Improving Task Responsiveness with Limited Preemptions

Yifan Wu and Marko Bertogna
Scuola Superiore S.Anna
Pisa, Italy
y.wu@sssup.it, marko@sssup.it

Abstract

The optimality of preemptive EDF *scheduling with relation to the achievable system utilization is a clear advantage of this scheduling policy for single processor real-time systems. However, recent works suggested that the run-time behavior of* EDF *might be improved by limiting the preemption support only to particular time instants, dividing each task into a sequence of non-preemptive chunks of execution, without affecting the schedulability of the system.*

In this paper, we will take a closer look to limited preemption EDF *scheduling (*LP-EDF*), evaluating the potential advantages offered by this policy in terms of response-time reduction and improved control performances. In particular, we will show how to increase the responsiveness of a control application by placing non-preemptive regions of maximal length at the end of the code of selected tasks. The effectiveness of the proposed method will be proved both analytically and by extensive simulations.*

1 Introduction

Limited-preemption EDF scheduling (LP-EDF) has been introduced by Baruah in [2] to join the beneficial effects of both preemptive and non-preemptive scheduling. The main benefit of non-preemptive scheduling is indeed the reduced number of context switches, with a limited scheduling overhead due to cache misses and to the additional need of storing the state of a preempted task in order to safely retrieve it when the task will be resumed. On the other side, executing each task non-preemptively might lead to limited schedulability performances due to the large blocking imposed on tasks with smaller deadlines. With LP-EDF, instead, a task is executed non-preemptively as long as this does not cause the system to become unschedulable. When the task executed for the maximum allowed non-preemptive interval, the processor is surrendered to the ready task having earliest deadline, according to EDF. Pseudopolynomial complexity algorithms are presented in [2, 4] to compute the durations of the maximum Non-Preemptive (NP) chunks for each task in the system. In this way, one can take advantage of the optimality of preemptive EDF with a reduced system overhead.

However, the benefits of limited preemption EDF are not limited to the smaller number of preemptions introduced in the system. We will show in this work how to exploit this technique to increase the responsiveness of a selected set of tasks, improving the control performances of a control system.

The remainder of the paper is organized as follows. Section 2 provides an overview of the main results on non-preemptive and limited-preemption scheduling. Section 3 defines our system model and the evaluated timing attributes. In Section 4, the existing response time analysis for preemptive EDF is briefly reminded, in order to introduce our extension to limited-preemption EDF. The effectiveness of the proposed approach is tested with an exhaustive set of experiments shown in Section 5. Finally, Section 6 states our conclusion.

2 Related Work

In this section, we briefly remind the main results on non-preemptive and limited-preemption scheduling.

In [12], Jeffay *et al.* proved that EDF is optimal even among non-preemptive work-conserving scheduling algorithms[1] for periodic or sporadic task sets. For these systems, a necessary and sufficient schedulability test with pseudo-polynomial complexity is provided. Moreover, it is shown that both the scheduling problem — find an algorithm that is able to schedule a feasible task set — and the feasibility problem — decide if a set of tasks can be scheduled without any deadline be missed — are NP-hard in the strong sense for concrete periodic task systems scheduled by non-preemptive algorithms[2].

Baruah and Chakraborty analyzed in [3] the schedulability of non-preemptive task sets under the recurring task model and showed that there exist polynomial time approximation algorithms for both preemptive and non-preemptive scheduling.

[1] A scheduling algorithm is work-conserving if the processor is never idled when a task is ready to execute. Note that EDF is not optimal among general non-preemptive schedulers (including non work-conserving ones).

[2] A concrete periodic task is a periodic task that comes with an assigned initial activation.

Mok and Poon presented in [15] sufficient conditions to guarantee the robustness (a.k.a. sustainability) of non-preemptive task systems, i.e., guaranteeing that the schedulability is not affected by the relaxation of one or more task timing requirements (like a decrease in the computation time or an increase in the period).

The idea of deferring preemptions until pre-determined points inside the task code has been first introduced by Burns in [7]. An algorithm called Fixed Priority with Deferred Preemptions (FPDP) has been proposed, describing as well an associated response time analysis. However, a flaw in the analysis has been later corrected by Bril *et al.* in [6].

With a similar idea, Baruah analyzed in [2] the Limited Preemption EDF scheduling algorithm (LP-EDF). The maximum amount of time for which a task may execute non-preemptively, without missing any deadline, is computed. Differently from the model adopted in [7, 6], there are no fixed preemption points, but the position of Non-Preempting Regions (NPR) may float inside the task code (provided it is shorter than the allowed length). The computation of the maximum NPR lengths has been later improved in [4].

A response time analysis for preemptive EDF has been described by Spuri *et al.* in [16, 17]. Such analysis has been extended to systems scheduled with non-preemptive EDF by George *et al.* in [10]. Palencia and Gonzalez applied similar techniques for more general (distributed) task systems in [11].

The idea of exploiting non-preemptive scheduling to improve control performances has been adopted by Buttazzo and Cervin in [8], where non-preemptive EDF is used to reduce task jitter.

3 System Model

We will consider a set τ composed by n periodic and sporadic real-time tasks. Each task τ_i is defined by a worst-case execution requirement C_i, a relative deadline D_i and a period, or minimum interarrival time, T_i (all parameters are assumed in the real numbers domain). Such a sporadic task generates an infinite sequence of jobs $\tau_{i,k}, k \in \mathbb{N}$, with the first job arriving at any time, and successive job-arrivals separated by at least T_i time units. Each job $\tau_{i,k}$ has an arrival time $a_{i,k}$, a finishing time $f_{i,k}$, and a deadline $d_{i,k} = a_{i,k} + D_i$. The starting time $s_{i,k}$ of job $\tau_{i,k}$ is the first time it is scheduled for execution.

We consider the scheduling of sporadic task systems upon a single processor, using the **Earliest Deadline First** (EDF) scheduling algorithm [13] with limited preemption (LP-EDF) [2, 4]. For each task τ_i, we will assume to know in advance the maximum amount of time for which it can execute non-preemptively. Such value can be computed using the techniques described in [2, 4], and will be denoted as Q_i. Note that $Q_i \leq C_i, \forall i$. Whenever $Q_i = C_i$, the task τ_i will always be executed non-preemptively.

Figure 1. Placement of the final NP chunk of a task τ_i.

3.1 Placement of NP chunks

In order to improve control performances for one or more tasks τ_i, we will place an NP region with length Q_i at the end of its worst-case execution, i.e., at time $C_i - Q_i$, as shown in Figure 1. This can be done considering the code executed by the task when it produces the largest worst-case execution time, and placing a preemptions disable command as close as possible to the instruction executed Q_i time-units before the end. When this is not possible, for instance because at time $C_i - Q_i$ the task is inside a loop, or it is calling a remote function which cannot be modified, the preemptions disable instruction is placed as soon as possible, resulting in a smaller non-preemptive region. Without losing generality, we assume Q_i to denote the effective length of such non-preemptive region.

3.2 Timing Attributes

To investigate the responsiveness of limited-preemption EDF, we will consider the following timing attributes:

- **Response Time** $R_{i,k}$ of a job $\tau_{i,k}$; it is the time that elapses between the job arrival time $a_{i,k}$ and its finishing time $f_{i,k}$:

$$R_{i,k} = f_{i,k} - a_{i,k}.$$

- **Input-Output Delay** (*IO delay*) $\Delta^{io}_{i,k}$ of a job $\tau_{i,k}$; it is the time between the start time $s_{i,k}$ and the finishing time $f_{i,k}$ of $\tau_{i,k}$:

$$\Delta^{io}_{i,k} = f_{i,k} - s_{i,k}.$$

- **Start Delay** (*Sampling delay*) $\Delta^{s}_{i,k}$ is the time between the arrival time $a_{i,k}$ and the start time $s_{i,k}$ of a job $\tau_{i,k}$:

$$\Delta^{s}_{i,k} = s_{i,k} - a_{i,k}.$$

Moreover, for each one of the above attributes $X_{i,k}$ (being X one of R, Δ^{io} and Δ^{s}), we define the corresponding task-based worst-case X_i and average $\bar{X}_i$ values, taking, respectively, the maximum and the average among all jobs of a task τ_i:

$$X_i = \max_k \{X_{i,k}\},$$

and

$$\bar{X}_i = \lim_{m \to \infty} \frac{\sum_{k=1}^{m} X_{i,k}}{m}. \quad (1)$$

Finally, three kinds of jitters will be defined, taking the maximum difference among all job values:

- **Response Jitter** j_i^r:

$$j_i^r = \max_k R_{i,k} - \min_k R_{i,k}.$$

- **Input-Output Jitter** (*IO jitter*) j_i:

$$j_i^{io} = \max_k \Delta_{i,k}^{io} - \min_k \Delta_{i,k}^{io}.$$

- **Start Jitter** (*Sampling jitter*) j_i:

$$j_i^s = \max_k \Delta_{i,k}^{s} - \min_k \Delta_{i,k}^{s}.$$

4 Response Time Analysis

The worst-case response time of a task in a system scheduled with EDF has been computed by Spuri et al. in [16, 17]. We briefly remind here the adopted technique.

Definition 1 (Deadline Busy Period)**.** *A deadline-d-busy period is an interval of continuous execution in which only instances with absolute deadline before* d *are scheduled*[3].

Theorem 1 (from [16, 17])**.** *The worst-case response time of a task* τ_i *is found in a deadline busy period in which all tasks but* τ_i *are released synchronously from the beginning of the deadline busy period, and at their maximum rate.*

4.1 Worst-Case Response Time of EDF

Exploiting Theorem 1, it is possible to compute the worst-case response time of each task τ_i, considering all deadline-$(a + D_i)$-busy periods for a meaningful set of possible release times a of jobs of τ_i, and taking the maximum response time among such jobs, as follows [16, 17]:

1. The maximum busy period length L is found considering a situation in which all tasks are released synchronously and at their maximum rate. Therefore, L is the smallest positive value satisfying the following equation:
$$L = \sum_{\tau_i \in \tau} \left\lceil \frac{L}{T_i} \right\rceil C_i.$$
2. For each task τ_i, the maximum deadline-busy period L_i is found considering all tasks but τ_i synchronously released at time $t = 0$. An algorithm for the computation of all L_i values is presented in Figure 2.
3. The length $L_i(a)$ of the deadline-$(a + D_i)$-busy period of a job of task τ_i that arrives a time units after the synchronous arrival of all other tasks is the smallest positive value satisfying the following equation:
$$L_i(a) = \left(1 + \left\lfloor \frac{a}{T_i} \right\rfloor\right) C_i + \sum_{\substack{j \neq i \\ D_j \leq a + D_i}} \min\left\{ \left\lceil \frac{L_i(a)}{T_j} \right\rceil, 1 + \left\lfloor \frac{a + D_i - D_j}{T_j} \right\rfloor \right\} C_j. \quad (2)$$

[3]Note that a *busy period* (without referring to any particular deadline) is instead just an interval of continuous execution.

COMPUTE BUSY PERIOD LENGTHS(τ)

```
1  A ≐ {a < L | a = kT_j + D_j, 1≤j≤n, k ∈ ℕ ∪ {0}};
2  L_{n+1} ← L;
3  for (i = n) down to 1
4     repeat
5        a ← max{ℓ ∈ A | ℓ ≤ L_{i+1} − C_i + D_i} − D_i;
6     until (L_i(a) > a)
7     L_i = L_i(a)
8  return L_1, ..., L_n
```

Figure 2. Algorithm for the computation of the maximum deadline busy period lengths.

4. The maximum response time of τ_i can be found as
$$R_i = \max_{a \in A_i} \{L_i(a) - a\}, \quad (3)$$
where
$$A_i \doteq \{a < L_i \mid a = kT_j + D_j - D_i, \ \forall j, \ k \in \mathbb{N} \cup \{0\}\}.$$

As proved in [10], if the task set utilization is strictly lower than 1, L exists and is pseudopolynomial, so that the algorithm may converge in a pseudopolynomial number of steps. If instead the total utilization is 1, the maximum busy period length L is bounded by the least common multiple of the periods of the tasks, when such value exists. In that case, the complexity of the algorithm is exponential.

In the next section, we will show how to modify the algorithm in order to take into account non-preemptive regions.

4.2 Worst-Case Response Time of LP-EDF

George et al. presented in [10] a method to compute the worst-case response time for task systems scheduled with non-preemptive EDF. The method takes into account the effect of priority inversion for the computation of the deadline busy period and can be used as well in analyzing the limited preemption EDF case.

Theorem 2. *The worst-case response time of a task* τ_i *is found in a deadline busy period for* τ_i *in which:*

- τ_i *has a job released at time* a *(and possibly other jobs released before);*
- *all tasks with relative deadline smaller than or equal to* $(a + D_i)$ *are released synchronously at time* $t = 0$ *and at their maximum rate;*
- *a further task* τ_j *with relative deadline greater than* $a + D_i$*, if any, starts executing a non-preemptive region of length* Q_j*, an arbitrarily small amount of time before* $t = 0$.

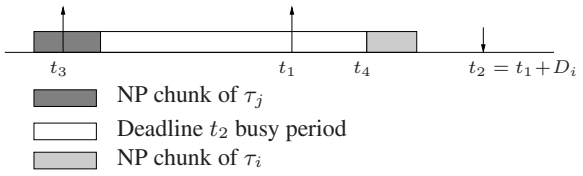


Figure 3. Worst-case response time scenario for Limited preemption EDF.

Proof. We adapt the proof contained in [10] to the case under consideration. Consider the scenario in Figure 3 where τ_i has a job with arrival time t_1 and absolute deadline $t_2 = t_1 + D_i$. Let t_4 be the start time of the NP chunk of τ_i, according to the rule defined in section 3.1. The actual execution time of the NP chunk is $\leq Q_i$. Finally let t_3 be the last time before or at t_1 such that there are no pending jobs with absolute deadlines before or at t_2.

By the choices made, t_3 must coincide with the release time of at least one job, and there cannot be idle time between t_3 and t_4. This means that the execution of τ_i's NP chunk of the job arrived at time t_1 is preceded by a busy period of those instances released between t_3 and t_4, and that have absolute deadlines before or at t_2, plus at most one other NP chunk released before t_3 with absolute deadline after t_2.

Consider now the scenario in which:

- all tasks but τ_i with relative deadline less than or equal to $(t_2 - t_3)$ are released from time $t = 0$ at their maximum rate;
- being $a = (t_1 - t_3)$, τ_i is released at time $\left(a - \left\lfloor \frac{a}{T_i} \right\rfloor T_i\right), \left(a - (\left\lfloor \frac{a}{T_i} \right\rfloor - 1)T_i\right), \ldots, a$;
- the task τ_j, if any, that attains the maximum values of $\max_{D_j > (t_2 - t_3)}\{Q_j\}$ is released an arbitrarily small amount of time before $t = 0$. That is, τ_j causes the worst possible priority inversion w.r.t. the absolute deadline $a + D_i$.

In the new scenario, the workload in the interval preceding the start time of the NP chunk of the considered instance of τ_i, released at time $a = t_1 - t_3$, cannot be less than in the previous scenario: the busy period preceding this NP chunk cannot be shorter, since it includes the worst-case priority inversion w.r.t the absolute deadline $a + D_i$, as well as the largest deadline-$(a + D_i)$-busy period preceding it. Therefore, τ_i's response time cannot diminish. □

Using Theorem 2, it is possible to adapt the algorithm described in Section 4.1 for the computation of the worst-case response time of a task τ_i to the limited preemption case. We will prove that we will only need to replace Equation (3) with

$$R_i = \max_{a \in A_i}\{L_i(a) - a + Q_i\}. \quad (4)$$

and Equation (2) with

$$L_i(a) = \max_{D_j > a + D_i}\{Q_j\} + \left(1 + \left\lfloor \frac{a}{T_i} \right\rfloor\right) C_i - Q_i + \sum_{\substack{j \neq i \\ D_j \leq a + D_i}} \min\left\{1 + \left\lfloor \frac{L_i(a)}{T_j} \right\rfloor, 1 + \left\lfloor \frac{a + D_i - D_j}{T_j} \right\rfloor\right\} C_j. \quad (5)$$

Proof. Differently from the worst-case response time analysis for preemptive EDF, where the considered busy period ends at the finishing time of the reference job of τ_i, in LP-EDF we focus on the busy period *preceding* the execution of the final non-preemptive region of the reference job (time t_4 in Figure 3). Hence all jobs released before this time should be taken into account.

Consider the scenario described in the proof of Theorem 2. Let $L_i(a)$ be the length of the busy period starting at time $t = 0$ and preceding the start time of the last non-preemptive region of a job of τ_i released at time a. Let $\tau_{i,k}$ be such job. The response time of $\tau_{i,k}$ is given by $L_i(a) - a + Q_i$. Taking the maximum over all jobs released by τ_i, Equation (4) follows.

Now, to prove that the length $L_i(a)$ can be determined by finding the smallest solution of Equation (5), note that the first term on the RHS accounts for the maximum possible priority inversion w.r.t. the absolute deadline $a + D_i$. The second and third terms correspond to the time needed to execute the jobs of τ_i released before or at time a, excluding the final NP chunk of $\tau_{i,k}$. Finally, the last term represents the time needed to execute the jobs of tasks $\tau_{j \neq i}$, with absolute deadlines $\leq (a + D_i)$, that are released before the beginning of the last NP region of $\tau_{i,k}$. Note that this term is slightly different from the corresponding term used in Equation (2), to make sure that the last NP region of $\tau_{i,k}$ has already started[4]. □

Observing Eq. (5), it is clear that the length of the busy period used to compute the worst-case response time of τ_i is not affected by NP regions of any task τ_j having $D_j \leq D_i$. In other words, the responsiveness of a task does not change if NP regions are introduced inside the code of one or more tasks having shorter deadlines.

By taking advantage of this observation, we will analyze in Section 5 the performances of the controller task having the largest relative deadline among tasks that have non-negligible NP regions.

5 Experimental Results

5.1 Experiment Setup

We consider a system with $n = 7$ hard real-time tasks. We will monitor the timing attributes listed in Section 3.2

[4]The analysis can be tightened adopting techniques used in [6] for Fixed Priority scheduling. In particular, the term $\left(1 + \left\lfloor \frac{L_i(a)}{T_j} \right\rfloor\right)$ can be replaced by the tighter term $\left(\left\lceil \frac{L_i(a)}{T_j} \right\rceil\right)$, whenever there is a positive blocking term (i.e., if $\max_{D_j > a + D_i}\{Q_j\} > 0$). See [6] for further details.

for task τ_1, having period $T_1 = 50ms$ and execution time $C_1 = 5ms$ (and, therefore, utilization $U_1 = 0.1$). The other 6 tasks are generated using UUNIFAST algorithm [5], with period T_i uniformly distributed in $[10,\ 100]\ ms$ and utilization U_i chosen according to a 6-dimensional uniform distribution to reach the desired total utilization. No particular task ordering is assumed. For all tasks, including τ_1, D_i equals T_i.

The system utilization varies from 0.2 to 1 with steps of 0.1. For each utilization, 500 task sets are randomly generated. For each task set τ, three scheduling policies are tested:

- Fully preemptive Earliest Deadline First policy, denoted as EDF;
- Limited-preemption EDF, placing the largest possible non-preemptive regions (whose lengths Q_i are computed using the algorithm described in [2]) at the end of the execution of each task; this policy will be denoted as LP-EDF;
- Limited-preemption EDF, placing the largest possible non-preemptive regions at the end of the execution of those tasks having a relative deadline $\leq D_1$, and scheduling the remaining ones with preemptive EDF; this policy will be denoted as LP-EDF*;

A schedule is generated for each one of the above policies, for a simulation length of 40 seconds, running in TrueTime [9].

We will measure the average values defined in Section 3.2, approximating Equation (1) with the the following expression:

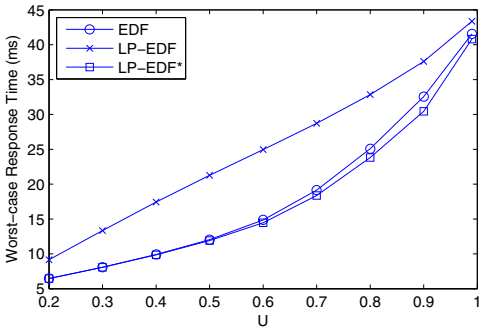
$$\bar{X}_i = \frac{\sum_{k=1}^{m_i} X_{i,k}}{m_i}, \tag{6}$$

where m_i is the total number of jobs of τ_i generated during the $40s$ of simulation. Since the simulation time is very large, Equation (6) approximates well Equation (1).

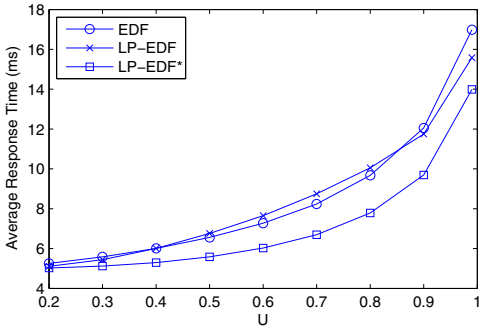
5.2 Responsiveness Results

In Figure 4, we show the worst-case and average response times of task τ_1. The worst-case response time is computed with the method described in Section 4, while the average response time is derived as described above.

The results show that the smallest worst-case response times are obtained with LP-EDF*, thanks to the NP region of task τ_1. The largest worst-case response times are instead obtained with LP-EDF, due to the blocking imposed by the non-preemptive regions of lower-priority tasks. The lower subgraph shows that, with limited preemption scheduling, the system can achieve better average response time. More specifically, LP-EDF* gives always the best (shortest) average response time, which are from 20% to 30% smaller than the average response times of preemptive EDF. Under LP-EDF, instead, the average response time is comparable to the EDF case, being slightly



(a) Worst-case Response Time of τ_1



(b) Average Response Time of τ_1

Figure 4. Response Time of τ_1.

shorter only when the system utilization is below 0.4, or above 0.85.

To highlight the improvement over preemptive EDF, we found it useful to introduce the **Relative Response Time Improvement** $\hat{R}_i^A$, defined as the difference between the average response time under preemptive EDF and under a particular scheduling policy A, divided by the task period, i.e.,

$$\hat{R}_i^A = \frac{\bar{R}_i^{\text{EDF}} - \bar{R}_i^A}{T_i}$$

where $\bar{R}_i^A$ is the average response time of τ_i under a policy A. Notice that $\hat{R}_i^A > 0$ means that the response time of τ_i is reduced (improved) under the policy A w.r.t. preemptive EDF.

The **Overall Relative Response Time Improvement** $\hat{R}$ is defined as the mean of $\hat{R}_i$ among all tasks τ_i:

$$\hat{R} = \frac{\sum_{\tau_i \in \tau} \hat{R}_i}{n}.$$

In Figure 5, the two solid curves for LP-EDF and LP-EDF* show that the average response time is improved using LP-EDF* and LP-EDF in highly loaded system, w.r.t the preemptive EDF case. Moreover, under LP-EDF*, τ_1 achieves *always* a much lower average response time, and $\hat{R}_1^{\text{LP-EDF}^*} > 0$ for all tested utilizations.

The dashed curves, however, show that the overall relative response time improvement $\hat{R}$ of the whole task set is negative, both for LP-EDF* and LP-EDF. This can be explained by the response-time "redistribution" that takes

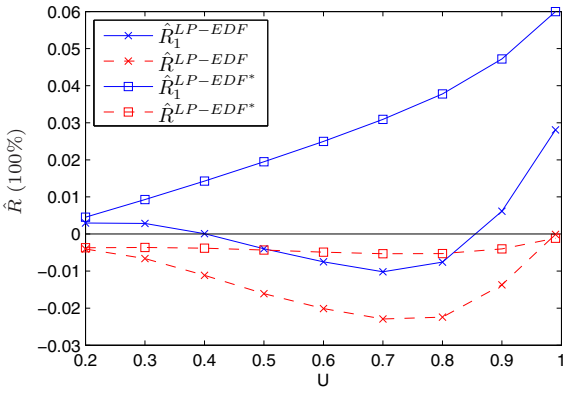


Figure 5. Relative Response Time Improvement ($U_1 = 0.1$).

places when introducing NP regions: the average response times of shorter-period tasks are increased due to the extra blocking time from lower priority tasks; instead, the average response time of longer-period tasks are reduced, thanks to the non-preemptive execution of their NP regions. However, because of the smaller periods, the increase in the relative response time of shorter-period tasks is more significant than the reduction for longer-period tasks. Nonetheless, the negative effect is relatively small in LP-EDF*, since there are less priority inversions for shorter-period tasks.

Figure 6 shows the timing attributes of τ_1. As depicted, both LP-EDF and LP-EDF* perform well in minimizing IO delay Δ_1^{io} and IO jitter j_1^{io}. It can be easily proved that the IO delay with limited preemptions is *always* smaller than with preemptive EDF, due to the smaller interference suffered by a task in its last chunk. Moreover, since once a task starts executing, it cannot be preempted by tasks with larger relative deadlines, the IO delay of τ_1 with LP-EDF is always identical to the IO delay with LP-EDF*, as shown in the figure. Both algorithms have IO delay Δ_1^{io} very close to $C_1 = 5ms$ (and IO jitter j_1^{io} close to zero), meaning that most of τ_1's jobs will be able to execute entirely non-preemptively.

Regarding the start delay Δ_1^s and the start jitter j_1^s, LP-EDF* has the same performances as preemptive EDF. This can be explained by noting that no job $\tau_{1,k}$ of task τ_1 can be blocked by the NP regions of tasks having smaller relative deadlines, since either they have an absolute deadline earlier than $d_{1,k}$, or they arrive after $a_{1,k}$. Instead, both the start delay Δ_1^s and the start jitter j_1^s increase with LP-EDF, due to the blocking of lower priority tasks.

Finally, the response jitter j_1^r is influenced by both j_1^s and j_1^{io}, being $R_{1,k} = \Delta_{1,k}^s + \Delta_{1,k}^{io}$. The smallest response jitter is obtained with LP-EDF*, while the larger response jitter of LP-EDF is due to its large start jitter.

Similar results are obtained changing τ_1's execution time. Due to space reasons, we include here only the case with $C_1 = 10ms$ (and $U_1 = 0.2$). The results for $\hat{R}$ and

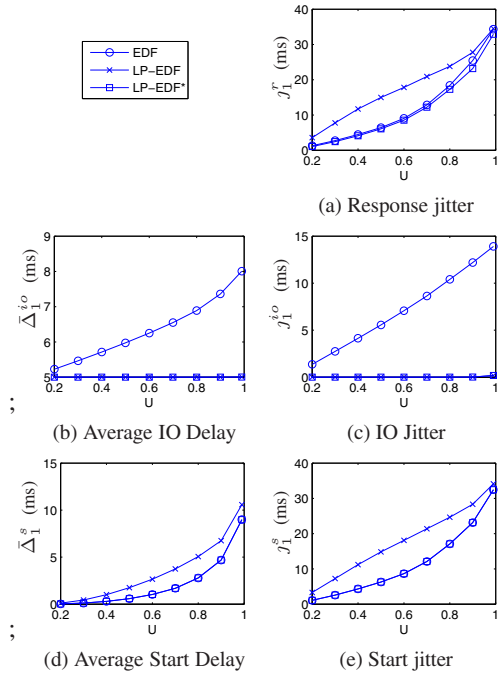


Figure 6. Timing Attributes of τ_1 ($U_1 = 0.1$).

the other timing attributes are shown, respectively, in Figure 7 and 8. Note that increasing τ_1's utilization, there is a more significant improvement (over preemptive EDF) in the average response time of τ_1 for both LP-EDF and LP-EDF*, as testified by the positive values of $\hat{R}_1^{\text{LP-EDF}}$ and $\hat{R}_1^{\text{LP-EDF}^*}$. Jitters and delays are similar to the previous case. Note that τ_1's IO delay and jitter are not always constant, but they increase for heavy loads, meaning that τ_1 is not always able to execute non-preemptively. Nevertheless, the values of Δ_1^{io} and j_1^{io} for LP-EDF and LP-EDF* are still significantly smaller than with EDF.

5.3 Control Performance Results

When a controller is implemented as a hard real-time task running in a multi-threaded environment, the scheduling-induced delay and jitter affect assumptions like the constant sampling period and the null, or constant, input-output delay, degrading control performances [14, 8]. In general, a control task achieves better performance if it experiences smaller delay and jitter at runtime. To show how limited preemption scheduling can be exploited to increase the responsiveness and, accordingly, the performances of a controller tasks, we considered the following benchmark control system, inspired by the example shown in [8].

An inverted pendulum with natural frequency of 6 rad/s is controlled by a Linear Quadratic Gaussian (LQG) controller [1]. The state-space model of the in-

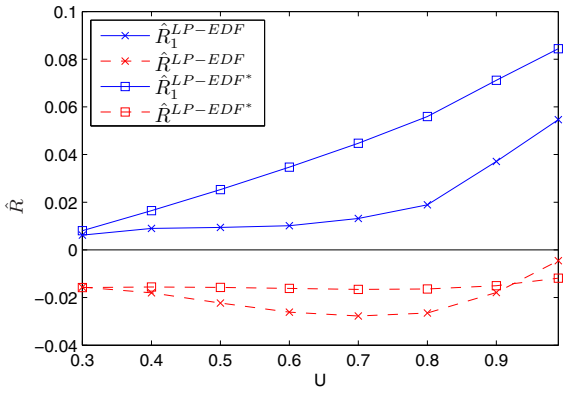


Figure 7. Relative Response Time Improvement ($U_1 = 0.2$).

verted pendulum is:

$$\frac{dx}{dt} = \begin{bmatrix} 0 & 1 \\ 36 & 0 \end{bmatrix} x + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u + \begin{bmatrix} 1 \\ 0 \end{bmatrix} v$$
$$y = \begin{bmatrix} 0 & 1 \end{bmatrix} x + e$$

where x is the state vector, u is the control signal, y is the measurement signal, v is a continuous-time Gaussian white-noise process with zero mean and variance 1, and e is a discrete-time Gaussian white-noise process with zero mean and variance 0.1.

A quadratic cost function J is provided to design the LQG controller as well as to evaluate the control performance.

$$J = E \lim_{t_p \to \infty} \frac{1}{t_p} \int_0^{t_p} \left(x^T \begin{bmatrix} 0 & 0 \\ 0 & 10 \end{bmatrix} x + u^2 \right) dt$$

where $[0, t_p]$ is the time span to be considered. Although from a theoretical point of view t_p should be ∞, in practice we could use a large enough value to evaluate the control performance. In our experiment, t_p is set to the simulation time, i.e., 40 seconds. Notice that the cost function is defined so as to minimize the state error and control energy. Therefore, a larger cost implies a worse control performance (see [8]).

The simulation setup is the same as in Section 5.1 with task τ_1 being the controller task. Hence, the sampling period of the controller is 50ms. Assuming sampling (input) and control signal actuation (output) happen, respectively, at the start time and at the finishing time of each job of τ_1, the lateness of the controller is equal the lateness of τ_1.

¿From Figure 9, we notice that by employing limited-preemption scheduling, the control performance improves (the cost is reduced) w.r.t. standard EDF, owing to the reduction of IO delay and jitter. The performance improvement of LP-EDF, however, is smaller than with LP-EDF* because of the negative effect on sampling delay and jitter. Nevertheless, the results demonstrate that the enhanced

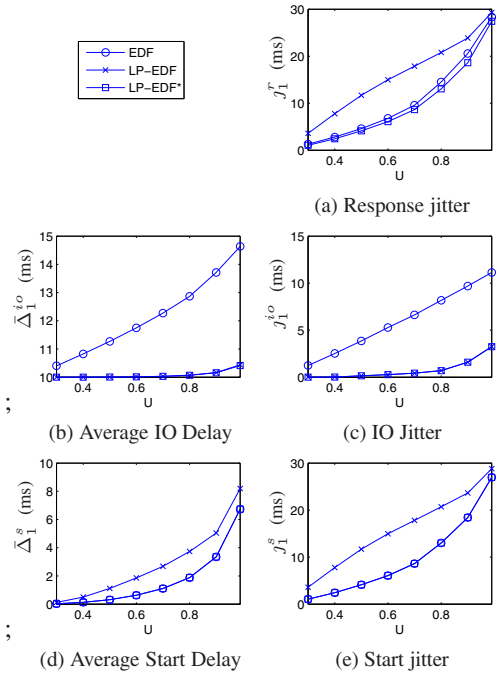


Figure 8. Timing Attributes of $\tau_1 (U_1 = 0.2)$.

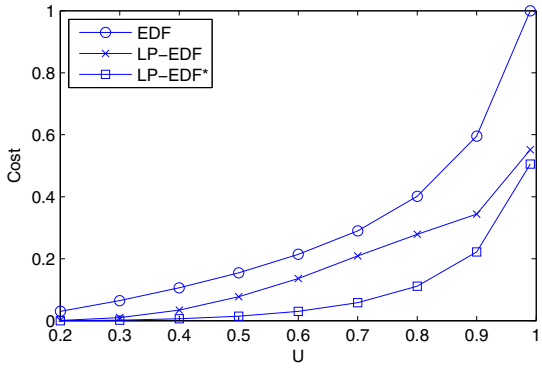


Figure 9. Control Performance of τ_1.

responsiveness obtained with limited preemption scheduling helps achieving better performances in real-time control systems.

6 Conclusions

We proposed the application of limited preemption EDF scheduling to improve the responsiveness of selected tasks in a uniprocessor real-time system. In particular, (i) we suggested to execute non-preemptively the last chunk of code of each control task, in order to improve the control performances; (ii) for each such task, we provided an algorithm to compute the worst-case response time, extending a previously proposed method for non-preemptive systems; (iii) we evaluated the proposed policy on a randomly generated task distribution, measuring average timing parameter that determine the performances of a con-

trol system. Our simulations, together with an example case-study, showed the effectiveness of the proposed approach.

References

[1] Karl Johan Åström and Björn Wittenmark. *Computer-Controlled Systems*. Prentice Hall, 3rd edition, 1997.

[2] Sanjoy Baruah. The limited-preemption uniprocessor scheduling of sporadic task systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, pages 137–144, Palma de Mallorca, Balearic Islands, Spain, July 2005. IEEE Computer Society Press.

[3] Sanjoy Baruah and Samarjit Chakraborty. Schedulability analysis of non-preemptive recurring real-time tasks. In *International Workshop on Parallel and Distributed Real-Time Systems (IPDPS)*, pages 8 pp.–, Rhodes, Greece, April 2006.

[4] Marko Bertogna and Sanjoy Baruah. Uniprocessor scheduling of sporadic task systems under preemption constraints. *IEEE Transactions on Computers*, 2009. In submission. Available for download at http://retis.sssup.it/ marko/publi.html.

[5] Enrico Bini and Giorgio C. Buttazzo. Biasing effects in schedulability measures. In *ECRTS '04: Proceedings of the 16th Euromicro Conference on Real-Time Systems*, pages 196–203, Catania, Italy, July 2004.

[6] Reinder J. Bril, Johan J. Lukkien, and Wim F. J. Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption revisited. In *ECRTS '07: Proceedings of the 19th Euromicro Conference on Real-Time Systems*, pages 269–279, 2007.

[7] A. Burns. Preemptive priority based scheduling: An appropriate engineering approach. *In S. Son, editor, Advances in Real-Time Systems*, pages 225–248, 1994.

[8] Giorgio Buttazzo and Anton Cervin. Comparative assessment and evaluation of jitter control methods. *Proc. of the 15th International Conference on Real-Time and Network Systems (RTNS2007)*, pages 137–144, March 29-30, 2007.

[9] Anton Cervin, Dan Henriksson, Bo Lincoln, Johan Eker, and Karl-Erik Årzén. How does control timing affect performance? Analysis and simulation of timing using Jitterbug and TrueTime. *IEEE Control Systems Magazine*, 23(3):16–30, June 2003.

[10] Laurent George, Nicolas Rivierre, and Marco Spuri. Preemptive and non-preemptive real-time uniprocessor scheduling. Technical Report RR-2966, INRIA: Institut National de Recherche en Informatique et en Automatique, 1996.

[11] José Carlos Palencia Gutiérrez and Michael González Harbour. Response time analysis of EDF distributed real-time systems. *Journal of Embedded Computing*, 1(2):225–237, 2005.

[12] K. Jeffay, D. Stanat, and C. Martel. On non-preemptive scheduling of periodic and sporadic tasks. In *Proceedings of the 12th Real-Time Systems Symposium*, pages 129–139, San Antonio, Texas, December 1991. IEEE Computer Society Press.

[13] C. L. Liu and James Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[14] Pau Martí. *Analysis and Design of Real-Time Control Systems with Varying Control Timing Constraints*. PhD thesis, Automatic Control Department, Technical University of Catalonia, July 2002.

[15] Aloysius K. Mok and Wing-Chi Poon. Non-preemptive robustness under reduced system load. In *RTSS '05: Proceedings of the 26th IEEE International Real-Time Systems Symposium*, pages 200–209, Washington, DC, USA, 2005. IEEE Computer Society.

[16] Marco Spuri. Analysis of deadline scheduled real-time systems. Technical Report 2772, Institut National de Recherche en Informatique et en Automatique, 1996.

[17] John A. Stankovic, Marco Spuri, Krithi Ramamritham, and Giorgio C. Buttazzo. *Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms*. Kluwer Academic Publishers, 101 Philip Drive, Assinippi Park Norwell, MA 02061, USA, 1998.