Dynamic Scheduling of Skippable Periodic Tasks: Issues and Proposals

Maryline Chetto, Audrey Marchand

Dynamic Scheduling of Skippable Periodic Tasks: Issues and Proposals

Maryline Silly-Chetto and Audrey Marchand
IRCCyN, University of Nantes, Nantes, FRANCE
Email: {maryline.chetto, audrey.marchand}@univ-nantes.fr

***Abstract*— This paper deals with dynamic scheduling in real-time systems that have Quality of Service requirements. We assume that tasks are periodic and may miss their deadlines, occasionally, as defined by the so-called Skip-Over model. In this paper, we present a dynamic scheduling algorithm, called RLP (Red as Late as possible, a variant of Earliest Deadline to make slack stealing and to get better performance in terms of ratio of periodic task instances which complete before their deadline). Simulation results show that RLP outperforms the two conventional skip-over algorithms, namely RTO and BWP, introduced about ten years ago. Then, we investigate a second criteria called fairness, aiming to balance individual success ratios of tasks. We present variants of RLP to improve fairness and report comparative simulation results. Finally, we present the integration of these QoS scheduling services into CLEOPATRE[1], a free open-source library which offers selectable real-time facilities on shelves.**

***Index Terms*— Real-time scheduling, Earliest Deadline, Fairness, Component-based architectures, Operating systems, Real-time Linux.**

I. INTRODUCTION

Real-time systems are computer systems in which the correctness of the system depends not only on the logical correctness of the computations performed, but also on time factors. Real-time systems can be classified in three categories: hard, soft and weakly-hard.

In hard real-time systems, all instances must be guaranteed to complete within their deadlines. In those critical control applications, missing a deadline may cause catastrophic consequences on the controlled system. For soft real-time systems, it is acceptable to miss some of the deadlines occasionally. It is still valuable for the system to finish the task, even if it is late.

In weakly-hard real-time systems, tasks are allowed to miss some of their deadlines, but there is no associated value if they finish after the deadline. Typical illustrating examples of systems with weakly-hard real-time requirements are multimedia systems in which it is not necessary to meet all the task deadlines as long as the deadline violations are adequately spaced.

This paper is based on "Dynamic Scheduling of Skippable Periodic Tasks in Weakly-Hard Real-Time Systems," by M. Silly-Chetto, and A. Marchand, which appeared in the Proceedings 14th IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS'07), Tucson, Arizona, USA, March 2007. © 2007 IEEE.

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

There have been some previous approaches to the specification and design of real-time systems that tolerate occasional losses of deadlines. Hamdaoui and Ramanathan in [1] introduced the idea of (m,k)-firm deadlines to model tasks that have to meet m deadlines every k consecutive invocations. The Skip-Over model was introduced by Koren and Shasha [2] with the notion of skip factor. It is a particular case of the (m,k)-firm model. They reduce the overload by skipping some task invocations, thus exploiting skips to increase the feasible periodic load.

In this paper, we address the problem of the dynamic scheduling of periodic task sets with skip constraints. In this context, the objective of a scheduling algorithm is to maximize the effective QoS (Quality of Service) of periodic tasks defined as the number of task instances which complete before their deadline.

The remainder of this paper is organized as follows: Section 2 presents relevant background materials about the Skip-Over model. We describe two basic scheduling algorithms, namely RTO and BWP which are based on this model. In section 3, we recall the foundation of EDL (Earliest Deadline as Late as possible) algorithm, a specific method to optimize system slack by running the hard deadline tasks at the latest time while still guaranteeing their timing requirements [3]. Then, we show how to use EDL for providing an efficient scheduling algorithm called RLP (Red as Late as Possible) for the Skip-Over model. In Section 4, we provide an illustrative example. Simulation results are reported in section 5 in order to show RLP performances compared to RTO and BWP. In section 6, we provide an algorithmic description of the RLP scheduler. In section 7, we briefly describe the main results of an experiment that show improvements in terms of fairness thanks to two dynamic scheduling strategies, namely RLP-LF and RLP-MS. We present, in section 8 the integration of these QoS scheduling services into CLEOPATRE, a free open-source library which offers selectable real-time facilities on shelves and we report measures in terms of footprint and time overheads. Section 9 summarizes our contribution and gives directions for future works.

II. BACKGROUND AND RELATED WORK

A. The skip-over model

In what follows, we consider 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 at any time and 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. This parameter represents the tolerance of this task to miss deadlines. That means that 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 a hard periodic task. So, the skip parameter can be viewed as a QoS metric (the higher s_i, the better the quality of service).

Every task T_i is divided into instances where each instance occurs during a single period of the task. Every instance of a task is either red or blue [2]. A red task instance must complete before its deadline; A blue task instance can be aborted at any time. However, if a blue instance completes successfully, the next task instance is still blue.

B. RTO and BWP algorithms

Two scheduling algorithms were introduced about ten years ago by Koren and Shasha [2]. Under the *Red Tasks Only* (RTO) algorithm, red instances are scheduled as soon as possible according to *Earliest Deadline First* (EDF) algorithm [5], while blue ones are always rejected.

The *Blue When Possible* (BWP) algorithm is an improvement of RTO. Indeed, BWP schedules blue instances whenever their execution does not prevent the red ones from completing within their deadlines. In other words, blue instances are served in background relatively to red instances.

III. THE RLP ALGORITHM

A. Earliest Deadline as Late as possible

Let us review the fundamental properties of EDF algorithm, stated in [3] and [7] which are the basic foundation of our approach for scheduling tasks in the skip-over model. In general, the implementation of EDF consists in executing tasks according to their urgency, as soon as possible with no inserted idle time. Such implementation is known as EDS (*Earliest Deadline as Soon as possible*).

Nevertheless, in some applications, this implementation presents drawbacks, for example when soft aperiodic tasks need to be served with minimal response times. In that case, it is preferable to postpone execution of periodic tasks, executing them by the so called EDL (*Earliest Deadline as Late as possible*) strategy. Such approach is known as Slack Stealing since it makes any spare processing time available as soon as possible. In doing so, it effectively steals slack from the hard deadline periodic tasks.

A means of determining the maximum amount of slack which may be stolen, without jeopardizing the hard timing constraints, is thus key to the operation of the EDL algorithm. In [3], we described how the slack available at any current time can be found. This is done by mapping out the processor schedule produced by EDL for the periodic tasks from the current time up to the end of the current hyper-period (the least common multiple of task periods). This schedule is constructed dynamically whenever necessary and is computed from a static EDL schedule which is constructed off-line and memorized by means of the two following vectors:

- K, called static deadline vector. K represents the time instants from 0 to the end of the first hyper-period, at which idle times occur and is constructed from the distinct deadlines of periodic tasks.
- D, called static idle time vector. D represents the lengths of the idle times which start at time instants of K.

The complexity for computing the EDL static schedule is O(N) where N is the total number of periodic instances in the hyperperiod.

At run time, the dynamic EDL schedule is updated from the static one by taking into account the execution of current ready tasks. It is described by means of the two following vectors:

- K_t, called dynamic deadline vector. K_t represents the time instants posterior to t in the current hyper-period ,at which idle times occur.
- D_t, called dynamic idle time vector. D_t represents the lengths of the idle times that start at time instants given by K_t.

The complexity for computing the EDL dynamic schedule is O(K.n) where n is the number of periodic tasks, and K is equal to $\lfloor R/p \rfloor$, where R and p are respectively the longest deadline and the shortest period of current ready tasks [7].

B. Principles of RLP algorithm

The objective of RLP algorithm is to bring forward the execution of blue task instances so as to minimize the ratio of aborted blue instances, thus enhancing the QoS (i.e., the total number of task completions) of periodic tasks. From this perspective, RLP scheduling algorithm, which is a dynamic scheduling algorithm, is specified by the following behavior:

- if there are no blue task instances in the system, red task instances are scheduled as soon as possible according to the EDF (Earliest Deadline First) algorithm.
- if blue task instances are present in the system, they are scheduled as soon as possible according to the EDF algorithm (note that it could be according to any other heuristic), while red task instances are processed as late as possible according to the EDL algorithm. Deadline ties are always broken in favor of the task with the earliest release time.

The main idea of this approach is to take advantage of the slack of red periodic task instances. Determination of the latest start time for every red request of the periodic task set requires preliminary construction of the schedule as described previously and taking skips into account [8].

In the EDL schedule established at time t, we assume that the instance following immediately a blue instance which is part of the current periodic instance set at time t, is red. Indeed, none of the blue task instances is guaranteed to complete within its deadline.

Moreover, in [7] it was proved that the online computation of the slack time is required only at time instants corresponding to the arrival of a request while no other is already present on the machine. In our case, the EDL sequence is constructed not only when a blue task is released (and no other was already present) but also after a blue task completion if blue tasks remain in the system (the next task instance of the completed blue task has then to be considered as a blue one). Note that blue tasks are executed in the idle times computed by EDL and are of same importance beside red tasks (contrary to BWP which always assigns higher priority to red tasks).

IV. Illustrative Example

To illustrate RLP, let us consider a set of five periodic tasks $\mathcal{T} = \{T_1, T_2, T_3, T_4, T_5\}$ whose parameters are described in Table 1. We assume that all the tasks have the same skip parameter s_i = 2. We note that the processor utilization factor for this task set is equal to 1.15 and consequently some instances will necessarily miss their deadlines. It can be observed on Figure 1 that, thanks to RLP scheduling, the number of deadline violations relative to blue task instances has been reduced to three (for the same task set, five deadline violations occurs with a BWP schedule).

TABLE I.
Task parameters

Task	T_1	T_2	T_3	T_4	T_5
c_i	3	4	1	7	2
p_i	30	20	15	12	10

They occur at time instants t = 40 (task T_5), and t = 60 (tasks T_4 and T_5). Observe that T_4 first blue task instance which would fail to complete within its deadline with the BWP strategy, has enough time to succeed in the RLP schedule, since the execution of T_2 and T_1 first red task instances is postponed. Until time t = 10, red task instances are scheduled as soon as possible. From time t = 10 to the end of the hyper-period (defined as the least common multiple of task periods), red task instances do execute as late as possible in the presence of blue task instances, thus enhancing the QoS of periodic tasks.

V. Experimental Results

A. Simulation parameters

The simulation context includes 50 periodic task sets, each consisting of 10 tasks with a least common multiple equal to 3360. Tasks are defined under QoS constraints with uniform s_i. Their worst-case execution time depends on the setting of the periodic load U_p. Deadlines are equal to the periods and greater than or equal to the computation times. Simulations have been processed over 10 hyper-periods. Measurements rely on the ratio of periodic tasks instances which complete before their deadline. The evaluation is done by varying the periodic task load, U_p.

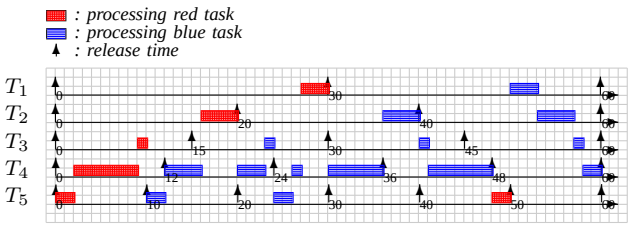

Figure 1. A RLP schedule

B. Observations

Simulation results reported in Figure 2 and Figure 3 are carried out for a skip parameter s_i equal to 2 and 6 respectively, varying the periodic load and measuring the percentage of periodic task instances that complete successfully. We observe that, for any skip parameter and any processor workload, BWP and RLP outperform RTO for which the resulting QoS is constant and minimal. For $U_p \leq 1$, the processor is under-loaded, and both BWP and RLP success in completing all blue tasks instances which are respectively executed after and before red task instances. In overload situations, RLP reveals better than BWP and, higher is the skip parameter more significant is the advantage of RLP over BWP.

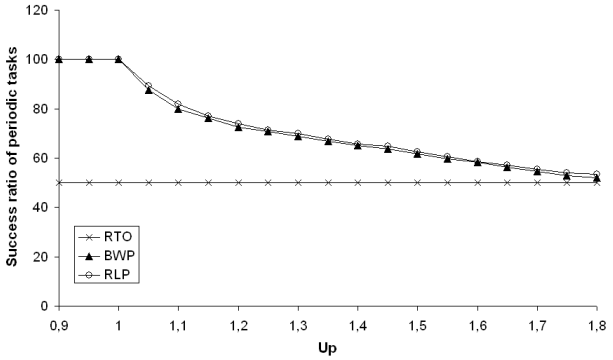

Figure 2. QoS for uniform s_i=2

VI. Implementing the RLP Scheduler

The RLP scheduler is performed by the RLP schedule() function, reported hereafter. In our implementation, the scheduler maintains three task lists which are sorted in increasing order of deadline: waiting list, red ready list and blue ready list.

- waiting list: list of waiting tasks.
- red ready list: list of red scheduled tasks
- blue ready list: list of blue scheduled tasks

Note that tasks in the red ready list are always performed before any one present in the blue ready list. At RLP

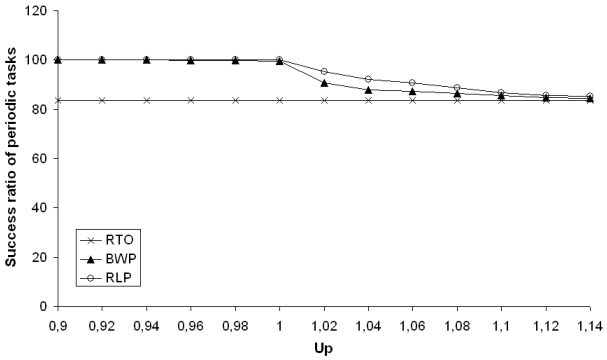

Figure 3. QoS for uniform s_i=6

schedule() invocation time, the currently running task is the default candidate to run next.

```
RLP schedule(t : current time)
begin
  /*Checking blue ready list in order 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 skip value=1
      Put task into waiting list
  endwhile
  /*Checking waiting list in order to release tasks*/
  while (task=next(waiting list)=not(∅))
      if (task→release time>t)
        break
      endif
      if ((task→current skip value < task→max skip value)
      and (Slack(t)=0))
        Pull task from waiting list
        Put task into red ready list
      else
        if (blue ready list=∅)
          Compute EDL_schedule
        endif
        if (Slack(t)!=0)
          Pull task from waiting list
          Put task into blue ready list
        endif
      endif
      task→current skip value+=1
  endwhile
  if ((blue ready list=not(∅)) and (Slack(t)!=0))
      /*Checking red ready list in order to suspend task*/
      while (task=next(red ready list)=not(∅))
        Pull task from red ready list
        Put task into waiting list
      endwhile
  endif
end
```

The RLP schedule() routine proceeds in three steps. In the first one, it examines blue ready list in order to abort one or several blue tasks which have reached their deadline. The waiting list is scanned in the second step so as to resume tasks whose release time is less than or equal to current time. Red tasks are put in the red ready list when there is no slack at current time, contrary to blue ones released only when there is an idle time. Slack value at time t is the output of the Slack(t) function, obtained from the EDL schedule. Such schedule is defined by computing the length of every processor idle time which follows every task deadline in the current hyper-period. In the last step, the red ready list is examined in order to suspend red ready tasks (released before current time), provided the blue ready list is not empty and there is slack at current time i.e. surplus processing time.

VII. Fairness Issues

A. Motivation example

Our interest here is not quantifying absolute QoS (defined here as the global success ratio of task instances while guaranteeing skip constraints) but measuring it in a relative manner in order to compare several scheduling strategies. In some cases, performance evaluation of a firm scheduling strategy must be performed not only on the basis of the global success ratio but rather must describe the behavior of every task which is characterized by its individual success ratio. This is the issue we propose to illustrate here, with the description of a real-time telesurveillance application (Figure 4).

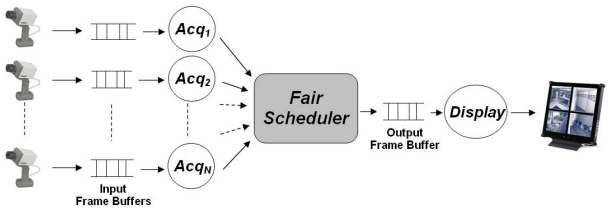

Figure 4. A real-time telesurveillance application

Video data are first captured and digitized through video capture devices such as video cameras. Then, each video capture task "Acq_i" reads the input video buffer relative to the camera it is attached, thus periodically acquiring incoming frames. Downstream from the chain, another task named "*Display*" is in charge of continously consuming frames from an output frame buffer and sending the acquired video frames to a final display device composed of various telesurveillance screens.

Among others, one important problem part of the management of telesurveillance systems is the rate at which data are refreshed on the display device. Indeed, by definition, such a system must provide pictures as recent as possible to be useful, and this must equally concern all the screens of the device. The leading solution of this problem is to ensure not only a mininum acceptable refreshing rate (*i.e.* a minimum QoS level) but also an equal one for the different screens of the display device. So, one of the key features of the scheduler here is to provide a fair service to all the tasks in charge of the frame acquisition upstream from the chain of the system (tasks "Acq_i").

Consequently, our objective in this section is to bring to light how to improve fairness while guaranteeing a Quality of Service as good as possible.

B. Definitions

In the present approach, fairness resembles the balancing of the task success ratios as specified by the following definition:

***Definition** 1: A scheduling algorithm X is* fairer *than a scheduling algorithm Y if the biggest difference between the individual success ratios of tasks obtained with X is lower to the one obtained with Y.*

Note that fairness does not refer to the ability of the system to maintain a certain level of performance. This characteristic would still permit aberrant behavior of the system such as degrading the global success ratio but keeping individual success balanced nevertheless.

C. Proposed Algorithms

The need of designing systems which are as fair as possible has led us to study novel scheduling strategies. Underlying the assessment that scheduling algorithms relying only on dynamic priorities assigned according to the Earliest Deadline (ED) criterion are not fair (see section D.), we defined new blue tasks scheduling algorithms for the RLP algorithm.

The proposed scheduling algorithms are inspired by the work described in [6] which deals with the problem of analyzing the performance of real-time control systems that feature the Deadline Mechanism [4] for on-line recovery from timing faults. From our results on the Deadline Mechanism, we propose to evaluate the two following strategies in the context of a weakly-hard real-time system:

- RLP-LF (RLP - Last Failure) algorithm schedules at each time instant, the ready blue task whose number of successive successes from the last failure is least. The earliest deadline rule is used to break ties between blue tasks of equal priorities.
- RLP-MS (RLP - Minimum Success) algorithm schedules at each time instant, the ready blue task whose individual success ratio memorized from the initialization time is least. As for the RLP-LF case, ties are broken in favour of the task with the earliest deadline.

These two variants of RLP scheduling algorithm ensure that any task will have the highest priority within finite time and no task will indefinitely keep the highest priority.

D. Comparative evaluation

The simulation context includes 50 periodic task sets, each consisting of 10 tasks with a least common multiple equal to 3360. Tasks are defined under QoS contraints with uniform s_i = 2. Their worst-case execution time depends on the setting of the periodic load Up and is randomly generated in order to reflect the greatest number of applications. Deadlines are equal to the periods and greater than or equal to the computation times. Simulations have been processed over 10 hyper-periods.

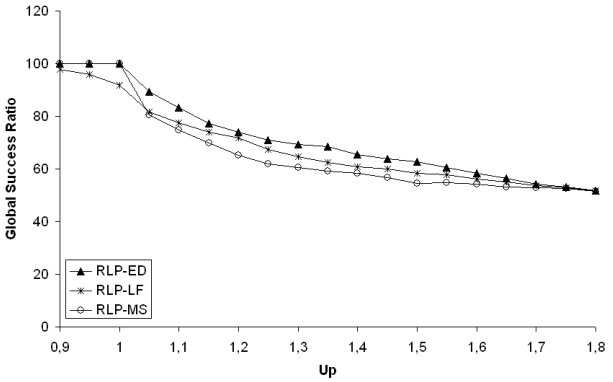

Figure 5. Measurement of the Global Success Ratio

1) Measuring the global success ratio: Figure 5 enables us to conclude that RLP-ED is better in terms of global success ratio than all other variants of RLP since Earliest Deadline is the optimal scheduling algorithm for deadline critical tasks. Consequently, if fairness is of no importance for a given application, we can consider RLP-ED as the best one with a global success ratio that attains 100% when the processor is under-loaded. We note that in overload situations, RLP-MS is the worst one, even if the gap with RLP-LF is less than 10%.

2) Measuring individual success ratios: Figure 6 brings out the fact that RLP-ED lacks fairness since it gives more importance to some tasks. Individual success ratios are really scattered around the curve depicting the global success ratio.

The distribution of the individual success ratios has been significantly improved by the proposed algorithms, RLP-LF and RLP-MS (see Figures 7 and 8). Individual success ratios of tasks have been gathered around the global success ratio curve. Whereas the mean distance between two individual success ratios is respectively equal to 30.88% for RLP-ED and 12.65% for RLP-LF, it is reduced to 1.24% for RLP-MS on the average. Similarly, as regards the maximal difference of individual success ratios, we observe a big gap between the different strategies.

VIII. Integration in a Free Operating System

A. The Cleopatre library

A library of free software components was developed within the French National project CLEOPATRE (Software Open Components on the Shelf for Embedded Real-Time Applications) in order to provide more efficient and better service to real-time applications. Our purpose was to enrich the real-time facilities of real-time Linux versions, such as RTLinux [9] or RTAI [10]. RTAI was the solution adopted for this project because we wanted the CLEOPATRE components to be distributed under the LGPL license which is also the one used in the RTAI project.

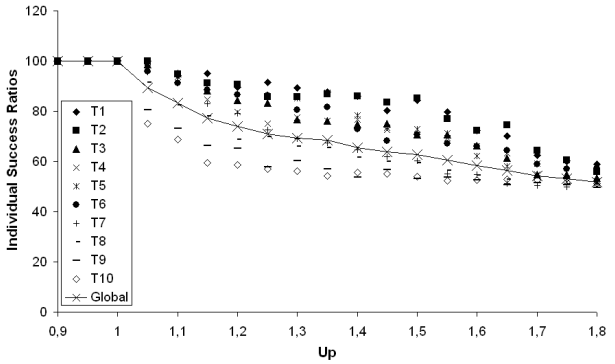

Figure 6. Measurement of the Individual Success Ratios (RLP-ED)

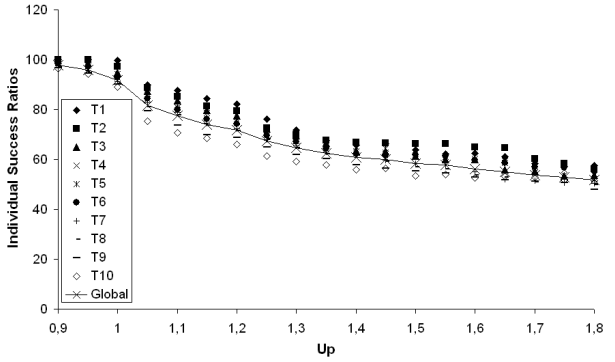

Figure 7. Measurement of the Individual Success Ratios (RLP-LF)

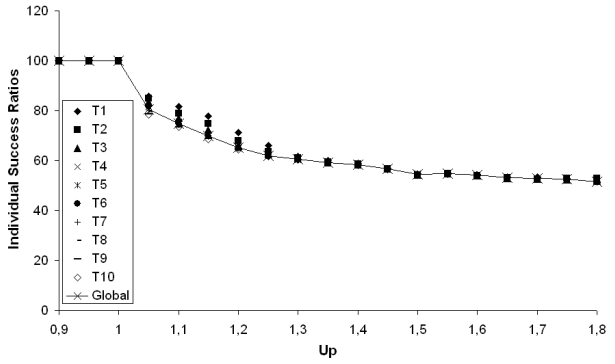

Figure 8. Measurement of the Individual Success Ratios (RLP-MS)

The CLEOPATRE library offers selectable COTS (Commercial-Off-The-Shelf) components dedicated to dynamic scheduling, aperiodic task service, resource control access, fault-tolerance and now, QoS scheduling (see Figure 9).

An additional layer named TCL (Task Control Layer) interfaces all the CLEOPATRE components. It has been added as a dynamic module in $RTAI DIR/modules/TCL.o, and represents an enhancement of the legacy RTAI scheduler defined in $RTAI DIR/modules/rt sched.o. CLEOPATRE applications are highly portable to any new CPU architecture thanks to this OS abstraction layer which makes the library of services,

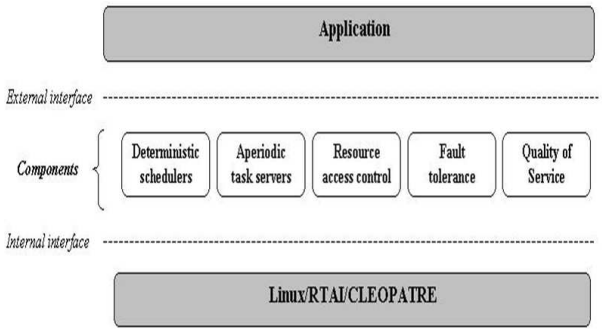

Figure 9. The CLEOPATRE framework

generic. The CLEOPATRE Off-the-Shelf components are optional except the OS abstraction layer (TCL) and the scheduler.

At most one component per shelf can be selected. Since all components of a given shelf have the same programming interface, they are interchangeable. Everything needed to use and develop CLEOPATRE can be downloaded from the web site of the project: http://cleopatre.rts-software.org.

RTO, BWP and RLP algorithms have been put into an additional shelf called Quality of Service. The QoS services are available as independent software components. This enables developers to build their own application-specific operating system.

B. Data structures and API

The basic data structure of our QoS schedulers is the task descriptor, defined in $RTAI DIR/include/QoS.h as struct QoSTaskStruct. This one contains nine fields for every task gathered and described in the following data structure:

```
typedef struct QoSTaskStruct QoSTaskType;
struct QoSTaskStruct{
void (*fct) (QoSTaskType *); /*pointer to task function*/
TaskType TCL_task; /*low-level descriptor*/
TimeType critical_delay; /*deadline*/
TimeType period;
TimeType release_time;
unsigned int max_skipvalue; /*maximum tolerance to skips*/
unsigned int current_skipvalue; /*dynamic skip parameter*/
unsigned int current_shift; /*shift w.r.t a RTO sequence*/
unsigned int slack; /*slack time of the task*/
};
```

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.

The user interface for the QoS schedulers is composed of the following functions:

- QoS_create : create a new task
- QoS_resume : resume a task
- QoS_wait : wait till next period
- QoS_delete : delete a task

C. Overheads and footprints

For embedded real-time applications, the memory footprint and disk footprint of the operating system are generally key issues as well as the time overhead incurred by its execution. Measurements of footprints for the schedulers are given in Table 2.

TABLE II.
FOOTPRINTS

Scheduler	Hard disk size (KB)	Memory size (KB)
RTO	3.2	2.3
BWP	4.1	3.2
RLP	9.7	7.6

We observe that RLP requires less than ten KB. We have made some experiments to get a quantitative evaluation about the overhead introduced by the RLP scheduler. The tests have consisted in measuring the overhead for different number of tasks (5, 10, 15, 20,...) with all periods equal to 10 milliseconds. Periods of all tasks are harmonic, leading up to an hyper-period equal to 3360 ticks. Measurements were performed over a period of 1000 seconds on a computer system with a 400 MHz Pentium II processor with 384 Mo RAM. Results are shown in Figure 10.

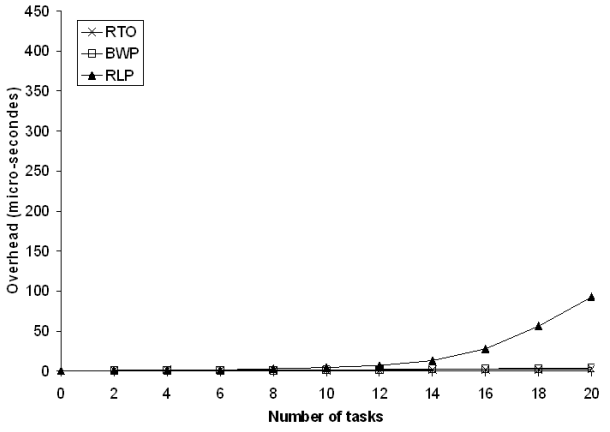

Figure 10. Overheads of RTO, BWP and RLP

The timings shown hereafter were performed with a 400 MHz Pentium II by using the Time Stamp Counter (TSC) available with every modern Intel processor. As it can been seen from Figure 10, the overhead of the QoS schedulers scales with the number of installed tasks. We note that BWP mean execution time is quite higher than the one observed for RTO. This is caused by the blue task management performed under BWP. The curve obtained for RLP is mainly due to the amount of time spent on the EDL schedule (performed only when a blue task instance is released or completed). As a matter of fact, we observe that overheads are closely related to algorithm efficiencies. An interesting feature of this component approach is that the selected scheduler can be tuned to balance performance versus complexity, so easily conforming to applications requirements.

D. A programming example

Success of real-time systems comes from both ease of use and performances. Writing code to run with CLEOPATRE is as simple as writing a C language program to run under Linux. The scheduling of QoS tasks is performed in the QoS schedule() function as described in 5. The scheduling occurs on timer handler activation (each 8254 interrupt).

Consider a periodic task set $\mathcal{T}$ composed of two tasks. The program implemented under Cleopatre is described below:

```
/*—— Headers of required components ——*/
#include <TCL.h>
#include <QoS.h>
#include <simul.h>
/*—— Timer clock period (10ms) ——*/
#define TIMERTICKS 10000000
/*——— Declarations of QoS tasks ———*/
QoSTaskType T1;
QoSTaskType T2;
/*——— Code description of QoS tasks ——*/
void CodeT1() {simul.wait(4);}
void CodeT2() {simul.wait(1);}
/*—— Initialization ——*/
int init_module(void)
{
  TCLCreateType create={0, 2000, 0, 0};
  /********* Initializing QoS tasks **********/
  QoS_create(&T1, CodeT1, 4, 20, 20, 2, create);
  QoS_create(&T2, CodeT2, 1, 15, 15, 2, create);
  /************** Resuming QoS tasks ***************/
  QoS_resume(&T1, 100);
  QoS_resume(&T2, 100);
  /************ Starting the real-time mode **********/
  TCL.begin(TIMERTICKS, 2000);
  return 0;
}
/*—— Ending and deleting QoS tasks ———*/
void cleanup_module(void)
{
  /***********Deleting QoS tasks**********/
  QoS_delete(&T1);
  QoS_delete(&T2);
  /********** Exit from the real-time mode **********/
  TCL.end();
}
```

IX. CONCLUDING REMARKS AND EXTENSIONS

The paper has described scheduling algorithms dedicated to uni-processor systems that may experience overload. We have considered the Skip-over model where all tasks are periodic and characterized by a skip factor. Because using specific properties of Earliest Deadline, we have shown that the so-called RLP algorithm performs better than other skip-over strategies when the resulting QoS is measured in terms of global success ratio. With the emergence of lots of weakly-hard real time applications that can tolerate a certain deadline missed, the understanding real time must be improved instead of just guaranteeing global QoS. The real time schedule theory need to be expanded to investigate a second criteria,

fairness, aiming to balance individual success ratios of tasks. The experimental study reported in this paper has permitted to compare results from these two points of view. Recently, this approach was extended to cope with aperiodic tasks that arrive at unpredictable times [8]. Aperiodic tasks may have strict deadline or no deadline at all. The objective of the skip-over scheduler is then to serve all the tasks by accounting for both timing and QoS constraints.

Finally, we have integrated these QoS functionalities in CLEOPATRE which is a portable, open-source, free to download and royalty free RTOS that can be used in commercial applications through the LPGL license.

REFERENCES

[1] M. Hamdaoui, P. Ramanathan, "A Dynamic Priority Assignment Technique for Streams with (m,k)-firm deadlines". *IEEE Transactions on Computers*, Vol. 44, No. 4, pp 1443-1451, 1995.

[2] G. Koren., D. Shasha, "Skip-Over Algorithms and Complexity for Overloaded Systems that Allow Skips". *Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS'95)*, Pisa, Italy, 1995.

[3] H. Chetto, M.Chetto., "Some Results of the Earliest Deadline Scheduling Algorithm". *IEEE Transactions on Software Engineering*, Vol. 15, No. 10, pp 1261-1269, 1989.

[4] A-L. Liestman and R-H. Campbell, "A fault tolerant scheduling problem", *In Proceedings of the IEEE Transactions on Software Engineering*, Vol. 12, No. 10, pp 1089-1095, 1986.

[5] C. Liu, J.W. Layland "Scheduling algorithms for multiprogramming in a hard real-time environment". *Journal of ACM*, vol.20 n1 pp.46-61, 1973.

[6] M. Silly-Chetto,"On the stability of scheduling algorithms for real-time contro", *IMACS/IEEE-SMC Computational Engineering in Systems Applications Multiconference*, Lille, France, 1996.

[7] M. Silly-Chetto, "The EDL Server for Scheduling Periodic and Soft Aperiodic Tasks with Resource Constraints", *Journal of Real-Time Systems*, Kluwer Academic Publishers, Vol. 17, pp 1-25, 1999.

[8] A. Marchand, M. Silly-Chetto, "Dynamic Real-Time Scheduling of Firm Periodic Tasks with Hard and Soft Aperiodic Tasks". *Journal of Real-Time Systems*. Vol. 32, No. 1, pp 21-47, 2006.

[9] P. Mantegazza, "DIAPM RTAI for Linux: Why's, what's and how's", *Real Time Linux Workshop*, University de Technology of Vienna, 1999.

[10] V. Yodaiken, "The RTLinux Approach to Real-Time" - FSM Labs Inc., August 2004.

[11] A.Marchand and M. Silly-Chetto, "Stability and Robustsness Issues in Scheduling Periodic Tasks with Firm Real-Time Requirements", *In Proc. Euromicro Conference on Real-Time Systems, WIP Session*, Dresden, July 2006.

Maryline Silly-Chetto received the degree of Docteur de 3ième cycle in control engineering and the degree of Habilitée à Diriger des Recherches in Computer Science from the University of Nantes, France, in 1984 and 1993, respectively. From 1984 to 1985, she held the position of Assistant professor of Computer Science at the University of Rennes, while her research was with the Institut de Recherche en Informatique et Systèmes Aléatoires, Rennes. In 1986, she returned to Nantes and is currently a professor with the Institute of Technology of the University of Nantes. She is conducting her research at IRCCyN. Her main research interests include scheduling and fault-tolerance technologies for real-time applications. She has published more than 60 journal articles and conference papers in the area of real-time operating systems. She is the leader of a French national R&D project, namely Cleopatre, supported by the French government, which aims to provide free open source real-time solutions.

Audrey Marchand graduated in Computer Engineering at Polytechnic School of the University of Nantes (France), in 2002. She received her PhD degree in computer science from the University of Nantes in 2006. She is currently a post-doctoral researcher at Polytechnic University of Valencia in Spain. Her research interests include real-time scheduling theory, aperiodic service mechanisms, quality of service guarantees in soft real-time systems, and Linux-based real-time operating systems and applications.