QoS and Aperiodic Tasks Scheduling for Real-time Linux Applications

Audrey Marchand and Maryline Silly-Chetto
LINA (Laboratoire d'Informatique de Nantes Atlantique)
Rue Christian Pauc, 44300 Nantes cedex 03, France
{audrey.marchand, maryline.chetto}@iut-nantes.univ-nantes.fr

Abstract

There has been increasing interest in the real-time community for Quality of Service (QoS) based systems, such as multimedia and telecommunication systems. In this paper, we deal with scheduling components integrating new QoS functionalities under Linux/RTAI (Real-Time Application Interface) [Rta04]. The work relies on a periodic task model which allows occasional deadline violations, together with soft aperiodic tasks. This model discards selected task instances in such a way that the overall performance of the real-time system remains acceptable even in case of overload. The key objective is to exploit the skips in order to minimize the response time of soft aperiodic requests, using the EDL (Earliest Deadline as Late as possible) server, which is a dynamic slack stealing algorithm. The level of QoS, i.e. the skip factor, is fixed by the application programmer. Simulation results show the performance of two algorithms, namely EDL-RTO and EDL-BWP. Both the mean reponse time of aperiodic requests and the ratio of periodic tasks completions are considered. We also provide results to estimate the preemption cost induced by these algorithms. Finally, integration of these QoS scheduling strategies has been done in the open-source library of CLEOPATRE [1], a patch to Linux/RTAI.

1 Introduction

A real-time control application usually involves tasks which produce results under deadline constraints. Traditional classification of real-time systems stands for three classes to characterize the real-time requirement of such systems : hard, soft and firm. In hard real-time systems, all instances must be guaranteed to complete within their deadlines. For soft systems, it is acceptable to miss some of the deadlines occasionally. In firm systems, tasks are also allowed to miss some of their deadlines. Typical illustrating examples of systems with firm real-time requirements are multimedia and automotive control systems.

In recent years, many new real-time applications have emerged in which it is not necessary to meet all the task deadlines as long as the deadline violations are adequately spaced. These new scheduling techniques can deal with the problem of maintaining satisfactory performance under overload conditions.

The Skip-Over model was introduced by Koren and Shasha [10] with the notion of *skip factor* s. 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 [9] 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.

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 analyze an algorithm, based on a variant of Earliest Deadline First (EDF) scheduling, in order to exploit skips under the Total Bandwith Server (TBS).

The scope of the paper is here to minimize the response time of the soft aperiodic tasks in the presence of periodic tasks with skips. Our approach tends to distribute the spare time saved by skips for enhancing the response time of aperiodic requests. The paper is organized as follows : the next section introduces the CLEOPATRE project. Section

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

2 is a background about dynamic scheduling of periodic tasks with skips and dynamic servicing of soft aperiodic requests. We show in Section 3 how to schedule periodic tasks with skips together with soft aperiodic requests, using the EDL server. And we present simulation results to evaluate the performance of this dynamic scheduling approach in terms of response time, ratio of tasks completions and tasks preemption rate.

2 CLEOPATRE : a patch to Linux/RTAI

The work presented here is part of a French national project, CLEOPATRE (Software Open Components on the Shelf for Embedded Real-Time Applications) [2]. The objective is first to create a library of free software components for the development of real-time systems and second, to participate in the evolution of an opened community standard, Linux. Cleopatre produces a RTOS that is composed by a Linux kernel together with a modified Linux/RTAI (Real-Time Application Interface) extension.

In CLEOPATRE project, we are concerned about providing a framework which permits to develop hard, soft and firm applications using a library of selectable components dedicated to dynamic scheduling, resource control access, fault-tolerance and QoS management. The different modules are completely independent of each other. The kernel is fully modular in terms of scheduling policies, aperiodic servers, and concurrency control protocols. All modules are dynamically loadable, thus allowing the user to easily fulfil its specific needs for the development of a real-time application. Note that the proceeded changes keep the compatibility with the Linux/RTAI based applications.

In this paper, we are interested in the enrichment of existing shelves with Quality of Service facilities. These QoS facilities rely on RTO and BWP algorithms of the Skip-Over model [10], described in the following section.

3 Background material

3.1 Dynamic scheduling of periodic tasks with skips

We are here interested in the problem of scheduling periodic tasks which allow occasional deadline violations (i.e. skippable periodic tasks), on a uniprocessor system. We assume that tasks can be preempted and that they do not have precedence constraints. A task T_i is characterized by a worst-case computation time c_i, a period p_i, a relative deadline equal to its period, and a skip parameter s_i, which gives the tolerance of this task to missing deadlines. The distance between two consecutive skips must be at least s_i periods. When s_i equals to infinity, no skips are allowed and T_i is equivalent to a hard periodic task. One can view the skip parameter as a QoS metric (the higher s_i, the better the quality of service).

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. This is the colorful terminology introduced by Koren and Shasha [10]. 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 first algorithm proposed by Koren and Shasha 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. As illustrated in Figure 1, we can see that the distance between every two skips is exactly s_i periods.

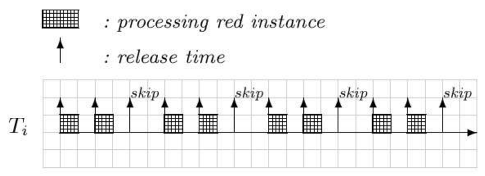

FIGURE 1: *RTO scheduling ($s_i = 3$)*

The second algorithm studied 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. Figure 2 shows an example of the possible sequence of instances of a BWP task.

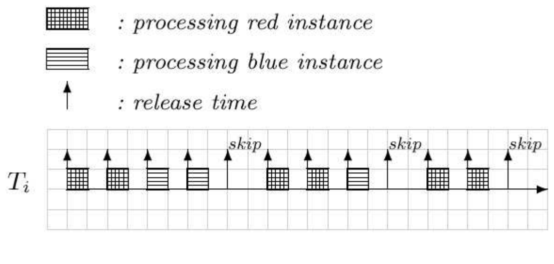

FIGURE 2: *BWP scheduling ($s_i = 3$)*

3.2 The EDL server for servicing soft aperiodic requests

The EDL (Earliest Deadline as Late as possible) server [5, 12] is a dynamic scheduling algorithm which is able to deal with the arrival of aperiodic tasks in the presence of periodic tasks. The objective is to minimize the response time of aperiodic tasks while ensuring that the deadlines of periodic tasks are always guaranteed. We assume that aperiodic tasks are served on a FCFS (First Come First Serve) basis.

The EDL server 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. More precisely, when an aperiodic request occurs, the EDL sequence is constructed on-line, and idle times computed in the EDL sequence are then exploited to execute the aperiodic tasks. Note the EDL sequence is constructed only at time instants corresponding to the arrival of a new aperiodic request while no other one is present.

The EDL server has been proved to be optimal in terms of average response time of aperiodic tasks [12]. In the next section, we show how to adapt the EDL server to periodic tasks with QoS constraints, i.e with a skip factor.

We give here an illustration of the EDL server (see Figure 3) applied for a hybrid set of basic periodic tasks and soft aperiodic tasks. We consider a set $\mathcal{T} = \{T_1, T_2\}$ of two periodic tasks $T_1 = (3, 10)$ and $T_2 = (3, 6)$. Periodic tasks are executed as soon as possible by the EDF (*Earliest Deadline First)* algorithm up to the arrival of a soft aperiodic task, say at time $\tau = 5$. The aperiodic task requires 4 units of execution time. At the time of arrival, the EDL schedule is computed on-line so as to determine the localization and the duration of the idle times, thus postponing the execution of the periodic tasks.

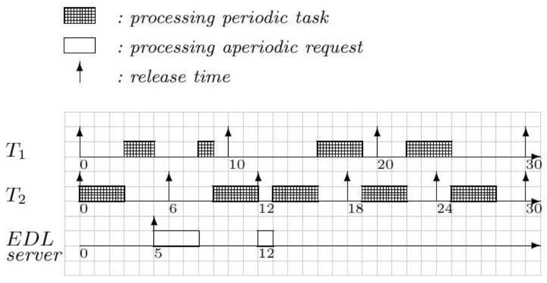

FIGURE 3: *Illustration of the EDL server*

In this example, the dynamic idle time vectors obtained in executing all the periodic tasks as late as possible from time $\tau = 5$, gives an optimal response time to the aperiodic request, provided they are executed in the EDL schedule produced at time $\tau = 5$. The aperiodic request does indeed receive service for 4 units.

4 Scheduling periodic tasks with skips and soft aperiodic requests

4.1 Theoretical fundation

We study here the EDL server adjustment to periodic tasks with skips defined according to the Skip-Over model. We remind that a QoS level defined by the user, has always to be guaranteed for periodic tasks.

Let us consider periodic tasks with skips T_i (c_i, p_i, s_i) where c_i represents the worst-case computation time, p_i the period, and s_i the skip factor of the task, according to the Skip-Over model previously defined. In order to integrate skips, we have to proceed to the determination of the idle times localization and duration on the basis of the Red Tasks Only (RTO) model, so as to maximize the spare time saved by the skipped instances. The key idea to take skips into account in the EDL server, is to integrate the skip factor s_i in the original EDL formulae for the localization and duration of idle times. Let assume that an aperiodic request occurs at time τ. Idle times computed from time τ, are then recorded in a deadline vector $\mathcal{K}$ constructed only from the distinct deadlines of red instances. In other words, every instant k_i corresponds to the deadline of a red task. Idle times for RTO and BWP models, are then computed on-line from an extension of the recurrent relations used with a basic periodic model.

Figure 4 shows the EDL-RTO server behavior with a set $\mathcal{T} = \{T_1(2, 6, 2), T_2(2, 4, 2)\}$ of two periodic tasks with skips.

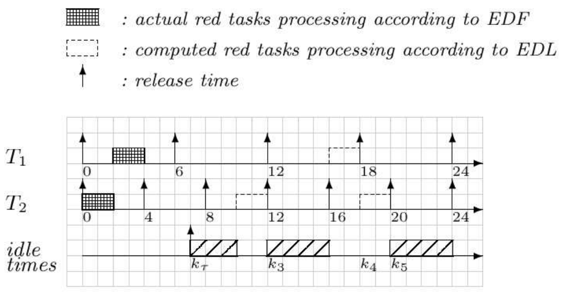
: actual red tasks processing according to EDF
: computed red tasks processing according to EDL
: release time
T_1
T_2
idle times
0
6
12
18
24
4
8
16
20
k_τ
k_3
k_4
k_5

FIGURE 4: *Idle times computation with EDL-RTO*

When there are no aperiodic tasks within the system, periodic tasks are scheduled as soon as possible, that is to say up to time $\tau = 7$, where an aperiodic request occurs. Then, idle times are determined by scheduling red tasks as late as possible. Afterwards, the aperiodic request will receive service within these idle times intervals, so as to have an optimal response time.

Let us consider now the EDL-BWP server (see Figure 5) applied to the same set of periodic tasks with skips. At time $t = 4$, T_2 blue instance attempts to execute and completes successfully at time $t = 6$, thus introducing a shift (equal to exactly one period) in the idle times sequence computed at time $\tau = 7$. Note that the immediate idle time starting at time k_τ is larger than the one observed with EDL-RTO. In this case, aperiodic requests with worst-case execution time larger than 4 units, will have a better response time under EDL-BWP.

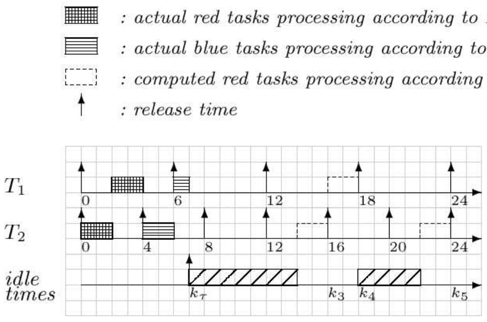
: actual red tasks processing according to EDF
: actual blue tasks processing according to EDF
: computed red tasks processing according to EDL
: release time
T_1
T_2
idle times
0
6
12
18
24
4
8
16
20
k_τ
k_3
k_4
k_5

FIGURE 5: *Idle times computation with EDL-BWP*

4.2 Simulation results

In this section, we summarize the results of simulation studies which compare the performance of the EDL server to that of the Background server (BG server) for soft aperiodic tasks. Experiments tend to evaluate the response time performance for the four models EDL-RTO, EDL-BWP, BG-RTO and BG-BWP. We simulated these algorithms throughout 3 hyperperiods. The results obtained are the averages over a group of 100 sets of 5 periodic tasks having periods ranging from 10 to 60. Deadlines are equal to the periods and greater than or equal to the computation times. Request times of aperiodic tasks are randomly generated. The aperiodic load was varied by changing the average execution time. The EDL server is evaluated with respect to the BG server. The latter is a conventional joint scheduling algorithm which simply consists in scheduling aperiodic requests when there is no periodic activity. Applied to RTO periodic tasks, the BG server schedules the aperiodic tasks when there is no red tasks ready for execution. Used with BWP periodic tasks, it offers the same behavior while blue tasks are executed when there is neither red tasks, nor aperiodic tasks to execute. The BG server is interesting thanks to its efficient implementation.

For the first experiment, the periodic load was held constant ($U_s = 40\%$) while the aperiodic load was increased from 20% to 110%. Skip value was fixed to $s_i = 2$. Results are described on Figure 6.

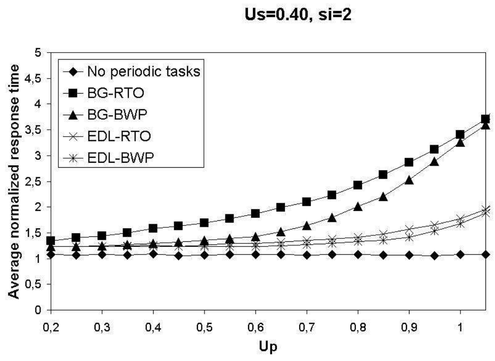

FIGURE 6: *Mean aperiodic response time varying aperiodic load ($s_i = 2$)*

Results show that the EDL algorithm offers significant performance improvements, in terms of enhancement of the aperiodic response times, over the BG algorithm. We note that the higher the periodic load, the wider the performance advantage of the EDL server over the BG server. Moreover, the two servers offer better performance when used with the BWP model.

In the second experiment, we fixed the periodic load ($U_p = 40\%$) while varying the aperiodic load from 5% to 80%. Results reinforce again the EDL server's advantage over the BG server and show its capacity to enhance aperiodic responsiveness (see Figure 7).

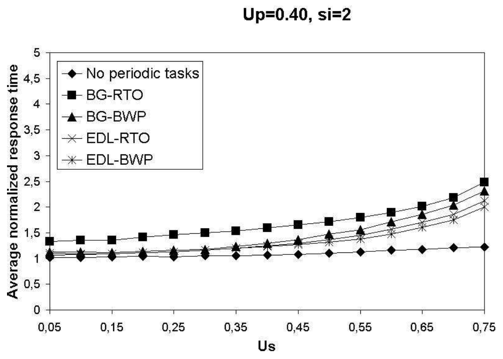

FIGURE 7: *Mean aperiodic response time varying periodic load ($s_i = 2$)*

For the third experiment, we evaluate the level of QoS for the periodic skippable tasks, according to the tolerated losses of deadlines, when the periodic and aperiodic loads vary.

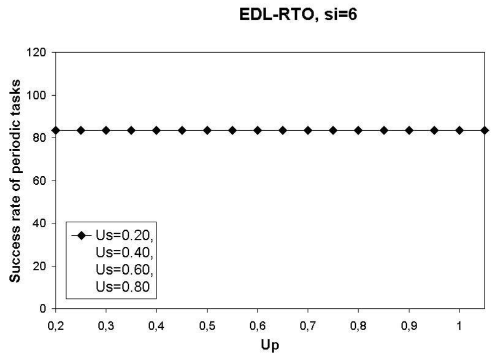

FIGURE 8: *Success rate of periodic tasks under EDL-RTO ($s_i = 6$)*

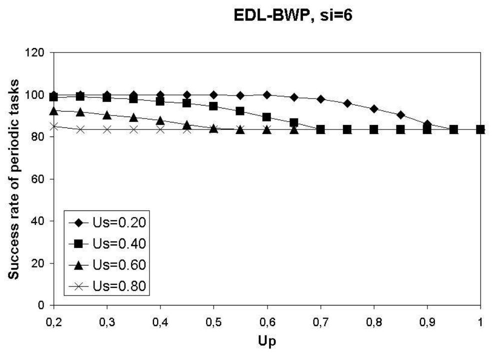

FIGURE 9: *Success rate of periodic tasks under EDL-BWP ($s_i = 6$)*

We note that the BWP model outperforms the RTO model in which the level of QoS is still constant whatever are the periodic or aperiodic loads applied. For $s_i = 6$, the QoS remains constant at a rate of 5/6=83%. The QoS relating to BWP tasks is better than the one observed for RTO tasks as long as the total load (taking skips into account) remains lower than 1. Beyond that point, QoS measurements are identical for RTO and BWP.

Finally, we study the preemption rate induced by those algorithms, varying the periodic load.

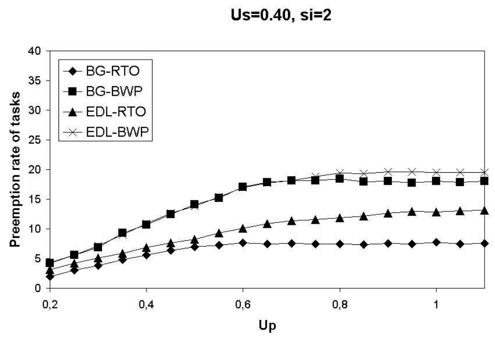

FIGURE 10: *Preemption rate of tasks varying periodic load ($s_i = 2$)*

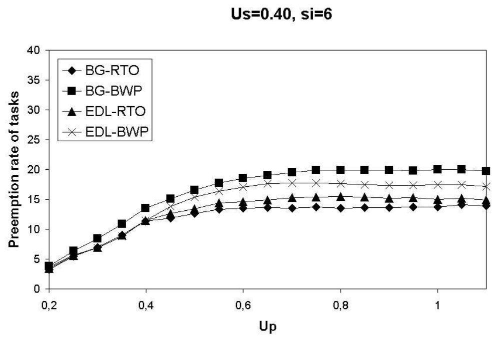

FIGURE 11: *Preemption rate of tasks varying periodic load ($s_i = 6$)*

The results of the study are as follows: for a low QoS level (i.e. small value of s_i), EDL-BWP presents the highest preemption rate among all the algorithms, while BG-RTO offers the best performance from this point of view. Nevertheless, for a high QoS level (i.e. high value of s_i), EDL-RTO and BG-RTO present a low preemption level, while BG-BWP suffers from the highest number of preemptions.

4.3 Integration to CLEOPATRE library

The CLEOPATRE library offers different classes of services. Scheduling components offer two scheduling algorithms: Deadline Monotonic (DM) and Earliest Deadline First (EDF). Concerning the shelf that provides aperiodic tasks servicing, three servers have been implemented, in order to cope with soft and hard aperiodic tasks arrivals: Background server, Total Bandwidth server (TBS) [6], and Earliest Deadline as Late as possible server (EDL). 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) [13]. Two fault-tolerance mechanisms have been implemented: the Deadline Mechanism [11] and the Imprecise Computation model [7]. Dynamic scheduling of periodic tasks with skips, namely RTO and BWP algorithms, have been added into the shelf of scheduling components.

An additional layer named TCL (Task Control Layer) which has been added as a dynamic module in `$RTAI_DIR/modules/TCL.o`, represents an enhancement of the legacy RTAI scheduler defined in `$RTAI_DIR/modules/rt_sched.o`. TCL interfaces all the new scheduling components. It is responsible for managing low-level `TaskType` (extended RTAI `RT_TASK` task descriptor) tasks through various functions (`TCL_CREATE, TCL_DESTROY, TCL_KILL, TCL_READY, TCL_BLOCK` and `TCL_SCHEDULE)` [8].

4.3.1 Data Structures

The basic data structure of the two QoS schedulers, named RTO and BWP, is the *task descriptor*, defined in `$RTAI_DIR/include/QoS.h` as `struct QoSTaskStruct`. This one contains seven fields for every task: `(*fct)(QoSTaskType)` which points to the task function, `TCL_task` which is the low-level task descriptor, `critical_delay` which defines the critical delay of the task, `period` its period of activation, `release_time` the time at which the task is released, `max_skipvalue` the static skip parameter and `current_skipvalue` the dynamic skip parameter.

```
typedef struct QoSTaskStruct QoSTaskType;
struct QoSTaskStruct{
  void (*fct) (QoSTaskType *);
  TaskType TCL_task;
  TimeType critical_delay;
  TimeType period;
  TimeType release_time;
  unsigned int max_skipvalue;
  unsigned int current_skipvalue;
};
```

The user interface for the QoS schedulers is given in Table 1.

QoS functions	Description
QoS_CREATE	create a new real-time task
QoS_RESUME	resume a real-time task
QoS_WAIT	wait till next period
QoS_DELETE	delete a real-time task

TABLE 1: *QoS schedulers' interface*

At initialization time, the user has to set the usual parameters for all tasks (period p_i, critical delay d_i,...) and also the additional skip parameter s_i for all QoS tasks.

4.3.2 Algorithms

The scheduling of RTO and BWP tasks is performed in the `QoS_schedule()` function. The scheduling occurs on timer handler activation (each 8254 interrupt). In our implementation, the scheduler maintains three linked lists: `waiting_list`, `red_ready_list` and `blue_ready_list`.

- `waiting_list`: List of waiting tasks. All tasks are sorted in increasing order of deadline

- `red_ready_list`: List of red scheduled tasks sorted in increasing order of deadline

- `blue_ready_list`: List of blue scheduled tasks sorted in increasing order of deadline

At `QoS_schedule()` time, the currently running task is the default candidate to run next. A task is considered schedulable if it is not already running and it is enabled for dispatch on the CPU.

In RTO module, the `QoS_schedule()` routine attempts to release tasks from the `waiting_list` list. If a task is found with a `release_time` parameter lesser than or equal to current time, then it is put into the `red_ready_list` list.

The scheduling decision is in the worst-case in $O(N^2)$, where all the N tasks have to be released at the same time . The algorithmic description of RTO is given in Figure 12.

```
QoS_schedule(t :  current time)
Begin
    /*Checking waiting_list to release tasks*/
    While (task=next(waiting_list)=not(∅))
        If (task->release_time<t)
            break
        EndIf
        Pull task from waiting_list
        Place task to red_ready_list
    EndWhile
End
```

FIGURE 12: *Algorithmic description of RTO*

The `QoS_schedule()` routine of BWP (see Figure 13) operates in two distinct phases. In the first phase, it examines `blue_ready_list` in order to abort, if necessary, blue tasks whose deadlines are greater than or equal to current time. The `waiting_list` list is scanned in the second phase so as to resume tasks whose release time is lesser than or equal to current time.

```
QoS_schedule(t : current time)
Begin
    /*Checking blue_ready_list to abort tasks*/
    While (task=next(blue_ready_list)=not(∅))
        If (task->release_time+task->critical_delay<t)
            break
        EndIf
        Pull task from blue_ready_list
        task->release_time+=task->period
        task->current_skipvalue=1
        Place task to waiting_list
    EndWhile
    /*Checking waiting_list to release tasks*/
    While (task=next(waiting_list)=not(∅))
        If (task->release_time<t)
            break
        EndIf
        Pull task from waiting_list
        If (task->current_skipvalue<task->max_skipvalue)
            Place task to red_ready_list
        Else
            Place task to blue_ready_list
        EndIf
    EndWhile
End
```

FIGURE 13: *Algorithmic description of BWP*

4.4 Measuring overhead

4.4.1 Description

We have made some experiments with the implementation to make a quantitative evaluation of the overhead introduced by the QoS schedulers.

Theses tests consist of measuring the overhead introduced when scheduling different number of tasks (1, 10, 20, 30,...) with periods of 10 milliseconds each one. Periods of all tasks are harmonic, leading up to a hyperperiod of 3360 ticks. Measurements were performed over a period of 1000 seconds on a computer system with a 1,7 Ghz Pentium 4 processor with 512 Mo RAM.

4.4.2 Results

The overhead we show for RTO and BWP scheduling components, indicates the amount of time spent performing scheduling tasks. As it can be seen from Figure 14, the overhead of the QoS schedulers scales with the number of installed tasks.

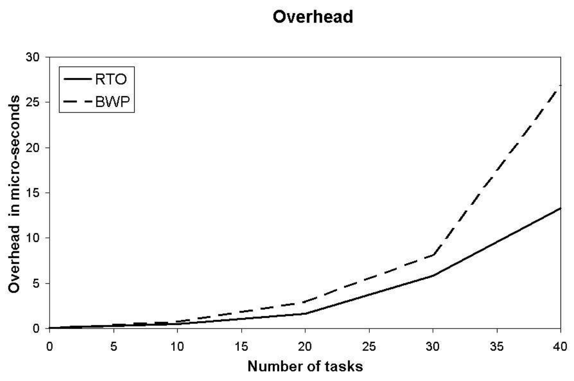

FIGURE 14: *Dynamic overhead of RTO and BWP schedulers*

It can be observed that the overhead introduced by RTO scheduling runtime increases in $O(N^2)$ where N is the number of tasks in the system. Also, we note that BWP scheduling runtime overhead tends to be twice RTO runtime overhead, when the number of scheduled tasks increases. The relating explanation is that the BWP scheduler examines all blue runnable tasks on each activation. If necessary, it has to abort some of them and to put them back in the list of waiting tasks which is sorted in increasing deadlines. Therefore, the mean runtime execution is obviously increased by the amount of time spent managing blue tasks.

5 Conclusion

This paper pointed out the need of more flexible scheduling solutions for systems involving both periodic and soft aperiodic tasks, due to the increasing interest for real-time applications dealing with multimedia and active monitoring systems. We described a new hybrid approach based on dynamic slack stealing, using the EDL server applied to the Skip-Over model. From the results, it can be stated that EDL exploits the spare time saved by skipped instances to minimize aperiodic response times. Through the examples we saw the impact of the skip parameter on responsiveness for aperiodic requests.

These new QoS functionalities are available under Linux/RTAI as CLEOPATRE components which can be dynamically loaded. The application programmer has just to set the scheduler with the additional parameter s_i in order to make use of QoS facilities.

References

[1] http://www.aero.polimi.it/rtai/

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

[3] G. C. Buttazzo, M. Caccamo, 1999, *Minimizing Aperiodic Response Times in a Firm Real-Time Environment*, IEEE Trans. Software Eng., Vol.25, No.1, pp22–32

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

[5] H. Chetto and M. Chetto, 1989, *Some results of the earliest deadline scheduling algorithm*, In Proceedings of the IEEE Transactions on Software Engineering, Vol.15, No.10, pp1261–1269

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

[7] J-Y. Chung, J-W-S. Liu and K. Lin, 1990, *Scheduling periodic jobs that allow imprecise results*, In Proceedings of the IEEE Transactions on Computers, Vol.39, No.9, pp1156–1174

[8] T. Garcia, A. Marchand and M. Silly-Chetto, 2003, *Cleopatre: a R&D project for providing new real-time functionalities to Linux/RTAI*, In Proceedings of the Fifth Real-Time Linux Workshop

[9] M. Hamdaoui and P. Ramanathan, 1995, *A dynamic priority assignment technique for streams with (m,k)-firm deadlines*, IEEE Transactions on Computers, Vol.44, No.4, pp1443–1451

[10] G. Koren and D. Shasha, 1995, *Skip-Over algorithms and complexity for overloaded systems that allow skips*, In Proceedings of the 16th IEEE Real-Time Systems Symposium

[11] A-L. Liestman and R-H. Campbell, 1986, *A fault tolerant scheduling problem*, In Proceedings of the IEEE Transaction on Software Engineering, Vol.12, No.10, pp1089–1095

[12] M. Silly, 1999, *The EDL server for scheduling periodic and soft aperiodic tasks with resource constraints*, The Journal of Real-Time Systems, Kluwer Academic Publishers, Vol.17, pp1–25

[13] L. Sha, R. Rajkumar, and J-P. Lehoczky, 1990, *Priority inheritance protocols: An approach to real-time synchronization*, In IEEE Transactions on Computers, pp1175-1185