# We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

**4,800**
Open access books available

**122,000**
International authors and editors

**135M**
Downloads

Our authors are among the

**154**
Countries delivered to

**TOP 1%**
most cited scientists

**12.2%**
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

## Interested in publishing with us?
## Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

# 9

# Quality of Service Scheduling in the Firm Real-Time Systems

Audrey Queudet-Marchand and Maryline Chetto
*University of Nantes, IRCCyN UMR CNRS 6597*
*France*

## 1. Introduction

### 1.1 What does firm real-time mean?

Real-time systems are those in which the time at which the results are produced is important. The correctness of the result of a task is not only related to its logical correctness, but also to when the results occur (Stankovic, 1988). In order to characterize their requirements, real-time systems are traditionaly classified as follows: *hard, soft* and *firm* (Liu, 2000). It is imperative that all time constraints are met in hard real-time systems. In contrast, firm or soft real-time systems do not have as stringent timeliness requirements allowing for some degree of tardiness (soft) or miss ratio (firm). Many researches within the soft and firm real-time area have focused on minimizing tardiness and/or miss ratio but without quantifying acceptable levels. In this chapter, we focus on scheduling firm periodic tasks which have additional requirements. These requirements specify the minimum acceptable completion ratios that should be met in order to maintain system correctness. In a hard real-time system all hard deadline tasks must meet their deadlines to maintain system correctness; otherwise, the system has failed. In contrast, a deadline is considered to be soft if it can be missed occasionally. A task that misses a single soft deadline is not considered a failure. Correctness in a soft real-time system is determined by the degree to which timeliness has been enforced for the entire task set. However, the completion of a tardy firm deadline task is not meaningful since late delivery of the result is considered to be of no value to the real-time system.

Although firm deadlines can occasionally be missed, there is normally an upper limit to the number of misses within a defined interval. The hard real-time paradigm is well established and it has received considerable attention by researchers and practitioners within the academe and the industry alike. Numerous techniques and algorithms – especially in the area of scheduling – have been developed. Most scheduling algorithms developed for soft and firm real-time systems lack the ability to enforce constraints on the upper limit of misses. Unbounded consecutive time constraint violations may occur without such an enforcement. Realistically, if consecutive instances of a task fail to complete before their deadlines, then the system will eventually suffer from a failure. This indicates that there are additional constraints. These constraints express the minimum degree of timeliness that must be enforced for firm real-time tasks. This is the subject of this chapter.

### 1.2 Quality of Service (QoS) requirements

*Quality of Service (QoS)* generally refers to a broad collection of networking technologies and techniques. QoS refers to the ability of a network to deliver predictable results. Traditional QoS characteristics include availability, bandwidth, delay or delay jitter (Huston, 2000). However, QoS demands that no task be late (Krings & Azadmanesh, 1999) in hard real-time systems. QoS represents the quantified ratio of tasks that may not be executed – i.e. the total amount of work not contributing to the value of the system – in firm real-time systems. Delay and delay jitter are eliminated from consideration in both cases. Perfect QoS characterization proves to be difficult at the application level. It would be desirable that the scheduling policy adapts to changes in user QoS requirements. Such policy should strive to achieve the desired QoS in an environment with variable resources as well as complex and variable application demands. Consequently, this chapter adresses consideration of the control of the deadline miss ratios as a QoS concern.

### 1.3 Targeted applications

#### 1.3.1 Typical example: a wireless autonomous surveillance system

Let us consider a real-time monitoring system in charge of sampling key environmental indicators such as ambient temperature, carbon dioxide levels, relative humidity, wind speed/direction or solar radiation. Wireless sensor networks are practical and cost effective solutions for such monitoring. The hardware of a node basically includes various sensors, analog-to-digital converters, data storage, a radio (transceiver) and a microprocessor. Environmental data are collected periodically from the sensors and communicated from the sensor node to a base station as depicted in Figure 1. Note that the sampling rate of sensor data may vary from 5 seconds to 10 minutes according to the nature of the measurement.

A real-time monitoring system must provide updated data within strict time constraints. It is essential to have an efficient real-time scheduling of all the periodic sampling tasks.

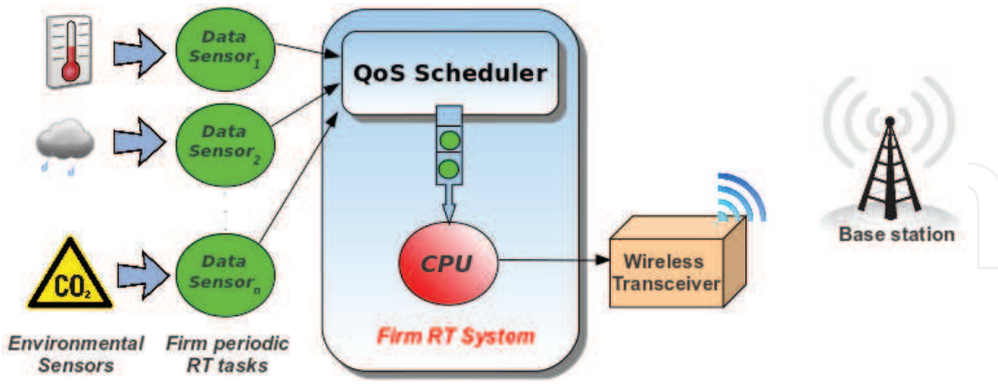



Fig. 1. Simplified architecture of a real-time wireless surveillance application

#### 1.3.2 Scheduling issue

Such a real-time system is often operated in environments that are subject to significant uncertainties. Some parameters such as emergency events, asynchronous demands from external devices (e.g. base station requests for statistical computations on sampled data) or

even energy starvation cannot be accurately characterized at design time. The occurrence of such situations will temporarily make the system overloaded (i.e. the processing power required to handle all the tasks will exceed the system capacity). The scheduling will then consist in determining the sequence of execution of sampling tasks in order to provide the best QoS.

The scheduling will play a significant role because of its ability to guarantee an acceptable sampling rate for all the tasks. The scheduler aims to gracefully degrade the QoS (i.e. sampling rate) to a lower but still acceptable level – e.g. a recording at 15 values per minute instead of 30 values per minute for wind speed – in such an overload situation. The execution of some (least important) tasks will be skipped. For instance, it will be less harmful to an air quality surveillance system to skip one wind speed record than to interrupt the transmission of the carbon dioxide level. Given this observation, one gets a better understanding of the real-time CPU scheduling flexibility needed in such applications.

In this chapter, we address the problem of the dynamic scheduling of periodic tasks with firm constraints. The scope of this study concerns maximizing the actual QoS of periodic tasks *i.e.* the ratio of instances which complete before deadline.

## 2. Scheduling skippable periodic tasks

### 2.1 Related work

Different approaches have been proposed in order to specify firm real-time systems. In (Hamdaoui & Ramanathan, 1995), the concept of *(m,k)-firm* deadlines permits us to model tasks that have to meet $m$ deadlines every $k$ consecutive instances. The *Distance-Based Priority (DBP)* scheme increases the priority of a job in danger of missing more than $m$ deadlines over a sliding window of $k$ instances for service. In order to specify a task that tolerates $x$ deadlines missed over a finite range or window among $y$ consecutive instances, a *windowed lost rate* is also proposed in (West & Poellabauer, 2000). In (Bernat et al., 2001), the authors describe a more general specification of the distribution of met and lost deadlines. *Virtual Deadline Scheduling (VDS)* (West et al., 2004) and *Dynamic Window-Constrained Scheduling (DWCS)* (Zhang et al., 2004) are other existing schedulers provably superior to *DBP* for a number of specific and non-trivial situations.

The notion of *skip factor* is presented in (Koren & Shasha, 1995). The skip factor of a task equal to $s$ means that the task will have one instance skipped out of $s$. It is a specific case of the *(m,k)-firm* model with $m = k - 1$. Skipping some task instances then permits us to transform an overload situation into an underload one. Making optimal use of skips has been proved to be an NP-hard problem. ($m$,$k$)-hard schedulers are presented in (Bernat & Burns, 1997). Most of these approaches require off-line feasibility tests to ensure a predictable service.

Scheduling hybrid task sets composed of skippable periodic and soft aperiodic tasks has been studied in (Buttazzo & Caccamo, 1999; Caccamo & Buttazzo, 1997). A scheduling algorithm based on a variant of *Earliest Deadline First (EDF)* exploits skips under the *Total Bandwith Server (TBS)*. In our previous work (Marchand & Silly-Chetto, 2005; 2006), we make use of the same approach with the *Earliest Deadline as Late as possible server (EDL)*. These results led us to propose a raw version of the *Red tasks as Late as Possible (RLP)* algorithm (idle time schedule based on red tasks only) (Marchand, 2006; Marchand & Chetto, 2008).

In contrast, tasks with soft real-time constraints still have a value even when completing after their deadlines. In this case, task overruns can cause overload situations that may be managed by overrun handling mechanisms such as *Overrun Server Method (OSM)* (Tia et al., 1995), *CApacity SHaring (CASH)* (Caccamo et al., 2000) or *Randomized Dropping (RD)* (Bello & Kim, 2007). A more complete survey on overrun handling approaches in soft real-time systems can be found in (Asiaban et al. , 2009).

### 2.2 The skip-over model

Each periodic 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 factor $s_i$ – which gives the tolerance of this task to missing deadlines – $2 \leq s_i \leq \infty$. Every periodic task instance can be either *red* or *blue* under the terminology introduced in (Koren & Shasha, 1995). A red instance must complete before its deadline; a blue instance can be aborted at any time. The operational specification of a skippable periodic task $T_i$ is composed of four characteristics: (1) the distance between two consecutive skips must be at least $s_i$ periods, (2) if a *blue* instance is skipped, then the next $s_i - 1$ instances are necessarily *red*, (3) if a *blue* instance completes successfully, the next instance is also *blue* and (4) the first $s_i - 1$ instances are *red*. The assumption $s_i \geq 2$ implies that, if a *blue* instance is skipped, then the next one must be *red*. The assumption $s_i = \infty$ signifies that no skip is authorized for task $T_i$. Skips permit us to schedule systems that might otherwise be overloaded. The system is overloaded since $U_p = \sum_{i=1}^{n} \frac{c_i}{p_i} = \frac{4}{6} + \frac{1}{2} = 1.17$ as shown in Figure 2. Allowing $T_2$ to skip one instance over three enables us to produce a feasible schedule.

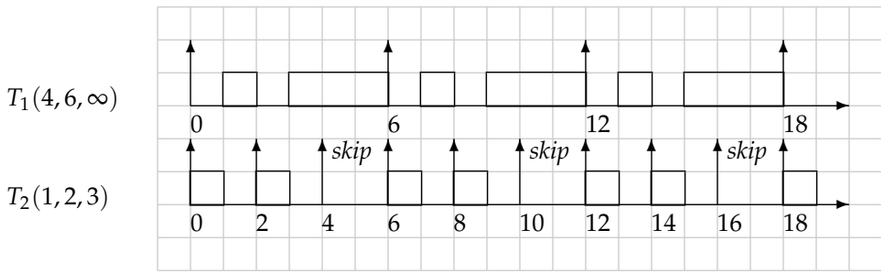



Fig. 2. A schedule with the Skip-Over model

### 2.3 Feasibility test for skippable periodic task sets

(Liu &Layland, 1973) show that a task set $\{T_i(c_i, p_i); 1 \leq i \leq n\}$ is schedulable if and only if its *cumulative processor utilization* (ignoring skips) is not greater than 1, i.e.,

$$\sum_{i=1}^{n} \frac{c_i}{p_i} \leq 1. \tag{1}$$

(Koren & Shasha, 1995) prove that the problem of determining whether a set of periodic occasionally skippable tasks is schedulable, is NP-hard. However, they prove the following necessary schedulability condition for a given set $\Gamma = \{T_i(p_i, c_i, s_i)\}$ of skippable periodic tasks:

$$\sum_{i=1}^{n} \frac{c_i(s_i - 1)}{p_i s_i} \leq 1. \tag{2}$$

(Caccamo & Buttazzo, 1997) introduce the notion of *equivalent utilization factor* defined as follows.

**DEFINITION 1.** *Given a set* $\Gamma = \{T_i(p_i, c_i, s_i)\}$ *of n skippable periodic tasks, the equivalent utilization factor is defined as:*

$$U_p^* = \max_{L \geq 0} \frac{\sum_i D(i, [0, L])}{L} \tag{3}$$

*where*

$$D(i, [0, L]) = (\lfloor \frac{L}{p_i} - \frac{L}{p_i s_i} \rfloor) c_i. \tag{4}$$

They also provide a sufficient condition in (Caccamo & Buttazzo, 1998) for guaranteeing a feasible schedule of a set of skippable tasks:

**THEOREM 1.** *A set* $\Gamma$ *of skippable periodic tasks is schedulable if*

$$U_p^* \leq 1. \tag{5}$$

### 2.4 Skip-over scheduling algorithms

#### 2.4.1 RTO (Red Tasks Only)

The first algorithm called Red Task Only (RTO) (Koren & Shasha, 1995) always rejects the blue instances whereas the red ones are scheduled according to EDF. Deadline ties are broken in favor of the task with the earliest release time. Generally speaking, RTO is not optimal. However, it becomes optimal under the particular deeply red task model where all tasks are synchronously activated and the first $s_i - 1$ instances of every task $T_i$ are red. The scheduling decision runs in the worst-case in $O(n^2)$ where all the $n$ tasks are released simultaneously.

Figure 3 depicts a RTO schedule for the task set $\mathcal{T} = \{T_0, T_1, T_2, T_3\}$. Table 1 gives the characteristics of $\mathcal{T}$. Tasks have uniform skip factor $s_i = 2$. The total processor utilization $U_p = \sum \frac{c_i}{p_i}$ is equal to 1.19. The equivalent processor utilization $U_p^*$ is equal to 0.79. This consequently guarantees the feasibility of the task set under minimal QoS.

| Task | $T_0$ | $T_1$ | $T_2$ | $T_3$ |
|---|---|---|---|---|
| $c_i$ | 4 | 6 | 9 | 4 |
| $p_i$ | 36 | 24 | 18 | 12 |

Table 1. A basic periodic task set

The schedule produced by RTO exhibits the lowest acceptable QoS level for the task set . All blue instances are systematically rejected every $s_i$ periods for each task.

#### 2.4.2 BWP (Blue When Possible)

The second scheduling algorithm called Blue When Possible (BWP) algorithm (Koren & Shasha, 1995) is an improvement of RTO. Blue instances can execute only if there are no red ready instances. Deadline ties are still broken in favor of the task with the earliest release time. BWP improves RTO in that it offers a higher QoS resulting from the successful completions of blue instances.

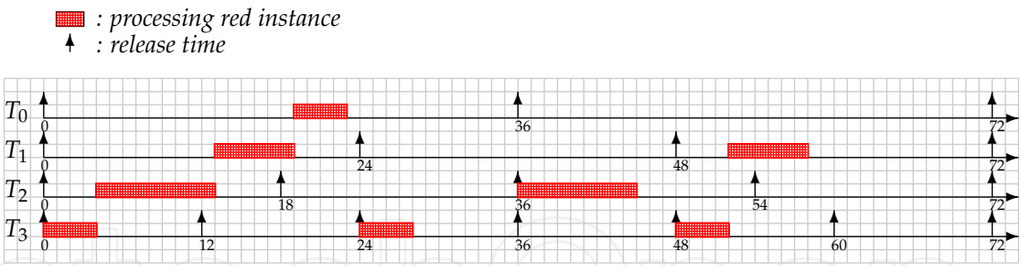



Fig. 3. A schedule produced by the RTO scheduling algorithm ($s_i = 2$)

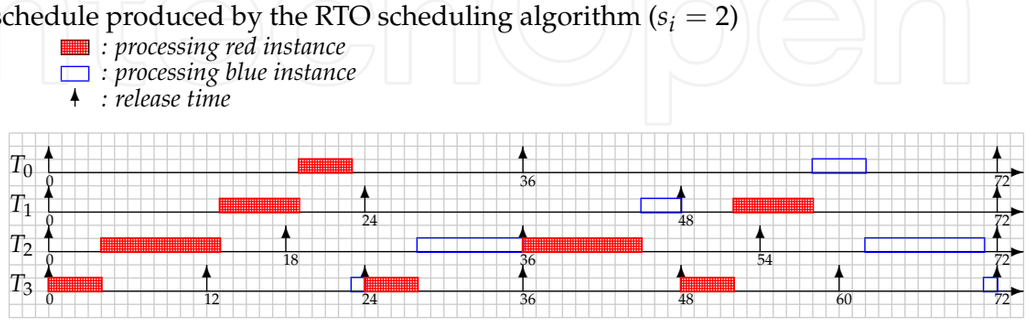



Fig. 4. A schedule produced by the BWP scheduling algorithm ($s_i = 2$)

Figure 4 illustrates a BWP schedule for the task set $\mathcal{T}$ (see Table 1).

As can be seen, BWP increases the total number of task instances that complete successfully. Five deadlines of blue instances are missed at instants $t = 24$ (task $T_3$), $t = 36$ (task $T_2$), $t = 48$ (tasks $T_1$ and $T_3$) and $t = 72$ (task $T_3$). In contrast, all deadlines of blue instances are missed under RTO which represents a total of seven instances.

## 3. Superior skip-over scheduling algorithms

### 3.1 CPU idle times determination under EDL

The basic foundation of our scheduling approach for enhancing the QoS of skippable periodic tasks relies on the *Earliest Deadline as Late as possible (EDL)* algorithm (Chetto & Chetto, 1989). Thus, we will review the fundamental properties of this algorithm. Such an 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. We described in Chetto & Chetto (1989) 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 hyperperiod (the least common multiple of task periods). This schedule is constructed dynamically whenever necessary. It is computed from a static EDL schedule constructed off-line and memorized by means of the following two vectors:

- $K$, called static deadline vector. $K$ represents the instants from 0 to the end of the first hyperperiod – 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 instants of $K$.

The dynamic EDL schedule is updated at run-time from the static one. It takes into account the execution of the current ready tasks. It is described by means of the following two vectors:

- $K_t$, called dynamic deadline vector. $K_t$ represents the instants $k_i$ from $t$ in the current hyperperiod at which idle times occur.
- $D_t$, called dynamic idle time vector. $D_t$ represents the lengths of the idle times that start at instants $k_i$ given by $K_t$.

Assume now that, given the task set $\mathcal{T} = \{T_1(3,10,10); T_2(3,6,6)\}$, we want to compute idle times from instant $t = 5$ while tasks have been processed by EDF from 0 to $t$. The resulting schedule is depicted in Figure 5. Note that $f^{EDL} = 1$ if the processor is idle at $t$, 0 otherwise.

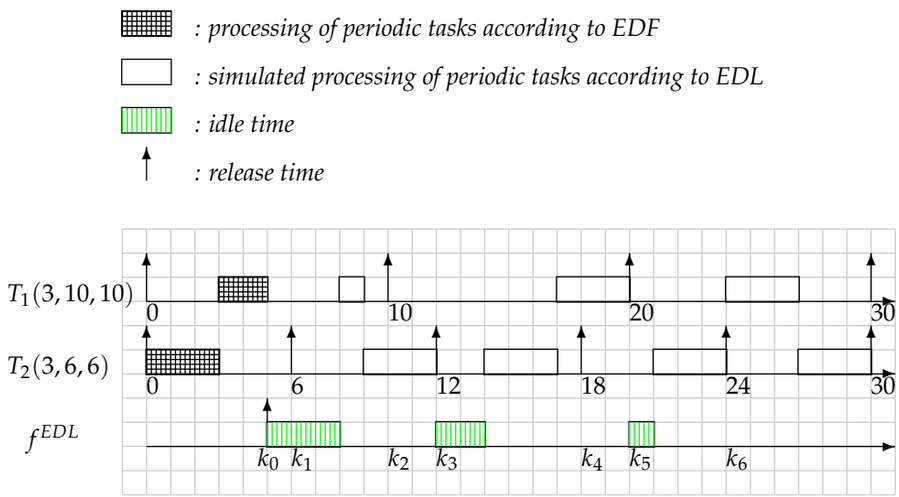



Fig. 5. EDL computation of dynamic idle times at time $t = 5$

Next, tasks are scheduled as late as possible according to EDL from time $t = 5$ to the end of the hyperperiod. Nonzero idle times resulting from the computation of vectors $K_t$ and $D_t$ appear at times $t = 5$, $t = 6$, $t = 12$ and $t = 20$.

Chetto & Chetto (1989) showed that the EDL schedule computation can be efficiently used for improving the service of aperiodic tasks. By definition, soft aperiodic requests must not compromise the guarantees given for periodic tasks and should be completed as soon as possible. No acceptance test is performed for soft aperiodic requests; they are served on a best-effort basis within the computed idle times, the goal being to minimize their response times. Concerning hard aperiodic tasks, each task is subject to an acceptance-rejection test upon arrival. Hard aperiodic tasks can indeed easily be admitted or rejected on the basis of the knowledge of idle times localization.

In the next sections, we are first interested in using EDL to build a schedule on the red instances only so as to execute the blue instances as soon as possible in the remaining EDL idle times (see section 3.2 The RLP algorithm). In a second phase, EDL will allow us to derive a test for deciding on-line whether a blue instance can be accepted for execution or not (see section 3.3 The RLP/T algorithm).

### 3.2 The RLP algorithm

BWP executes blue instances in background beside red ones. Processor time is often wasted due to the abortion of uncompleted blue instances that have reached their deadlines. Figure 4 shows that task $T_2$ is aborted at time $t = 36$. This leads to 8 units of wasted processor time.

#### 3.2.1 Algorithm description

The *Red tasks as Late as Possible (RLP)* algorithm (Marchand & Chetto, 2008) brings forward the execution of blue instances so as to enhance the actual QoS (*i.e.*, the total number of successful executions). From this perspective, RLP runs as follows:

- if no blue instance waits for execution, red instances execute as soon as possible according to the EDF scheduling rule.
- else (i.e. at least one blue instance is ready for execution), blue instances execute as soon as possible according to EDF scheduling (note that it could be according to any other scheduling heuristic), and red instances are processed as late as possible according to EDL.

Figure 6 gives the pseudo-code of the RLP algorithm. RLP 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 instances waiting for their next release,
- *red ready list*: list of red instances ready for execution,
- *blue ready list* : list of blue instances ready for execution.

At every instant $t$, the scheduler performs the following actions:

1. it updates all the three lists: instances may be released or aborted according to their current state (i.e. waiting or ready red/blue instances),
2. if $t$ belongs to an EDL idle time, it selects the first instance in the blue ready list for execution. Otherwise it selects the first instance in the red ready list.

The main idea of this approach is to take advantage of the slack of red instances. The determination of the latest start time for every red instance requires preliminary construction of the schedule by a variant of the EDL algorithm taking skips into account (Marchand & Silly-Chetto, 2006). We assume in the EDL schedule established at time $\tau$ that the instance following immediately a blue one - which is part of the current periodic instance set at time $\tau$ - is red. Indeed, none of the blue instances is guaranteed to complete within its deadline.

We proved in (Silly, 1999) that the online computation of the slack time is required only at instants which corresponds to the arrival of a request while no other is already present on the machine. The EDL sequence is constructed here not only when a blue instance is released - and no other one 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 instances are executed in the EDL idle times with the same importance as red instances, contrary to BWP which always assigns higher priority to red instances.

```
Algorithm RLP(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 (f_EDL(t)=0))
        /*red task release*/
        Pull task from waiting list
        Put task into red ready list
      else
        if (blue ready list=∅)
          Compute EDL_schedule
        endif
        if (f_EDL(t)!=0)
          /*blue task release*/
          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 (f_EDL(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
```

Fig. 6. RLP scheduling algorithm

### 3.2.2 Illustrative example

Consider the periodic task set $\mathcal{T}$ defined in Table 1. The relating RLP scheduling is illustrated in Figure 7. The number of deadline misses has been reduced to four. Missed deadlines occur at instants $t = 36$ (task $T_3$), $t = 54$ (task $T_2$) and $t = 72$ (tasks $T_1$ and $T_3$). Observe that the first blue instance $T_2$ which failed to complete within its deadline under BWP scheduling (see Figure 4) has enough time to succeed under RLP scheduling. The execution of the first red instances of $T_1$ and $T_0$ is postponed. Red instances are scheduled as soon as possible until time $t = 12$ and execute as late as possible in the presence of blue instances from time $t = 12$ up to the end of the hyperperiod. This enhances the actual QoS of periodic tasks.

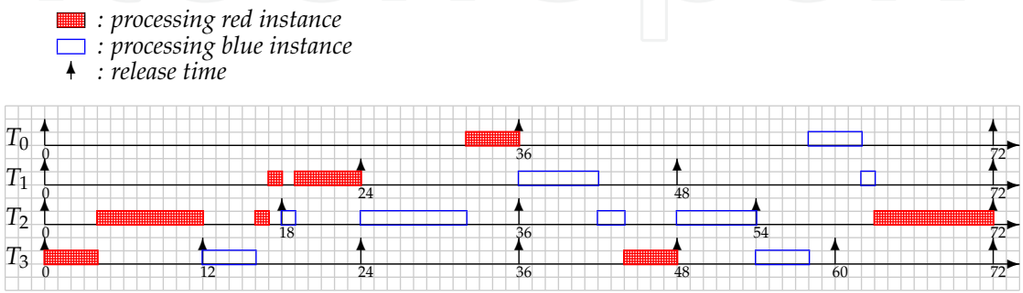



Fig. 7. A schedule produced by the RLP scheduling algorithm ($s_i = 2$)

### 3.3 The RLP/T algorithm

#### 3.3.1 Algorithm description

The *Red tasks as Late as Possible with blue acceptance Test (RLP/T)* algorithm (Marchand & Chetto, 2008) is an improvement of RLP designed to maximize even more the actual QoS.

RLP/T runs as follows: red instances enter the system directly at their arrival time whereas blue instances integrate the system upon acceptance. A blue instance is scheduled as soon as possible together with red ones once accepted. All the ready instances are of the same importance. Deadline ties are broken in favor of the task with the earliest release time.

Processor idle times are computed according to the EDL strategy once a new blue instance is released. We assume that the instance immediately following a blue instance is also blue in the EDL schedule established at time $\tau$. All blue instances previously accepted at $\tau$ are guaranteed by the schedulability test. It ensures there are enough idle times to accommodate the new blue instance within its deadline, as described hereafter.

#### 3.3.2 Acceptance test of blue instances under RLP/T

The question we ask now can be formulated as follows: "Given any occurring blue instance $B$, can $B$ be accepted?". $B$ will be accepted provided a valid schedule exists, i.e. a schedule in which $B$ will complete within its deadline while all periodic instances previously accepted will still meet their deadlines. Let $\tau$ be the current time which coincides with the release of a blue instance $B$. $B(r, c, d)$ is characterized by its release time $r$, its execution time $c$ and its deadline $d$, with $r + c \leq d$. We assume that the system supports several uncompleted blue instances at time $\tau$ previously accepted. Let's denote by $\mathcal{B}(\tau) = \{B_i(c_i(\tau), d_i), i\text{=1 to } blue(\tau)\}$,

the blue instance set at time $\tau$. The value $c_i(\tau)$ is called *dynamic execution time* and represents the remaining execution time of $B_i$ at $\tau$. $\mathcal{B}(\tau)$ is ordered such that $i < j$ implies $d_i \leq d_j$.

Theorem 2 presents the acceptance test of blue instances within a system involving RLP skippable tasks. This test is based on theoretical results established in (Silly-Chetto et al., 1990) for the acceptance of sporadic requests that occur in a system composed of non-skippable periodic tasks.

**THEOREM 2.** *Instance B is accepted if and only if, for every instance* $B_i \in \mathcal{B}(\tau) \cup \{B\}$ *such that* $d_i \geq d$, *we have* $\delta_i(\tau) \geq 0$, *with* $\delta_i(\tau)$ *defined as:*

$$\delta_i(\tau) = \Omega^{EDL}_{\mathcal{T}(\tau)}(\tau, d_i) - \sum_{j=1}^{i} c_j(\tau) \tag{6}$$

$\delta_i(\tau)$ is called *slack of instance* $B_i$ *at time* $\tau$. It defines the maximum units of time during which $B_i$ could be delayed without violating its deadline. $\Omega^{EDL}_{\mathcal{T}(\tau)}(\tau, d_i)$ denotes the total units of time that the processor is idle in the time interval $[\tau, d_i]$. The total computation time required by blue instances within $[\tau, d_i]$ is given by $\sum_{j=1}^{i} c_j(\tau)$.

The acceptance test is based on the computation of EDL idle times which gives the slack of any blue instances. Then, this slack is compared to zero. The acceptance test runs in $O(\lfloor \frac{R}{p} \rfloor n + blue(\tau))$ in the worst-case, where $n$ is the number of periodic tasks, $R$ is the longest deadline and $p$ is the shortest period. $blue(\tau)$ denotes the number of blue instances at time $\tau$ whose deadline is greater or equal to the deadline of $B_i$. A specific updating of additional data structures with slack tables may reduce the complexity to $O(n + blue(\tau))$ as proved in (Tia et al., 1994).

#### 3.3.3 Illustrative example

Figure 8 gives an illustration of RLP/T scheduling for the periodic task set $\mathcal{T}$ defined in Table 1. Clearly, RLP/T improves on both RLP and BWP. Only three deadline violations relative to blue instances are observed: at instants $t = 36$ (task $T_3$), $t = 54$ (task $T_2$) and $t = 72$ (task $T_3$). The acceptance test contributes to compensating for the time wasted in starting the execution of blue instances which are not able to complete before deadline. The blue instance $T_2$ released at time $t = 36$ is aborted at time $t = 54$ – 8 units of time were indeed wasted – in the RLP case (see Figure 7). This rejection performed with RLP/T permits us to save time recovered for the successful completion of the blue instance $T_1$ released at time $t = 48$.

## 4. Performance analysis

### 4.1 Simulation details

We report part of a performance analysis composed of three simulation experiments in order to evaluate RLP/T with respect to RTO, BWP and RLP.

We successively measure:

- *the QoS* (i.e. the ratio of instances that complete within their deadline),
- *the CPU wasted time ratio* (i.e. the percentage of useless processing time),

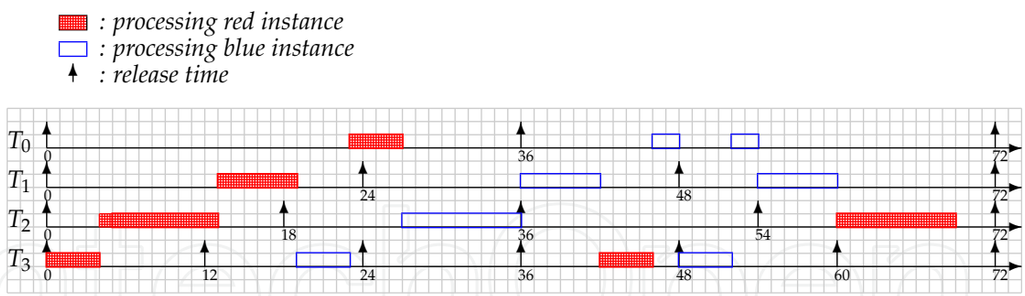


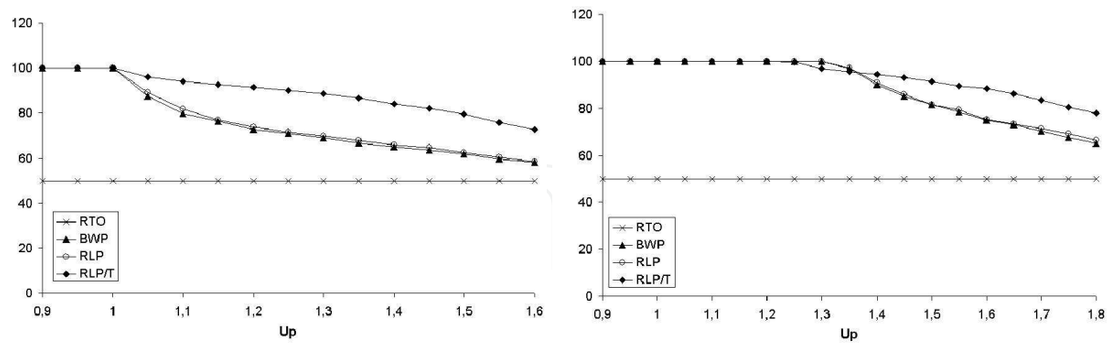

Fig. 8. A schedule produced by the RLP/T scheduling algorithm ($s_i = 2$)

- *the CPU idle time ratio* (i.e. the percentage of time during which the processor is not processing any task).

We make the processor utilization $U_p$ vary. The simulator generates 50 sets of periodic tasks. Each set contains 10 tasks with a least common multiple of periods equal to 3360 time units. The tasks have a uniform skip factor $s_i$. Worst-case computation times depend on $U_p$. Deadlines are equal to periods and greater than or equal to computation times. Simulations have been processed over 10 hyperperiods.

#### 4.1.1 Experiment 1

Figure 9 depicts the simulation results for $s_i$=2. The results are given for an actual computation time (ACET) equal to 100% and 75% of the worst-case computation time (WCET) respectively. Let us recall that the tasks have variable actual computation times assumed to be less than an estimated worst-case computation time. The assumption that a task consumes its WCET in every activation is not necessarily true. This implies that the actual CPU utilization never exceeds the estimated one used in the schedulability test.




(a) ACET=WCET (b) ACET=0.75*WCET

Fig. 9. QoS for $s_i$=2

BWP and RLP outperform RTO – for which the QoS is constant and minimal – for any processor workload. Both BWP and RLP succeed in completing all blue instances that respectively execute after and before red instances for $Up \leq 1$. RLP and BWP give almost the same performances under overload. Nevertheless, RLP/T provides a significant improvement in performance compared with RLP.

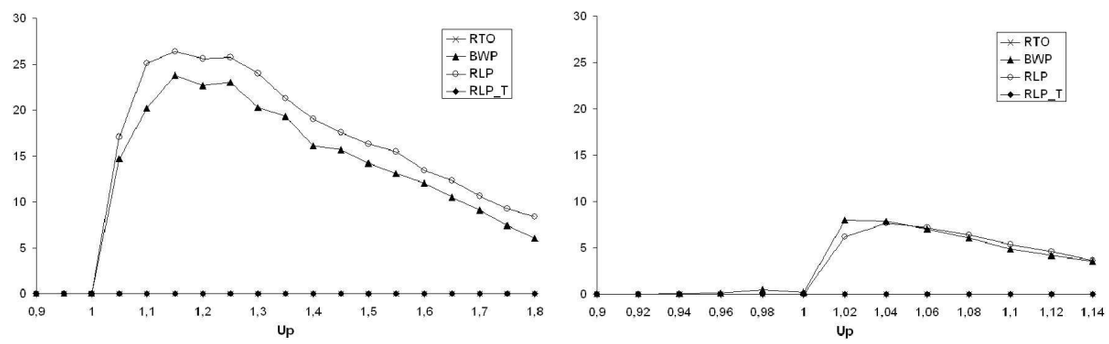
The QoS observed for BWP, RLP and RLP/T is higher for a given processor utilization when the task's computation time is less than the task's worst-case execution time. As the amount of time given by $WCET - ACET$ is not used by each instance, additional CPU time permits us to successfully complete a higher number of instances.

Moreover, note that BWP and RLP outperform RLP/T for low overloads with ACET=0.75* WCET. This comes from the admission test in RLP/T which uses WCET values and not ACET ones. Consequently, RLP/T rejects instances that after all could have been accepted on the basis of their ACET. This is exactly what we observe for $U_p$ equal to 130%: RLP/T temporarily offers lower performances than BWP and RLP. Note that this phenomenon is no longer observable once the skip factors are higher (e.g. $s_i = 6$).

Finally, other tests (Marchand, 2006) – not reported here – show that the higher the skip factor is, the more significant the advantage of RLP/T over the other scheduling algorithms.

#### 4.1.2 Experiment 2

We study here the CPU time wasted in incomplete executions of blue instances. The simulation results for $s_i = 2$ and $s_i = 6$ are depicted in Figure 10.




(a) $s_i = 2$ (b) $s_i = 6$

Fig. 10. Wasted CPU time for low and high skips

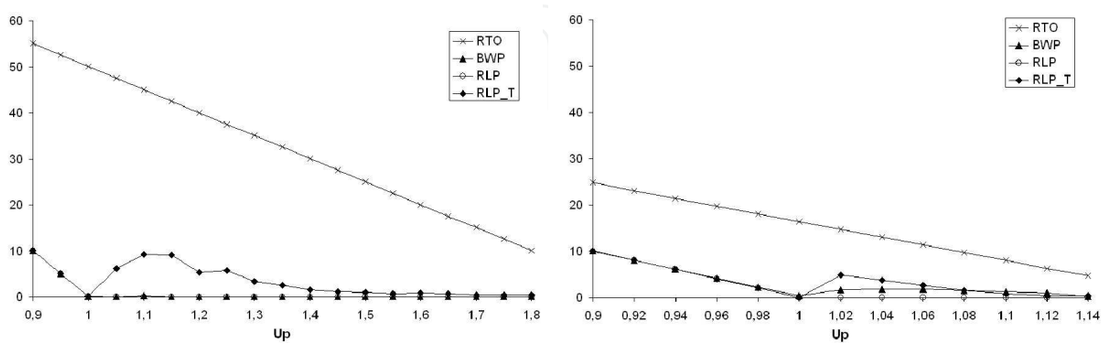
The wasted CPU time is equal to zero for RTO since all red instances execute successfully. It is also equal to zero under RLP/T for any CPU utilization. This is due to the admission test that prevents from the abortion of blue instances. A blue instance is accepted if and only if it can complete before deadline.

The wasted CPU time is always positive under BWP and RLP once the system is overloaded ($U_p > 1$). BWP and RLP involve the largest wasted CPU time – 24% et 26% respectively – for $U_p = 115\%$ and $s_i = 2$. The BWP and RLP curves present a decline beyond that load. More red instances have to be executed under high overload. Less available CPU time is consequently available for the execution of blue instances.

Additional results reported in (Marchand, 2006) show that wasted CPU time is all the less significant as skip factors grow.

#### 4.1.3 Experiment 3

Finally, we study the CPU idle time ratio given by the percentage of time during which the processor is not processing any task. This measure quantifies the ability to face a dynamic processing surplus (e.g. the arrival of an aperiodic task). Simulation results for $s_i = 2$ and $s_i = 6$ are presented in Figure 11.




(a) $s_i = 2$ (b) $s_i = 6$

Fig. 11. CPU idle time for low and high skips

We note that the CPU idle time ratio under RTO is the highest one compared with all strategies. This ratio declines in a linear fashion according to $U_p$. It varies from 55% for $U_p = 90\%$ to 10% for $U_p = 180\%$ and $s_i = 2$. Note the singular points of the curves $s_i = 2$ and $s_i = 6$: when $U_p = 100\%$ idle time ratios are respectively equal to $\frac{1}{2} = 50\%$ and $\frac{1}{6} = 16.7\%$ which correspond exactly to the skip factors.

Idle time ratios are identical and positive – e.g. idle time $= 10\%$ for $U_p = 90\%$ – when $U_p < 100\%$ under BWP, RLP and RLP/T. They decline in a linear fashion until reaching a zero value for $U_p = 100\%$.

Results differ for overloaded systems ($U_p > 100\%$). RLP involves no CPU idle time whatever the skip factors are. We observe that BWP involves a low idle time ratio only under low skip ratios. RLP/T clearly appears as the most efficient strategy, still offering idle time under light overloads. For example, the idle time ratio under RLP/T for $s_i = 2$ and $U_p = 115\%$ is equal to 9%. RLP/T gives a low and still positive idle time ratio even when the system is highly overloaded.

In summary, RLP/T proves to be the most suitable scheduling strategy to cope with transient overloads while providing the highest Quality of Service.

### 4.2 Integration into an open-source operating system

#### 4.2.1 The CLEOPATRE Library

From 2002 until 2006, we developed a library of free software components within the French National project *CLEOPATRE (Software Open Components on the Shelf for Embedded Real-Time Applications)*. This project aims to provide efficient services to real-time applications (Silly et al., 2007). It enriches real-time Linux variants with enhanced real-time facilities.

CLEOPATRE components have been prototyped under *Linux/RTAI (Real-Time Application Interface)* (Racciu & Mantegazza, 2006) and distributed under the LGPL license. The LGPL allows proprietary code to be linked to the GNU C library, glibc. When a program is linked with a library – whether statically or using a shared library – the combination of the two is legally speaking a combined work, a derivative of the original library. Companies do not have to release the source to code that which has been dynamically linked to an LGPLed library. This makes the use of such codes much more attractive.

The CLEOPATRE library offers selectable COTS (Commercial-Off-The-Shelf) components dedicated to dynamic scheduling, aperiodic task servicing, resource control access, fault-tolerance and QoS scheduling. An additional task named *TCL (Task Control Logic)* interfaces all the CLEOPATRE components and has the highest priority. It has been added as a dynamic module in $*RTAIDIR*/*modules*/*TCL.o* and interfaces with the legacy RTAI scheduler defined in $*RTAIDIR*/*modules*/*rtai_sched.o*, as depicted in Figure 12.

The CLEOPATRE interface is totally independent from the RTAI core layer. It can be directly used with Xenomai – which supports the RTAI API – and easily adapted to any other real-time Linux extension.

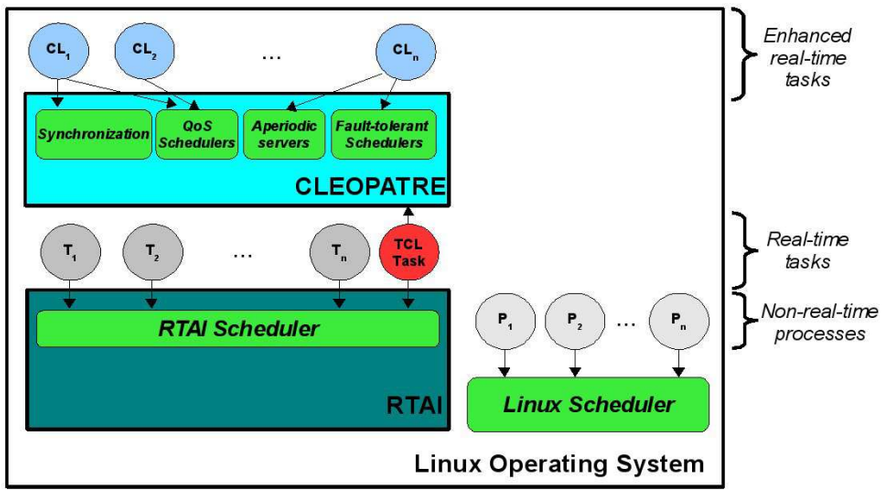



Fig. 12. Cleopatre Library within Linux/RTAI

CLEOPATRE applications are highly portable to any new CPU architecture thanks to this OS abstraction layer which makes the library of services generic (Silly et al., 2007). The CLEOPATRE Off-the-Shelf components are optional except for 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 become interchangeable. Everything needed to use or extend CLEOPATRE can be downloaded from the project web site *http://cleopatre.rts-software.org*.

### 4.2.2 Overheads and footprints

The memory and disk footprints of the operating system turn out to be key issues for embedded real-time applications as well as the time overhead incurred by the operating system itself. Table 2 gives the footprints for the schedulers provided by CLEOPATRE.

| **QoS components** | Hard disk size (KB) | Memory size (KB) |
|---|---|---|
| *RTO* | 3.2 | 2.3 |
| *BWP* | 4.1 | 3.2 |
| *RLP* | 9.7 | 7.6 |
| *RLP/T* | 13.3 | 10.8 |

Table 2. Footprints of QoS components

The smallest footprint of an application using a QoS scheduler comes to 52.4 KB in memory (65.2 KB on hard disk). This corresponds to the total load due to RTAI, the TCL task and the RTO scheduler. On the contrary, the greatest footprint corresponds to the RLP/T scheduler (i.e. 60.9 KB in memory and 75.3 KB on hard disk). Any QoS scheduler, including RLP/T scheduler, easily fits into the flash memory of an embedded system.

We conducted experiments to obtain a quantitative evaluation of the overhead led by the QoS schedulers. We measured the overhead for various numbers of tasks (5, 10, 15, 20,...) with all periods equal to 10 milliseconds. Periods are harmonic with a hyperperiod equal to 3360 timer ticks. The measurements were performed over a period of 1000 seconds on a computer system with a 400 MHz Pentium II processor with 384 Mo RAM. Figure 13 shows the resulting overhead.

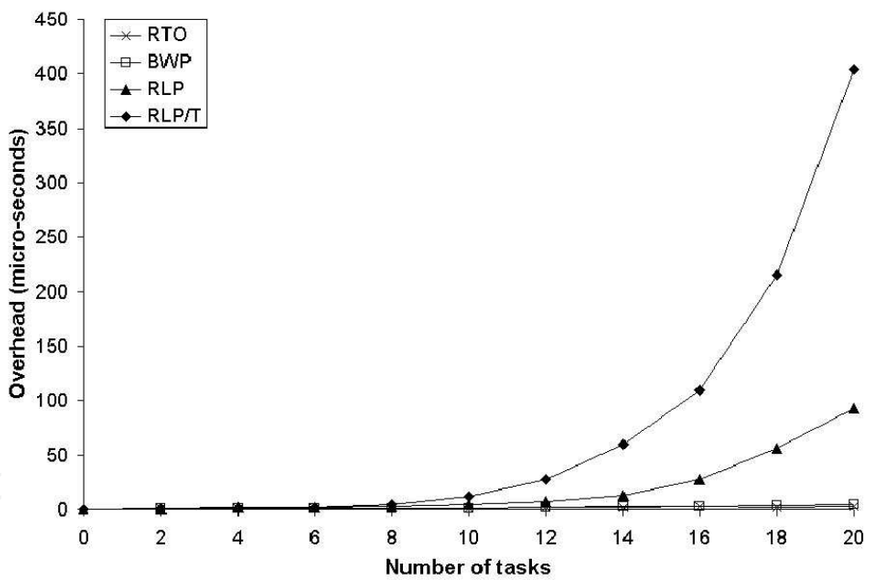



Fig. 13. Dynamic overhead of the QoS schedulers

The average overhead led by the QoS schedulers scales with the number of installed tasks. BWP exhibits an average execution time that is substantially higher than the RTO. This comes from the management of blue instances under BWP. The curve obtained for RLP and RLP/T mainly comes from the amount of time spent on the EDL schedule (performed only when a blue instance is released or completed). As a matter of fact, we observe that overhead is closely related to efficiency. An interesting feature of the component approach lies in that the selected scheduler can be tuned to balance performance versus complexity, and thus easily conforms to implementation requirements.

## 5. Conclusion

### 5.1 Summary

While it is imperative that all time constraints – generally expressed in terms of deadlines – are met in hard real-time systems, firm real-time systems do not have as stringent timeliness requirements since they allow for some degree of miss ratio. Video reception and multimedia-oriented mobile applications are typical firm real-time applications that require the need for a suitable real-time scheduler which represents the central key service in any operating system. The proliferation of these applications has motivated many research efforts over the last twenty years in order to produce a scheduling framework that explicitly addresses their specific requirements and improves the global Quality of Service.

A firm real-time system is typically characterized by dynamic changes in workloads (tasks have variable actual execution times). It consequently needs a scheduler able to handle possible overload situations and to allow the system to achieve graceful degradation by skipping some tasks. The scheduler has to supply a dynamic mechanism that determines on-line the task to be shed from the system. Multimedia systems are typically systems in which performance is sensitive to the distribution of skips: if skips occur for several consecutive instances of the same task, then the system performance may be totally unacceptable leading to some form of instability. To overcome the shortcoming of the Quality of Service metrics only based on the average rate of dropped tasks, Koren and Shasha proposed the skip-over model in which a periodic task with a skip factor of $s$ is allowed to have one instance skipped out of $s$ consecutive instances (Koren & Shasha, 1995).

In this chapter, we have considered the skip-over model where independant tasks run periodically on a uni-processor architecture and can be preempted at any time. Additionally, they have a skip factor. We described two on-line scheduling algorithms respectively named RLP and RLP/T, the latter being based on an admission control mechanism. The results of an experimental study indicate that improvements with both RLP and RLP/T are quite significant compared with the two basic algorithms introduced by Koren and Shasha. We have integrated all the QoS schedulers presented in this chapter as software components which are part of the CLEOPATRE open-source library. We have performed their evaluation under a real-time Linux-based operating system, namely Linux/RTAI. The observed overheads and footprints enabled us to state their ability to be used even for embedded applications with severe memory and timeliness requirements.

### 5.2 Future work

#### 5.2.1 QoS and energy harvesting

Many embedded systems work in insecure or remote sites (e.g. wireless intelligent sensors). The new generation of these systems will be smaller and more energy efficient while still offering sufficient performance. A typical example is data farming where sensors are spread over an area to supervise the environment and send collected data for further processing to a base station. Sensors are deployed and then must stay operational for a long period of time, in the range of months or even years.

One way to prolong the lifetime of such autonomous systems is to harvest the required energy from the environment. Energy Harvesting is defined as the process of capturing energy from one or more natural energy sources accumulating it and storing it for later use (Priya & Inman,

2009). Energy harvesting from a natural source where a remote application is deployed and where energy is inexhaustible appears as an attractive alternative to inconvenient traditional batteries. Energy harvesting with solar panels is one of the most popular technologies. Nowadays, many real life applications using energy harvesting are operational. Wireless sensor network systems, including ZigBee systems, benefit from this technology.

A wireless sensor has timing constraints that must be satisfied and consume only as much energy as the energy harvester can collect from the environment. But the harvested energy is highly dependent on the environment and the power drained from most environmental energy sources is not constant over time. Consequently, energy consumption coming from the execution of tasks should be continuously adjusted in order to maximize the Quality of Service and not only to minimize the energy consumption. The main challenge of research is to provide an energy-aware scheduling algorithm that will schedule tasks so as to consider jointly two kinds of constraints: time (i.e. deadlines) and energy availability.

To address the above problem, we proposed in a recent paper (El Ghor et al., 2011) an efficient scheduling algorithm called EDeg which is based on both the energy stored and the energy estimated to be harvested in the future. We performed a series of experiments based on the rate of missed deadlines in order to compare EDeg with other scheduling methods. Experimental results show that EDeg significantly outperforms the classical greedy schedulers, including EDF.

We are now extending this scheduling strategy to the skip-over model. The objective is to reduce the rate of missed deadlines when the system lacks either time or energy, by taking into account skip factors. To summarize, our current work focuses on the same problem studied in this chapter but considers the specific issue of real-time energy harvesting systems.

#### 5.2.2 QoS and multicore systems

While real-time applications are becoming more and more concurrent and complex, the drive toward multicore systems seems inevitable. Multicore processors solve the problem of heat that has been slowing processor growth in the past while providing increased performance. We propose in a recent paper (Abdallah et al., 2011) to tackle the problem of distributing skippable periodic tasks over such platforms. Our contribution is twofold. First, we design a schedulability test for multicore task sets under QoS constraints. Second, based on this test, we propose new partitioned scheduling heuristics to assign tasks with QoS constraints to processors so as to minimize the number of processors used. In conclusion, this new line of investigation extends the work presented in this chapter to multicore platforms.

## 6. References

Abdallah, N., Queudet, A., Chetto, M. & Chehade, R.-H. (2011). Partitioned EDF Scheduling in Multicore systems with Quality of Service constraints. *Proceedings of IEEE International Conference on Electronics, Circuits, and Systems*, December 2011, Beirut (Lebanon).

Asiaban, S., Moghaddam, M.-E. & Abbaspour, M. (2009). A Real-Time Scheduling Algorithm for Soft Periodic Tasks. *International Journal of Digital Content Technology and its Applications*, 3(4):100–111.

Bernat, G. & Burns, A. (1997). Combining (n/m)-hard deadlines and dual priority scheduling, *Proceedings of the 18th IEEE Real-Time Systems Symposium*, pp 46-57, December 1997, San Francisco (USA).

Bernat, G., Burns, A. & Llamosi, A. (2001). Weakly-hard real-time systems, *IEEE Transactions on Computers*, 50(4):308-321.

Bello, L.-L. & Kim, K. (2007). Overrun handling approaches for overload-prone soft real-time systems. *Advances in Engineering Software*, 38(11-12):780–794.

Buttazzo, G.-C. & Caccamo, M. (1999). Minimizing Aperiodic Response Times in a Firm Real-Time Environment, *IEEE Transaction on Software Engineering*, 25(1):22–32.

Caccamo, M. & Buttazzo, G.-C. (1997). Exploiting skips in periodic tasks for enhancing aperiodic responsivess, *Proceedings of the 18th IEEE Real-Time Systems Symposium*, December 1997, San Francisco (USA).

Caccamo, M., & Buttazzo, G.-C. (1998). Optimal Scheduling for Fault-Tolerant and Firm Real-Time Systems. *Proceedings of the IEEE Real-Time Computing Systems and Applications*, October 1998, Hiroshima (Japan).

Caccamo, M., Buttazzo, G. & Sha, L. (2000). Capacity sharing for overrun control. *Proceedings of the 21st IEEE Real-Time Systems Symposium*, November 2000, Orlando (USA).

Chetto, H. & Chetto, M. (1989). Some Results of the Earliest Deadline Scheduling Algorithm. *IEEE Transactions on Software Engineering*, 15(10):1261–1269.

El Ghor, H., Chetto, M. & Chehade, R.-H. (2011). A real-time scheduling framework for embedded systems with environmental energy harvesting. *Computers & Electrical Engineering Journal*, 37(4):498–510.

Hamdaoui, M. & Ramanathan, P. (1995). A Dynamic Priority Assignment Technique for Streams with (m,k)-firm deadlines. *IEEE Transactions on Computers*, 44(4):1443–1451.

Huston, G. (2000). Quality of Service - Fact or Fiction? *The Internet Protocol Journal*, 3(1):1–40.

Krings, A.W. & Azadmanesh, M.H. (1999). QoS Considerations In Real-Time Scheduling, *Proceedings of the 2nd International Conference on Parallel Computing Systems*, August 1999, Ensenada (Mexico).

Koren, G. & Shasha, D. (1995). Skip-Over Algorithms and Complexity for Overloaded Systems that Allow Skips. *Proceedings of the 16th IEEE Real-Time Systems Symposium*, December 1995, Pisa (Italy).

Liestman, A-L. & Campbell, R-H. (1986). A fault tolerant scheduling problem. *Proceedings of the IEEE Transaction on Software Engineering*, 12(10):1089–1095.

Liu, J.-W.-S. (2000). *Real-Time Systems*, Prentice-Hall.

Liu, C-L. & Layland, J-W. (1973). Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the Association for Computing Machinery*, 20(1):46–61.

Marchand, A. (2006). Ordonnancement Temps Réel avec Contraintes de Qualité de Service - De la théorie à l'intégration. *PhD Thesis*, University of Nantes (France), October 2006.

Marchand, A. & Chetto, M. (2008). Quality of Service Scheduling in Real-Time Systems. *International Journal on Computers, Communications & Control*, 3(4):354–366.

Marchand, A. & Silly-Chetto, M. (2005). QoS Scheduling Components based on Firm Real-Time Requirements, *Proceedings of the ACS/IEEE International Conference on Computer Systems and Applications*, January 2005, Le Caire (Egypt).

Marchand, A. & Silly-Chetto, M. (2006). Dynamic Real-Time Scheduling of Firm Periodic Tasks with Hard and Soft Aperiodic Tasks. *Journal of Real-Time Systems*, 32(1-2):21–47.

Priya, S., Inman, D.-J. (2009). *Energy Harvesting Technologies*, Springer.

Racciu, G. & Mantegazza, P. (2006). RTAI User Manual 3.4 rev 0.3. URL:www.rtai.org

Silly, M. (1999). The EDL Server for Scheduling Periodic and Soft Aperiodic Tasks with Resource Constraints. *The Journal of Real-Time Systems*, 17(1): 87-111.

Silly-Chetto, M., Chetto, H. & Elyounsi, N (1990). An Optimal Algorithm for Guranteeing Sporadic Tasks in Hard Real-Time Systems. *IEEE Symposium on Parallel and Distributed Processing*, December 1990, Dallas (USA).

Silly-Chetto, M., Garcia-Fernandez T. & Marchand-Queudet, A. (2007). CLEOPATRE: Open-source Operating System Facilities for Real-Time Embedded Applications. *The Journal of Computing and Information Technology*, 15(2): 131-142.

Stankovic, J.-A. (1988). Misconceptions about real-time computing: a serious problem for next-generation systems. *IEEE Computer*, 21(10):10–19.

Tia, T., Liu, J., Sun, J. & Ha, R. (1994). A Linear-Time Optimal Acceptance Test for Scheduling of Hard Real-Time Tasks, *Tech. report*, University of Illinois, Urbana-Champaign (USA).

Tia, T.-S., Deng, Z., Shankar, M., Storch, M., Sun, J., Wu, L.-C. & Liu, J.-W.-S. (1995). Probabilistic performance guarantee for real-time tasks with varying computation times, *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, May 1995, Chicago (USA).

West, R. & Poellabauer, C. (2000). Analysis of a Window-constrained scheduler for real-time and best-effort packet streams, *Proceedings of the 21st IEEE Real-Time Systems Symposium*, December 2000, Orlando (USA).

West, R., Zhang, Y., Schwan, K. & Poellabauer, C. (2004). Dynamic window-constrained scheduling of real-time streams in media servers, *IEEE Transactions on Computers*, 53(6):744–759.

Zhang, Y., West, R. & Qi, X. (2004). A virtual deadline scheduler for window-constrained service guarantees, *Technical Report*, No. 2004-013, Boston University (USA).


**Real-Time Systems, Architecture, Scheduling, and Application**

Edited by Dr. Seyed Morteza Babamir

ISBN 978-953-51-0510-7
Hard cover, 334 pages
**Publisher** InTech
**Published online** 11, April, 2012
**Published in print edition** April, 2012

This book is a rich text for introducing diverse aspects of real-time systems including architecture, specification and verification, scheduling and real world applications. It is useful for advanced graduate students and researchers in a wide range of disciplines impacted by embedded computing and software. Since the book covers the most recent advances in real-time systems and communications networks, it serves as a vehicle for technology transition within the real-time systems community of systems architects, designers, technologists, and system analysts. Real-time applications are used in daily operations, such as engine and break mechanisms in cars, traffic light and air-traffic control and heart beat and blood pressure monitoring. This book includes 15 chapters arranged in 4 sections, Architecture (chapters 1-4), Specification and Verification (chapters 5-6), Scheduling (chapters 7-9) and Real word applications (chapters 10-15).

**How to reference**

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Audrey Queudet-Marchand and Maryline Chetto (2012). Quality of Service Scheduling in the Firm Real-Time Systems, Real-Time Systems, Architecture, Scheduling, and Application, Dr. Seyed Morteza Babamir (Ed.), ISBN: 978-953-51-0510-7, InTech, Available from: http://www.intechopen.com/books/real-time-systems-architecture-scheduling-and-application/quality-of-service-scheduling-for-firm-real-time-systems

INTECH
open science | open minds

**InTech Europe**
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

**InTech China**
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen