Limitation of the Hyper-Period in Real-Time Periodic Task Set Generation

Joël Goossens, Christophe Macq
Université Libre de Bruxelles, Département d'Informatique
Bld. du Triomphe C.P. 212, B-1050 Brussels, Belgium.
Joel.Goossens,Christophe.Macq@ulb.ac.be

Abstract: This paper presents a method used in order to generate arbitrary task systems, compound of periodic and independent tasks. The originality of this methods relays on a judicious choice of the periods of the tasks in order to reduce the lcm of the periods (and the simulation duration required in order to study such a system). The authors prove that the lcm of n integers might be very high: they propose an upper bound of the lcm of n integers, given by e^m where m is the least prime number greater than the greatest integer in the set of the n integers. Then, they present a method in order to choose the periods of the tasks which imply a bounded lcm of the periods. The method relies on the decomposition of an integer in distinct prime factors. A second algorithm is proposed in order to obtain the other temporal parameters of the task system to generate. Then, an example is given in order to illustrate the method: several task systems are generated for performance analysis of classical scheduling algorithms, in term of preemption.

Index Terms: real-time system, hard real-time scheduling, simulation, periodic task set, hyper-period, preemption, deadline monotonic, earliest deadline first, least laxity first.

1 Introduction

Real-time systems are characterized by stringent timing constraints (generally expressed in the form of a deadline); hence, the correctness of a task computation depends not only on its logical or computational results, but also on the instant when the result is made available. The most important feature of real-time systems

is their *predictability*, i.e., the ability to determine whether the system is capable (or not) to meet all the timing requirements of the tasks. Examples of such systems include the control of engines, traffic, nuclear power plants, time-critical packet communications, aircraft avionics and robotics. This leads to interesting problems, and even in apparently simple cases, like systems composed of independent periodic tasks on a single processor, the behavior of the schedules may be surprisingly complex.

The algorithms used for the scheduling of periodic hard real-time tasks are generally priority-driven preemptive algorithms. These algorithms assign priorities to tasks according to some policy. At each instant, the processor is assigned to the highest priority task which is ready to run, preempting (if necessary) a lower priority running task.

One generally distinguishes *static schedulers*, where each task receives a distinct priority beforehand, and *dynamic schedulers*, where each request of each task receives some priority, which may even change with time. Interesting sub-cases are *synchronous systems*, where all tasks are started at the same time (otherwise the system is said to be *asynchronous*, or *offset free* if the offsets—i.e., the times at which the first requests occur—are not fixed by the problem but may be chosen by the scheduler [4, 3]); *implicit deadline* systems, where each deadline coincides with the period (i.e., each request must simply be completed before the next request of the same task occurs); *constrained deadline* systems, where the deadlines are not greater than the periods and *arbitrary deadline* if no constraint exists between the deadline and the period.

Synchronous implicit deadline systems with static schedulers are particularly popular in the literature and among practitioners, not because of their generality or efficiency, but simply because in this case, an easy-to-implement optimal scheduler is known (the rate monotonic scheduler, RMS for short, which gives higher priorities to lower periods [10]) and there is a very simple and fast sufficient feasibility test based on the utilization factor [10]. A scheduler is optimal if there is a scheduling rule without deadline miss, the system is also feasible with these scheduling rules. If we want to be a bit more liberal and allow synchronous constrained deadline systems, we still know an optimal scheduler (the deadline monotonic scheduler, DMS for short, which gives higher priorities to lower deadline delays [9]) but we now need to compute explicitly the first response time of each task instance, for instance by Tindell's iteration[1], to determine the feasibility of the system.

If we now turn to asynchronous or arbitrary deadline systems, the situation is less favorable since DMS is not an optimal scheduling rule anymore; Audlsey gives an optimal static priority assignment which considers $\mathcal{O}(n^2)$ distinct priority assignment (see [2]). A periodic real-time system repeats its task arrival pattern after an interval called hyper-period (this notion shall be expressed with more details in the next section). Moreover, to check the feasibility of the system, it is necessary to determine all the response times on a period which may go up to slightly more than a full hyper-period[4].

Better scheduling performances may be obtained with dynamic strategies (since

static ones may be considered as a special case anyway), and a first curious feature is that one knows two simple optimal scheduling rules: the *deadline driven scheduler* (DDS for short, also known in the literature as the *earliest deadline scheduler* or the *earliest deadline first*) which attributes the CPU to the active request with the least deadline [10], and the *least laxity first* scheduler (LLF for short) which attributes the CPU to the active request with the least difference between the delay before the deadline and the CPU time still needed to complete the request [11, 12, 7]. Both are optimal, whatever the offsets and the deadline delays are. For implicit deadline systems there is a very simple feasibility check: a system is feasible if and only if the utilization factor is not greater than 1. Unfortunately, for constrained or arbitrary deadline systems, being synchronous or not, their feasibility with dynamic schedulers is generally intrinsically exponential in terms of the number of tasks. Again, it may be necessary to determine all the response times on a period which may go up to twice the hyper-period [8].

The problem that appears here is that the hyper-period grows exponentially (as a function of the biggest period and with the number of tasks). It is why it is important to try to limit the hyper-period of real-time systems.

The paper is structured as follows: section 2 presents our model of computation and our assumptions; section 3 presents a pseudo-random algorithm which generates a set of periodic real-time tasks with a "limited" hyper-period; section 4 proposes a first application of our task set generator: a comparison of the number of preemptions induced by the "popular" real-time scheduling rules. Section 5 summarizes the situation.

2 Computational Model and Assumptions

We shall consider in this paper the problem of scheduling a set of periodic real-time tasks. The set is composed of n periodic tasks $\tau_1, \ldots, \tau_n$. Each periodic task τ_i is characterized by the quadruple (T_i, D_i, C_i, O_i) with $0 < C_i \leq D_i$, $C_i \leq T_i$ and $O_i \geq 0$, i.e., by a period T_i, a hard deadline delay D_i, an execution time C_i, and an offset O_i, giving the instant of the first request. The requests of τ_i are separated by T_i time units and occur at time $O_i + (k-1)T_i$ $(k = 1, \ldots)$. The execution time required for each request is C_i time units; C_i can be considered as the worst-case execution time for a request of τ_i. The execution of the k^{th} request of task τ_i, which occurs at time $O_i + (k-1)T_i$, must finish before or at time $O_i + (k-1)T_i + D_i$, which is its deadline; the deadline failure is fatal for the system: the deadlines are considered to be hard. All timing characteristics of the tasks in our model of computation are assumed to be natural integers.

For instance consider the following task set: $\{\tau_1 = \{C_1 = 4, T_1 = 10, D_1 = 8, O_1 = 2\}, \tau_2 = \{C_3 = 5, T_2 = 15, D_2 = 9, O_1 = 0\}\}$. Figure 1 gives the schedule of the (static) deadline monotonic priority rule; Figure 2 gives the schedules of the (dynamic) deadline driven scheduler and Figure 3 gives the schedules of the (dynamic) least laxity first algorithm. In those figures, $\downarrow$ represents a task request,

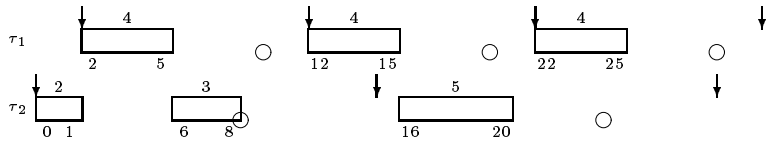


Figure 1: Deadline Monotonic Scheduling; the schedule repeats from time 30.

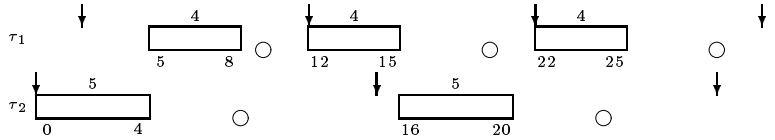


Figure 2: Deadline Driven Scheduler; the schedule repeats from time 30.

○ a deadline and  an execution of c units between time units a and b, included; in the special case where $a = b$ we omit b in our representation.

From these schedules interesting and preliminary remarks follow. All the schedules are feasible. Both deadline driven scheduler and the least laxity first algorithm are optimal but the number of preemptions are rather different and this number is considerably greater with the least laxity first rule; for instance at time $t = 2$ the laxities are identical and we observe a trashing situation until time $t = 8$, when τ_2 ends its execution. It may be noticed that we shall in section 4 study in detail the number of preemptions induced by various "popular" real-time scheduling rules. The dynamic strategies are ambiguous in some circumstances, e.g., with the least laxity first algorithm and at time $t = 2$ the laxity of both the requests are identical; we give first the CPU to the request of τ_1. The same phenomenon exists for the deadline driven scheduler since deadlines of active requests can coincide. In order to avoid ambiguities, we shall give the CPU to the task with the highest priority and the smallest index. With the previous assumption, it is easy to see that all the schedules repeat from time $t = 30$. In addition we have assumed that the switching times (including scheduling) may be neglected and that the tasks are independent. In the following, $P = \text{lcm}\{T_i \mid i = 1, \ldots, n\}$ denotes the hyper-period of the system and $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$ denotes the utilization factor or the periodic load of the system, i.e., the (long term) fraction of the processor time spent in the execution of the task set (if feasible).

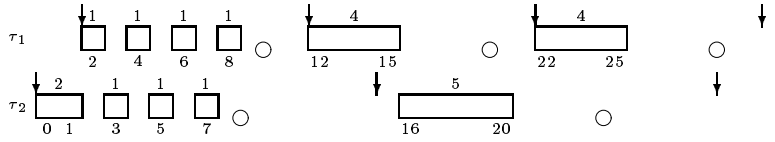


Figure 3: Least laxity first scheduling; the schedule repeats from time 30.

3 Task Set Generation

Except for very special cases, the feasibility check of asynchronous systems consists in simulating its evolution in some feasibility interval, i.e. a finite interval such that it is sure that no deadline will ever be missed iff, when we only keep the requests made in this interval, all deadlines for them in this interval are met. For instance, a feasibility interval for an asynchronous constrained deadline system using a static scheduler is $[0, S_n + P[$, where $S_1 = O_1$, $S_i = \max\{O_i, O_i + \lceil \frac{S_{i-1}-O_i}{T_i} \rceil T_i\}$ $(i = 2, \ldots, n)$; moreover the schedule is periodic from time S_n with a period of P (see [4] for details). A feasibility interval for an asynchronous constrained deadline system using the deadline driven scheduler or the least laxity first algorithm is $[O^{\max}, O^{\max} + 2P[$ where $O^{\max}$ is the maximal offset (see Leung and Merill [8] for details); moreover the schedule is periodic from $O^{\max} + P$, with a period of P. This duration could be laid only to $t_c + 1 + P$ with t_c the date of the last acyclic idle slot, calculated on the fly (with $-1 \leq t_c < O^{\max} + P$). Since for the most cases, $t_c = -1$ the simulation duration would be reduced in most cases to $O^{\max} + P$ (see [5] for more details). The very same interval remains a feasibility interval for arbitrary deadline systems for any static schedulers, for the deadline driven scheduler and for the least laxity first algorithm under the condition that $U \leq 1$ (see [3] for details). It follows that a simulation of the periodic part of a schedule, or a feasibility interval is proportional to P, the hyper-period of the system. Hence, statistical analyses of the schedules of periodic tasks must consider an interval proportional to P. For this reason it seems interesting to dispose of an algorithm to randomly choose the periodic tasks in the more general way possible but with the restriction that the hyper-period remains reasonable, in order to perform those statistical analyses in a reasonable time. This is not too unrealistic, since the user may often slightly act on the periods of the various tasks in order to get such a feature. The problem is of course that P may grow exponentially with the number n of tasks. To see this element, we will first demonstrate that the biggest hyper-period we can generate with task sets characterized by periods less or equal to a given number (m) grows exponentially with this number, as exhibited by the following lemma. Next, we will show that the hyper-period grows exponentially with the task number n, up to reach very quickly the limit defined by the biggest period m.

Lemma 1 *Let $u_1, \ldots, u_n$ be n natural integers such that $0 < u_i \leq m$ $(1 \leq i \leq n)$. We have that*

$$P \leq \lambda(m) \tag{1}$$

where $\lambda(m) = \prod_{1 \leq i \leq \pi(m)} p_i^{\lfloor \log_{p_i} m \rfloor}$, p_i denotes the i^{th} prime number and $\pi(m)$ is the prime counting function (i.e., $\#\{i \mid i \text{ is prime} \wedge i \leq m\}$). Moreover, $\lambda(m) \sim e^m$.

Proof. According to the "fundamental theorem of arithmetic" [6], each positive integer u can be expressed in the form

$$u = 2^{u_2} \times 3^{u_3} \times 5^{u_5} \times 7^{u_7} \times \cdots = \prod_{\{p \,|\, p \text{ is prime}\}} p^{u_p}$$

where the exponents $u_2, u_3, \ldots$ are uniquely determined non negative integers and where all but a finite number of exponents are null. If we have 2 positive integers u and v, and if both of them have been canonically factored into primes we have

$$\text{lcm}\{u, v\} = \prod_{\{p \,|\, p \text{ is prime}\}} p^{\max\{u_p, v_p\}}$$

and for n integers we have:

$$\text{lcm}\{u_1, u_2, \ldots, u_n\} = \prod_{\{p \,|\, p \text{ is prime}\}} p^{\max\{u_{1,p_i}, u_{2,p_i}, \cdots, u_{n,p_i}\}}$$

$(u_i = \prod_p p^{u_{i,p}})$. Since $0 \leq u_i \leq m$, and from the definition of the function λ it follows that:

$$P = \text{lcm}\{u_1, \ldots, u_n\} = \prod_{1 \leq i \leq \pi(m)} p_i^{\max\{u_{1,p_i}, \cdots, u_{n,p_i}\}}$$

Each factor of the previous equation (say p_i^x) is less than m consequently $x \leq \log_{p_i} m$. We can conclude that

$$P \leq \prod_{1 \leq i \leq \pi(i)} p_i^{\lfloor log_{p_i} m \rfloor} = \lambda(m).$$

Next, we can show that

$$\sqrt{m} < p_i^{\lfloor \log_{p_i} m \rfloor} \leq m$$

We know an approximation for $\pi(n)$ (results demonstrated in 1896, independently by J. Hadamard and C.-J. de la Valée Poussin):

$$\pi(n) \sim \frac{n}{\ln n}$$

We have

$$m^{\frac{m}{2 \ln m}} < \lambda(m) \leq m^{\frac{m}{\ln m}}$$

and

$$\lim_{m \to \infty} m^{\frac{m}{\ln m}} = e^m$$

Thus

$$\lambda(m) \sim e^m$$

■

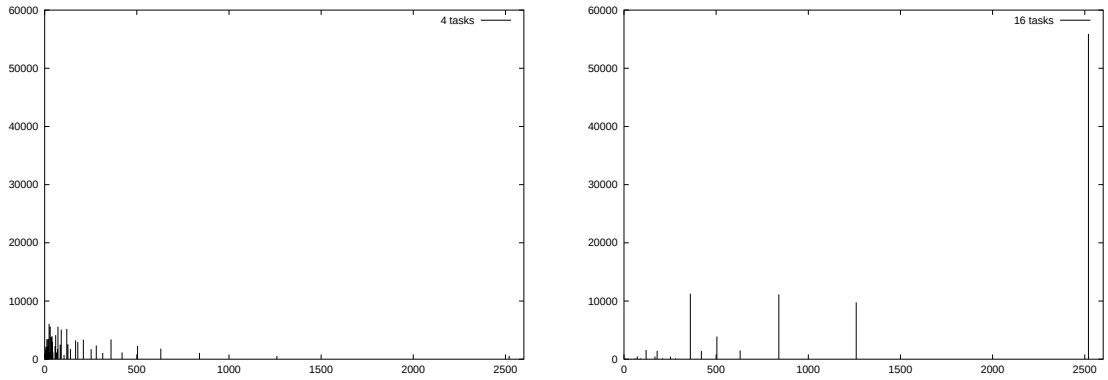
We have shown that $\lambda(m)$ grows exponentially with m, the biggest period we have in our task set. Now, we will illustrate the growing of the hyper-period as a function of n, the number of tasks.

To do this, we will generate an important number of random task sets with periods chosen in $[1, 10]$. We first generate random task sets (100,000) with 4 tasks, then with 8, 16, 32, 64 and 128 tasks. The Table 1 shows the main characteristics of the hyper-period for these task sets.

	4 tasks	8 tasks	16 tasks	32 tasks	64 tasks	128 tasks
Mean	142	682	1709	2397	2517	2520
Minimum	2	4	12	72	360	2520
Maximum	2520	2520	2520	2520	2520	2520
Number of times we have the maximum	574	12955	55907	92848	99843	100000

Table 1: Characterization of the hyper-period for 100,000 random task sets.

Figure 4 shows the distribution of the hyper-period for task sets characterized by (a) 4 tasks and (b) 8 tasks. We can easily see that the probability of apparition of the biggest hyper-period (2520 when periods are in $[1, 10]$) grows exponentially with the number of tasks (see also the last row of the Table 1).



(a) (b)

Figure 4: Distribution of the hyper-period for (a) 4 tasks and (b) 8 tasks for 100,000 task sets.

For a big n, the probability we reach $\lambda(m)$, the biggest possible hyper-period with periods less or equal to m, becomes very important (93% when we have 64 tasks in the previous simulations).

Our main goal is to generate with a (pseudo-)random algorithm n periods $(T_1, \ldots, T_n)$ in such a way that their least common multiple remains bounded, with the idea in mind that users can act slightly on periods (increase or decrease them) to avoid this exponential phenomenon. For this reason we shall choose our periods from a predetermined set of prime numbers $q_1, \ldots, q_r$ and (pseudo-)random exponents $e_1, \ldots, e_r$ and choose T_i as follows : $T_i = q_1^{e_1} \times q_2^{e_2} \times \cdots \times q_r^{e_r}$.

Our algorithm uses a matrix ($\mathcal{M}$) representing all the primes we have selected and their respective exponents. We can notice that in fact, our algorithm can use a

set of vectors, but we shall use a "matrix" in the following for simplicity. Each prime number (say q_i), corresponds to a line in the matrix (the i^{th}) or the i^{th} sub-vector. These numbers represent a probabilistic distribution for the powers of q_i. In fact, for each power found in this line, we have a probability of choice proportional to its occurrence number in the list.

Now, we can generate an exponent for a prime number q_i simply in choosing a $\mathcal{M}_{i,j}$, where j is a pseudo-random number chosen uniformly in the interval $[1, |\mathcal{M}_i|]$, where $|\mathcal{M}_i|$ denotes the number of powers of the i^{th} prime number (indices are here assumed to start from 1). The following $\mathcal{M}$ represents the pattern we can find in this matrix.

$$\mathcal{M} = \begin{pmatrix} q_1^0 & q_1^1 & q_1^1 & q_1^2 & q_1^2 & q_1^2 & q_1^3 & q_1^4 & q_1^4 \\ q_2^0 & q_2^1 & q_2^1 & q_2^2 & q_2^2 & q_2^2 & q_2^3 & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & & & & \\ q_r^0 & q_r^0 & q_r^0 & q_r^1 & q_r^1 & & & & \end{pmatrix}$$

The number of rows for $\mathcal{M}$ is equal to r, the number of primes and the number of columns is equal to the granularity of the distribution.

We can see a simple practical example (Example 2) of such choices. In this example, there are 5 prime numbers (2,3,5,7 and 11) and their greatest exponent varies between 1 and 4.

Example 2

$$\mathcal{M} = \begin{pmatrix} 1 & 2 & 2 & 4 & 4 & 4 & 8 & 16 & 16 \\ 1 & 3 & 3 & 9 & 9 & 9 & 27 & & \\ 1 & 5 & 5 & 25 & 25 & 25 & & & \\ 1 & 1 & 7 & 7 & 7 & 49 & & & \\ 1 & 1 & 1 & 11 & 11 & & & & \end{pmatrix}$$

■

Algorithm 1 is thus used to generate periods such that the hyper-period is limited (we shall show later how this hyper-period is limited). Notice that we use in this algorithm the following functions: $Rand(\alpha_1, \alpha_2)$ which returns a pseudo-random real number uniformly distributed in the interval $[\alpha_1, \alpha_2]$ and $Round(x)$ which returns the closest integer to x.

Algorithm 1 Method used to determine a period.

Parameters: the matrix $\mathcal{M}$;
$period \leftarrow 1$;
for each line i of the matrix $\mathcal{M}$ **do**
  $p \leftarrow Round(Rand(1, |\mathcal{M}_i|))$;
  $period \leftarrow period \times \mathcal{M}_{i,p}$;
end for

This algorithm generates periods between $\prod_{1 \leq i \leq r} \min\{\mathcal{M}_{i,j},\ 1 \leq j \leq |\mathcal{M}_i|\}$ and $\prod_{1 \leq i \leq r} \max\{\mathcal{M}_{i,j},\ 1 \leq j \leq |\mathcal{M}_i|\}$. With the Example 2, periods are in the interval $[1, 5821200]$. It may be noticed that with this algorithm the upper bound for the hyper-period is also equal to the upper bound for the periods ($\prod_{1 \leq i \leq r} \max\{\mathcal{M}_{i,j},\ 1 \leq j \leq |\mathcal{M}_i|\}$); this can be shown with a similar reasoning than the one used in the proof of Lemma 1.

We can control the size of the greatest hyper-period (which may be equal to the size of the biggest period we generate), with an adequate choice of prime numbers with their respective powers.

We shall now consider the second algorithm (Algorithm 2) which determines the other task characteristics (i.e., C_i's, O_i's, D_i's).

Algorithm 2 Method used to generate task sets.

Parameters: u_1, u_2, d_1, d_2, o_1, o_2, $\mathcal{M}$, $\mathcal{V}$, U and n;
$current\_load \leftarrow 0$;
$task\_set \leftarrow \phi$;
$i \leftarrow 0$;
while $current\_load < U$ **and** $i < n$ **do**
  $i \leftarrow i + 1$;
  Generate a period T_i according to the Algorithm 1, using $\mathcal{M}$ and $\mathcal{V}$;
  $C_i \leftarrow \max(1, Round(Rand(u_1, u_2) \times T_i))$;
  $O_i \leftarrow Round(Rand(o_1, o_2) \times T_i)$;
  $D_i \leftarrow Round((T_i - C_i) \times Rand(d_1, d_2)) + C_i$;
  if $current\_load + \frac{C_i}{T_i} \leq 1$ **then**
    $current\_load = current\_load + \frac{C_i}{T_i}$;
    $task\_set = task\_set \cup \{(T_i, D_i, C_i, O_i)\}$;
  end if
end while
$n \leftarrow i$;

Notice that the parameters u_1, u_2, d_1, d_2, o_1 and o_2 are real numbers.

u_1 and u_2 are such that $0 \leq u_1 \leq u_2 \leq 1$ and are used to choose a computational time less than T_i but proportional to T_i.

The parameters o_1 and o_2 control the asynchronous aspect of the system and are such that $0 \leq o_1 \leq o_2$. It may be noticed that $o_1 = o_2 = 0$ corresponds to synchronous case and from [4] we know that without loss of generality (with respect to the feasibility of asynchronous systems) we can restrict to $o_1 \geq \frac{1}{T_i}$ and $o_2 < 1$ for asynchronous cases. This algorithm constructs a task set composed of n tasks with a periodic load near U. The exact periodic load this algorithm generates is equal to $current\_load$. We can also notice than n is adjusted at the end of the algorithm if we reach the periodic load before we generate the desired number of tasks.

The parameters d_1 and d_2 control the laxity of the tasks and are such that $0 \leq d_1 \leq d_2$. Notice that $d_1 = d_2 = 1$ corresponds to implicit deadlines systems,

$d_2 = 1$ corresponds to constrained deadline systems, otherwise we have arbitrary deadline systems.

It may be noticed that with our algorithm, all the values in the interval $[1, \prod_{1 \leq i \leq r} \max\{\mathcal{M}_{i,j},\ 1 \leq j \leq |\mathcal{M}_i|\}]$ for the periods cannot be generated. In fact, we can only generate $\prod_{1 \leq i \leq r} \#\{\mathcal{M}_{i,j} \mid 1 \leq j \leq |\mathcal{M}_i|\}$ different periods, out of $\prod_{1 \leq i \leq r} \mathcal{M}_{i,|\mathcal{M}_i|}$ possible values (notice that the # operator gives the number of different elements of a given set). This number of periods we can generate represents only all the different possibilities our algorithm can choose with a matrix $\mathcal{M}$. In Example 2, for instance, only 360 periods are possible among the $5,821,200$ numbers in the range.

Here are some guidelines to choose adequately the primes and their powers. If we need many different periods, we should choose primes as little as possible $(2, 3, 5, \cdots)$ and take powers as function as the biggest hyper-period we can tolerate. If we will less different periods, but periods that are really far one to the others, primes should be choosen bigger. The powers are equally choosen such that we do not exceed the biggest hyper-period we can tolerate. We can easily choose the minimum for the periods with the choice of the least powers for each primes. Finally, we can also influence the average for the periods with the distribution of powers: if we need a system with a little average for periods, we should construct $\mathcal{M}$ such that little powers have a great frequency facing the frequency of the big powers.

4 Application

In this section we shall see a first application of our algorithm. Our goal is to compare the actual number of preemptions induced by some popular scheduling algorithms for hard real-time periodic tasks. First, we shall consider the periodic task sets generated by our algorithm.

Task Characteristics

In our simulations we used the following parameters:

$$\mathcal{M} = \begin{pmatrix} 1 & 1 & 1 & 1 & 4 & 4 & 4 & 8 & \\ 1 & 3 & 3 & 3 & 3 & 9 & 9 & 27 & 27 \\ 1 & 5 & & & & & & & \\ 1 & 7 & 7 & 7 & & & & & \\ 1 & 1 & 13 & & & & & & \\ 1 & 1 & 1 & 17 & 17 & & & & \\ 1 & 1 & 1 & 1 & 19 & & & & \end{pmatrix}$$

The deadlines are taken in the interval $[C_i + 1, T_i]$, with $d_1 = 0$ and $d_2 = 1$ (i.e., we study constrained deadline systems). We have also chosen $o_1 = 1/T_i$ and $o_2 = 1$ (i.e., we study asynchronous system). Regarding the computation times we have chosen $u_1 = 0$ and $u_2 = \frac{1}{25}$.

According to Algorithm 1 we have that $P \leq \prod_{1 \leq i \leq r} \mathcal{M}_{i,|\mathcal{M}_i|} = 31,744,440$ and that the periods are included in the interval $[1, 31744440]$. Figure 5 (a) shows the actual distribution of the hyper-period we got for 10^3 task sets generated by Algorithm 1. We can observe that the greatest hyper-period (the last impulse) appears really often (more than 710 times on 1,000 task sets). The other value are mainly distributed in the first interval (between 1 and slightly more than 1.6×10^6).

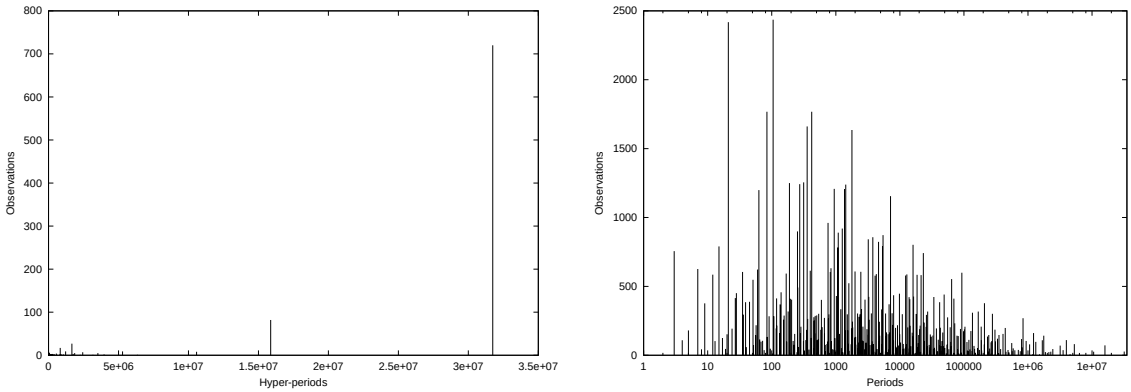
And it is why it is important to choose judiciously $\mathcal{M}$ to avoid to generate task set we can not simulate in a reasonable time. The average hyper-period is in fact equal to 24,382,211, showing the absorbing power of higher values in such distributions.

Figure 5 (b) shows the actual distribution of the task periods (notice that the X-axis is in logarithmic scale); the average is $\simeq 10^5$; we observe that few periods are very frequent, and when the periods grow, there are less and less observations and these observations are less and less frequent.

Figure 6 (a) presents the distribution of the number of tasks in a task set, which varies between 1 and 55, with an average around 19. Figure 6 (b) shows the distribution of the number of tasks in a task set in function of the utilization factor; according to Algorithm 2 we expected this phenomenon, where the number of tasks is proportional to the utilization factor.

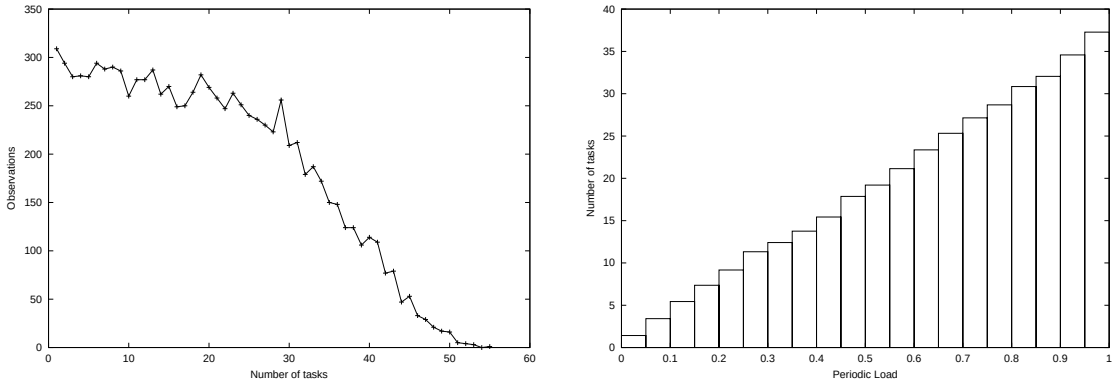
Number of preemptions

We shall now present our study about the number of preemptions. Preemption means in our context the fact that the system temporarily stops a task (before its completion) to switch to another task (with a higher priority). Scheduling theory generally assumes that the preemption time may be neglected. In fact, a preemption implies an additional context switching for the operating system and such operations are relatively heavy for the processor. For this reason it seems interesting to have a good idea of the actual number of preemptions induced by popular scheduling algorithms.



(a) (b)

Figure 5: Distribution of Hyper-periods (a) and distribution of periods (b).



(a) (b)

Figure 6: The distribution of number of tasks (a) and the number of tasks as a function of the periodic load (b).

As we have already seen in Figures 1, 2 and 3, the algorithm used to schedule real-time tasks has a definite impact on the number of preemptions. On these examples, LLF generates 6 preemptions, while EDF does not generate any preemption and DMS generates only one preemption.

Our study will confirm this phenomenon on a large number of randomly chosen task sets (the task sets introduced in the previous section).

Since we consider asynchronous constrained deadline systems, we simulate each task set in the interval $[0, O^{max} + 2P[$ for the DMS, EDF and the LLF scheduling rules. Moreover we have only considered task sets which are schedulable with the 3 algorithms (i.e., the sets that do not miss any deadline using the DMS during all the time of the simulation; about 74% of all task sets).

Figure 7 shows the number of preemptions for 2 algorithms in function of the periodic load: DMS and LLF. It is not useful to represent EDF because results obtained for EDF and DMS are really imperceptible.

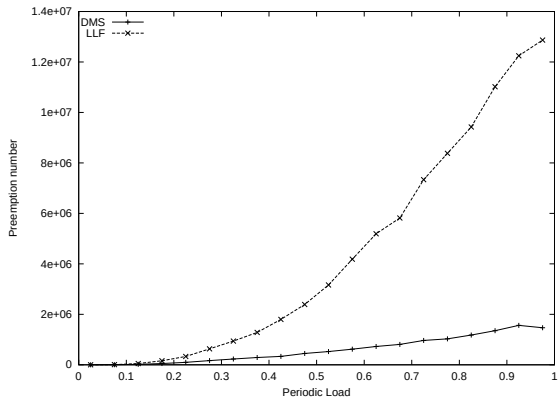


Figure 7: The preemption number vs the periodic load for DMS and LLF.

The trends observed during the first examples of scheduling (Figure 1, 2 and 3)

seems to be the real trends of theses scheduling algorithms: LLF generates about 6 times more preemptions on average than the two other algorithms (in fact, this policy generated between between 3 times and 9 times more preemptions than the two others policies for our task sets). We can observe this phenomenon on the Figure 8.

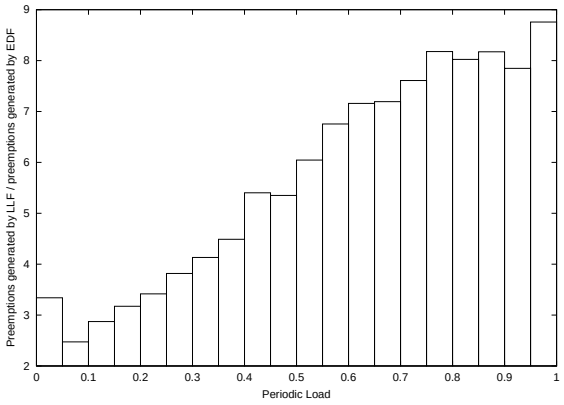


Figure 8: The preemptions generated by LLF divided by the preemptions generated by EDF, as a function of the periodic load.

The second important observation leads to the conclusion that EDF and DMS are really close to each other according to the number of preemptions (we cannot distinguish these algorithms on the Figure 7); the results are similar when we consider the number of preemptions as a function of the number of tasks, and it is relatively easy to understand why: the task number is proportional to the periodic load for all the task set we generate with our algorithm. The average preemption number of DMS and EDF are equal to 386,969 and 385,806 respectively. The standard deviation are also really near one to the other : $5,055,478$ for DMS and $5,042,753$ for EDF. The maximum of these two distributions are equal to $5,222,925$ and $5,222,761$ respectively for DMS and EDF.

When we consider the case where DMS generates less preemptions than EDF, we found 98 cases. Although, there are 192 cases where EDF generates less preemptions (on 649 cases). Figure 9 presents for each considered periodic load the difference between the preemption number of DMS and the preemption number of EDF. This Figure presents also the absolute value of this difference.

All these elements leads to the conclusion that EDF generates less preemptions than DMS. When the load becomes important (> 0.95), this trend seems to be inverted.

Although, the difference between the two distributions is nearly insignificant.

In the next part of this section, we will study the influence of the granularity for the task sets on the preemption number. The granularity of a task set can be defined as the $\gcd(\gcd(T_1, D_1, C_1, O_1), \cdots, \gcd(T_n, D_n, C_n, O_n))$ and can be adapted by real-time designer in function of the delay between clock ticks. Two of the three algorithms studied have an interesting property : the preemption number is

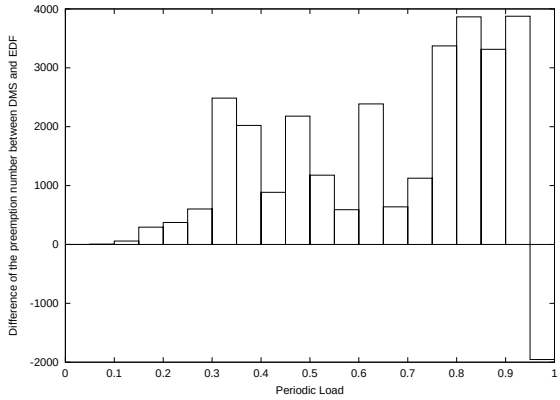


Figure 9: Difference between the preemption number of DMS and EDF vs the periodic load.

independent of the granularity of the system; this is the case for DMS and EDF. But granularity has an influence on the preemption number for LLF. We can observe this phenomenon on the Figure 10.

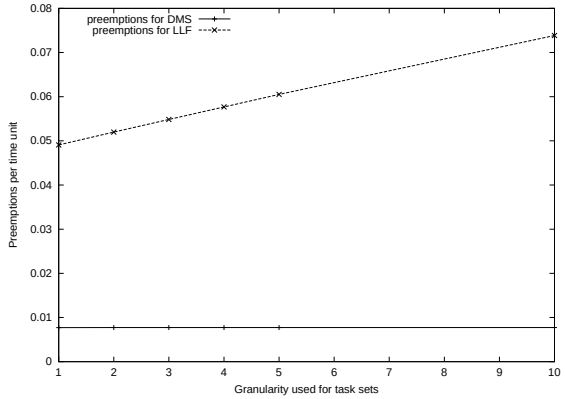


Figure 10: Influence of the granularity for DMS and LLF.

We see that the preemption number is directly proportional to the granularity of the tasks for LLF. This can lead to important system degradations when the frequency of the internal clock is very small or when the implementer has no other choice than to choose an important granularity.

5 Conclusion

In this paper, we have shown that the hyper-period of a real-time system grows exponentially with the greatest possible period, if care is not taken to control this phenomenon. This leads to the conclusion that it is relatively difficult to generate task sets with large periods which can be simulated during its whole periodic part. A part of the contribution of this paper is the presentation of an algorithm that

generates task sets with a limited hyper-period (we can control the maximum hyper-period of all the task sets we generate) and such that it generates big periods (periods can be as great as the hyper-period).

Next we have presented the characteristics of 1,000 task sets used (so generated) to study the behavior — facing the number of preemptions — of the most popular scheduling algorithms described in the literature : Deadline Monotonic Scheduling, Earliest Deadline First and Least Laxity First.

It appeared that the LLF algorithm generates much more preemptions that the other two algorithms (about 6 times more, on the average). A second useful observation was that the preemption numbers of the DMS and EDF algorithms are very similar. A finer study allowed however to conclude that EDF generates on the average slightly less preemptions than DMS.

We have also shown that the granularity of the system has an influence on the preemption number for LLF; the preemption number for this policy is directly proportional to the granularity.

Acknowledgment

The authors gratefully acknowledge the significant contributions of R. Devillers.

References

[1] Audsley, A. N., Burns, A., Richardson, M., and Tindell, K. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal* (1993), 284–292.

[2] Audsley, N. C. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Tech. rep., University of York, England, 1991.

[3] Goossens, J. *Scheduling of Hard Real-Time Periodic Systems with Various Kinds of Deadline and Offset Constraints.* PhD thesis, Université Libre de Bruxelles, Belgium, 1999.

[4] Goossens, J., and Devillers, R. The non-optimality of the monotonic priority assignments for hard real-time offset free systems. *Real-Time Systems 13*, 2 (September 1997), 107–126.

[5] Grolleau, E., and Choquet-Geniet, A. Cyclicité des ordonnancements de tâches périodiques différées. In *Conférence RTS'2001* (Paris, 2000), pp. 216–229.

[6] Knuth, D. E. *The Art of Computer Programming, Vol 2, Seminumerical Algorithms*, 3 ed. Addison-Wesley, Reading, USA, 1998.

[7] Leung, J. Y.-T. An new algorithm for scheduling periodic, real-time tasks. *Algorithmica 4* (1989), 209–219.

[8] Leung, J. Y.-T., and Merrill, M. A note on preemptive scheduling of periodic, real-time tasks. *Information processing letters 11*, 3 (November 1980), 115–118.

[9] Leung, J. Y.-T., and Whitehead, J. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation 2* (1982), 237–250.

[10] Liu, C. L., and Layland, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery 20*, 1 (January 1973), 46–61.

[11] Mok, A., and Dertouzos, M. Multiprocessor scheduling in a hard real-time environment. In *Proceedings of the Seventh Texas Conference on Computing Systems* (1978).

[12] Mok, A. K.-L. *Fundamental Design Problems of Distributed Systems for The Hard-Real-Time Environment.* PhD thesis, Massachusetts Institute of Technology, 1983.