QoS Scheduling Components based on Firm Real-time Requirements

Audrey Marchand and Maryline Silly-Chetto
LINA (Laboratoire d'Informatique de Nantes Atlantique)
Rue Christian Pauc
44306 Nantes cedex 03 France
{audrey.marchand, maryline.chetto}@iut-nantes.univ-nantes.fr

Abstract

In the last years, multimedia and feedback control applications have emphasized the importance of appropriate resource allocation policies in real-time systems, introducing the problem of controlling and adapting the quality of service (QoS) provided by an application. The basic idea is then to avoid overload situations by scaling down the applications' resource requirements, while sustaining a specified QoS level for the system. In this paper, we are interested in the problem of dynamic QoS driven CPU allocation for hybrid sets of tasks, consisting of firm periodic tasks (i.e, tasks allowing occasional skips of instances) and soft aperiodic requests. The approach seeks to establish a compromise between minimizing the response time of aperiodic requests and maximizing the QoS of periodic tasks. First, we briefly present the library of free software components developed within the French National project CLEOPATRE [1]*. Then, we focus on the functioning of the QoS management component. Finally, we present some simulation results to underline the interest of such a scheduling scheme for real-time applications' developers.*

1. Introduction

A real-time system is a system in which the correctness of a computation depends not only on obtaining the right result, but also upon providing the result on time.

To date, solutions were widely proposed for scheduling hard real-time tasks [11]. However, some types of time-sensitive applications have gained momentum in the last few years with a particular focus on control and multimedia fields. In those emergent applications, real-time constraints are firm, *i.e.*, occasional deadline violations entail only QoS degradations without any danger for the integrity of the system. As a matter of fact, we have to observe that some degree of missed deadlines can be tolerated by periodic tasks, without jeopardizing the whole behavior of the real-time system. For instance, it is not necessary to decode every frames within a video streaming application, as long as the decoding rate do not overcome the threshold of human perception. Therefore it is important for a system to accommodate specific user quality requirements and delivery as good service as it can from the users' perspective. In this context, QoS schedulers are worth considering compared with conventional solutions, since they provide a well-defined and controllable behavior of the system according to quantitatively measurable parameters.

In this work, we present a programming model suitable to real-time applications formed by periodic tasks which tolerate some deadline misses, and soft aperiodic tasks (*i.e.*, tasks with no deadline). The paper is divided into 9 sections. The next section reports some works in the area of firm real-time requirements. Section 3 presents the CLEOPATRE project with its library of free software components. Section 4 talks about the Skip-Over job skipping model. Section 5 introduces the EDL (Earliest Deadline as Late as possible) aperiodic tasks server. Section 6 and 7 describe the joint scheduling of skippable periodic tasks and soft aperiodic requests, using the EDL server applied to RTO and BWP tasks respectively. Section 8 shows the performance results using such a hybrid scheme, in terms of the minimization of the aperiodic responsiveness. Finally, conclusions and future work are discussed in Section 9.

2. Related work

In the last years, the need for some QoS requirements attached to a real-time application has emerged. Different strategies allow a negotiated level of service to be offered for each of the tasks in the system.

In [8] the authors extended the conventional concept of *firm* deadline to the definition of *(m,k)-firm* deadline, where each periodic task must have at least m deadlines met in each window of k activations. Koren and Shasha [9] pro-

[1] work supported by the French research office, grant number 01 K 0742

posed the *skip factor* constraint for dealing with overloaded systems. If a task has a skip factor of s, it will have one invocation skipped out of s. It is a particular case of the *(m,k)-firm* model [8] where $m = k - 1$. They reduce the overload by skipping some task invocations, thus exploiting skips to increase the feasible periodic load. This approach gives a solution to the scheduling problem of overloaded systems, while representing a system Quality of Service requirement for real-time applications. Broadly speaking, the Skip-Over scheduling algorithms guarantee the timing correctness of the real-time application. One interesting result is that making optimal use of skips is a NP-hard problem.

In [3, 4], Caccamo and Buttazzo follow this work by scheduling hybrid task sets consisting of skippable periodic and soft aperiodic tasks. They propose and analyse an algorithm, based on a variant of Earliest Deadline First (EDF) scheduling, in order to exploit skips under the Total Bandwith Server (TBS).

West and Poellabauer in [14] proposed a *windowed lost rate*, that specifies a task can tolerate x deadlines missed over a finite range or window, among consecutive y instances. In [2], Bernat *et al.* introduce a general framework for specifying tolerance of missed deadlines under the definition of *weakly hard* constraints.

The scope of the paper is here to minimize the response time of the soft aperiodic tasks under the EDL server, in the presence of periodic tasks meeting Skip-Over requirements. The main emphasis of the paper is on the specification of the EDL (Earliest Deadline as Late as possible) server under Skip-Over constraints. Our approach tends to distribute the spare time saved by skips for enhancing the response time of aperiodic requests.

3. The CLEOPATRE project

3.1. Scientific issues

The emergence of more and more complex industrial applications pointed out the the need of academic research transfer as far as real-time computing is concerned. In this context, the objective of the French national project CLEOPATRE (Software Open Components on the Shelf for Embedded Real-Time Applications) [1] is to integrate validated research results within the Real-time Linux extension named Linux/RTAI (Real-Time Application Interface). This project tends to bring solutions for the development of real-time embedded applications, offering generic open-source software components. CLEOPATRE will provide one framework which permits to develop hard, soft and firm applications using a library of selectable components dedicated to dynamic scheduling, aperiodic tasks service, resource control access, fault-tolerance and QoS management.

3.2. The COTS library

New service abstractions and interfaces have been developed as COTS (Commercial-Off-The-Shelf) components, in order to provide more efficient and better service to real-time applications. The kernel is fully modular in terms of scheduling policies, aperiodic servers, and concurrency control protocols (see Figure 1). All modules are dynamically loadable, thus allowing the user to easily fulfill his specific needs for the development of his real-time application. The modified Linux/RTAI kernel appears at the low-level. Note that the proceeded changes keep the compatibility with the Linux/RTAI native kernel applications.

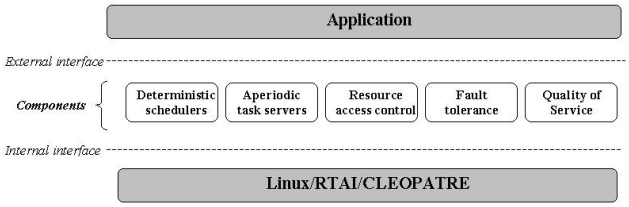

Figure 1. The CLEOPATRE library

Scheduling components offer various scheduling algorithms. The first one is the Deadline Monotonic (DM) algorithm based on fixed priority, which executes first the task with the shortest deadline. It is optimal among fixed priority schedulers. The second one is the Earliest Deadline First (EDF) algorithm based on dynamic priority, which executes tasks in the order of their absolute deadline. It is optimal among all scheduling algorithms, outperforming DM on the criterion of schedulable utilization.

Concerning the shelf that provides aperiodic tasks service, three aperiodic task server components have been implemented, in order to avoid timing faults on aperiodic requests. The first one, called the Background server, schedules aperiodic tasks when no periodic activity is present (*i.e.*, no periodic tasks are ready for execution). It guarantees that aperiodic tasks will not affect the periodic tasks behavior. Another server is the Total Bandwidth server (TBS) [5], which gives a virtual deadline to aperiodic requests according to the system load, at their arrival time. TBS was proved to be an optimal server. Finally, the Earliest Deadline as Late as possible server (EDL) [6, 13], is a dynamic slack stealer in the sense that, when an aperiodic task occurs, it dynamically computes the idle times available for aperiodic tasks when the periodic tasks are scheduled as late as possible. When no aperiodic activity is present, periodic tasks are scheduled as soon as possible according to the EDF algorithm. The EDL server has been proved to be optimal [13].

Five resource management protocols are available: FIFO (First In First Out), Priority, SPP (Super Priority Protocol),

PIP (Priority Inheritance Protocol) and PCP (Priority Ceiling Protocol)[12]. In the SPP protocol, all critical tasks holding a resource are scheduled non-preemptively. According to the PIP protocol, when a task holds a resource, thus blocking another priority task, it temporarily inherits the priority of the blocked task until it releases the resource [12]. The PCP protocol represents an enhancement of the PIP protocol, since it prevents deadlocks while reducing blocking time.

Two fault-tolerance mechanisms have been implemented. The Deadline Mechanism is an error recovery method, where each fault-tolerant task is characterized by a primary and a backup version [10]. Whenever the primary part fails, the scheduler executes the backup process. The Imprecise Computation model [7] is a fault-tolerant method, especially adapted to iterative programs where tasks are made up of mandatory and optional parts. Whenever the mandatory instance completes successfully, the optional instance is activated in order to refine the result produced.

In this paper, our work focuses on the provision of QoS safe mechanisms at the kernel-level of Linux/RTAI. Namely, we provide QoS guarantees for real-time applications. These QoS components rely on the *RTO (Red Tasks Only)* and *BWP (Blue When Possible)* algorithms of the Skip-Over model [9].

3.3. Open-source distribution project

Although cost reduction is one of the primary reasons to go with an open-source solution, there are many other compelling reasons for end users as well as software developers. Nowadays, open-source softwares are more and more attractive: there is no licence fee charged for the software, the source code is freely available, and there are no restrictions on the use of the software, even for commercial purposes. CLEOPATRE components will be delivered as open-source components under the GNU Lesser General Public License (LGPL). This license applies to proprietary solutions, and is quite different from the ordinary General Public License. Indeed, it can be used for certain libraries in order to permit linking those libraries into non-free programs. Consequently, a program using a LGPL library is not subjected to the LGPL conditions, thus encouraging the adoption and distribution of open-source libraries within the industrial community.

4. Scheduling Qos periodic tasks

We are now interested in the problem of QoS driven CPU allocation to periodic tasks which allow occasional deadline violations (*i.e.*, skippable periodic tasks), on a uniprocessor system.

4.1. Terminology and assumptions

We consider a system consisting of periodic tasks which can be preempted and do not have precedence constraints. The i-th task is denoted as T_i. The period of T_i is denoted by p_i. We assume that its relative deadline d_i is equal to its period. The worst-case execution time of T_i is represented by c_i. Finally, the tolerance of task T_i to missing deadline is denoted by a skip parameter s_i, $2 \leq s_i \leq \infty$. The distance between two consecutive skips must be at least s_i periods. When s_i=∞, no skips are allowed and T_i is equivalent to a hard periodic task.

A task T_i is divided into instances where each instance occurs during a single period of the task. Every instance of a task can be *red* or *blue*. These are the colors used by Koren and Shasha in [9]. A red task instance must complete before its deadline; a blue task instance can be aborted at any time. When a task misses its deadline, we say that the task (or deadline) instance was *skipped*. The fact that $s_i \geq 2$ implies that, if an instance is skipped, then the next $s_i - 1$ instances will be red.

4.2. QoS scheduling algorithms

The algorithm proposed in [9] is the *Red Tasks Only (RTO)* algorithm. The red instances are scheduled according to EDF, while the blue ones are always rejected. In the deeply red model where all tasks are synchronously activated and the first $s_i - 1$ instances of every task T_i are red, this algorithm is optimal. The distance between every two skips is exactly s_i periods.

The second algorithm described by Koren and Shasha is the *Blue When Possible (BWP)* algorithm which is an improvement of the first one. Indeed, BWP schedules blue instances whenever their execution does not prevent the red ones from completing within their deadlines. In that sense, it operates in a more flexible way. If there are no ready red tasks, then the blue instances are dispatched according different heuristics (*e.g.*, any one, the blue task with the earliest deadline, the blue task with the latest deadline).

4.3. Example

Consider the set $\mathcal{T}$=$\{T_1, T_2\}$ of two basic periodic tasks $T_1(1, 2)$ and $T_2(4, 6)$. In this case, the effective utilization factor is U_p=$\frac{1}{2}$+$\frac{4}{6}$=1.17 > 1. The system is overloaded, but tasks can be schedulable if T_1 skips one instance every 3, as illustrated on Figure 2 where s_1=3 and s_2=∞.

In the following section, we take an interest in the description of the EDL mechanism and we report some results from investigations into the problem of making optimum use of the remaining processor idle time in scheduling periodic tasks as late as possible.

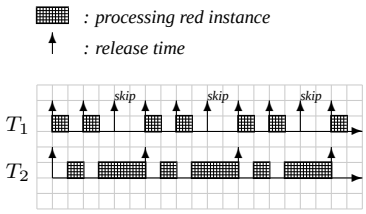

Figure 2. Skip-Over RTO example

5. Scheduling aperiodic tasks

5.1. Definition of the EDL server

The EDL *(Earliest Deadline as Late as possible)* server [6] is a dynamic scheduling algorithm which is able to deal with the arrival of aperiodic tasks. It consists in processing the periodic tasks as soon as possible when no aperiodic activity is present. Whenever an aperiodic request occurs, all periodic tasks are executed as late as possible, while ensuring that all deadlines are met. In other words, the EDL algorithm takes advantage of the effective laxity (*i.e.*, the interval between the completion time and the deadline) of periodic tasks to minimize the aperiodic response time.

In [6], Chetto and Chetto presented a simple way for determining the localization and duration of idle times in any window of a schedule produced by EDL. Let us introduce the terminology used by the authors. With f_Y^X they denote the availability function defined with respect to a task set Y and a scheduling algorithm X,

$$f_Y^X(t) = \begin{cases} 1 & \text{if the processor is idle at } t \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

For any instants t_1 and t_2, the integral $\int_{t_1}^{t_2} f_Y^X(t)dt$ gives the total units of idle times in the time interval $[t_1, t_2]$, denoted by $\Omega_Y^X(t_1, t_2)$. It has been proved [13] that the EDL server requires on-line computation of the processor idle time only at time instants corresponding to the arrival of a new aperiodic task. Moreover, the following result was proved: the start time of every idle period coincides with the deadline of a periodic request. Formulae for computing the static and dynamic idle time intervals are given in section 5.2 and 5.3 respectively. The fundamental property of EDL [6] is that it guarantees the maximum idle time in a given interval for any set of periodic tasks.

Theorem 5.1 *Let X be any preemptive scheduling algorithm. For any instant t,*

$$\Omega_{\mathcal{T}}^X(0,t) \leq \Omega_{\mathcal{T}}^{EDL}(0,t) \quad (2)$$

Another main result relies on the complexity of the determination of the idle times duration. As proved in [13], the on-line computation is performed in $O(\lfloor \frac{R}{p} \rfloor n)$ where n denotes the number of periodic tasks, R the longest deadline and p the shortest period.

In the next sections, we are interested in determining the localization and duration of idle times. Both static and dynamic cases are studied. Formulae and comments are illustrated using the set $\mathcal{T}$=$\{T_1, T_2\}$ of two periodic tasks $T_1(3, 10)$ and $T_2(3, 6)$.

5.2. Static idle time determination

First, we consider a set of n periodic tasks denoted by $\mathcal{T}$=$\{T_1, ..., T_n\}$ on a uniprocessor system. Each task is characterized by its worst-case computation time c_i and a period p_i. Let P=lcm$(p_1, p_2, ..., p_n)$ the hyperperiod be equal to the least common multiple of the task periods. $f_{\mathcal{T}}^{EDL}$ can be represented by means of two vectors: the *static deadline vector* $\mathcal{K}$ and the *static idle time vector* $\mathcal{D}$. $\mathcal{K}$=$(k_0, k_1, ..., k_i, k_{i+1}, ..., k_q)$ represents the times at which idle times occur within $[0, P]$ and is constructed from the distinct deadlines of periodic tasks. We have k_0=0, k_q=P, and $k_i < k_{i+1}$ for all $1 \leq i \leq n$. $\mathcal{D}$=$(\Delta_0, \Delta_1, ..., \Delta_i, \Delta_{i+1}, ..., \Delta_q)$ is used to gather the lengths of the idle times, with Δ_i representing the length of the idle time that starts at time k_i. $\mathcal{D}$ can be obtained thanks to the recurrent formula [13]:

$$\Delta_q = 0 \quad (3)$$

$$\Delta_i = \max(0, F_i), \quad i = q-1 \textit{ down to } 0 \quad (4)$$

where F_i=$(P - k_i) - \sum_{j=1}^{n} \lceil \frac{P-k_i}{p_j} \rceil c_j - \sum_{k=i+1}^{q} \Delta_k$

From formulae 3 and 4 applied to the set of basic periodic tasks $\mathcal{T}$ previously introduced, we can carry out the *static deadline vector* $\mathcal{K}$=$(0, 6, 10, 12, 18, 20, 24)$ and the *static idle time vector* $\mathcal{D}$=$(3, 0, 0, 2, 0, 1, 0)$. The computation of the static idle time intervals gives us the EDL schedule represented in Figure 3.

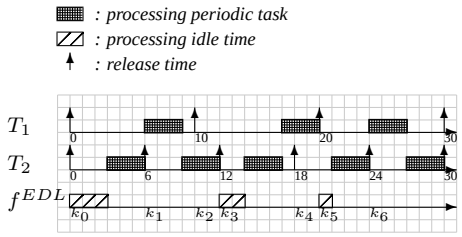

Figure 3. EDL schedule produced at time zero

5.3. Dynamic idle time determination

Let τ be the current time which coincides with the arrival of an aperiodic task. Each periodic task T_j is characterized by its static parameters c_j and p_j, by its dynamic execution

time $A_j(\tau)$ and the deadline d_j of its current request at time τ. Let index h such that k_h=max $\{d; d \in \mathcal{K}$ et $d < \tau\}$. Let M be the greatest deadline of current aperiodic request and then index f such that k_f=min $\{k_i; k_i > M\}$. $f^{EDL}_{\mathcal{T}(\tau)}$ can be represented by means of two vectors: the *dynamic deadline vector* $\mathcal{K}(\tau)$=$(\tau, k_{h+1}, ..., k_i, ..., k_q)$ and the *dynamic idle time vector* $\mathcal{D}(\tau)$=$(\Delta_h(\tau), \Delta_{h+1}(\tau), ..., \Delta_i(\tau), ..., \Delta_q(\tau))$ with $h < i \leq q$ which represents the length of the idle time that follows time k_i. $\mathcal{D}(\tau)$ is then completely defined by the following relations [6]:

$$\Delta_i(\tau) = \Delta_i, \quad i = q \text{ down to } f \tag{5}$$

$$\Delta_i(\tau) = \max(0, F_i(\tau)), \quad i = f-1 \text{ down to } h+1 \tag{6}$$

where $F_i(\tau)$=$(P - k_i) - \sum_{j=1}^{n} \lceil \frac{P-k_i}{p_j} \rceil c_j + \sum_{\substack{j=1 \\ d_j > k_i}}^{n} A_j(\tau)$

$$- \sum_{k=i+1}^{q} \Delta_k(\tau)$$

and $\Delta_h(\tau) = (P - \tau) - \sum_{j=1}^{n} (\lceil \frac{P-\tau}{p_j} \rceil c_j - A_j(\tau))$

$$- \sum_{k=h+1}^{q} \Delta_k(\tau)$$

We can now apply those computational results to the same task set $\mathcal{T}$. This set of tasks is scheduled as soon as possible according to the EDF (Earliest Deadline First) algorithm up to time 5. We suppose that an aperiodic request occurs at time τ=5, involving an on-line computation of the idle times. Assume that we wish to compute the maximum processing time available within $[5, 30]$ for $\mathcal{T}$. From formulae 5 and 6, we carry out the *dynamic deadline vector* $\mathcal{K}(5)$=$(5, 6, 10, 12, 18, 20, 24)$ and the *dynamic idle time vector* $\mathcal{D}(5)$=$(3, 2, 0, 2, 0, 1, 0)$, as illustrated in Figure 4.

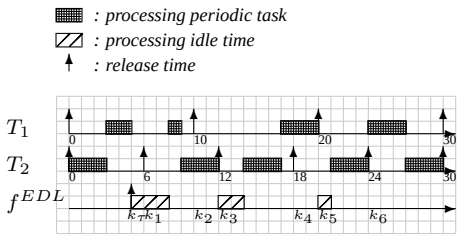

Figure 4. EDL schedule produced at time 5

6. EDL together with RTO tasks $T_i(c_i, p_i, s_i)$

6.1. Static case study

Let P=lcm$(s_1p_1, s_2p_2, ..., s_np_n)$. $\mathcal{K}$ represents all the instants in the hyperperiod $[0, P]$, leading up to an idle time. It is constructed from the distinct deadlines of *red* instances. The distance between two consecutive skips is exactly s_i periods in which we can observe m red instances followed by a single blue instance, with m=$(s_i - 1)$. Those m red instances, individually considered are periodic with a period equal to $s_i.p_i$ and a deadline equal to p_i. Consequently, k_i instants are defined as follows:

$$k_i = (x.s_i + k).p_i \tag{7}$$

with x=0, ..., $(\frac{P}{s_i.p_i} - 1)$, k=1, ..., m, and $k_i < k_{i+1}$, k_0=min $\{p_i; 1 \leq i \leq n\}$, k_q=P − min $\{p_i; 1 \leq i \leq n\}$.

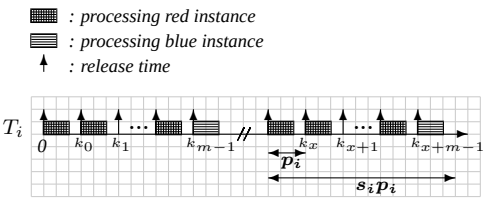

Figure 5. Time instants $k_i(x, k)$ under RTO

$\mathcal{D}$=$(\Delta_0, \Delta_1, ..., \Delta_i, \Delta_{i+1}, ..., \Delta_q)$ represents the length of idle times starting at time instants k_i. It can be obtained thanks to the following recurrent formulae, taking skips into account:

$$\Delta_q = \min\{p_i; 1 \leq i \leq n\} \tag{8}$$

$$\Delta_i = \max(0, F_i), \quad \textit{for } i = q-1 \textit{ down to } 0 \tag{9}$$

with F_i=$(P - k_i) - \sum_{j=1}^{n} (\lceil \frac{P-k_i}{p_j} \rceil - \lceil \frac{P-k_i}{p_j.s_j} \rceil) c_j - \sum_{k=i+1}^{q} \Delta_k$

6.2. Dynamic case study

$\mathcal{K}(\tau)$=$(\tau, k_{h+1}, ..., k_i, ..., k_q)$ gathers the instants greater or equal than τ (current time), leading up to an idle time. As for the static case, all the k_i correspond to the distinct deadlines of *red* tasks.

$\mathcal{D}(\tau)$=$(\Delta_h(\tau), \Delta_{h+1}(\tau), ..., \Delta_i(\tau), ..., \Delta_q(\tau))$ represents the length of the idle times associated to vector $\mathcal{K}(\tau)$. Every periodic task T_i is characterized by the processing time $A_i(\tau)$ allocated to its current instance at time τ. This instance has also a dynamic deadline denoted by d_i. Let M be the greatest deadline of the current periodic request and the index f such as k_f=min $\{k_i; k_i > M\}$. $\mathcal{D}(\tau)$ is completely defined by the following recurrent relations:

$$\Delta_i(\tau) = \Delta_i, \quad \textit{for } i = q \textit{ down to } f \tag{10}$$

$$\Delta_i(\tau) = \max(0, F_i(\tau)), \quad \textit{for } i = f-1 \textit{ down to } h+1 \tag{11}$$

with $F_i(\tau)$=$(P - k_i) - \sum_{j=1}^{n} (\lceil \frac{P-k_i}{p_j} \rceil - \lceil \frac{P-k_i}{p_j.s_j} \rceil) c_j$

$$+ \sum_{\substack{j=1 \\ d_j > k_i}}^{n} A_j(\tau) - \sum_{k=i+1}^{q} \Delta_k(\tau)$$

and $\Delta_h(\tau) = (P - \tau) - \sum_{j=1}^{n} (\lceil \frac{P-\tau}{p_j} \rceil - \lceil \frac{P-\tau}{p_j.s_j} \rceil) c_j - A_j(\tau))$

$$- \sum_{k=h+1}^{q} \Delta_k(\tau)$$

As regards the computational complexity with the RTO model, we observe that the determination of the idle times is also performed in $O(\lfloor \frac{R}{p} \rfloor n)$ where n denotes the number of periodic tasks, R the longest deadline and p the shortest period.

6.3. Illustration of EDL applied to RTO

Consider a simple example where a task set $\mathcal{T}$ contains two skippable periodic tasks $T_1(4, 10, 2)$ and $T_2(4, 6, 2)$. This system is not schedulable with a basic periodic model because the utilization factor here ($4/10$+$4/6$) is higher than 1. Allowing skips allows us to schedule the system that would otherwise be overloaded. Thus, the utilization factor taking skips (4/20+4/12) is less than 1. We suppose that an aperiodic request with an execution time of 5 units occurs at time τ=12. Idle times can then be recovered to process additional tasks and more particularly, to schedule aperiodic requests as soon as possible, while the execution of periodic activities is postponed. When there are no aperiodic activities in the system, the EDF algorithm is used to schedule periodic tasks as soon as possible. Whenever a new aperiodic request occurs, the idle times of EDL applied to $\mathcal{T}$ are computed on-line to optimize its response time.

First, we give an illustration under the RTO model. We can carry out the static deadline vector $\mathcal{K}$=$(0, 6, 10, 18)$ and the static idle time vector $\mathcal{D}$=$(2, 0, 4, 4)$. The function $f_{\mathcal{T}}^{EDL}$ with $\mathcal{T}$=$\{T_1, T_2\}$ is depicted in Figure 6 illustrating the EDL schedule for $\mathcal{T}$ produced at time zero. At the aperiodic arrival time, we have $\mathcal{K}(12)$=$(12, 18)$ and

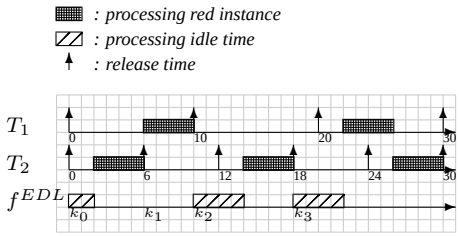

Figure 6. EDL schedule produced at time zero

$\mathcal{D}(12)$=$(2, 4)$. It is interesting to note that the response time of the aperiodic request is minimized, provided the task is executed in the idle times of the EDL schedule produced at its arrival time. The schedule actually executed is depicted in Figure 7. The actual aperiodic response time is 9.

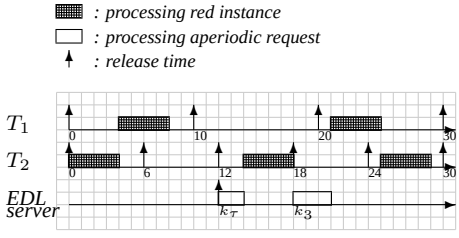

Figure 7. The EDL server for the RTO model

7. EDL together with BWP tasks $T_i(c_i, p_i, s_i)$

7.1. Dynamic case study

Compared with the RTO model, we observe that, at time $t = \tau$, the current periodic requests can be either red or blue. Indeed, in the BWP model, when there is no red instances to execute, the scheduler tends to execute blue instances which can be aborted at any time. In addition, if a blue instance completes successfully, the next instance is still blue, introducing a shift in the original RTO sequence of instances. This *move forward* at time τ can be modelled by an additional parameter $m_i(\tau)$ associated to task T_i, which is incremented modulo s_i on each completion of a blue instance. Consequently, k_i instants in vector $\mathcal{K}$=$(k_0, k_1, ..., k_i, k_{i+1}, ..., k_q)$ are redefined as follows:

$$k_i = (x.s_i + k + m_i(\tau)).p_i \tag{12}$$

with x=$0, ..., (\frac{P}{s_i.p_i} - 1)$, k=$1, ..., m$ et $0 \leq m_i(\tau) < s_i$.

The BWP sequence of instances observed on $[k_i, P]$ is the same as the RTO one observed on $[k_i - m_j(\tau)p_j, P - m_j(\tau)p_j]$ where we integrate the move forward effect $m_j(\tau)$, with $0 \leq m_j(\tau) < s_i$ associated to task T_j. $\mathcal{D}(\tau)$ is then completely defined by the new recurrent formulae:

$$\Delta_i(\tau) = \max(0, F_i(\tau)), \quad \textit{for } i = q \textit{ down to } \tau \tag{13}$$

with $F_i(\tau)$=$(P - k_i)$

$$- \sum_{j=1}^{n} (\lceil \tfrac{P-k_i}{p_j} \rceil - \lceil \tfrac{P-k_i+m_j(\tau).p_j}{p_j.s_j} \rceil + \lceil \tfrac{m_j(\tau).p_j}{p_j.s_j} \rceil) c_j$$

$$+ \sum_{\substack{j=1, d_j > k_i \\ completed\ blue/red}}^{n} A_j(\tau) - \sum_{k=i+1}^{q} \Delta_k(\tau)$$

As for the complexity for determining the duration of idle times under the BWP model, it is constant in $O(N)$ where N denotes the total number of periodic requests released in the hyperperiod $[0, P]$. Note that the on-line computation has always to be performed on the whole current hyperperiod, because of the unpredictable shifts in the RTO sequence, caused by blue instances completions.

7.2. Illustration of EDL applied to BWP

Consider now the same hybrid task set applied to the BWP model. The idle times are computed on-line using the current periodic tasks, as shown in Figure 8.

Note that the schedule produced for T_1 is the same as previously whereas the one made for T_2 is moved back a row, due to the successful completion of T_2 blue instance at time t=12, thus enhancing aperiodic responsiveness. Then, the soft aperiodic request is scheduled according to the newly computed idle times: $\mathcal{K}(12)$=$(12, 24)$ and $\mathcal{D}(12)$=$(8, 2)$. The service received by the aperiodic task is depicted in Figure 9. T_1 and T_2 blue instances are executed once the aperiodic request has completed its execution, but

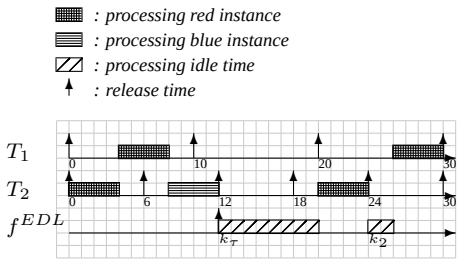

Figure 8. EDL schedule at the aperiodic arrival

they are aborted at time t=18 and t=20 respectively, because violating their respective deadlines. Note that the next instances are red in order to respect the skips constraints.

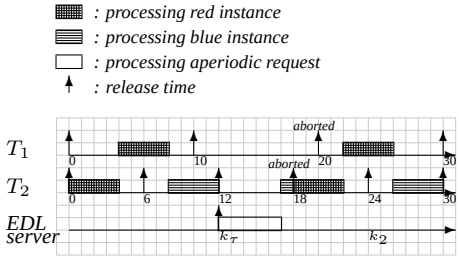

Figure 9. The EDL server for the BWP model

In this example, the actual aperiodic response time is 5, which gives a better result than in the EDL-RTO case. This is an optimal response time. Nevertheless, we saw that the computational complexity for the BWP model is higher that the one observed for RTO tasks.

8. Simulation results

All the experiments rely on a task set of five skippable periodic tasks with uniform parameters s_i. Simulation results are carried out for several periodic and aperiodic loads, with two different skip parameters. We randomly generated a large number of aperiodic task sets, each of which contains five tasks. In the graphs, the average aperiodic response time is normalized with respect to their computation time. We will use the following notation : U_p represents the periodic load without skips (*i.e.*, with a basic periodic model), U_p^* the periodic load taking skips into account, and U_s the aperiodic load.

8.1. EDL server vs BG server

In this section, we present some experimental results to compare the performance of EDL to that of the Background server (BG), for periodic tasks with skips together with soft aperiodic tasks. Used with the RTO model, the Background server schedules aperiodic requests when there are no red instances to execute. Used with the BWP model, we observe the same scheduling scheme (first, the red instances scheduled, then the aperiodic requests) with the blue instances executed where there are neither red instances nor aperiodic requests ready for execution. With the EDL server, blue instances are also scheduled in background in relation to red instances and aperiodic requests.

Varying periodic load

This first experimental result shows the performance of the EDL server, as a function of the periodic load, for two skip parameters. The load was varied by changing the average periodic service time. Figure 10 describes the performance results for uniform skip parameters s_i=2 and s_i=6. The aperiodic load U_s was held constant at 40%. The periodic load U_p was increased in such a way that U_p^*+$U_s \leq 1$ remains satisfied [4].

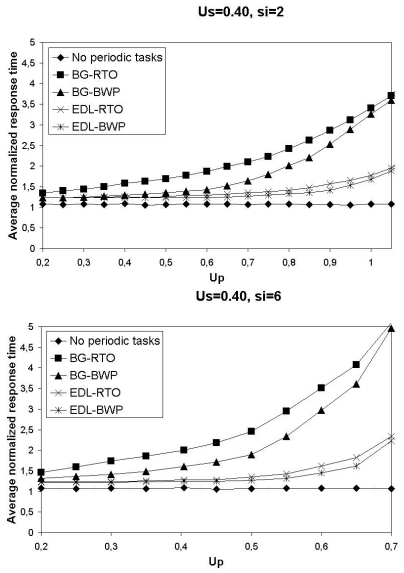

Figure 10. Average aperiodic response time

In view of these results, we can say that, on the whole, the EDL server achieves better performance than the BG server. We note that the higher the periodic load, the wider the performance advantage of the EDL server over the BG server. Moreover, we note that the two servers offer better results when used with BWP tasks. In addition, we observe than the periodic activity does not have a detrimental effect on the average response time of aperiodic requests served with the EDL server, up to a total load of 80%. In contrast, the presence of periodic tasks in the system has a significant impact on the results observed with the BG server for the same loads. Finally, the variations of the skip parameter

value show that, for wide loads, the aperiodic response time is all the more improved that the QoS observed for the periodic tasks is smaller.

Varying aperiodic load

The performance of the EDL server has also been evaluated as a function of the aperiodic load. The load was varied by changing the average aperiodic service time. In this experiment, U_p was held constant at 40%, while the aperiodic load U_s was increased such that U_p^*+$U_s \leq 1$. Figure 11 describes the performance results for uniform skip parameters, s_i=2 and s_i=6.

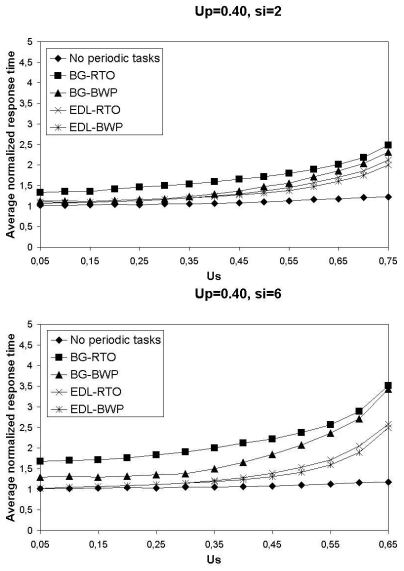

Figure 11. Average aperiodic response time

On the whole, EDL achieves good performance by exploiting the spare time saved by the skipped instances, compared with the performance of the BG server. The BWP model is still more interesting to use. In addition, we can say again that the higher the aperiodic load, the larger the interest of using the EDL server. Once again, we note that periodic tasks do not have much influence on the EDL server performance up to a total load of 80%, whereas the BG server performance is always conditioned with respect to the periodic activity. Furthermore, we observe the effect of the skip parameter on the improvement of the aperiodic response time. Indeed, we can say that it is worthwhile here to exploit the spare time saved by skipped instances, in order to enhance the aperiodic responsiveness.

9. Conclusions

The integration of QoS requirements in real-time systems involving both periodic and aperiodic activity, is motivated by the interest of emerging applications dealing with multimedia and active monitoring. We described an approach for the uniprocessor case, using the EDL server applied to the Skip-Over model. Simulation results indicate that this flexible scheduling scheme do improve the responsiveness of soft aperiodic requests. Through the examples we saw that the RTO algorithm is interesting thanks to its easy of use. Nevertheless, it is worthwhile to exploit the BWP scheduling approach showing better performances, but with high complexity computations. Future theoretical work includes QoS components enrichment to handle sporadic tasks which involves adjustments of the basic approach for the acceptance problem.

References

[1] http://www.cleopatre-project.org.

[2] G. Bernat, A. Burns, and A. Llamosi. Weakly-hard real-time systems. *IEEE Transactions on Computers*, 50(4):308–321, April 2001.

[3] G. C. Buttazzo and M. Caccamo. Minimizing aperiodic response times in a firm real-time environment. *IEEE Trans. Software Eng.*, 25(1):22–32, 1999.

[4] M. Caccamo and G. Buttazzo. Exploiting skips in periodic tasks for enhancing aperiodic responsivess. *18th IEEE Real-Time Systems Symposium*, December 1997.

[5] M. Caccamo, G. Lipari, and G. Buttazzo. Sharing resources among periodic and aperiodic tasks with dynamic deadlines. *20th IEEE Real-Time Systems Symposium*, December 1999.

[6] H. Chetto and M. Chetto. Some results of the earliest deadline scheduling algorithm. *IEEE Transactions on Software Engineering*, 15(10):1261–1269, October 1989.

[7] J. Y. Chung, J. W. S. Liu, and K. Lin. Scheduling periodic jobs that allow imprecise results. *IEEE Transactions on Computers*, 39(9):1156–1174, 1990.

[8] M. Hamdaoui and P. Ramanathan. A dynamic priority assignment technique for streams with (m,k)-firm deadlines. *IEEE Transcations on Computers*, 44(4):1443–1451, December 1995.

[9] G. Koren and D. Shasha. Skip-over algorithms and complexity for overloaded systems that allow skips. *16th IEEE Real-Time Systems Symposium*, 1995.

[10] A. L. Liestman and R. H. Campbell. A fault tolerant scheduling problem. *IEEE Transactions on Software Engineering*, 12(10):1086–1095, November 1986.

[11] J. W. S. Liu. *Real-Time Systems*. Prentice-Hall, 2000.

[12] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, pages 1175–1185, 1990.

[13] M. Silly. The edl server for scheduling periodic and soft aperiodic tasks with resource constraints. *The Journal of Real-Time Systems, Kluwer Academic Publishers*, 17:1–25, 1999.

[14] R. West and C. Poellabauer. Analysis of a window-constrained scheduler for real-time and best-effort packet streams. *21st IEEE Real-Time Systems Symposium*, November 2000.