A Dynamic Scheduler for Real-Time Periodic Tasks with Quality of Service Requirements

MARYLINE SILLY-CHETTO and AUDREY MARCHAND
Institut de Recherche en Communication et Cybernétique de Nantes
UMR CNRS 6597 BP 92101 - 1 Rue de la Noé, F-44321 Nantes cedex 3
FRANCE

Abstract: - This paper is concerned with dynamic scheduling in overload 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. Every task is characterized by a skip factor which represents the minimal Quality of Service (QoS) level, required by the concerned application. The objective consists in scheduling tasks in order to guarantee the QoS constraint and maximize the ratio of periodic task instances which complete before their deadline. A comparative evaluation of scheduling algorithms can then be realized by measuring this ratio. In this paper, we present a dynamic scheduling algorithm, called RLP (Red as Late as possible) based on the EDL (Earliest Deadline as Late as possible) strategy. Simulation results are reported in order to show performance improvement obtained by RLP, in comparison with the two classical skip-over algorithms, namely RTO and BWP, introduced about ten years ago.

Key-Words: - Real-time systems, Periodic tasks, Scheduling, Quality of Service, Skip over model, Earliest Deadline

1 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. What is common for most real-time tasks is the Worst Case Execution Time (WCET), which has to be calculated in order to make predictions about the system behaviour, e.g., to guarantee timing requirements.

Real-time systems can be classified in three categories: hard, soft and firm.

- 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 firm 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 firm 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.

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 concept 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 contraints. In this context, the objective of a scheduling algorithm is to maximize the effective QoS of periodic tasks given by the number of task instances which complete before their deadline\(^1\). 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).

\(^1\) work supported by the French research office, grant number 01 K 0742.
Possible) for the Skip-Over model. Simulation results are reported in section 4 in order to show RLP performances compared to RTO and BWP. In section 5, we provide an algorithmic description of the RLP scheduler and we report measures from an implementation experience, in terms of footprint and time overheads. Section 6 summarizes our contribution and gives directions for future works.

2 Background materials

2.1 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.

2.2 RTO and BWP algorithms

Two scheduling algorithms were introduced about ten years ago. The first one proposed by Koren and Shasha is the Red Tasks Only (RTO) algorithm. Red instances are scheduled as soon as possible according to Earliest Deadline First (EDF) algorithm [4], while blue ones are always rejected.

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

To illustrate BWP, let us consider a set of five periodic tasks $T = \{T_0, T_1, T_2, T_3, T_4\}$ 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.

Compared with RTO where no blue instance is executed, necessarily, BWP performs better. We observe that five deadline violations relative to blue task instances occur at time instants $t = 24$ (task $T_3$), $t = 30$ (tasks $T_2$ and $T_4$) and $t = 60$ (tasks $T_3$ and $T_4$), thus reducing the effective QoS.

### Table 1: Task parameters

<table>
<thead>
<tr>
<th>$T_i$</th>
<th>$C_i$</th>
<th>$P_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_0$</td>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>$T_1$</td>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>$T_2$</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>$T_3$</td>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>$T_4$</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>

Figure 1: BWP schedule

3 The RLP algorithm

3.1 Earliest Deadline as Late as possible

Let us review the fundamental properties of Earliest Deadline First algorithm (EDF), stated in [3] [5] which are the basic foundation of our approach for scheduling tasks in the skip-over model. In general, 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 hyperperiod (the least common multiple of task periods).

### 3.2 Principles of RLP

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 behaviour:

- 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 by a variant of the EDL algorithm taking skips into account [6]. 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 [5] 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).

### 3.3 Illustrative example

Let us consider the task set defined in section 2 et let us apply RLP to it. It can be observed that, thanks to RLP scheduling, the number of violations of deadline relative to blue task instances has been reduced to three. They occur at time instants \( t = 40 \) (task \( T_3 \)), and \( t = 60 \) (tasks \( T_3 \) and \( T_4 \)). Observe that \( T_3 \) first blue task instance which failed to complete within its deadline in the BWP case (see Figure 1), has enough time to succeed in the RLP case, since the execution of \( T_1 \) and \( T_0 \) 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 hyperperiod (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.

![RLP schedule](image)

**Figure 2**: RLP schedule

### 4 Performance measurements

#### 4.1 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 contraints 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 hyperperiods. Measurements rely on the ratio of periodic tasks which complete before their deadline. The evaluation is done by varying the periodic task load, \( U_p \).

#### 4.2 Observations

The results obtained for \( s_i = 6 \) (one instance every six can be aborted) are described on Figure 3. From the graph, we observe that BWP and RLP
outperform RTO for which the resulting QoS is constant whatever the periodic load. The QoS remains constant at the minimal rate given by \( \frac{5}{6} \) i.e 83%. Additional simulations, not reported in this paper, have been realized and show that higher is the skip parameter more significant is the advantage of RLP over RTO and BWP.

![Figure 3: QoS of periodic tasks varying periodic load](image)

**5 Implementation**

**5.1 Algorithmic description**

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

```c
void RLP_schedule(t : current time)
begin
while (task=next(waiting list)=not(Ø))
    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
        Place task into red ready list
    endif
    else
        if (blue ready list=Ø)
            Compute EDL_schedule
        endif
        if (Slack(t)!=0)
            Pull task from waiting list
            Place task into blue ready list
        endif
    endif
    endwhile
endwhile
/*Checking waiting list in order to release tasks*/
while (task=next(red ready list))=not(Ø))
if (blue ready list=not(Ø)) and (Slack(t)=0)
    Pull task from red ready list
    Place task into waiting list
endif
endif
endwhile
end
```

The RLP `schedule()` routine proceeds in three steps. In the first one, it examines blue ready list in order to or 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 hyperperiod. 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.

**5.2 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. RTO, BWP and RLP scheduling algorithms have been implemented into a Linux based real-time operating system, namely Cleopatre. All real-time services are available as independent software components,
dispatched in different shelves of a library, some of them being optional. This enables developers to build their own application-specific operating system from a wide set of configurable components including QoS schedulers. Measurements of footprints for the schedulers are given in Table 2.

<table>
<thead>
<tr>
<th>Scheduler</th>
<th>Hard Disk size (Kb)</th>
<th>Memory size (Kb)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTO</td>
<td>3.2</td>
<td>2.3</td>
</tr>
<tr>
<td>BWP</td>
<td>4.1</td>
<td>3.2</td>
</tr>
<tr>
<td>RLP</td>
<td>9.7</td>
<td>7.6</td>
</tr>
</tbody>
</table>

Table 2: Footprints

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 hyperperiod 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 4.

![Figure 4: Overheads of RTO, BWP and RLP](image)

As it can be seen from Figure 4, 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.

6 Conclusions and future work

The paper has described a scheduling algorithm dedicated to uniprocessor systems that 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 showed 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.

Recently, this approach was extended to cope with aperiodic tasks that arrive at unpredictable times. 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.

In this paper, the order used by the scheduler for executing the blue instances is based on the relative deadline but could be selected based on a particular performance metric. For future work, we plan to compare different methods for ordering ready blue instances, aiming to balance individual success ratios for stability and robustness motivations.

References: