Chen, A. K. Mok, and T.-W. Kuo. Utilization bound revisited. *IEEE Transaction on Computers*, 52(3), 2003.

[7] C.-C. Han and H.-y. Tyan. A better polynomial-time schedulability test for real-time fixed-priority scheduling algorithm. In *Proc. of the* 18^{th} *IEEE Real-Time Systems Symposium*, 1997.

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

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

[10] C.-G. Lee and L. Sha. Enhanced utilization bounds for QoS management. *Accepted for Publication in IEEE Transaction on Computers*, 2003.

[11] J. Lehoczky, L. Sha, and Y. Ding. The rate-monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proc. of the* 10^{th} *IEEE Real-Time Systems Symposium*, 1989.

[12] J. P. Lehoczky, L. Sha, and J. Strosnider. Enhanced aperiodic responsiveness in hard real-time environment. In *Proc. of the* 8^{th} *IEEE Real-Time Systems Symposium*, 1987.

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

[14] Y. Manabe and S. Aoyagi. A feasibility decision algorithm for rate monotonic scheduling of periodic real-time tasks. In *Proc. of the* 1^{st} *Real-Time Technology and Applications Symposium*, 1995.

[15] D. Park, S. Natarajan, and M. J. Kim. A generalized utilization bound test for fixed-priority real-time scheduling. In *Proc. of the* 2^{nd} *International Workshop on Real-Time Systems and Applications*, 1995.

[16] M. Sjödin and H. Hansson. Improved response-time analysis calculations. In *Proc. of the* 19^{th} *IEEE Real-Time Systems Symposium*, 1998.