Worst case response time versus worst case offset configuration using the deadline driven scheduler

Joël Goossens
Université Libre de Bruxelles
Département d’Informatique
Bld. du Triomphe C.P. 212, B-1050 Brussels, Belgium.

Abstract
We study the worst case response in the scheduling of hard real-time periodic task sets with the deadline driven scheduler. We show that the worst case response times do not always occur in the synchronous schedule; even worse, the worst response times for the various tasks do not necessarily occur in the same asynchronous schedule.

Index terms: real-time system, hard real-time scheduling, synchronous system, asynchronous system, deadline driven scheduling, response time.

1 Introduction
In real-time systems, the processes must not only deliver a correct answer, but they must also do so in due time, generally expressed in term of a deadline. This leads to interesting
problems, and even in apparently simple cases, like systems composed of independent periodic tasks on a single preemptive processor, the behavior of the schedules may be surprisingly complex. The main scheduling problem is to guarantee that all (hard) deadlines are always met.

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 [9, 5]); 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 systems, where no constraint exists between the deadline and the period (notice that if the deadline is greater than the period, many requests of a single task may be simultaneously active, even if the system is feasible, i.e., all deadlines are met).

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 [14]) and there is a very simple and fast sufficient feasibility test based on the utilization factor [14]. 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 [13]) but there is no known simple and fast schedulability test; however, in this case, for each task, the response time (i.e., the time between the arrival of the request and the completion of its processing) of the first request is the worst one (among all requests of this task in all asynchronous schedules; this holds for any static scheduling rule [14, 5], not only for DMS or RMS), and Audsley [1] and Tindell [20] have shown how to compute it reasonably fast without performing a true simulation, which leads to a necessary and sufficient feasibility test. If we now turn to synchronous arbitrary deadline systems, the situation is less favorable since DMS is not an optimal scheduling rule anymore; Audsley gives an optimal static priority assignment which considers $O(n^2)$ distinct priority assignments (see [1] for details); and the first request of each task has not necessarily the worst response time [11], so that it may be necessary to check subsequent response times (by simulation [7] or by direct computations [4]) till the first idle period or idle point [5, 10]. For asynchronous or offset free systems, the situation is similar: Audsley’s algorithm remains optimal. For offset free systems, we do not know an optimal offset allocation (apart from checking all non-equivalent possibilities [5, 1] and computing many response times [5]).

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 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 nearest deadline [14], 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 \cite{15, 16, 12}. Both are optimal in the sense that, whatever the offsets and the deadlines, if there is a scheduling rule without deadline miss, the system is also feasible with these scheduling rules. While DDS and LLF are equivalent with respect to the feasibility of the systems, they generally behave quite differently; for instance DDS yields less preemptions than LLF (which may be important if, contrary to what is usually assumed in those theories, preemptions are not negligible for the performances of the system); this is due to the fact that with DDS the relative priorities of two (active) requests do not change with time, while they do with LLF; that is why, in the following, we shall only consider the deadline driven scheduler. 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 deadline systems, being synchronous or not, their feasibility with dynamic schedulers is generally intrinsically exponential in terms of the number of tasks. For instance, it is no longer true that for each task the first request in the synchronous case has the worst response time, so that it is necessary to determine more response times to make sure that the system is feasible or not. However the worst case response time remains very interesting in the dynamic case; it is useful not only for the analysis of single processor systems, but also for the analysis of distributed real-time systems, e.g., in the holistic schedulability analysis by Tindell and Clark \cite{19}. Spuri and colleagues \cite{17, 18} have shown how to compute for arbitrary deadline systems the worst case response times with the deadline driven scheduler and outlined the fact that this worst case does not always occur in the initial period of the synchronous schedule. In this paper we shall also exhibit two interesting properties concerning the worst case response time, not considered in the work of Spuri; indeed in the dynamic case, the situation is very different from the static one where the synchronous case is the worst case regarding the feasibility of the system and also provides the worst case response times in the beginning of the synchronous schedule. In the dynamic case the synchronous case remains the worst case situation regarding the feasibility of the system, but the worst case response times do not always occur in the synchronous schedule; even worse, we shall show that the worst response times for the various tasks of the system do not necessarily occur in the same (asynchronous) schedule. We shall exhibit this phenomenon and comment this feature which may seem counter-intuitive.

The rest of the paper is organized as follows: we shall first detail our context and specify our model of computation and our assumptions; we shall then give preliminary results concerning the schedulability of our systems; and finally we consider the worst case response time notion and exhibit the two claimed interesting properties.

## 2 Computational model and assumptions

We shall consider in this paper the problem of scheduling a set of periodic hard real-time tasks. The set is composed of \( n \) periodic tasks \( \tau_1, \ldots, \tau_n \). Each periodic task \( \tau_i \) 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 $\tau_i$ are separated by $T_i$ time units and occur at time $O_i + (k - 1)T_i$ ($k = 1, 2, \ldots$). 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 $\tau_i$. The execution of the $k$th request of task $\tau_i$ (which we shall denote $\delta_k^i$), which occurs at time $O_i + (k - 1)T_i$, must finish before or at time $O_i + (k - 1)T_i + D_i$; 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. We study the problem of scheduling a task set for a single processor system with DDS. In addition we shall assume that the switching times (including scheduling, hence the special interest devoted to the less time consuming dynamic strategies) may be neglected and that the tasks are independent.

### 3 Schedulability of the task set

First, asynchronous and arbitrary deadline systems, which are the most general systems considered in this paper, and we introduce a very general property based on the quantities $\eta_k(t, t')$ which denote, for $t \leq t'$, the number of values $k \in \mathbb{N}$ such that $t \leq O_i + k \cdot T_i$ and $O_i + k \cdot T_i + D_i \leq t'$.

That is, $\eta_k(t, t')$ is the number of requests of task $\tau_i$ which occur in the interval $[t, t')$ with a deadline less than or equal to $t'$. Any feasible scheduling algorithm must give at least $\eta_k(t, t') \cdot C_i$ CPU time units to $\tau_i$ in this interval; this is a necessary condition for the schedulability of the system in this interval, whatever the scheduling rule. For a deadline driven scheduler, it occurs that the condition is in some sense also sufficient.

**Lemma 1 ([10])** An asynchronous and arbitrary deadline system $R$ is feasible up to time $t$ with a deadline driven scheduler iff $\sum_{k=1}^{n} \eta_k(t', t'') \cdot C_i \leq t'' - t'$ for all $0 \leq t' < t'' \leq t$.

This slightly generalizes a similar property first proved in [2, 3]; we shall later see the interest of this property. Synchronous systems (i.e., when $O_i = 0 \ \forall i$) constitute the worst possible case in terms of schedulability:

**Theorem 2 ([14])** Let $S = \{\tau_i = (C_i, D_i, T_i) \mid i = 1, \ldots, n\}$ with arbitrary deadlines. If $S$ is schedulable (i.e., all deadlines are met) in the synchronous case using the deadline driven scheduler, this is also the case in all asynchronous situations.

Nevertheless, synchronous systems are very popular in the literature, probably due to the fact that they occur quite often in the applications, but also from the fact that there are much smaller feasibility intervals (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) in the synchronous case than in the asynchronous one. Liu and Layland [14] noticed that, for the synchronous case, it is not necessary to look at a full hyper-period (i.e., the interval $[0, P)$ where $P \overset{\text{def}}{=} \text{lcm}\{T_i \mid 1 \leq i \leq n\}$) to detect deadline failures: the interval $[0, L')$, where $L'$ is the position of the first idle slot, is a
feasibility interval. If \( U = \sum_{i=1}^{n} \frac{C_i}{T_i} < 1 \), we always have \( I' < P \); but if \( U = 1 \) there is no idle slot after the system started at time 0. In a previous work [10] we have improved this feasibility test by introducing the notion of an idle point.

**Definition 3** \( x \in \mathbb{N} \) is an idle point of the schedule of a system if all requests occurring strictly before \( x \) have completed their execution before or at time \( x \). □

If the system is idle in an interval \([a, b]\), all (integer) instants between \( a \) and \( b \) (included) are idle points by the ones where either a request starts (0 is always an idle point of this kind), or gets completed, or both are clearly more interesting than the ones inside an idle slot. It may easily be seen that if \( U \leq 1 \), then \( P \) is an idle point.

In recent works we have shown other interests of the idle point notion, in particular for parallelizing the schedule generation of uniprocessor scheduling algorithm [7] and parallelizing the preprocessing algorithm of on-line admission control problem [8].

**Theorem 4 ([10])** When the deadline driven scheduler is used to schedule a synchronous set of tasks with arbitrary deadlines on a single processor, there is no processor idle point prior to an overflow, but the origin. □

Consequently the interval \([0, L]\) is a feasibility interval for synchronous arbitrary deadline systems (\( L \) is the position of the first idle point, but the origin). It may be shown [10] that \( L \) is the least strictly positive solution of the equation \( L = \sum_{i=1}^{n} \left[ \frac{L}{T_i} \right] C_i \) and that the latter may be solved by a simple iterative procedure. The original interval of Liu and Layland (i.e., \([0, I']\)) is often called the first busy period in the literature; we shall call the interval \([0, L]\) the first elementary busy period in order to distinguish them. It may be noticed that an idle point is equivalent to an idle slot of zero length some authors used (see for instance [17, 18]) to extend (without a proof) Liu and Layland’s results.

### 4 Worst case response time

Recently, Spuri [17, 18] has extended the computation of the worst case response time, already considered for static priority rules, to the dynamic case (more exactly to the deadline driven scheduler). It may be noticed that Spuri considered the scheduling of sporadically periodic tasks in the worst case scenario: the tasks are released at their maximum rate, i.e., they are periodic.

We define the worst case response time as follows:

**Definition 5** For a task \( \tau_i \), we define the worst case response time (we shall call it \( r_i \)) as the largest response time of \( \tau_i \) among all request of \( \tau_i \) in all synchronous and asynchronous situations. □

It is important at this point to specify that the Spuri’s work concerns arbitrary asynchronous (or offset free) systems, but its main result provides a necessary and sufficient condition for the feasibility of arbitrary and synchronous systems.
Theorem 6 ([17, 18]) A synchronous system with arbitrary deadlines is feasible iff the worst case response time of each task \((\tau_i)\) is less than or equal to the deadline: \(r_i \leq D_i\) for \(i = 1, \ldots, n\).

Spuri gave an algorithm to compute for each task its worst case response time (the reader can find details in [17, 18]). He also outlined the fact that the worst case response time for a task does not always occur in the first busy period of the synchronous schedule, while the latter is a feasibility interval. This is exhibited with the following example.

Example 7 Consider the synchronous system \(S = \{\tau_1 = \{C_1 = 2, D_1 = 3, T_1 = 4\}, \{C_2 = 3, D_2 = 6, T_2 = 7\}\}\). The length \(L\) of the first elementary busy period is in this case 7, the first idle slot is the time unit 19—time units being numbered from 0—the response time of the first request of \(\tau_2\) is 5 while the response time of the 4\(^{th}\) request of \(\tau_2\), starting at time 21, is 6 (see Figure 1; in those figures, \(|\) represents a task request, \(\bigcirc\) a deadline and \(\longrightarrow\) 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).

But Spuri has not outlined another interesting point: the worst case response time does not necessarily occur in the worst case, i.e., the synchronous case, as exhibited in the following example.

Example 8 Consider the following system:

<table>
<thead>
<tr>
<th>(\tau_i)</th>
<th>(T_i)</th>
<th>(D_i)</th>
<th>(C_i)</th>
<th>(r_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\tau_1)</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(\tau_2)</td>
<td>8</td>
<td>7</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>(\tau_3)</td>
<td>12</td>
<td>6</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

In the synchronous case, the largest response time of a request of \(\tau_3\) is 2, as exhibited in Figure 2. But the worst case response time of a request of \(\tau_3\) is 3, this case occurring in asynchronous situations (e.g., \(O_1 = 0, O_2 = 2, O_3 = 3\), as illustrated by Figure 3).

Contrary to our intuition the worst case response time does not necessarily occur in the worst case (with respect to schedulability), nor in the first elementary busy period (see Example 7). We shall see why this is not contradictory. Regarding the feasibility of
the system, from Lemma 1, an asynchronous system is feasible iff \( \sum_{k=1}^{n} \eta_k(t, t') \cdot C_i \leq t' - t \) for all \( 0 \leq t < t' \), and it may be shown that the function \( \sum_{k=1}^{n} \eta_k(t, t') \cdot C_i \), which combines characteristics of the various tasks, is maximum in the synchronous case (see Theorem 2) and for some \( t \leq t' \leq L \) (see Theorem 4). The response time of a particular request depends on the relative priorities between the various requests, which depend on the request/deadline configurations (we have presented in previous work a general calculation procedure for determining the response time of a particular request, details can be found in [4]). Since the synchronous case (and in particular its first elementary busy period \([0, L]\)) does not necessarily include all the configurations, it follows that the worst case response time for a particular request does not necessarily occur in the synchronous schedule (and a fortiori in the interval \([0, L]\) of the latter).

Now, we know that the worst case response time of a particular task does not necessarily occur in the synchronous schedule; it is also important at this point to notice that the worst case response times of the various tasks, i.e., \( r_1, \ldots, r_n \), do not necessarily occur in the same asynchronous schedule, as exhibited in the following example.

**Example 9** Consider the following system:

<table>
<thead>
<tr>
<th>( T_i )</th>
<th>( D_i )</th>
<th>( C_i )</th>
<th>( r_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tau_1 )</td>
<td>4</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>( \tau_2 )</td>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

We shall show that the worst case responses of \( \tau_1 \) and \( \tau_2 \) never occur in the same asynchronous situation. A priori there is an infinite number of asynchronous situations, but we have shown in previous work (see for instance [6]) that we can consider only
lcm \{ T_i | i = 1, \ldots, n \} non-equivalent asynchronous situations (regarding the distribution of the response times); in our case there are 2 non-equivalent asynchronous situations given by (for instance): \( \{ O_1 = 0, O_2 = 0 \} \), and \( \{ O_1 = 0, O_2 = 1 \} \). The largest response time for a request of \( \tau_1 \) occurs in the first (a)synchronous situation (see Figure 4: the response time of the first request of \( \tau_1 \) is 4 = \( D_1 - 1 \)) but does not occur in the second one (see Figure 5: the largest response time of \( \tau_1 \) is 3 here); symmetrically the largest response time of a request of \( \tau_2 \) is 3 and occurs in the second asynchronous situation, while the largest response time of \( \tau_2 \) in the first situation is 2. It may be noticed that we have considered here an arbitrary deadline system, but it is not difficult to see that the phenomenon holds also for constrained deadline systems, since if in the previous system we subtract 1 from each \( D_i \) we get the same schedule since the relative priority ordering is not modified and the response times remain less than or equal to their deadline delays.

5 Conclusion

In this paper we have considered the scheduling problem of hard real-time periodic task sets using the deadline driven scheduler. We have exhibited two interesting features concerning the worst case response time, not considered in the work of Spuri; indeed we show that in the dynamic case the situation is very different than in the static one where the synchronous case is the worst case regarding the feasibility of the system and also provides the worst case response times in the beginning of the same (synchronous) schedule. In the dynamic case the synchronous case remains the worst case situation regarding the feasibility of the system but the worst case response times do not always occur in the synchronous schedule; still worse we have shown that the worst response times for the various tasks of the system do not necessarily occur in the same (asynchronous) schedule. We have exhibited this phenomenon and commented why this apparently counter-intuitive feature is possible.
Acknowledgment

The author gratefully acknowledges the significant contributions of R. Devillers.

References


