INRIA

Analysis of Deadline Scheduled Real-Time Systems

Marco Spuri

N˙ 2772

Janvier 1996

_____ PROGRAMME 1 _____

*R apport
de recherche*

Analysis of Deadline Scheduled Real-Time Systems

Marco Spuri *

Abstract: A uniform, flexible approach is proposed for analysing the feasibility of deadline scheduled real-time systems. In its most general formulation, the analysis assumes sporadically periodic tasks with arbitrary deadlines, release jitter, and shared resources. System overheads of a tick driven scheduler implementation, and scheduling of soft aperiodic tasks are also accounted for.

A procedure for the computation of task worst-case response times is also described for the same model. While this problem has been largely studied in the context of fixed priority systems, we are not aware of other works that have proposed a solution to it when deadline scheduling is assumed. The worst-case response time evaluation is a fundamental tool for analysing *end-to-end* timing constraints in distributed systems [21].

Key-words: real-time, scheduling, feasibility analysis, response times.

(Résumé : tsvp)

Analyse de Systèmes Temps-Réel à Ordonnancement par Échéance la plus Proche en Premier

Résumé :   Une technique uniforme et flexible est proposée pour analyser la faisabilité de systèmes temps-réel à ordonnancement par échéance la plus proche en premier (EDF). Dans sa formulation la plus générale, l'analyse considère des tâches sporadiquement périodiques à échéances arbitraires, à jigues sur les instants d'activation, et avec partage de ressources. Les coûts induits par l'ordonnancement des tâches apériodiques (à échéances non strictes) et par une implantation basée sur une horloge sont pris en compte.

Une procédure pour le calcul du pire temps de réponse des tâches est décrite pour le modèle considéré. Bien que ce problème ait déjà été étudié pour des systèmes à priorités fixes, nous ne connaissons aucun travail sur ce sujet lorsque l'ordonnancement EDF est utilisé. L'évaluation des pires temps de réponse est fondamentale pour l'analyse de contraintes temporelles de *bout-en-bout* dans les systèmes distribués [21].

Mots-clé :   temps-réel, ordonnancement, analyse de faisabilité, temps de réponse.

1 Introduction

Real-time systems are characterized by tasks with stringent timing constraints, that must be met in order to guarantee correctness and safety. One of the main issues of such systems is *predictability*. Central to this issue is the study of suitable scheduling algorithms that let us analyse *a priori* the *feasibility* of the system, that is, establish whether the timing constraints are going to be met or there are potential failures, before the system is actually built.

Within uniprocessor systems, two well known algorithms, Rate Monotonic and Earliest Deadline First (EDF), have been shown optimal with respect to fixed and dynamic priority pre-emptive schemes, in the fundamental work of Liu and Layland [11]. In this work, the two algorithms were studied under a number of restrictions, among which:

- all tasks were strictly periodic;

- deadlines were equal to periods;

- all tasks were independent.

Later, several papers have appeared in the literature in order to extend the analysis of Liu and Layland to more general and useful models. Suitable concurrency control protocols [2, 4, 14] have been proposed for handling shared resources. Tasks have been allowed to have deadlines shorter than their periods [1, 3, 9]. Aperiodic scheduling has also received much attention [6, 8, 10, 15, 16, 17].

Quite recently, Tindell *et al.* [20] presented further extensions to the existing theory for analysing fixed priority pre-emptive systems. The major contributions of their approach are

- arbitrary deadlines (*i.e.* deadlines may also be greater than periods),

- periodic and sporadic tasks,

- release jitter (a delay between the arrival and the actual release of a task instance),

- 'bursty' tasks.

A very interesting aspect of their approach is that the analysis is entirely based on the computation of task worst-case response times. This computation not only is important for assessing the feasibility of a uniprocessor system, but it also plays a fundamental role when we extend our attention to real-time distributed systems. Distributed applications are characterized by precedence relationships between their tasks. If the tasks are statically allocated to single processors, *end-to-end* timing constraints can be analysed by a theory which assumes release jitter [1]: "All tasks are defined to arrive at the same time, but a precedence constrained task on one processor can have its release delayed awaiting the arrival of a message from its predecessors. The worst-case release jitter of such a subtask can be computed by knowing the response times of predecessor tasks located on other processors." See [22, 21] for a general treatment.

In this paper, we focus our attention on deadline scheduled systems, and we propose a flexible approach for analysing such systems, which extends previous results in a very similar way. The motivations of our research are that EDF scheduling always achieves higher processor utilization than fixed priority scheduling, and that we believe a suitable analysis can be developed for analysing *end-to-end* timing constraints also when EDF scheduling is locally used in each single processor, and even in the network protocol.

Another reason for which we believe EDF scheduling should be seriously considered as a candidate for actual real-time systems is that on the contrary to Rate Monotonic, or even to the more general Deadline Monotonic, which have shown not to be optimal when arbitrary deadlines are assumed [9], EDF scheduling stays optimal indeed. By means of a simple interchange argument, Dertouzos [5] proved that any feasible pre-emptive schedule can be easily transformed into an EDF pre-emptive schedule without affecting its feasibility.

Hence, similarly to [20], our goal is then to examine complex systems with sporadically periodic tasks (*i.e.* bursty tasks), which can have arbitrary deadlines as well as arbitrary release jitter, and can share resources by locking and unlocking semaphores through a suitable concurrency control protocol. Overheads of tick driven schedulers and servers for aperiodic requests are also considered. The analysis is based on the concept of *busy period*, an interval of time in which the processor is kept busy, and simple extensions of the original Liu and Layland's theory [11]. The resulting feasibility analysis is then a generalization of some of the results described in [3].

Moreover, in our analysis we do not deal merely with feasibility assessment, but we also describe a procedure for the computation of the task worst-case response times under deadline scheduling. As already stated, the solution to this problem is a fundamental tool for the so called *holistic schedulability analysis* for distributed real-time systems [21]: "The release jitter of a message depends on the worst-case response time of the sender task. The worst-case response time of a task depends, in part, on the response times of messages. A message response time depends its release jitter [22]."

We believe that the holistic analysis can now also be extended to distributed systems with local EDF schedulers, even if this is the subject of current research and it will not be further treated in this paper. We are also not aware of other works dealing with the problem of response time evaluation under EDF scheduling. That is why we believe that the major contribution of our work is the solution to this problem.

The paper is organized as follows. In Section 2 the notation and the assumptions used throughout the paper are described. The basic feasibility analysis for periodic and sporadic task sets with arbitrary deadlines is treated in Section 3, while in Section 4 we describe the procedure for the computation of worst-case response times under the same model. Sections 5, 6, 7, 8, and 9, deal with several extensions to the basic model: release jitter, sporadically periodic tasks, resource sharing, inclusion of soft aperiodic tasks, and tick scheduling overheads, respectively. In all extensions, both the feasibility analysis and the worst-case response time computation are treated in detail. A case study is presented in Section 10. Finally, our conclusions are stated in Section 11. In Appendix A we have reported the prooves of a couple of properties concerning the busy period analysis which we do

not consider essential for the comprehension of the paper. Furthermore, in Appendix B the complete description of our analysis is summarized.

2   Computational Model and Notation

This work is mainly inspired by the results presented in [1, 20]. Our goal is to find an approach for the analysis of problems similar to those studied there, in which the basic assumption of fixed priority pre-emptive scheduling is replaced by the assumption of EDF pre-emptive scheduling. More specifically, we assume the same processor workload model, *i.e.* task arrival laws and properties, while the scheduling mechanism is now deadline-based: at any time, the ready task with the earliest deadline is run. For this reason, we also adopt, as far as possible, the same notation as in [1, 20].

In the paper we consider the scheduling of tasks on a single processor. A *task* consists of an infinite number of *requests*, or *instances*, whose *arrival* times are separated by a minimum time T, called *period* (according to the conventional notation, this assumption is common to periodic and sporadic tasks). We assume that task instances may arrive at any time. However, the arrival must be recognized by a run-time dispatcher, which then will place the instance in a notional run-time queue. The instance is then said to be *released*. The time between a task's arrival and its release[1] is known as *release jitter*.

Each task instance may execute for a bounded amount of computation C, called *worst-case execution time*. The computation should complete within a time D (*relative deadline*) after the arrival. The notional run-queue is ordered according to the actual task's deadlines, earliest first, that is, we assume an EDF [11] pre-emptive dispatching. Tasks may also share resources, by locking and unlocking semaphores according to a protocol like the Priority Ceiling [14, 4] or the Stack Resource Policy [2].

In a later section we also extend the workload model with the so called *sporadically periodic* tasks [1]. This sort of tasks is intended to model the behaviour of events which may arrive at a certain rate for a number of times, and then not re-arrive for a longer time. For example, there are interrupts which behave in this way (they are also termed bursty sporadics). Sporadically periodic tasks are assigned two periods: an *inner period* (t) and an *outer period* (T). The outer period is the worst-case inter-arrival time between bursts. The inner period is the worst-case inter-arrival time between task instances within a burst. There is a bounded number of arrivals to each burst. Furthermore, the total time for the burst (*i.e.* the number of inner arrivals multiplied by the inner period) must be less than or equal to the outer period. A task that is not a bursty task is simply modelled as one that has an inner period equal to the outer period, and at most one inner arrival.

A glossary of the notation used in this paper follows:

C_i The worst-case computation time of task i on each release.

D_i The deadline of task i, measured relative to the arrival time of the task.

---

[1] Note that the release of a task can also be delayed by other factors, such as a distributed synchronization.

B_i The worst-case blocking time task i can experience due to the operation of the concurrency control protocol.

J_i The worst-case release jitter of task i (*i.e.* the worst-case delay between the arrival and its release).

T_i The outer period of task i.

t_i The inner period of task i.

n_i The worst-case number of arrivals of task i per outer period.

r_i The worst-case response time of task i, measured from the arrival time to the completion time.

$I_i(t)$ The number of instances of task i released before time t.

$H_i(t)$ The number of instances of task i with deadline before or at t.

$s_i(a)$ The release time of the first instance of task i, in an arrival pattern where another i's instance arrives at time a.

L The length of the first busy period in the most demanding arrival pattern.

U_S The processor utilization allocated to the aperiodic server.

3   Basic Feasibility Analysis

In this section, as well as in the following one, we assume a simple model in which all tasks have null release jitter, do not share resources, and are not bursty (*i.e.* $\forall i, J_i = 0, B_i = 0, t_i = T_i, n_i = 1$). A basic feasibility analysis is presented and is later extended to handle more general models.

Given a task set, according to the assumptions, we can have different patterns of arrivals (recall that the period is only a worst-case inter-arrival time). It is not difficult to show that the worst-case pattern is that in which the task instances are released as soon as possible, that is, the first instance is released at time $t = 0$, and the others are then released according to the task's period (this pattern is termed *asap* in the rest of this paper). The result is quite intuitive, and it can be proven using the same argument as in Theorem 6 of the well known work by Liu and Layland [11].

Theorem 3.1 (Liu and Layland) *When the deadline driven scheduling algorithm is used to schedule a set of tasks on a processor, if there is an overflow for a certain arrival pattern, then there is an overflow without idle time prior to it in the pattern in which all task instances are released as soon as possible (*i.e. *in the* asap *arrival pattern).*

Proof. We can apply the same argument as Liu and Layland did. Assume there is a pattern which causes an overflow at time t. Let t' be the end of the last processor idle period before t, or 0 if there are any. t' must be the arrival time of at least one instance. If we ideally "shift" left all other instances arrived after t', up to t' and according to their maximum arrival rate, we can only increase the processor workload in the interval $[t', t]$. Since there was no processor idle time between t' and t, there will be no processor idle time after the shift. That is, an overflow will still occur at or before t. If we now only consider the pattern from time t' on, we have obtained the *asap* pattern and an overflow with no processor idle period prior to it. □

This theorem suggests studying the schedule of the *asap* pattern in the first *busy period*, *i.e.* in the first interval from time $t = 0$ up to the first processor idle time. The concept of busy period is already known in the literature [9, 7] and we will briefly see that it is a useful tool for the analysis of a task set even with a more general model.

The length L of the busy period can be computed by means of a simple iterative formula:

$$\begin{cases} L^{(0)} & = & \sum_{i=1}^{n} C_i, \\ L^{(m+1)} & = & W(L^{(m)}), \end{cases} \tag{1}$$

where $W(t)$ is the cumulative workload at time t, *i.e.* the sum of the computation times of the task instances arrived before time t:

$$W(t) = \sum_{i=1}^{n} \left\lceil \frac{t}{T_i} \right\rceil C_i.$$

The computation in Equation (1) is stopped when two consecutive values are found equal, that is, $L^{(m+1)} = L^{(m)}$. L is then set to $L^{(m)}$. It can be easily proven that the sequence $L^{(m)}$ converges to L in a finite number of steps if the overall processor utilization of the task set is less then or equal to 1, that is, if

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1.$$

(Note that if the condition does not hold the task set cannot be feasible). The proof is shown in Appendix A.

We now have to control whether in the busy period there are missed deadlines. To do this efficiently, we can look again at the argument of Theorem 3.1. According to it, we have an overflow at time t if the sum of the computation times of all instances with deadline before or at t is greater than t. This workload can be easily computed as

$$\sum_{D_i \leq t} \left(1 + \left\lfloor \frac{t \Leftrightarrow D_i}{T_i} \right\rfloor \right) C_i.$$

That is, a necessary and sufficient condition for the feasibility of the task set is that for all actual deadlines d in the first busy period

$$d \geq \sum_{D_i \leq d} \left(1 + \left\lfloor \frac{d \Leftrightarrow D_i}{T_i} \right\rfloor \right) C_i. \tag{2}$$

Note that Equation (2) replicates the result of Theorem 3.1 of [3]. However, we have presented the result in the more general case of arbitrary deadlines (in [3] it is assumed $D_i \leq T_i$ for all i). Furthermore, we not only believe that the argument used here to achieve the result is quite intuitive, but as we will briefly show, it can also be used to compute the maximum response time of each task, even in a more complex framework.[2]

4   Finding Worst-Case Response Times

The worst-case response time r_i of a task i is the maximum time between an i's instance arrival and its completion. As already stated, the computation of r_i is important if we want to analyse a global distributed system: the response time of a task is the maximum jitter experienced by the messages it sends over the network.

Finding r_i is not a trivial task when EDF scheduling is assumed. Contrary to our intuition, the worst-case response time of a task is not always found in the first busy period, which is not exactly the equivalent of the *critical instant* under fixed priority scheduling. However, the concept of busy period is still useful. The idea is that the completion time of a task's instance with deadline d, must be the end of a busy period in which all executed instances have deadlines less than or equal to d. In order to find the longest one among all such periods, we can then apply an argument similar to that of Theorem 3.1.

Lemma 4.1 *The worst-case response time of a task i is found in a busy period in which all other tasks are released* asap, *that is, they are released synchronously at the beginning of the period and then at their maximum rate (see Figure 1b).*

Proof. Consider a task i's instance, like in Figure 1a, with arrival time a and deadline $d = a + D_i$, respectively. Let t_2 be its completion time, according to the EDF schedule. Let t_1 be the last time before t_2, such that there are no pending instances with arrival time earlier than t_1 and deadline less than or equal to d. Since the i's instance is released at $t = a$, $t_1 \leq a$. Furthermore, by choice of t_1 and t_2, t_1 must be the arrival time of a task's instance, and there is no idle time in $[t_1, t_2]$. That is, $[t_1, t_2]$ is a busy period in which only instances with deadlines less than or equal to d execute.

At this point, if we "shift" left all instances of tasks different from i, in such a way to obtain an *asap* arrival pattern, like in Figure 1b, starting at t_1, the workload of instances with deadlines less than or equal to d cannot diminish in $[t_1, t_2]$. That is, the completion time of the i's instance considered can only increase.   □

---

[2] In [17], a similar technique is used to build a more realistic analysis in which both the system overheads and the interrupt handling are taken into account.
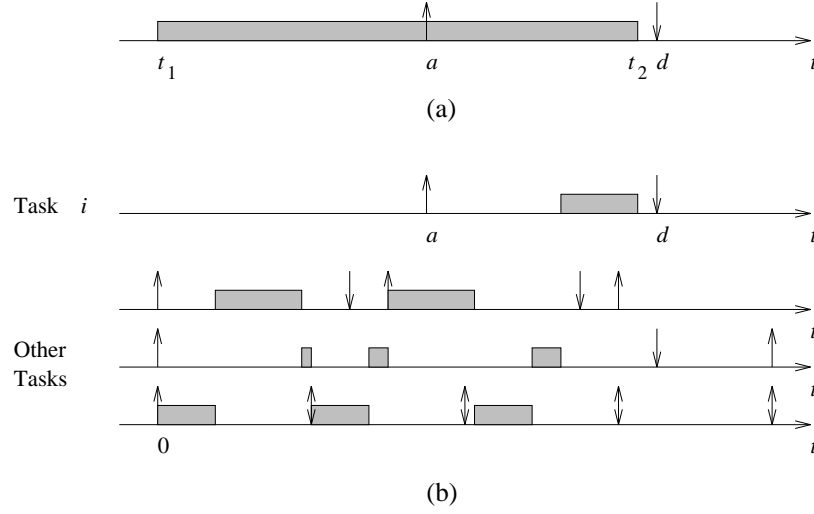
Figure 1: a) Busy period preceding an instance completion time. b) *Asap* arrival pattern possibly giving the worst-case response time for task i.

The previous lemma suggests the following algorithm for computing the worst-case response time of a task i: we only need to compute the length of the busy periods of task instances with deadlines less than or equal to that of the i's instance considered, for a number of arrival patterns like the one shown in Figure 1b. All tasks but i are released synchronously and at their maximum rate at time $t = 0$. Our attention will be on the i's instance released at time $t = a$, $a \geq 0$, and possibly preceded by other instances of task i (these instances may contribute to increase the busy period length).

In particular, given $a \geq 0$, we consider an arrival pattern in which the first instance of i is released at time

$$s_i(a) = a \Leftrightarrow \left\lfloor \frac{a}{T_i} \right\rfloor T_i.$$

If all other tasks are initially released at time $t = 0$, at time t, $\left\lceil \frac{t}{T_j} \right\rceil$ instances of j will have been released, for each $j \neq i$. However, at most only $1 + \left\lfloor \frac{a+D_i-D_j}{T_j} \right\rfloor$ of them can have a deadline less than or equal[3] to $d = a + D_i$. That is, the higher priority workload arrived up to time t is

$$W_i(a,t) = \sum_{\substack{j \neq i \\ D_j \leq a + D_i}} \min\left\{ \left\lceil \frac{t}{T_j} \right\rceil, 1 + \left\lfloor \frac{a + D_i \Leftrightarrow D_j}{T_j} \right\rfloor \right\} C_j \; + \; \delta_i(a,t)C_i,$$

---

[3]Note that we do not assume any particular policy for breaking deadline ties. Hence, in the worst-case instances sharing the same deadline should be considered as having higher priorities.

i	D_i	T_i	C_i	r_i	a_i
1	4	4	1	2	11
2	9	6	2	7	6
3	6	8	2	4	9
4	12	16	2	10	3

Table 1: Task set parameters for our example of response time computation.

a	$r_3(a)$
0	3
2	2
3	2
6	2
8	2
9	4
10	4

Table 2: Computation of r_3, the worst-case response time of task 3.

where

$$\delta_i(a,t) = \begin{cases} \min\left\{ \left\lceil \frac{t - s_i(a)}{T_i} \right\rceil, 1 + \left\lfloor \frac{a}{T_i} \right\rfloor \right\} & \text{if } t > s_i(a), \\ 0 & \text{otherwise.} \end{cases}$$

The length $L_i(a)$ of the resulting busy period relative to the deadline d can then be computed with the following iterative formula:

$$\begin{cases} L_i^{(0)}(a) & = & \sum\limits_{\substack{j \neq i \\ D_j \leq a + D_i}} C_j + I_{\{s_i(a)=0\}} C_i, \\ L_i^{(m+1)}(a) & = & W_i\left(a, L_i^{(m)}(a)\right), \end{cases} \tag{3}$$

where

$$I_{\{s_i(a)=0\}} = \begin{cases} 1 & \text{if } s_i(a) = 0, \\ 0 & \text{otherwise.} \end{cases}$$

As for Equation (1), the convergence of Equation (3) in a finite number of steps is ensured by the condition

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1.$$

The reason is that the maximum length of any busy period is L, the length of the first busy period of the *asap* arrival pattern (the property is shown in Appendix A). At each step of Equation (3) either $L_i^{(m+1)}(a)$ is equal to $L_i^{(m)}$, in which case the computation is halted, or $L_i^{(m+1)}(a)$ is $L_i^{(m)}$ increased at least by a quantity $C_{\min}$, the minimum computation time among all tasks. Being $L_i(a)$ bounded by L, its value is thus reached in a finite number of steps.

Once we have determined $L_i(a)$, the worst-case response time relative to a is

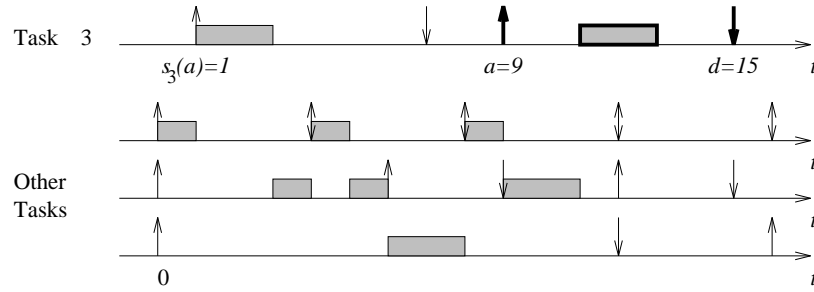$$r_i(a) = \max\{C_i, L_i(a) \Leftrightarrow a\}.$$

Figure 2: Task 3 worst-case response time with arrival at time $a = 9$.

Finally, according to Lemma 4.1, the worst-case response time of task i is

$$r_i = \max_{a \geq 0} \{r_i(a)\} . \tag{4}$$

For which values of a should we evaluate the function $r_i(a)$ in Equation (4)? An upper bound is given by L, which we know to be the upper bound of any busy period length. Hence, the significant values are in the interval $[0, L \Leftrightarrow C_i)$. Furthermore, it is not difficult to see that the local maxima of $L_i(a)$ are found for those values of a such that in the arrival pattern there is at least an instance of a task different from i with deadline equal to d, or all tasks are synchronized, *i.e.* $s_i(a) = 0$.

As an example, let us consider a set of four tasks, with the parameters shown in Table 1. The processor utilization of the task set is 95.83%. In the last two columns we have reported the worst-case response times of the tasks and the values of a for which they have been found. In Table 2 we have reported the computation of r_3. Note that the maximum is not achieved in the first busy period. The schedule for $a = 9$ is depicted in Figure 2 (upward arrows indicate instance releases, downward arrows indicate deadlines).

5 The Release Jitter Problem

In this section we remove the assumption of null release jitter. Which is, we now assume that after each arrival, a task i may be delayed for a maximum time J_i before being actually released. As previously stated, this may be due to several reasons among which a tick scheduling or a distributed synchronization.

When a task experiences jitter, we can have arrival patterns in which two consecutive releases of the same task are separated by an interval of time shorter than T. Thus, intuitively, the worst-case arrival pattern is one in which all tasks experience their shortest inter-release times at the beginning of the schedule. This intuition is confirmed by Theorem 3.1, which is still valid even when jitter is considered. Once again, the worst-case arrival pattern is referred to as *asap*, since all task instances are released as soon as possible. In this case, the
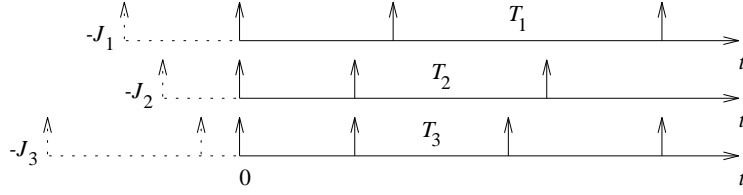
Figure 3: *Asap* arrival pattern for a task set with release jitter.

pattern is obtained by releasing the first instance of each task at time $t = 0$, all others are then released at time $t = \max\{kT_i \Leftrightarrow J_i, 0\}$, $\forall i$ and $\forall k > 0$. See Figure 3 for an example.

In order to check the feasibility of the *asap* pattern, we can apply the same methodology used in Section 3. We only need to modify our formulae to fit our new *asap* pattern.

In particular, Equation (1) must be modified in the definition of the cumulative workload $W(t)$, which becomes

$$W(t) = \sum_{i=1}^{n} \left\lceil \frac{t + J_i}{T_i} \right\rceil C_i.$$

Furthermore, for each task i, the number of instances with deadline before or at t is now

$$1 + \left\lfloor \frac{(t + J_i) \Leftrightarrow D_i}{T_i} \right\rfloor,$$

since the situation is like having the first instance arrival ideally at time $t = \Leftrightarrow J_i$, and all others equally T_i spaced later: all instances ideally arrived at time $t < 0$ are actually released at time $t = 0$.

Consequently, Equation (2) becomes

$$d \geq \sum_{D_i \leq d + J_i} \left(1 + \left\lfloor \frac{d + J_i \Leftrightarrow D_i}{T_i} \right\rfloor \right) C_i.$$

The evaluation of the worst-case response times is also a simple extension of the theory seen in the previous section. The argument of Lemma 4.1 can be applied to the new model too. The difference is again in the *asap* arrival pattern, owing to the initial release jitter. An example of the new patterns examined to find the worst-case response time of task i is shown in Figure 4.

As previously, given a, we include in the pattern all possible i's instances, so that there is one arrival at time $t = a$. In order to include the most possible instances, we force the first one to experience a release jitter J_i and we require its release time to be greater than or equal to 0 (its ideal arrival time may be before time 0). In this way, the first i's instance has release time

$$s_i(a) = a + J_i \Leftrightarrow \left\lfloor \frac{a + J_i}{T_i} \right\rfloor T_i.$$

Note that if $a + J_i < T_i$ the previous formula gives $s_i(a) = a + J_i$, *i.e.* the instance arrived at time $t = a$ is actually released J_i units of time later.

The computation of the higher priority workload arrived up to time t is now

$$W_i(a,t) = \sum_{\substack{j \neq i \\ D_j \leq a + D_i + J_j}} \min\left\{ \left\lceil \frac{t + J_j}{T_j} \right\rceil, 1 + \left\lfloor \frac{a + D_i + J_j \Leftrightarrow D_j}{T_j} \right\rfloor \right\} C_j \; + \; \delta_i(a,t)C_i,$$

where

$$\delta_i(a,t) = \begin{cases} \min\left\{ \left\lceil \frac{t - s_i(a) + J_i}{T_i} \right\rceil, 1 + \left\lfloor \frac{a + J_i}{T_i} \right\rfloor \right\} & \text{if } t > s_i(a), \\ 0 & \text{otherwise.} \end{cases}$$

The length of the resulting busy period relative to the deadline $d = a + D_i$ can then be computed through Equation (3), which we only need to modify in the definition of the initial value $L_i^{(0)}(a)$, that becomes

$$L_i^{(0)}(a) = \sum_{\substack{j \neq i \\ D_j \leq a + D_i + J_j}} C_j \; + \; I_{\{s_i(a)=0\}} C_i.$$

The worst-case response time relative to a is then

$$r_i(a) = \max\{J_i + C_i, L_i(a) \Leftrightarrow a\}.$$

Then, Equation (4) can finally be applied to find r_i. There is a slight difference, however, in the meaning of the variable a: in the previous case a was at the same time arrival and release time of the instance considered; in presence of release jitter it may be only the arrival time, with the release time possibly being up to J_i units of time later. In fact, the value of $r_i(a)$ in Equation (4) must be evaluated for all significant values of a such that the release time of the i's instance considered is greater than or equal to 0. Thus our attention will be on the interval $[\Leftrightarrow J_i, L \Leftrightarrow J_i \Leftrightarrow C_i)$:

$$r_i = \max_{a \in [-J_i, L - J_i - C_i)} \{r_i(a)\}.$$

The argument stated at the end of the previous section to further limit the values of a for which we have to compute $r_i(a)$ can still be fully applied also in presence of jitter.

6   Sporadically Periodic Tasks

In this section we extend our analysis by including sporadically periodic tasks. This sort of tasks models bursty activities and has been described in Section 2. In the analysis, we assume that $n_i t_i \leq T_i$ (*i.e.* the inner arrivals are all required to fit in an outer period) and that each instance may suffer a maximum release jitter J_i. The analysis is extended by following the same approach as in the previous section.
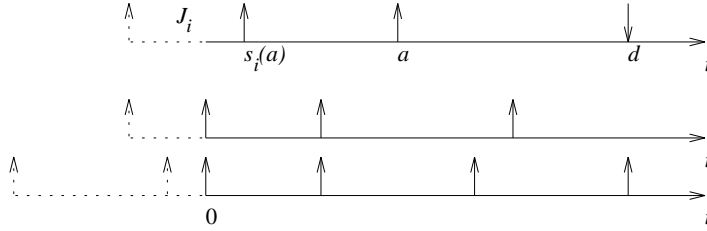
Figure 4: *Asap* arrival pattern for the evaluation of task *i*'s response time.
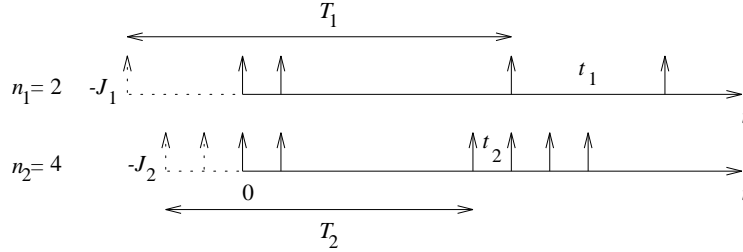

Figure 5: *Asap* arrival pattern of sporadically periodic tasks.

For the feasibility part, we start noting that once again we should look at the *asap* arrival pattern, which is the worst in terms of processor loading factor. The pattern is depicted in Figure 5. As previously, it is such that the first instances of all tasks are released at time $t = 0$, and are ideally experiencing their maximum jitter. All the following instances are then released as soon as possible.

The validity of Theorem 3.1 is not affected by the new model, as can be easily realized. We can thus apply the usual argument of the first busy period in the schedule. Equation (1) must be again modified in the definition of $W(t)$ according to the new model, in order to correctly compute the length L of the busy period.

The definition of the cumulative workload released up to time t is now a bit trickier. If we recall that for each task i, the first instance ideally arrives at time $t = \Leftrightarrow J_i$, but it is actually released at time $t = 0$, as shown in Figure 5, we can compute the number of instances of i arrived and released by time t, $I_i(t)$, as the sum of:

- n_i times the number of outer periods entirely fitted in an interval of $t + J_i$ units of time, and

- the minimum between n_i and the number of inner periods (rounded to the smallest larger integer) which fit in the last part of the interval $(t+J_i \Leftrightarrow \left\lfloor \frac{t+J_i}{T_i} \right\rfloor T_i$ wide) preceding t.

That is,

$$I_i(t) = \left\lfloor \frac{t + J_i}{T_i} \right\rfloor n_i + \min\left\{ n_i, \left\lceil \frac{t + J_i \Leftrightarrow \left\lfloor \frac{t+J_i}{T_i} \right\rfloor T_i}{t_i} \right\rceil \right\} .$$

$W(t)$ then becomes[4]

$$W(t) = \sum_{i=1}^{n} I_i(t)C_i.$$

With a similar argument, we can also determine the number of instances of task i with deadline before or at t, $H_i(t)$, which is thus the sum of:

- n_i times the number of outer periods entirely fitted in an interval of $t + J_i \Leftrightarrow D_i$ units of time, and

- the minimum between n_i and the number of inner periods (rounded to the largest smaller integer) which fit in the last part of the interval $(t + J_i \Leftrightarrow D_i \Leftrightarrow \left\lfloor \frac{t+J_i-D_i}{T_i} \right\rfloor T_i$ wide) preceding t, increased by 1.

That is,

$$H_i(t) = \left\lfloor \frac{t + J_i \Leftrightarrow D_i}{T_i} \right\rfloor n_i + \min\left\{ n_i, 1 + \left\lfloor \frac{t + J_i \Leftrightarrow D_i \Leftrightarrow \left\lfloor \frac{t+J_i-D_i}{T_i} \right\rfloor T_i}{t_i} \right\rfloor \right\}$$

Equation 2 can then be rewritten as

$$d \geq \sum_{D_i \leq d+J_i} H_i(d)C_i.$$

Finally, we have to modify our procedure for the computation of the worst-case response time of each task. The followed approach is always the same, since also the applicability of Lemma 4.1 is not affected by the new model. Thus, given a task i we consider all arrival patterns in which all other tasks are released *asap*, while i has an arrival at time $t = a$, and possibly some others previously.

The first step is to compute correctly $s_i(a)$, the release time of i's first instance. The difference from the previous situations is that we can now make a first choice on the basis of the outer period, and then we can refine it according to the inner period. In particular, we can first determine the release time of the first instance as if the instance with arrival time $t = a$ should be the first one of an outer period:

$$S_i(a) = a + J_i \Leftrightarrow \left\lfloor \frac{a + J_i}{T_i} \right\rfloor T_i.$$

---

[4]It is not difficult to see that being the new model more general than the previous one, the computation of $W(t)$ reduces to those previously shown when used with simpler or mixed models, in which we have tasks without a bursty behaviour and/or without release jitter.

Then, we can see whether other instances (up to $n_i \Leftrightarrow 1$) can be released earlier, by computing

$$s_i(a) = S_i(a) \Leftrightarrow \min \left\{ n_i \Leftrightarrow 1, \left\lfloor \frac{S_i(a)}{t_i} \right\rfloor \right\} t_i.$$

Note that if task i is not bursty, the computation is the same as in the previous section.

The computation of the higher priority workload arrived up to time t also has to be modified according to the new model. The number of instances of task j which have deadlines before or at $a + D_i$ is $H_j(a + D_i)$. Similarly, the number of instances of task j released by time t is $I_j(t)$. The higher priority workload arrived up to time t is thus

$$W_i(a, t) = \sum_{\substack{j \neq i \\ D_j \leq a + D_i + J_j}} \min \left\{ I_j(t), H_j(a + D_i) \right\} C_j \ + \ \delta_i(a, t) C_i,$$

where

$$\delta_i(a, t) = \begin{cases} \min \left\{ I_i \left(t \Leftrightarrow s_i(a) \right), H_i(a + D_i) \right\} & \text{if } t > s_i(a), \\ 0 & \text{otherwise.} \end{cases}$$

The rest of the procedure, *i.e.* the computation of the busy period relative to $d = a + D_i$, the computation of the worst-case response time relative to a, and the computation of the overall worst-case response time, remains unchanged. A description of the full analysis can be found in Appendix B.

7 Resource Sharing

If the tasks are allowed to share resources, in our analysis we must take into account additional terms, namely *blocking factors*, owing to the inevitable priority inversions. The maximum duration of such inversions can be bounded if shared resources are accessed by locking and unlocking semaphores according to a protocol like the Priority Ceiling [14, 4] or the Stack Resource Policy [2]. In particular, for each task i we can compute the worst-case blocking time B_i which the task can experience due to the operation of the concurrency control protocol.

We will explain how the feasibility analysis and the computation of the worst-case response times in the model of Section (5) should be modified. The details of a more general analysis can be found in Appendix B.

As we will briefly see, the required modifications on the analysis are only few. We first observe that the argument of Theorem 3.1 is still valid. We only need to be careful when checking the deadline of a task's instance: the instance being checked, or another one which precedes it in the schedule, may experience a blocking that we must carefully include as an additional term.

The length L of the first busy period is unaffected by the presence of blocking instead. Note that priority inversions may only "deviate" the schedule from its ordinary EDF characteristic. However, this does not change the workload arrived at any time t, which is in

fact independent from the scheduling algorithm (it only depends on the release pattern). Hence, L is still computed by means of iterative Equation (1).

As already stated, the deadlines in the first busy period can still be checked by Equation (2). In this case, however, we must also consider the blocking factor of the task with the largest $D_k \Leftrightarrow J_k$ value among those included in the sum of the right side. That is, if $k(d) = \max\{k : D_k \Leftrightarrow J_k \leq d\}$, now we must check:

$$d \geq \sum_{D_i \leq d + J_i} \left(1 + \left\lfloor \frac{d + J_i \Leftrightarrow D_i}{T_i} \right\rfloor \right) C_i \; + \; B_{k(d)}. \tag{5}$$

In order to explain why in the previous formula we only need to include $B_{k(d)}$, we first illustrate how the blocking factors are computed. Following Baker's approach [2], this computation can be carried out by reasoning in terms of task *preemption levels*: an instance of a task i is not allowed to preempt an instance of a task j unless the preemption level of i, π_i, is greater than the preemption level of j. In our model, a consistent assignment of preemption levels is such that

$$\pi_i > \pi_j \quad \Leftrightarrow \quad D_i \Leftrightarrow J_i < D_j \Leftrightarrow J_j.$$

To each semaphore s protecting critical sections we can associate a ceiling $\lceil s \rceil$ equal to the maximum preemption level of those tasks whose instances may lock it:

$$\lceil s \rceil = \max\{\pi_i : \; i \text{ may lock } s\}.$$

According to both protocols, the Stack Resource Policy and the Dynamic Priority Ceilings, a task i may be blocked either directly by a task with a lower preemption level, or indirectly by a similar task which directly blocks another task with a higher preemption level. If $cs_j(s)$ denotes the maximum time instances of task j may lock semaphore s, the blocking time of task i is thus defined as:

$$B_i = \max\{cs_j(s) : \; \lceil s \rceil \geq \pi_i \text{ and } \pi_j \leq \pi_i\}.$$

The correctness of Equation (5) is shown in the following theorem, in which we assume that the tasks are ordered by increasing value of $D_i \Leftrightarrow J_i$.

Theorem 7.1 *A sufficient condition for the feasibility of a set of tasks is that for each deadline d in the first busy period of the* asap *arrival pattern we have*

$$d \geq \sum_{D_i \leq d + J_i} \left(1 + \left\lfloor \frac{d + J_i \Leftrightarrow D_i}{T_i} \right\rfloor \right) C_i \; + \; B_{k(d)}.$$

Proof. The argument is very similar to that of Theorem 10 of [2]. Assume there is an overflow at time t for a given arrival pattern. Let t' be the last time before t such that there are no pending instances with release times before t' and deadlines before or at t.

Let $\mathcal{A}$ be the set of tasks that have instances with release times and deadlines in $[t', t]$. We have $\mathcal{A} \subseteq \{1, \ldots, k(t \Leftrightarrow t')\}$. According to Baker's argument, the only instances that execute in $[t', t]$ are those of tasks in $\mathcal{A}$, plus at most one instance of a task j that may block a task in $\mathcal{A}$. Similarly to Theorem 3.1 we can now ideally forget the schedule preceding t', and shift left all instances of the tasks $1, \ldots, k(t \Leftrightarrow t')$ released after t', up to t' and according to their maximum release rate. There are two cases:

$j \leq k(t \Leftrightarrow t')$ –   After the shift at least one instance of j is included in the workload executed in $[t', t]$, that is, we are in a worse situation.

$j > k(t \Leftrightarrow t')$ –   From the blocking point of view nothing changes. In this case j locks a semaphore s whose ceiling $\lceil s \rceil \geq \pi_{k(t-t')}$, since a task in $\mathcal{A}$ is blocked by j. Furthermore, $\pi_j < \pi_{k(t-t')}$. It follows that the instance of j executes in $[t', t]$ at most for a time $cs_j(s)$, which, by definition, is less then or equal to $B_{k(t-t')}$.

Hence we can conclude that after the shift the workload in $[t', t]$ has not diminished and that the only potential blocking experienced by some instance of a task in $\{1, \ldots, k(t \Leftrightarrow t')\}$ is still bounded by $B_{k(t-t')}$. An overflow will still occur at or before t. If it occurs before t we can repeat the argument on the new point and recompute the value $k(t \Leftrightarrow t')$. Without loss of generality, we can thus assume that the overflow still occurs at t. We have

$$\sum_{D_i - J_i \leq t-t'} \left(1 + \left\lfloor \frac{(t \Leftrightarrow t') + J_i \Leftrightarrow D_i}{T_i} \right\rfloor \right) C_i \; + \; B_{k(t-t')} \; > \; t \Leftrightarrow t',$$

a contradiction.   □

Similarly, the argument of Lemma 4.1 is still applicable for the computation of the worst-case response times. In Equation (3) we must consider the possible additional blocking time in the computation of $L_i^{(m+1)}$, which, according to the same argument of the previous theorem, becomes

$$L_i^{(m+1)}(a) = W_i\left(a, L_i^{(m)}(a)\right) + B_{k(a+D_i)}.$$

The worst-case response time relative to a becomes

$$r_i(a) = \max\left\{J_i + C_i + B_i, L_i(a) \Leftrightarrow a\right\}.$$

Finally, Equation (4) can be used to evaluate r_i. The interval in which the function $r_i(a)$ needs to be evaluated is $[\Leftrightarrow J_i, L \Leftrightarrow J_i \Leftrightarrow C_i \Leftrightarrow B_i)$. The argument of Section 4 to further restrict the set of the significant values for a is still valid.

8   Including Soft Real-Time Aperiodic Load

Periodic or sporadic hard tasks, as well as their sporadically periodic generalization, do not model all possible activities of a real-time system. In the literature, we find several works

in which hard tasks are jointly scheduled with soft or firm tasks [8, 15, 10, 16, 18, 19, 6, 13]. Soft tasks are characterized by soft deadlines, *i.e.* desired average response times, or by no deadlines at all, *i.e.* the goal is in this case to minimize the average response times. Firm tasks can be rejected if their deadlines cannot be met.

A common approach followed by several authors is the utilization of *bandwidth preserving* algorithms: they are all characterised by the definition of an aperiodic *server*, that is, a task devoted to servicing the aperiodic requests entering the system. The goal of the server is usually to preserve processor bandwidth, *i.e.* processor utilization, for minimizing the response times of aperiodic tasks, without jeopardizing the hard tasks schedule.

Examples of aperiodic servers are the *Priority Exchange* [8] and the *Sporadic Server* [15] algorithms, first described for fixed priority systems, and then also extended to deadline scheduled systems in [16] and [6] respectively. Both servers basically reserve a *capacity* C_S, *i.e.* processor time, each T_S units of time. They differ in the way the capacity is replenished. However, in the worst case it is not difficult to see that they behave like periodic tasks, thus they can easily be included in the feasibility analysis we have described so far.

Instead, we put our attention on another algorithm, the *Total Bandwidth* server, which has been first described in [16] and then improved and extended in [18]. This algorithm catches our attention because of its very simple practical implementation and also because it has been proven more performing than others by extensive simulations. Furthermore, it has been extended to handle firm aperiodic tasks too.

The idea of the Total Bandwidth server is very simple: whenever a new aperiodic task enters the system, the maximum available bandwidth is immediately assigned to it by computing a suitable deadline; the task is then scheduled according to this deadline. In practice, the designer of the system assigns a processor utilization U_S to the server (note that we could have more than one server). If an aperiodic task arrives at time a, when it is to be serviced by the server (we do not make any assumption on the server queuing strategy) it receives a deadline

$$d = \max\{a, \bar{d}_{prev}, f_{prev}\} + \frac{C}{U_S},$$

where $\bar{d}_{prev}$ is the "corrected" deadline and f_{prev} is the completion time of the previous scheduled aperiodic request. Once the task completes, its corrected deadline is computed as

$$\bar{d} = \max\{a, \bar{d}_{prev}, f_{prev}\} + \frac{\bar{C}}{U_S},$$

where $\bar{C}$ is its actual execution time ($\bar{C} \leq C$). In [18], it is proven that in any period of time $[t_1, t_2]$, the global computation time demanded by the server with deadlines less than or equal to t_2 is upper bounded by $(t_2 \Leftrightarrow t_1)U_S$.

In order to extend our analysis, first note that the argument of Theorem 3.1 is still valid. The only difference is that we now cannot say anything about the busy period length, since the presence of the server may take the processor busy for an arbitrarily long period of time (we do not assume any restrictions on the aperiodic task computation times). However,

since in any interval the fraction of the time possibly assigned to the server is U_S, according to our intuition it should be sufficient to consider this fraction in our arguments.

Let us modify the computation of L, the length of the busy period, by first defining the workload $W(t)$ as follows:

$$W(t) = \sum_{i=1}^{n} \left\lceil \frac{t}{T_i} \right\rceil C_i \; + \; tU_S.$$

Then Equation (1) becomes

$$\begin{cases} L^{(0)} & = & \frac{1}{1-U_S} \sum_{i=1}^{n} C_i, \\ L^{(m+1)} & = & \frac{W(L^{(m)}) - L^{(m)} U_S}{1 - U_S}. \end{cases} \tag{6}$$

For an explanation see Appendix A. Note that the convergence of the computation is now ensured by the condition

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \; + \; U_S \leq 1.$$

Lemma 8.1 *If there is a schedule with a time overflow, then there is an overflow within L in the schedule of the* asap *arrival pattern, that is, there exists $t < L$:*

$$t < \sum_{D_i \leq t} \left(1 + \left\lfloor \frac{t \Leftrightarrow D_i}{T_i} \right\rfloor \right) C_i \; + \; tU_S.$$

Proof. Assume there is an overflow in the schedule at time t_2. Let t_1 be the last instant of time preceding t_2 such that there are no pending instances with arrival time [5] earlier than t_1 and deadline before or at t_2. By choice of t_1, in the interval $[t_1, t_2]$ there is no idle and only tasks with deadlines before or at t_2 are executed. Since there is an overflow at t_2, we must have

$$t_2 \Leftrightarrow t_1 < \sum_{D_i \leq t_2 - t_1} \left(1 + \left\lfloor \frac{t \Leftrightarrow D_i}{T_i} \right\rfloor \right) C_i \; + \; (t_2 \Leftrightarrow t_1)U_S.$$

If $t_2 \Leftrightarrow t_1 < L$, let $t = t_2 \Leftrightarrow t_1$ and we are done. More generally, let

$$t = (t_2 \Leftrightarrow t_1) \Leftrightarrow \left\lfloor \frac{t_2 \Leftrightarrow t_1}{L} \right\rfloor L.$$

Of course $t < L$. By the hypothesis we have

$$\begin{aligned} t & < \sum_{D_i \leq t_2 - t_1} \left(1 + \left\lfloor \frac{t_2 \Leftrightarrow t_1 \Leftrightarrow D_i}{T_i} \right\rfloor \right) C_i \; + \; (t_2 \Leftrightarrow t_1)U_S \Leftrightarrow \left\lfloor \frac{t_2 \Leftrightarrow t_1}{L} \right\rfloor W(L) \\ & = \sum_{D_i \leq t_2 - t_1} \left(1 + \left\lfloor \frac{t_2 \Leftrightarrow t_1 \Leftrightarrow D_i}{T_i} \right\rfloor \right) C_i \Leftrightarrow \left\lfloor \frac{t_2 \Leftrightarrow t_1}{L} \right\rfloor \sum_{i=1}^{n} \left\lceil \frac{L}{T_i} \right\rceil C_i + \left(t_2 \Leftrightarrow t_1 \Leftrightarrow \left\lfloor \frac{t_2 \Leftrightarrow t_1}{L} \right\rfloor L \right) U_S \end{aligned}$$

---

[5] For aperiodic instances we must consider the "corrected" arrival time $\max\{a, \bar{d}_{prev}, f_{prev}\}$.

$$\leq \sum_{D_i \leq t_2 - t_1} \left(1 + \left\lfloor \frac{t_2 - t_1 - D_i}{T_i} \right\rfloor - \left\lfloor \frac{t_2 - t_1}{L} \right\rfloor \left\lceil \frac{L}{T_i} \right\rceil \right) C_i \; + \; tU_S$$

$$\leq \sum_{D_i \leq t_2 - t_1} \left(1 + \left\lfloor \frac{t_2 - t_1 - D_i}{T_i} \right\rfloor - \left\lceil \frac{\lfloor \frac{t_2 - t_1}{L} \rfloor L}{T_i} \right\rceil \right) C_i \; + \; tU_S$$

$$\leq \sum_{D_i \leq t_2 - t_1} \left(1 + \left\lfloor \frac{t_2 - t_1 - \lfloor \frac{t_2 - t_1}{L} \rfloor L - D_i}{T_i} \right\rfloor \right) C_i \; + \; tU_S$$

$$= \sum_{D_i \leq t_2 - t_1} \left(1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor \right) C_i \; + \; tU_S .$$

$\square$

Note that during the proof we used the following properties:

- If a is integer, then $a\lceil b \rceil \geq \lceil ab \rceil$.

- $\lfloor a \rfloor - \lceil b \rceil \leq \lfloor a - b \rfloor$.

The previous lemma lets us extend, once again, the necessary and sufficient condition of Equation (2), obtaining the following one:

$$t \geq \sum_{D_i \leq t_2 - t_1} \left(1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor \right) C_i \; + \; tU_S .$$

It is not difficult to see that the previous condition still has to be checked only at the deadlines of all tasks in the busy period. In fact, assume it is violated at time t. Let d be the last deadline before t among all task instances in the schedule. We have

$$
\begin{aligned}
\sum_{D_i \leq d} \left(1 + \left\lfloor \frac{d - D_i}{T_i} \right\rfloor \right) C_i \; + \; dU_S \; &= \; \sum_{D_i \leq t} \left(1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor \right) C_i \; + \; tU_S + (d - t)U_S \\
&> \; t + (d - t)U_S \\
&\geq \; t + (d - t) \\
&= \; d.
\end{aligned}
$$

That is, the condition is also violated at a task deadline.

As expected, also Lemma 4.1 remains valid. Hence the procedure for the evaluation of the worst-case response time of a task is basically the same as in the previous sections. We now have an additional term in the definition of the higher priority workload $W_i(a, t)$ because of the server, which, in the busy period relative to the deadline $d = a + D_i$, can demand at most a computation time $(a + D_i)U_S$. Thus, the new definition of $W_i(a, t)$ is the following

$$W_i(a, t) = \sum_{\substack{j \neq i \\ D_j \leq a + D_i}} \min\left\{ \left\lceil \frac{t}{T_j} \right\rceil , 1 + \left\lfloor \frac{a + D_i - D_j}{T_j} \right\rfloor \right\} C_j \; + \; \delta_i(a, t)C_i \; + \; (a + D_i)U_S .$$

The rest of the computation remains unchanged.

9   Accounting for System Overheads

In this section we extend our analysis in order to take into account the costs of an actual priority pre-emptive scheduler implementation. We start noting that the considerations are very similar to those for a fixed priority scheduler, at least as far as the implementation is concerned. Thus, we will follow exactly the approach described by Tindell *et al.* in [20].

According to them "Tick scheduling is a common way of implementing a priority pre-emptive scheduler: a periodic clock interrupt runs a scheduler which polls for the arrivals of tasks; any arrived tasks are placed in a notional priority ordered run-queue. The scheduler then dispatches the highest priority task on the run-queue." In the most general case, task instances can arrive at any time, and hence can suffer a worst-case release jitter of T_{tick}, the period of the tick scheduler (unless there are periodic tasks with periods multiple of T_{tick}).

Normally, the tick scheduler uses two queues: the *pending queue*, which holds a deadline ordered list of tasks awaiting their start conditions, and the *run queue*, a priority-ordered list of runnable tasks. "At each clock interrupt the scheduler scans the pending queue for tasks which are now runnable and transfers them to the run queue." The system overhead we are going to take into account is the time needed to handle the two queues, and more precisely the time needed to move tasks from one queue to another one.

In particular, as Tindell *et al.* did, we will consider the following implementation costs:

C_{tick} The worst-case computation time of the periodic timer interrupt.

C_{QL} The cost to take the first task from the pending queue.

C_{QS} The cost to take any possible subsequent task from the pending queue.

According to Tindell *et al.*'s analysis, the tick scheduling overheads over a window of width w are

$$OV(w) = T(w)C_{tick} + \min\{T(w), K(w)\}C_{QL} + \max\{K(w) \Leftrightarrow T(w), 0\}C_{QS}, \qquad (7)$$

where $T(w)$ is the number of timer interrupts within the window:

$$T(w) = \left\lceil \frac{w}{T_{tick}} \right\rceil$$

and $K(w)$ is the worst-case number of times tasks move from the pending queue to the run queue:

$$K(w) = \sum_{i=1}^{n} \left\lceil \frac{w + J_i}{T_i} \right\rceil .$$

In order to extend our analysis, we now have to generalize Theorem 3.1. Again, the generalization is achieved looking at the paper of Liu and Layland [11]: we simply have to

reformulate Theorem 8 in order to fulfill our needs. This theorem was proven for a task set scheduled by the deadline driven algorithm, in a system in which the processor time is accumulated by a certain *availability function*, that is, only a fraction of the processor time is devoted to the task schedule. The attention of Liu and Layland was on sublinear functions, that is, functions for which for all t and T

$$f(T) \leq f(t + T) \Leftrightarrow f(t).$$

The reason is that when there is a task set scheduled by fixed priority scheduling and another task set scheduled when the processor is not occupied by tasks of the first set (*i.e.* in background), then the availability function for the second task set can be shown to be sublinear. Our model, in which all tasks are scheduled when the processor is not busy executing tick scheduler code, fits perfectly in this description.

Theorem 9.1 (Liu and Layland) *When the deadline driven scheduling algorithm is used to schedule a set of tasks on a processor whose availability function is sublinear, if there is an overflow for a certain arrival pattern, then there is an overflow without idle time prior to it in the pattern in which all task instances are released as soon as possible (*i.e. in the asap *arrival pattern).*

Proof. Similar to that of Theorem 3.1. □

The feasibility analysis must be modified accordingly. The computation of the busy period length must take into account the additional load due to the tick scheduler. Thus the workload arrived at time t becomes

$$W(t) = OV(t) + \sum_{i=1}^{n} \left\lceil \frac{t + J_i}{T_i} \right\rceil C_i.$$

Equation (7) can be used to evaluate the availability function:

$$a(t) \geq \max\{t \Leftrightarrow OV(t), 0\}.$$

A sufficient condition for the feasibility of the task set is then

$$d \Leftrightarrow OV(d) \geq \sum_{D_i \leq d + J_i} \left(1 + \left\lfloor \frac{d + J_i \Leftrightarrow D_i}{T_i} \right\rfloor \right) C_i$$

for all deadlines in the first busy period. Note that the condition is a generalization of Theorem 9 of [11].

Applying the same argument as for the previous theorem, also Lemma 4.1 can be generalized according to our mixed scheduling model. That is, we can still evaluate the worst-case response times with the usual approach. However, we now have to consider the new availability function. Practically, we only need to modify the definition of $W_i(a, t)$, the higher

Semaphore	Locked by	Time held
1	Task 9	900
2	Task 9	300
2	Task 15	1350
3	Task 6	400
3	Task 10	400
4	Task 3	100
4	Task 9	300
5	Task 11	750
5	Task 15	750

Table 3: List of semaphores and locking pattern for the GAP task set.

priority workload arrived up to time t, taking into account the additional term due to the tick scheduler overheads:

$$W_i(a,t) = \sum_{\substack{j \neq i \\ D_j \leq a + D_i + J_j}} \min\left\{ \left\lceil \frac{t + J_j}{T_j} \right\rceil, 1 + \left\lfloor \frac{a + D_i + J_j \Leftrightarrow D_j}{T_j} \right\rfloor \right\} C_j + \delta_i(a,t)C_i + OV(t).$$

All the rest is unchanged.

10   Case Study

As other authors did, we too have applied our theory to the GAP (Generic Avionics Platform) task set, a small avionics case study described by Locke *et al.* in [12]. There are seventeen tasks in the GAP set, of which all but one are strictly periodic, with periods multiple of T_{tick}, therefore they do not suffer release jitter. Task 11 is a sporadic task, whose arrival is assumed to be polled by the tick scheduler, hence it may suffer a worst-case release jitter equal to T_{tick}. Some tasks also share resources that are accessed by locking and unlocking semaphores according to a hypothetical pattern, which we have assumed to be equal to that described by Tindell *et al.* in [20], and which is reported in Table 3. Similarly, we have also assumed a tick scheduler with the same parameters of their description:

$$C_{tick} = 66\mu s \quad T_{tick} = 1000\mu s \quad C_{QL} = 74\mu s \quad C_{QS} = 40\mu s.$$

According to Tindell *et al.*'s approach [20], there is no optimal priority assignment able to guarantee all tasks under a fixed priority system. Vice versa, according to our analysis, the GAP task set is indeed feasible under EDF scheduling. The worst-case response times computed with the theory described in the paper are reported in Table 4, where all times are given in microseconds. Note that all tasks with the same relative deadline have the same worst-case response time.

i	D_i	T_i	C_i	B_i	J_i	r_i	a_i
1	5000	200000	3000	0	0	4180	0
2	25000	25000	2000	300	0	12280	0
3	25000	25000	5000	300	0	12280	0
4	40000	40000	1000	300	0	20226	40000
5	50000	50000	3000	400	0	30226	30000
6	50000	50000	5000	400	0	30226	30000
7	59000	59000	8000	400	0	39226	21000
8	80000	80000	9000	1350	0	60226	0
9	80000	80000	2000	1350	0	60226	0
10	100000	100000	5000	1350	0	74150	0
11	200000	200000	1000	1350	1000	168558	0
12	200000	200000	3000	0	0	168558	0
13	200000	200000	1000	0	0	168558	0
14	200000	200000	1000	0	0	168558	0
15	200000	200000	3000	0	0	168558	0
16	1000000	1000000	1000	0	0	198760	0
17	1000000	1000000	1000	0	0	198760	0

Table 4: GAP task set parameters.

11 Conclusions

In this paper a new flexible approach for analysing deadline pre-emptive scheduled real-time systems has been proposed. Both the feasibility assessment and the evaluation of worst-case response times of the given task set have been tackled. The analysis has been first described for a computational model assuming periodic and sporadic tasks with arbitrary deadlines, and later extended to include sporadically periodic tasks (*i.e.* bursty tasks) with release jitter and shared resources. Servers for soft aperiodic scheduling and costs of a real tick driven implementation have also been included. The resulting analysis is a generalization of previously known results. The avionic case study shown in Section 10 has confirmed that deadline scheduling achieves higher processor utilization than fixed priority scheduling.

The evaluation of worst-case response times, in a model that assumes release jitter, has been shown as a fundamental tool for the analysis of distributed systems in which local processors use a fixed priority pre-emptive schedulers [21, 22]. We believe that the same argument of the *holistic schedulability analysis* can now also be applied to systems with local deadline schedulers. This is the subject of current research and will be described in a future paper.

References

[1] Audsley N., Burns A., Richardson M., Tindell K., and Wellings A.J., "Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling," *Software Engineering Journal*, September 1993.

[2] Baker T.P., "Stack-Based Scheduling of Real-time Processes," *The Journal of Real-Time Systems* **3**, pp. 67-99, 1991.

[3] Baruah S.K., Rosier L.E., and Howell R.R., "Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor," *The Journal of Real-Time Systems* **2**, 1990.

[4] Chen M. and Lin K., "Dynamic Priority Ceilings: A Concurrency Control Protocol for Real-Time Systems," *The Journal of Real-Time Systems* **2**, pp. 325-346, 1990.

[5] Dertouzos M.L., "Control Robotics; the Procedural Control of Physical Processes," *Information Processing* **74**, 1974.

[6] Ghazalie T.M. and Baker T.P., "Aperiodic Servers in a Deadline Scheduling Environment," *The Journal of Real-Time Systems* **9**, pp. 31-67, 1995.

[7] Katcher D.I., Lehoczky J.P., and Strosnider J.K., "Scheduling Models of Dynamic Priority Schedulers," Research Report CMUCDS-93-4, Carnegie Mellon University, Pittsburgh, April 1993.

[8] Lehoczky J.P., Sha L., and Strosnider J.K., "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments," *Proc. of the 8th IEEE Real-Time Systems Symposium*, December 1987.

[9] Lehoczky J.P., "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines," *Proc. of the 11th IEEE Real-Time Systems Symposium*, December 1990.

[10] Lehoczky J.P. and Ramos-Thuel S., "An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems," *Proc. of the 13th IEEE Real-Time Systems Symposium*, December 1992.

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

[12] Locke C.D., Vogel D.R., and Mesler T.J., "Building a Predictable Avionics Platform in Ada: A Case Study," *Proc. of the IEEE Real-Time Systems Symposium*, 1991.

[13] Ramamritham K., "Dynamic Priority Scheduling," in *Real-Time Systems: Specification, Verification and Analysis*, edited by Mathai Joseph, Prentice Hall, November 1995.

[14] Sha L., Rajkumar R. and Lehoczky J.P., "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Transactions on Computers* **39**(9), September 1990.

[15] Sprunt B., Sha L., and Lehoczky J.P., "Aperiodic Task Scheduling for Hard-Real-Time Systems," *The Journal of Real-Time Systems* **1**, pp. 27-60, 1989.

[16] Spuri M. and Buttazzo G., "Efficient Aperiodic Service under Earliest Deadline Scheduling," *Proc. of the 15th IEEE Real-Time Systems Symposium*, December 1994.

[17] Spuri M., "Earliest Deadline Scheduling in Real-Time Systems," Doctorate dissertation, Scuola Superiore S.Anna, Pisa, 1995.

[18] Spuri M., Buttazzo G., and Sensini F., "Robust Aperiodic Scheduling under Dynamic Priority Systems," *Proc. of the 16th IEEE Real-Time Systems Symposium*, December 1995.

[19] Spuri M. and Buttazzo G., "Scheduling Aperiodic Tasks in Dynamic Priority Systems," *The Journal of Real-Time Systems*, to appear.

[20] Tindell K., Burns A., and Wellings A.J., "An Extendible Approach for Analysing Fixed Priority Hard Real-Time Tasks," *The Journal of Real-Time Systems* **6**(2), 1994.

[21] Tindell K. and Clark J., "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems," *Microprocessors and Microprogramming* **40**, 1994.

[22] Tindell K., Burns A., and Wellings A.J., "Analysis of Hard Real-Time Communications," *The Journal of Real-Time Systems* **9**, 1995.

A Busy Period Properties

A.1 Convergence and Complexity of Iterative Formulae

Both in the feasibility analysis and in the evaluation of worst-case response times described in the paper, we compute busy period lengths by means of iterative formulae. We previously stated that the convergence of these formulae in a finite number of steps is ensured by the condition

$$\sum_{i=1}^{n} \frac{C_i}{T_i} + U_S \leq 1,$$

which is our first requirement, because if the condition does not hold the task set is surely not feasible.

We now prove the statement for the computation of the busy period length L in our model of Section 8, namely, Equation (6). First note that the workload function $W(t)$ is a
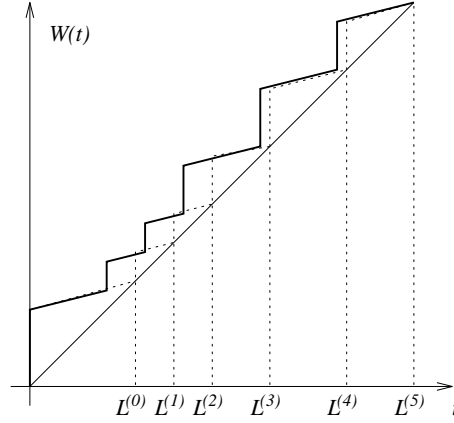
Figure 6: Workload function and busy period length computation.

non decreasing function, with $W(0^+) > 0$, as shown in Figure 6. Let H be the so called *hyperperiod* of the tasks, *i.e.* the least common multiple of their periods. We have

$$W(H) = \sum_{i=1}^{n} \left\lceil \frac{H}{T_i} \right\rceil C_i \ + \ H U_S = \sum_{i=1}^{n} H \frac{C_i}{T_i} \ + \ H U_S = H \left(\sum_{i=1}^{n} \frac{C_i}{T_i} \ + \ U_S \right) \leq H,$$

from which we can conclude that

$$L \leq H.$$

Furthermore, at each step $L^{(m+1)}$ is computed by solving the equation

$$W \left(L^{(m)} \right) + \left(t \Leftrightarrow L^{(m)} \right) U_S = t,$$

whose solution is

$$t = \frac{W \left(L^{(m)} \right) \Leftrightarrow L^{(m)} U_S}{1 \Leftrightarrow U_S}.$$

By looking at Figure 6, where a graphical representation of the computation is depicted, we can easily realize that either $L^{(m+1)}$ is equal to $L^{(m)}$, in which case we halt, or it is equal to $L^{(m)}$ plus at least $\frac{C_{\min}}{1-U_S}$, where $C_{\min}$ is the minimum task computation time. Hence in a finite number of steps the iteration reaches the final value of L.

The same argument can be used to prove that the time complexity of the computation of L is pseudo-polynomial, when

$$\sum_{i=1}^{n} \frac{C_i}{T_i} + U_S \leq p,$$

for some constant $p < 1$. We only need to find a tighter upper bound for L:

$$L = \sum_{i=1}^{n} \left\lceil \frac{L}{T_i} \right\rceil C_i + LU_S \leq \sum_{i=1}^{n} \left(1 + \frac{L}{T_i} \right) C_i + LU_S = \sum_{i=1}^{n} C_i + L \left(\sum_{i=1}^{n} \frac{C_i}{T_i} + U_S \right),$$

from which we obtain

$$L \leq \frac{\sum_{i=1}^{n} C_i}{1 \Leftrightarrow \left(\sum_{i=1}^{n} \frac{C_i}{T_i} + U_S \right)} \leq \frac{\sum_{i=1}^{n} C_i}{1 \Leftrightarrow p}.$$

Since each step of the iterative formula takes $O(n)$ time, the whole computation then takes $O(n \sum_{i=1}^{n} C_i)$ time, which is, as claimed, a psuedo-polynomial complexity. Note that this also implies a pseudo-polynomial time complexity for both the procedures of feasibility assessment and worst-case response times evaluation. Whether a full polynomial time algorithm exists for the feasibility assessment problem is still an open question [3].

A.2   Maximum Busy Period Length

Using again the 'shift' argument, we can easily prove that the length L of the first busy period in the schedule of the *asap* arrival pattern is the maximum length of any possible busy period in any schedule. Consider a busy period in the schedule of a given arrival pattern. Let L' be its length. The beginning of the period must coincide with the release of a task instance. If we ideally shift left all other subsequent task instances, so as to obtain an *asap* arrival pattern starting at the beginning of the busy period, the workload arrived within the period can only increase, *i.e.* the length of the new busy period cannot be less than L'. The busy period obtained in this way is exactly the first one in the schedule of the *asap* arrival pattern, that is, $L' \leq L$.

B   Full Analysis Description

In this appendix we fully describe the analysis for a system with the following characteristics:

- the task set is modelled by sporadically periodic arrival laws;

- all tasks can experience release jitter and can share resources by locking semaphores, according to a specific concurrency control protocol like the Dynamic Priority Ceiling or the Stack Resource Policy;

- a Total Bandwidth server with processor utilization U_S is included for handling either soft aperiodic tasks or firm aperiodic tasks (in a more general situation we could have more than one server, however, the analysis would not differ significantly);

- a tick scheduler with relative run-time costs is also considered.

We also assume that

$$\frac{C_{tick}}{T_{tick}} \; + \; \sum_{i=1}^{n} \frac{C_i}{T_i} \; + \; U_S \; \leq \; 1,$$

which is a necessary condition for the feasibility of the task set. Both the feasibility analysis and the computation of the worst-case response times are summarized.

B.1   Feasibility Analysis

The analysis proceeds in two steps: firstly we compute the length of the initial busy period in the schedule of the *asap* arrival pattern, and then we check all deadlines in the period. The busy period length L is computed by means of the following iterative formula:

$$\begin{cases} L^{(0)} & = & \frac{1}{1-U_S} \sum_{i=1}^{n} C_i, \\ L^{(m+1)} & = & \frac{W(L^{(m)}) - L^{(m)} U_S}{1-U_S}, \end{cases}$$

where $W(t)$ is the cumulative workload arrived before time t:

$$W(t) = OV(t) \; + \; \sum_{i=1}^{n} I_i(t)C_i \; + \; tU_S.$$

$I_i(t)$ is the number of instances of task i arrived and released by time t:

$$I_i(t) = \left\lfloor \frac{t+J_i}{T_i} \right\rfloor n_i + \min \left\{ n_i, \; \left\lceil \frac{t+J_i \Leftrightarrow \left\lfloor \frac{t+J_i}{T_i} \right\rfloor T_i}{t_i} \right\rceil \right\}.$$

$OV(t)$ is the overhead of the tick scheduler:

$$OV(t) = T(t)C_{tick} + \min\{T(t), K(t)\}C_{QL} + \max\{K(t) \Leftrightarrow T(t), 0\}C_{QS},$$

where $T(t)$ is the number of timer interrupts within time t:

$$T(t) = \left\lceil \frac{t}{T_{tick}} \right\rceil.$$

and $K(t)$ is the worst-case number of times tasks move from the pending queue to the run queue:

$$K(t) = \sum_{i=1}^{n} I_i(t).$$

Once determined L, the feasibility of the task set is established by checking for any deadline $d \leq L$ in the *asap* arrival pattern the following condition:

$$d \Leftrightarrow OV(d) \geq \sum_{D_i \leq d+J_i} H_i(d)C_i \; + \; dU_S \; + \; B_{k(d)},$$

where $H_i(t)$ is the number of instances of task i with deadline before or at t:

$$H_i(t) = \left\lfloor \frac{t + J_i \Leftrightarrow D_i}{T_i} \right\rfloor n_i + \min\left\{ n_i, 1 + \left\lfloor \frac{t + J_i \Leftrightarrow D_i \Leftrightarrow \left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor T_i}{t_i} \right\rfloor \right\} ,$$

and

$$k(d) = \max\{k : D_k \Leftrightarrow J_k \le d\}.$$

B.2 Worst-Case Response Times

The worst-case response time of task i is computed with the following formula:

$$r_i = \max_{a \in [-J_i, L - C_i - J_i - B_i)} \{r_i(a)\} ,$$

where $r_i(a)$ is the response time relative to the integer a:

$$r_i(a) = \max\{C_i + J_i + B_i, L_i(a) \Leftrightarrow a\} .$$

$L_i(a)$ is the length of the busy period relative to the deadline $d = a + D_i$, and it is computed by means of the following iterative formula:

$$\begin{cases} L_i^{(0)}(a) & = \sum\limits_{\substack{j \ne i \\ D_j \le a + D_i + J_j}} C_j + I_{\{s_i(a)=0\}} C_i, \\ L_i^{(m+1)}(a) & = W_i\left(a, L_i^{(m)}(a)\right) + B_{k(a+D_i)}, \end{cases}$$

where

$$I_{\{s_i(a)=0\}} = \begin{cases} 1 & \text{if } s_i(a) = 0, \\ 0 & \text{otherwise.} \end{cases}$$

$s_i(a)$ is the release time of the first instance of task i in the arrival pattern considered:

$$s_i(a) = S_i(a) \Leftrightarrow \min\left\{ n_i \Leftrightarrow 1, \left\lfloor \frac{S_i(a)}{t_i} \right\rfloor \right\} t_i,$$

where

$$S_i(a) = a + J_i \Leftrightarrow \left\lfloor \frac{a + J_i}{T_i} \right\rfloor T_i.$$

Finally, $W_i(a, t)$ is the higher priority workload relative to deadline $d = a + D_i$, released before time t:

$$W_i(a,t) = \sum_{\substack{j \ne i \\ D_j \le a + D_i + J_j}} \min\{I_j(t), H_j(a + D_i)\} C_j + \delta_i(a,t) C_i + (a + D_i) U_S + OV(t),$$

where

$$\delta_i(a,t) = \begin{cases} \min\{I_i(t \Leftrightarrow s_i(a)), H_i(a + D_i)\} & \text{if } t > s_i(a), \\ 0 & \text{otherwise.} \end{cases}$$