Constant-Time Admission Control for Partitioned EDF

Alejandro Masrur   Samarjit Chakraborty   Georg Färber
Institute for Real-Time Computer Systems, TU Munich, Germany
{Alejandro.Masrur, Samarjit.Chakraborty, Georg.Faerber}@rcs.ei.tum.de

Abstract

An admission control test is responsible for deciding whether a new task may be accepted by a set of running tasks, such that the already admitted and the new task are all schedulable. Admission control decisions have to be taken on-line and, hence, there is a strong interest in developing efficient algorithms for different setups. In this paper, we propose a novel constant-time admission control test for tasks scheduled on identical processors under partitioned Earliest Deadline First (EDF), i.e., once tasks have been assigned to a processor they remain on that processor. In particular, to model demanding real-time systems, we consider the case where relative deadlines may be less than the minimum separation between two consecutive task activations or jobs. The main advantage of the proposed test is that the time it takes is independent of the number of tasks currently admitted in the system. While it is possible to adapt polynomial-time schedulability tests from the literature to design a linear or even constant-time admission control for this setup, the test we propose provides a better accuracy/complexity ratio. We evaluate this test through a set of detailed experiments based on synthetic tasks and a realistic case study consisting of a real-time multimedia server.

1. Introduction

In many modern real-time systems such as interactive game servers, web servers, virtual worlds, and multimedia/communication systems, tasks have to be accepted or rejected on-line based on whether they can be feasibly scheduled or not. As a consequence, such systems require efficient admission control tests with fast and predictable running times. In particular, the increasing use of multiprocessor platforms makes it necessary to develop admission control algorithms that are able to manage multiple processing units at the same time.

In this paper, we propose such an admission control test for real-time tasks scheduled on identical processors under partitioned Earliest Deadline First (EDF). The Quality of Service (QoS) constraints associated with many of the applications mentioned above imply that deadlines be smaller than task periods or the minimum separation between any two consecutive jobs. For example, in an interactive application like a networked computer game, each game packet needs to be processed well before the arrival of the next packet. Here, a task is a new network connection or a user joining the game, which if accepted results in a sequence of network packets (jobs). While the case where relative deadlines are equal to periods is easier to handle and has been well studied in the literature, deadlines smaller than periods lead to additional complication and is still a topic of interest.

In addition to schedulability analysis, an admission control test for multiprocessors has to deal with the task allocation problem, which is known to be NP-hard in the strong sense [12]. Nevertheless, to perform an on-line allocation, it is possible to adapt one of the known bin-packing heuristics (e.g., First Fit (FF) [13]) to the considered setting. In our case, we are concerned with partitioned scheduling, i.e., tasks that are allocated to a processor stay on that processor as long as they remain active. As a consequence, we can use uniprocessor schedulability tests in combination with the allocation heuristic to determine whether a new task can be assigned to a given processor.

From here on, let p_i denote the minimum separation between two consecutive jobs and d_i the relative deadline of a task T_i. Existing *exact* schedulability tests for the case $d_i < p_i$ on uniprocessors, e.g., [6] and [19], have pseudo-polynomial complexity and are less suitable for admission control (i.e., on-line testing). The reason for this is that the running times of pseudo-polynomial schedulability tests depend on the deadlines of tasks, execution times, etc. Hence, it is complicated to precisely bound them in on-line settings where tasks come and go.

On the other hand, some polynomial-time *sufficient* tests have also been proposed, e.g., the density test [20, 15] and Devi's test [11]. Among all known polynomial-time tests for the case $d_i < p_i$, the density test has the minimum complexity. By combining the density test with FF, the admission decision for a new task on a multiprocessor system requires constant time $\mathcal{O}(1)$—of course, assuming that the number of processors is limited. Nevertheless, the resulting constant-time admission control has a poor accuracy. How a *polynomial-time* schedulability analysis is combined with FF to obtain an $\mathcal{O}(1)$ admission control is explained later in Section 6.

In [11], Devi showed an interesting accuracy improvement over the density test. However, Devi's test requires tasks to be sorted according to non-decreasing relative deadlines and, hence, has a higher complexity than the density test. Of course, it is possible to sort tasks on-line, so that we only need to add tasks to a sorted list as they arrive. However, if a new task is added to the system, using Devi's test also implies retesting all already accepted tasks with longer deadlines. This results in an admission control with linear complexity $\mathcal{O}(n)$ to test a new task in the system. As a consequence, the running time of this algorithm increases considerably as the number of accepted tasks grows, which is not desirable in many on-line settings.

Our contributions: Recall from above that the density test combined with FF results in a constant-time admission control algorithm for partitioned EDF. However, this solution is too pessimistic, particularly when the number of tasks grows. In order to increase the accuracy while retaining constant complexity, we propose a novel schedulability test to be combined with the FF heuristic. This test is based on calculating an upper bound on the *loading factor* produced on a given processor by the task set composed of all already running and the newly arriving task. The loading factor is defined as the maximum execution demand within a given time interval divided by the length of this interval [20]. Clearly, if the upper bound on the loading factor does not exceed 1, the maximum execution demand of all already running and the new task is always less than the available time for all possible time intervals. As a consequence, the new task does not affect the schedulability of the already running tasks and can be assigned to the processor.

To find an upper bound on the loading factor, it is necessary to compute the maximum execution demand of tasks for all possible time intervals. However, in this paper, we present an approximation technique that reduces the complexity of computing the maximum loading factor and allows designing a constant-time admission control test on its basis. The presented technique consists in partitioning the time line into non-overlapping intervals. The maximum execution demand and, thus, the loading factor are approximated this way by linear segments. Clearly, the approximation quality depends on the used number of linear segments. However, using more approximation segments increases the running time of the algorithm, which then has to be configured to achieve the desired accuracy and running time.

This paper is organized as follows: The next two sections provide a survey of related work and a description of the used task model and notation. The principles on which the proposed test is based will be presented in Section 4. Section 5 introduces and explains the proposed schedulability test, and the admission control algorithm for partitioned EDF will be discussed in Section 6. Section 7 presents a detailed comparison between the proposed admission control algorithm and well-known schedulability tests from the literature. Finally, some concluding remarks are presented in Section 8.

2. Related Work

As already stated, the FF heuristic can be combined with uniprocessor schedulability tests in order to derive admission control algorithms for partitioned EDF on multiprocessors. There are a number of approaches from the literature that can be used for this purpose, however, not all of them lead to constant-time admission control tests and we will have to sacrifice accuracy for the sake of efficiency.

On uniprocessor and for $d_i = p_i$, Liu and Layland [14] showed that an exact schedulability test can be performed in linear time. In this case, if the processor utilization does not exceed 1, tasks are schedulable under EDF [14]: $\sum_{i=1}^{n} \frac{e_i}{p_i} \leq 1$, where e_i is the worst-case execution time and n is the number of tasks.

For the considered case $d_i < p_i$, Baruah et al. [7] showed that the complexity increases considerably. However, assuming the processor utilization to be strictly less than 1, Baruah et al. [6, 7] proved that if a deadline is missed, this happens within a maximum time upper bound which can be computed. This result allows designing a pseudo-polynomial-time exact schedulability test for $d_i < p_i$. Another such algorithm based on a tighter time upper bound was presented by Ripoll et al. [19].

All exact schedulability tests for $d_i < p_i$, including the one of Albers and Slomka [2], incur pseudo-polynomial complexity and are normally not eligible for on-line testing. In order to reach polynomial complexity in testing schedulability for EDF with $d_i < p_i$, exactness must be sacrificed. Based on this idea, Liu in [15] and Stankovic et al. in [20] propose the density test, which consists in replacing p_i by d_i in the utilization test of Liu and Layland: $\sum_{i=1}^{n} \frac{e_i}{d_i} \leq 1$. Unfortunately, the accuracy of this test is rather poor.

To overcome the pessimism incurred by the density test, Devi [11] presented another approach based on sorting tasks according to their deadlines. However, Devi's test has a higher complexity (i.e., $\mathcal{O}(n \log n)$) than the density condition because tasks need to be sorted.

Other polynomial-time tests with better accuracy than both the density and Devi's test have been proposed for $d_i < p_i$, e.g., [10], [1] and [18]. Nevertheless, they all incur in a computational complexity of at least $\mathcal{O}(n \log n)$ as Devi's test.

So far we have discussed methods for uniprocessor schedulability analysis. There are also some contributions with respect to partitioned EDF on identical processors. For the case $d_i = p_i$, López et al. [17, 16] presented a schedulability test based on the concept of utilization bound. The key idea is that a given set of real-time tasks is schedulable on an identical multiprocessor system, if the total utilization does not exceed a certain bound.

More recently, Baruah and Fisher [3, 4, 5] presented and evaluated a polynomial-time algorithm to allocate tasks to identical processors considering the case $d_i < p_i$ and partitioned EDF. This algorithm is based on FF and on the first order approximation of the demand bound function proposed by Albers and Slomka [1]. Albers and Slomka proved in [2] that this first order approximated demand bound function is equal to Devi's condition [11]. Consequently, the algorithm presented by Baruah and Fisher is equivalent to combination of FF and Devi's test discussed previously. Baruah and Fisher's algorithm requires tasks to be sorted according to non-decreasing deadlines because of being based on Devi's test (Albers and Slomka's first order approximation). As a consequence, even if we can sort tasks on-line as they arrive, the algorithm of Baruah and Fisher yields an admission control test with linear rather than constant complexity to test a new task in the system. This is because, when a new task arrives to the system, all already accepted tasks with longer deadlines must be retested. Thus, the running time of this algorithm increases considerably for a large number of admitted tasks, which is normally not desirable for on-line testing in real-time systems.

In what follows, we introduce a schedulability test, which combined with FF results in a constant-time admission control test for identical processors. In contrast to the solution based on the density condition, the proposed test shows a much better accuracy as illustrated later.

3. Task Model and Notation

As we are concerned with the admission control problem, our task set is dynamic in the sense that tasks may arrive and leave from time to time. At a given time instant t, when a new task T_{new} arrives, $\mathbf{T}$ denotes the set of all tasks currently in the system plus T_{new}. In other words, $\mathbf{T}$ represents the state of the task system at the time instant at which T_{new}'s schedulability has to be tested. Throughout this paper, tasks are considered to be sporadic, independent and to run fully preemptively under partitioned EDF and on identical processors. We will use later $\mathbf{T}^l$ to denote any arbitrary subset of l tasks from $\mathbf{T}$.

Each task T_i in $\mathbf{T}$ is characterized by its relative deadline d_i, its worst-case execution time e_i and by its minimum inter-release time p_i, i.e., the minimum distance between two consecutive activations/jobs of T_i. Consequently, the ratio $u_i = \frac{e_i}{p_i}$ is T_i's *maximum* task utilization. As mentioned, relative deadlines d_i are assumed to be less than or equal to the respective minimum inter-release times p_i for all tasks.

As long as a task T_i is active in the system, it generates a potentially infinite succession of jobs. All jobs of T_i have the same worst-case execution time and relative deadline. Additionally, each job has its own release time and absolute deadline, i.e., the job's release time plus the relative deadline.

We denote $\mathbf{T}^l$'s *demand bound function* [6] by $h^l(t)$:

$$h^l(t) = \sum_{\forall T_i \in \mathbf{T}^l} \left(\left\lfloor \frac{t - d_i}{p_i} \right\rfloor + 1 \right) \cdot e_i.$$

The demand bound function $h^l(t)$ assumes the worst-case situation, namely, that jobs of all tasks in $\mathbf{T}^l$ are released together. Without loss of generality, the simultaneous release of jobs of all tasks in $\mathbf{T}^l$ is assumed to happen at time $t = 0$.

$\mathbf{T}^l$'s *loading factor* [20] is given by the ratio of its maximum execution demand in a given time interval divided by the length of this interval, and is denoted in this paper for $t > 0$ by:

$$\rho^l(t) = \frac{h^l(t)}{t}.$$

We denote by $\hat{\rho}^l = \max_{t>0} \left(\rho^l(t)\right)$ an upper bound on $\rho^l(t)$ and use $\hat{\rho}^l(t_x) = \max_{t_x \le t < t_{x+1}} \left(\rho^l(t)\right)$ to designate $\mathbf{T}^l$'s maximum loading factor in the time interval $[t_x, t_{x+1})$, where t_x and t_{x+1} are given points in time, $t_x < t_{x+1}$ and x is an integer number.

4. Theoretical Background

The schedulability test proposed in this paper computes an upper bound on the loading factor generated on a given processor by the task set $\mathbf{T}^l$. $\mathbf{T}^l$ consists of all tasks already running on the processor and a newly arriving task T_{new}. By definition, if the maximum loading factor does not exceed 1, $\mathbf{T}_l$ is schedulable (i.e., no deadline is missed) and the arriving task can be assigned to the processor.

The following two lemmas are concerned with finding such an upper bound on $\mathbf{T}^l$'s loading factor. As discussed later, we then apply these lemmas in the proposed schedulability test.

LEMMA 1 *Let $\mathbf{T}^{l-1}$ be a subset of $\mathbf{T}$, for which $\hat{\rho}^{l-1}(t_x)$ is the maximum loading factor within the time interval $[t_x, t_{x+1})$. If any task T_{new} is added to $\mathbf{T}^{l-1}$, where $t_x \le d_{new} < t_{x+1}$, the loading factor of the resulting subset $\mathbf{T}^l$ in $[t_x, t_{x+1})$ is bounded above by:*

$$\hat{\rho}^l(t_x) = \hat{\rho}^{l-1}(t_x) + \frac{e_{new}}{d_{new}}.$$

Proof: $\mathbf{T}^{l-1}$'s maximum loading factor in $[t_x, t_{x+1})$ is given by $\hat{\rho}^{l-1}(t_x)$. This means that $\rho^{l-1}(t) = \frac{h^{l-1}(t)}{t} \leq \hat{\rho}^{l-1}(t_x)$ holds in $[t_x, t_{x+1})$. It follows that $h^{l-1}(t) \leq t \cdot \hat{\rho}^{l-1}(t_x)$, i.e., $\mathbf{T}^{l-1}$'s demand bound function can be upper bounded in this interval. Hence, we can obtain an upper bound on $\mathbf{T}^l$'s loading factor in $[t_x, t_{x+1})$:

$$\rho^l(t) \leq \hat{\rho}^{l-1}(t_x) + \frac{\left(\lfloor \frac{t-d_{new}}{p_{new}} \rfloor + 1\right) \cdot e_{new}}{t}. \qquad (1)$$

Since $\hat{\rho}^{l-1}(t_x)$ is known and constant, we need to find the maximum possible value of the second term on the right-hand side of (1).

Now, this term is greater than zero if $t \geq d_{new}$ holds and, for $t = d_{new}$, it is equal to $\frac{e_{new}}{d_{new}}$. On the other hand, for $t > d_{new}$, $\lfloor \frac{t-d_{new}}{p_{new}} \rfloor$ increases at every $t = d_{new} + c \cdot p_{new}$ where $c > 0$ is a positive integer number. In this case, the second term of (1) can be expressed as $\frac{(c+1) \cdot e_{new}}{d_{new} + c \cdot p_{new}}$.

Further, it can be proven that $\frac{e_{new}}{d_{new}} \geq \frac{(c+1) \cdot e_{new}}{d_{new} + c \cdot p_{new}}$ holds for every integer number $c > 0$:

$$\begin{aligned} \frac{e_{new}}{d_{new}} &\geq \frac{(c+1) \cdot e_{new}}{d_{new} + c \cdot p_{new}}, \\ \frac{d_{new} + c \cdot p_{new}}{d_{new}} &\geq \frac{(c+1) \cdot e_{new}}{e_{new}}, \\ c\frac{p_{new}}{d_{new}} &\geq c. \end{aligned}$$

As $d_{new} \leq p_{new}$ holds, $\frac{e_{new}}{d_{new}}$ is the maximum possible value of this term and $\hat{\rho}^l(t_x) = \hat{\rho}^{l-1}(t_x) + \frac{e_{new}}{d_{new}}$ constitutes $\rho^l(t)$'s upper bound in $[t_x, t_{x+1})$. The lemma follows. □

LEMMA 2 *Let $\mathbf{T}^{l-1}$ be a subset of $\mathbf{T}$, for which $\hat{\rho}^{l-1}(t_x)$ is the maximum loading factor within the time interval $[t_x, t_{x+1})$. If any task T_{new} is added to $\mathbf{T}^{l-1}$, where $t_{x+1} > t_x \geq d_{new}$, the loading factor of the resulting subset $\mathbf{T}^l$ in $[t_x, t_{x+1})$ is bounded above by:*

$$\hat{\rho}^l(t_x) = \hat{\rho}^{l-1}(t_x) + \max\left(\frac{k \cdot e_{new}}{t_x}, \frac{(k+1) \cdot e_{new}}{t_k}\right),$$

where $k = \lfloor \frac{t_x - d_{new}}{p_{new}} \rfloor + 1$ and $t_k = d_{new} + k \cdot p_{new}$.

Proof: The maximum loading factor of $\mathbf{T}^{l-1}$ in $[t_x, t_{x+1})$ is given by $\hat{\rho}^{l-1}(t_x)$. That is, $\rho^{l-1}(t) = \frac{h^{l-1}(t)}{t} \leq \hat{\rho}^{l-1}(t_x)$ holds in $[t_x, t_{x+1})$, which yields $h^{l-1}(t) \leq t \cdot \hat{\rho}^{l-1}(t_x)$. As a consequence, the following inequality holds in $[t_x, t_{x+1})$, where the second term on the right-hand side is the loading factor of T_{new} alone:

$$\rho^l(t) \leq \hat{\rho}^{l-1}(t_x) + \frac{\left(\lfloor \frac{t-d_{new}}{p_{new}} \rfloor + 1\right) \cdot e_{new}}{t}. \qquad (2)$$

To find an upper bound on $\rho^l(t)$ in the interval $[t_x, t_{x+1})$, we first need to compute (2) at $t = t_x$:

$$\rho^l(t_x) \leq \hat{\rho}^{l-1}(t_x) + \frac{\left(\lfloor \frac{t_x-d_{new}}{p_{new}} \rfloor + 1\right) \cdot e_{new}}{t_x},$$

where $\lfloor \frac{t_x-d_{new}}{p_{new}} \rfloor + 1$ was denoted by k in this lemma. Further, removing the floor function in (2) and reordering, we obtain:

$$\rho^l(t) \leq \hat{\rho}^{l-1}(t_x) + \frac{(p_{new} - d_{new}) \cdot \frac{e_{new}}{p_{new}}}{t} + \frac{e_{new}}{p_{new}}. \qquad (3)$$

From (3), we know that the smallest possible t results in an upper bound on $\rho^l(t)$. However, we do not need to compute (3) until $t_k = d_{new} + k \cdot p_{new}$ for $k = \lfloor \frac{t_x - d_{new}}{p_{new}} \rfloor + 1$. This is because the loading factor for jobs in (t_x, t_k) cannot exceed the one at t_x. Replacing t by t_k in (3), we proceed as follows:

$$\begin{aligned} \rho^l(t) &\leq \hat{\rho}^{l-1}(t_x) + \frac{(p_{new} - d_{new}) \cdot \frac{e_{new}}{p_{new}}}{t_k} + \frac{e_{new}}{p_{new}}, \\ &\leq \hat{\rho}^{l-1}(t_x) + \frac{(p_{new} - d_{new}) \cdot \frac{e_{new}}{p_{new}} + t_k \cdot \frac{e_{new}}{p_{new}}}{t_k}. \end{aligned}$$

Now, taking into account that t_k is equal to $d_{new} + k \cdot p_{new}$, we finally obtain:

$$\rho^l(t) \leq \hat{\rho}^{l-1}(t_x) + \frac{(k+1) \cdot e_{new}}{d_{new} + k \cdot p_{new}}. \qquad (4)$$

Thus $\hat{\rho}^l(t_x)$, $\rho^l(t)$'s upper limit in $[t_x, t_{x+1})$, is given by $\hat{\rho}^{l-1}(t_x) + \max\left(\frac{k \cdot e_{new}}{t_x}, \frac{(k+1) \cdot e_{new}}{t_k}\right)$ and the lemma follows. □

5. Uniprocessor Schedulability

In this section, we introduce our schedulability test based on lemmas 1 and 2. Our test computes the maximum loading factor, which is produced on a given processor by the task set $\mathbf{T}^l$ of all already running and the newly arriving task. If the maximum loading factor does not exceed 1, the new task can be accommodated on that processor without causing deadline misses.

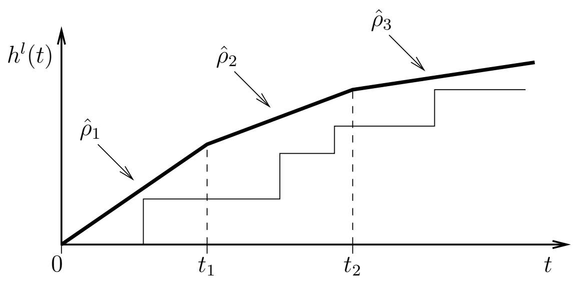

Figure 1. Approximation technique for computing the maximum loading factor

To calculate the maximum loading factor on a given processor, our algorithm partitions the time line into $b+1$ non-overlapping intervals considering that jobs of all tasks in $\mathbf{T}^l$ can be released simultaneously at time $t = 0$. This way, $\mathbf{T}^l$'s demand bound function and, hence, its loading factor are approximated by linear segments as illustrated in Figure 1. Clearly, the quality of the approximation depends on the number of segments we use. However, more segments result in a longer running time of the algorithm.

The first b intervals starting at time $t = 0$ have the same length t_{len}, while the last interval starts at $t_b = b \cdot t_{len}$ and covers the remaining time axis up to infinity. We enumerate intervals from $t = 0$ onwards, so that $[0, t_{len})$ is referred to as the first interval, $[t_{len}, 2t_{len})$ is the second one, $[t_b - t_{len}, t_b)$ is the b-th and $[t_b, \infty)$ is the $(b+1)$-th interval.

Figure 1 illustrates the used approximation technique for the case $b = 2$ (i.e., the time axis is divided into three). In Figure 1, $t_1 = t_{len}$ and $t_2 = 2t_{len}$ are the lower bounds of the second and third intervals. The maximum loading factor in the first time interval $[0, t_1)$ is represented by $\hat{\rho}_1$, whereas $\hat{\rho}_2$ and $\hat{\rho}_3$ denote the maximum loading factors in the second $[t_1, t_2)$ and third interval $[t_2, \infty)$ respectively.

Accommodating an arriving task into one of these intervals according to its deadline d_{new} allows more accuracy in computing the loading factor generated by $\mathbf{T}^l$ (i.e., the task set of all already accepted plus the new task) on the processor. This procedure can be associated to a restricted sorting of tasks according to deadlines so as to retain constant complexity. The execution demand of tasks $h^l(t)$ is approximated this way by $b+1$ slopes—in Figure 1, we have three slopes $\hat{\rho}_1$, $\hat{\rho}_2$ and $\hat{\rho}_3$. The maximum between $\hat{\rho}_1$ and $\hat{\rho}_2$ and $\hat{\rho}_3$ gives an upper bound on the loading factor of all tasks executing on the processor.

Figure 2 shows the pseudo-code of the proposed schedulability analysis in the form of an admission control test (i.e., the schedulability of a newly arriving task is tested on the system). The parameters b and t_b can be chosen arbitrarily to reach a desired accuracy/running time. In Figure 2, line 5, the length of the time intervals t_{len} is computed. Recall that the first b intervals have the same length and only the last interval $[t_b, \infty)$ is longer.

```
      // $T_{new}$: the newly arriving task
      // $b+1$: number of time intervals
      // $t_b$: start point of the last interval

5:    $t_{len} = \frac{t_b}{b}$;

      if $d_{new} \geq t_b$
         $\hat{\rho}_{b+1} = \hat{\rho}_{b+1} + \frac{e_{new}}{d_{new}}$;
      else
10:      $j = \lceil \frac{t_b - d_{new}}{t_{len}} \rceil$;
         $\hat{\rho}_{b-j+1} = \hat{\rho}_{b-j+1} + \frac{e_{new}}{d_{new}}$;

         for $x = b - j + 1$ to $b$
            $t_x = t_b - (b - x) \cdot t_{len}$;
15:         $k = \lfloor \frac{t_x - d_{new}}{p_{new}} \rfloor + 1$;
            $t_k = d_{new} + k \cdot p_{new}$;
            $\hat{\rho}_{x+1} = \hat{\rho}_{x+1} + \max\left(\frac{k \cdot e_{new}}{t_x}, \frac{(k+1) \cdot e_{new}}{t_k}\right)$;
         end
      end
20:
      if $\max_{1 \leq i \leq b+1}(\hat{\rho}_i) > 1$
         return("not schedulable");
      else
         return("schedulable");
      end
```

Figure 2. Proposed schedulability test for admission control based on Lemma 1 and Lemma 2

For an arriving T_{new}, if $d_{new} \geq t_b$ holds, we can apply Lemma 1 in line 8 to compute the maximum loading factor $\hat{\rho}_{b+1} = \hat{\rho}^l(t_b)$, i.e., $\mathbf{T}^l$'s maximum loading factor in $[t_b, \infty)$. As $d_{new} \geq t_b$ holds, adding T_{new} does not change the loading factor for intervals less than t_b.

On the other hand, if d_{new} is less than t_b, we need to find out in which interval the new deadline d_{new} fits. Now, the number of intervals of length t_{len} between d_{new} and t_b, including the one containing d_{new}, can be obtained calculating j in line 10. Clearly, including T_{new} in the system changes the loading factors from the $(b-j+1)$-th interval onwards.

Since d_{new} is greater than or equal to $t_{b-j} = t_b - j \cdot t_{len}$, $\mathbf{T}^l$'s maximum loading factor for the $(b-j+1)$-th interval, i.e., $\hat{\rho}_{b-j+1} = \hat{\rho}^l(t_{b-j})$, can be computed using Lemma 1—see Figure 2 line 11. From the $(b-j+2)$-th interval onwards, we can apply Lemma 2 because d_{new} is less than the lower bound of the $(b-j+2)$-th interval (i.e., t_{b-j+1}) and, consequently, it is less than the lower bounds of the following intervals. The maximum loading factors of these intervals are computed within the for-loop (lines 13 to 18). In line 14, the lower bound t_x of the corresponding interval is calculated. The values k and t_k required by Lemma 2 are computed in lines 15 and 16, whereas $\mathbf{T}^l$'s maximum loading factor for the $(x+1)$-th interval is obtained in line 17 (where x is the variable of the for-loop).

Finally, if the maximum loading factor in all the time intervals is not greater than 1 in line 21, $\mathbf{T}^l$ is schedulable on the processor and the new task T_{new} can be accepted. Otherwise, this algorithm rejects T_{new}.

6. Constant-Time Admission Control for Partitioned EDF

As discussed previously, we can use a bin-packing heuristic like FF [13] to perform an on-line allocation of tasks to processors. The FF heuristic consists in allocating tasks to a processor arrangement in a configured order starting by the same processor. A task can be assigned to a processor only if it is schedulable on that processor, i.e., if all currently running and the new task can meet their deadlines on the processor.

Now, FF can be combined with a proper schedulability test to solve the admission control problem under Partitioned EDF for the case $d_i < p_i$. Among the known tests for $d_i < p_i$, only the density condition leads to a constant-time admission control algorithm. In this case, the sum of densities for all tasks already running on a given processor (i.e., all tasks in $\mathbf{T}^{l-1}$) can be kept in memory: $\sum_{\forall T_i \in \mathbf{T}^{l-1}} \frac{e_i}{d_i}$. When a new task T_{new} arrives, only T_{new}'s density $\frac{e_{new}}{d_{new}}$ needs to be computed. If the sum of T_{new}'s density and densities in $\mathbf{T}^{l-1}$ does not exceed 1 for some processor, T_{new} can be assigned to it. As the number of processors is limited, the density test requires only $\mathcal{O}(1)$ time for testing a new task in the system.

However, to overcome the pessimism incurred by the density test, we use FF and the schedulability test of Figure 2. The test of Figure 2 requires constant time to compute how the new task changes the loading factor on a given processor and decide whether it can be assigned to that processor or not. (Notice that the for-loop in Figure 2 is limited by the constant b, which can be freely chosen to attain a desired accuracy.) Additionally, as the number of processors is limited, the resulting admission control algorithm also has constant complexity.

When a task T_{old} leaves the system, we have to update the loading factor on the corresponding processor. In this case, the algorithm of Figure 2 needs little modification. It is necessary to identify in which interval d_{old} is contained. Then, we need to compute again the *loading factor component* of the leaving task: $\frac{e_{old}}{d_{old}}$ or $\max\left(\frac{k \cdot e_{old}}{t_x}, \frac{(k+1) \cdot e_{old}}{t_k}\right)$ depending on the interval in which d_{old} is contained. And finally, instead of summing the loading factor component as for T_{new} in Figure 2 (lines 8, 11 and 17), T_{old}'s loading factor component has to be subtracted from maximum loading factor in the corresponding interval.

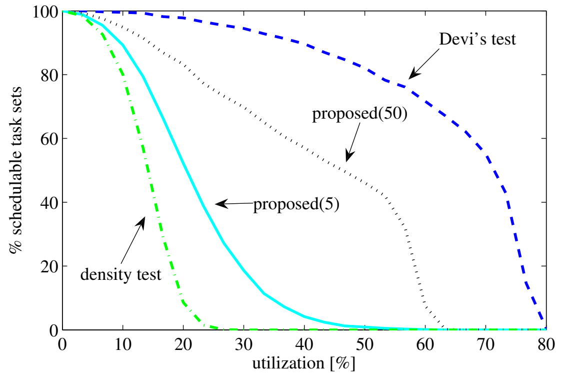

Figure 3. Schedulability versus utilization for 500 tasks

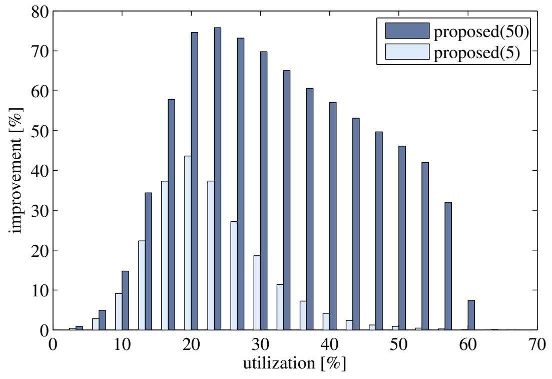

Figure 4. Improvement of the proposed test over the density test for 500 tasks

7 Experimental Results

In this section, we present some experimental results comparing the proposed admission control algorithm against possible approaches based on well-known techniques such as the density and Devi's test.

This section is structured in two subsections. The first one is concerned with a comparison of the mentioned methods on the basis of synthetic task sets, whereas, in the second subsection, we analyze a real-world application.

7.1 Synthetic Task Sets

The proposed test, the density test and Devi's test are compared with respect to their accuracy versus utilization for sets of 500 and 1000 tasks respectively. As discussed above, Devi's test has a higher complexity than the other two and cannot be used for constant-time admission control, however, it serves as a reference in this comparison. (Since Devi's test has a higher complexity, it results in a better acceptance ratio than all constant-time tests.)

The proposed test can be configured to compute an approximated maximum loading factor for different number of intervals $b + 1$. Clearly, the value of b influences the algorithm's performance. For larger values of b, the algorithms becomes more accurate, but, on the other hand, it has a longer running time. To illustrate the effect of b, we always consider two variants of this algorithm. In the case of 500 tasks, *proposed(5)* and *proposed(50)* are considered for $b = 5$ and $b = 50$ respectively, while *proposed(10)* (i.e., $b = 10$) and *proposed(100)* (i.e., $b = 100$) are taken into account for 1000 tasks per task set. The parameter t_b in Figure 2 was assigned to the average value of the relative deadlines in the task set. (In reality, it is not unusual to know the average relative deadlines of tasks.)

The accuracy of all the tests is measured as the percentage of schedulable task sets that they are able to accept on a single processor. Although we are concerned with the admission control problem, i.e., a new task should be admitted on a running system, the experimental results with synthetic tasks are presented in the form of whether an entire task set is schedulable or not. This way, if a given task set of n tasks is schedulable, it means that adding an n-th task to the set of $n - 1$ tasks was possible.

Task Set Generation: We first generated a random set of task utilizations u_i with *UUniFast* presented in [8, 9]. Then, we created p_i using a uniform distribution and obtained $e_i = u_i \cdot p_i$. The relative deadlines d_i were uniformly chosen from the range $[e_i, p_i]$. Additionally, we increased the utilization in uniform steps (a total of 24 steps) generating each time 10,000 different task sets.

Figure 3 shows the percentage of accepted task sets for the different algorithms and 500 tasks per set. Between 10% and 60% utilization, the proposed schedulability test can accept more task sets than the density test, whereas proposed(50) has a better performance than proposed(5). In the utilization interval $(15\%, 35\%)$, proposed(5) presents an improvement of around 20% more accepted task sets over the density test. The variant proposed(50) admits around 50% more task sets in the utilization interval $(15\%, 55\%)$. This is illustrated in Figure 4, where the length of bars represents the improvement over the density test.

In Figure 5, considering 1000 tasks per set, the proposed tests with $b = 10$ and $b = 100$ accept more task sets than the density test. The performance improvement over the density test is around 30% for $b = 10$ and in the utilization range $(15\%, 25\%)$. On the other hand, the variant proposed(100) ($b = 100$) outperforms the density test in approximately 50% more accepted task sets between 15% and almost 60% utilization. Figure 6 shows that the improvement over the density test where the longer bars stand for more accepted tasks sets.

The acceptance ratio of the proposed test can be bettered by improving the approximation of the loading factor. This can be achieved using more slopes in the approximation scheme, i.e., increasing b. However, if b is configured to be equal to $\lfloor 0.1 \cdot n \rfloor$ where n is the number of tasks (i.e., setting $b = 50$ when testing 500 tasks), the proposed test has an acceptable accuracy as shown in this section.

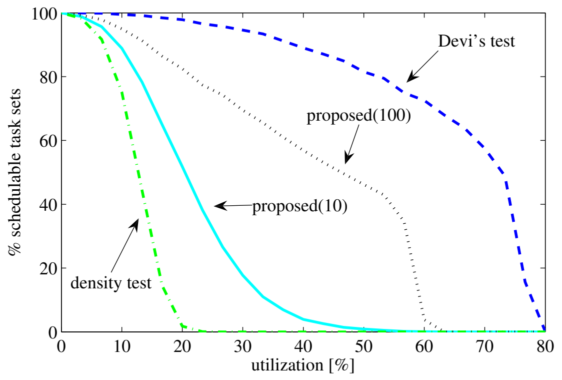

Figure 5. Schedulability versus utilization for 1000 tasks

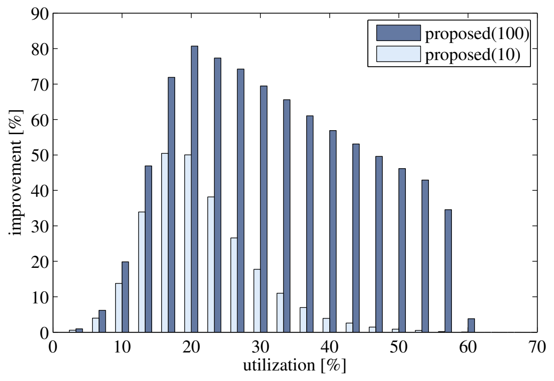

Figure 6. Improvement of the proposed test over the density test for 1000 tasks

7.2 Case Study

A comparison based on a real application illustrates the utility of the proposed constant-time admission control algorithm, i.e., the combination of FF with the test of Figure 2. Again, we consider two variants of this algorithm: *proposed(5)* and *proposed(10)* for $b = 5$ and $b = 10$ respectively. The parameter t_b was again set to the average or expected deadline value.

Besides comparing with the approach consisting of FF and the density test, we include in this comparison the algorithm based on FF and Devi's test as well. Clearly, combining FF and Devi's test results in an admission control with linear rather than constant complexity, i.e., the time taken by it depends on the number of tasks already accepted in the system. However, the combination of FF and Devi's test should serve as a reference in this comparison too.

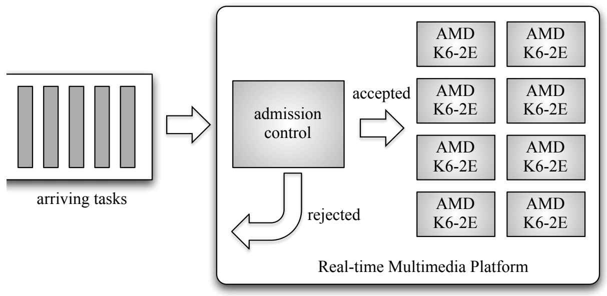

Figure 7. Setup for a real-time multimedia server

Task description	p_i	d_i	e_i
Matrix arithmetic	0.3176	0.0257	0.0009
Fast Fourier Transform	0.0192	0.0030	0.0016
Inverse FFT	0.0526	0.0055	0.0015
Compress JPEG	1.2821	0.1519	0.0560
Decompress JPEG	5.7866	0.4939	0.0450
High-pass gray-scale filter	0.5015	0.0494	0.0110
RGB to CYMK conversion	0.1073	0.0155	0.0077
RGB to YIQ conversion	0.0771	0.0208	0.0160
Image rotation	0.3597	0.0301	0.0021
Autocorrelation (sine)	0.0138	0.0014	0.0004

Table 1. Task pool based on E3S: parameters are given in seconds

The presented case study consists of a real-time multimedia server where requests from clients (tasks) are constantly arriving and have to be accommodated or rejected on-line. A number of modern multimedia applications such as high-definition video processing and interactive applications such as games are associated with tight timing constraints. In particular, the QoS requirements of these applications often necessitate that deadlines be less than the corresponding periods. As a consequence, many of the arriving tasks in our multimedia example have deadlines less than periods/minimum inter-release times.

The considered multimedia server consists of two to eight processors. In normal operation, only two of the processors are in active mode, while the other six remain in sleep mode and can be activated to cope with temporal increases of computation demand. The processors of this platform are identical and of type AMD K6-2E operating at 400 Mhz. Partitioned EDF scheduling is used, thus, once a task is assigned to a processor, it remains on that processor (i.e., task migration is not possible). Figure 7 shows a schematic representation of the described setup for the multimedia server in this example.

In order to obtain realistic tasks, we make use of the Embedded Systems Synthesis Benchmarks Suite (E3S) [21], which is based on embedded processor and task information from the Embedded Microprocessor Benchmark Consortium (EEMBC). We first created a *task pool* with tasks typically encountered in high-end multimedia applications such as autocorrelation, Fast Fourier Transform, compressing/decompressing JPEG, high-pass gray-scale filter, etc. Table 1 shows a brief description of tasks in the mentioned task pool together with their parameters (minimum inter-release time p_i, relative deadline d_i and worst-case execution time e_i).

An arbitrary task from this pool can arrive at any time to the multimedia server triggered by a client request. When a new task arrives, the admission control tests have to accept/reject it according to whether they can find a feasible allocation or not.

Figure 8 shows how many tasks the different admission control tests are able to accept when two processors are used. The density-based admission control can accept up to around 12 real-time multimedia tasks, after which it becomes pessimistic and starts rejecting almost all new tasks. On the other hand, the proposed admission control with $b = 5$ accepts around 5 additional tasks before starting to

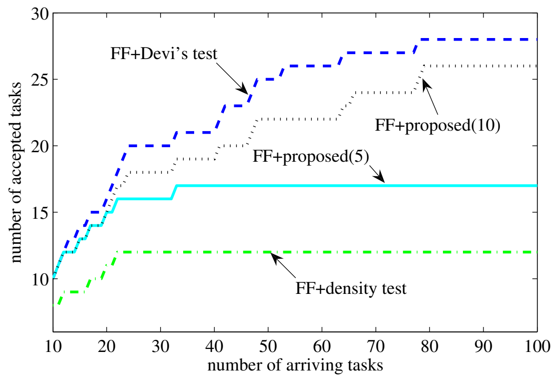

Figure 8. Number of accepted vs. number of arriving tasks considering two processors

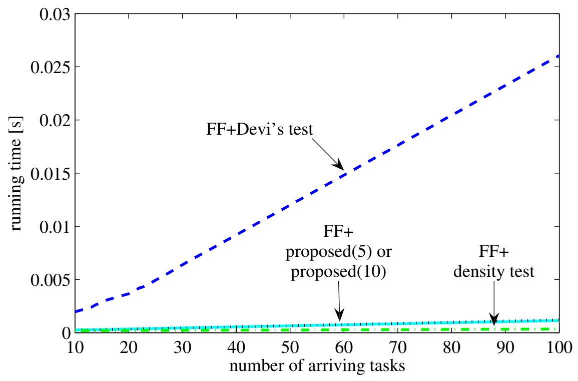

Figure 9. Running time vs. number of arriving tasks considering two processors

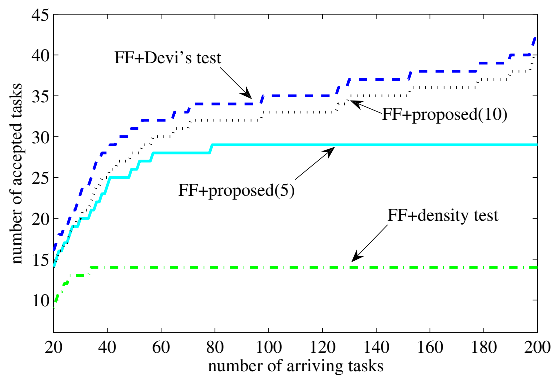

Figure 10. Number of accepted vs. number of arriving tasks considering four processors

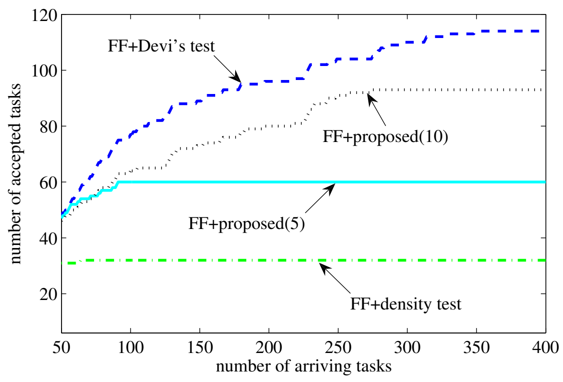

Figure 12. Number of accepted vs. number of arriving tasks considering eight processors

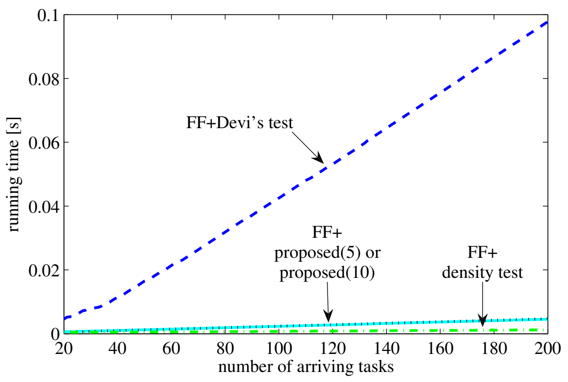

Figure 11. Running time vs. number of arriving tasks considering four processors

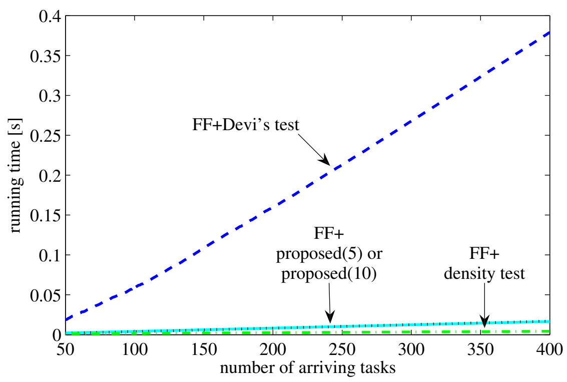

Figure 13. Running time vs. number of arriving tasks considering eight processors

reject other requests. The algorithm with $b = 10$ admits up to 15 more tasks than FF and the density test. This proposed algorithm with $b = 10$ accepts only 5 tasks less than the Devi's test approach.

Figure 9 shows the running times of algorithms as a function of the number of arriving tasks for the case of two active processors. The proposed test (for both $b = 5$ and $b = 10$) and the density test require around 1 to 2 orders of magnitude less time than the solution based on Devi's test.

Figures 10 and 11 are concerned with the case where four processor are in active mode. Because two more processors are available, the admission control algorithms are able to allocate more tasks than in the previous case. The density-based algorithm admits approximately 15 tasks before it starts denying acceptance to further requests. Proposed(5) (i.e., $b = 5$) is now able to allocate up to 27 tasks without much pessimism, after which it rejects almost all following tasks. Our algorithm with $b = 10$ can accommodate over 30 tasks. This algorithm is capable of allocating almost the same amount of tasks as the one based on Devi's test—see Figure 10. As shown in Figure 11, the algorithm based on Devi's test takes around 2 orders of magnitude longer time than the constant-time alternatives (again considering the running time as a function of the number of arriving tasks).

The performance of the admission control algorithms is shown in Figure 12 for eight active processors. Clearly, the number of accepted tasks increases because there is more computation capacity. The density-based admission algorithm is able to accommodate up to 30 tasks before it turns pessimistic and rejects further requests. The combination of FF and our test with $b = 5$ allows accepting around 30 more tasks than the density test, while the admission control based on the test with $b = 10$ is still more accurate and can assign approximately 90 tasks to the processors before becoming pessimistic. In this case, the Devi's test approach allows 10 more accepted tasks than the proposed test with $b = 10$. In Figure 13, the running times versus the number of arriving tasks are depicted for the case of eight processors in active mode. Here again, the admission control based on Devi's test still requires around 2 orders of magnitude additional running time than the constant-time algorithms.

8 Conclusions

In this paper, we presented an admission control test with constant complexity for partitioned EDF on identical processors and $d_i < p_i$ (i.e., this test requires a constant time to accept/reject a new task in the system with a fixed number of processors). The presented admission control is based on the combination of FF and a new schedulability test. Although the known density condition can also be used to derive a constant-time admission control, the test we propose is less pessimistic. We illustrated this by a set of detailed experiments with synthetic tasks and a case study consisting of a multimedia server.

The proposed test computes the maximum loading factor generated on a given processor by the set of all tasks already running on that processor plus the newly arriving task. By definition, if the maximum loading factor does not exceed 1, this task set is schedulable and the new task can be assigned to the considered processor. The test we propose partitions the time axis into intervals, such that the maximum loading factor is approximated by linear segments. The number of intervals determines the number of approximation segments. Clearly, if we use more segments, the approximation accuracy can be improved. However, many approximation segments increase the running time of the test.

How to configure the number of approximation intervals is a question that arises by our work. In our experiments with a large number of synthetic task sets, we showed that the proposed algorithm has a good performance, if we set the number of intervals to be 10% of the number of tasks (e.g., 50 intervals if we are expecting around 500 tasks to be active at the same time). However, this is an empirical result and some more analysis is still required.

In addition, a certain distribution of task parameters is often known in real-life applications (e.g., the expected workload on multimedia, communication or web servers is usually known). This previous knowledge about tasks can surly be used to configure the proposed test for a better performance. As future work, we plan to analyze common statistical distributions of task parameters and workload estimations on commercial servers, so as to use this information for improving the schedulability analysis.

Acknowledgments: We would like to thank the reviews for their valuable contributions to improving this paper.

References

[1] K. Albers and F. Slomka. An event stream driven approximation for the analysis of real-time systems. *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, June 2004.

[2] K. Albers and F. Slomka. Efficient feasibility analysis for real-time systems with EDF scheduling. *Proceedings of the DATE 05 Conference*, March 2005.

[3] S. Baruah and N. Fisher. The partitioned multiprocessor scheduling of sporadic task systems. *Proceedings of the 26th Real-Time Systems Symposium*, pages 321–329, December 2005.

[4] S. Baruah and N. Fisher. The partitioned multiprocessor scheduling of deadline-constrained sporadic task systems. *IEEE Transactions on Computers*, 55(7):918–923, 2006.

[5] S. Baruah and N. Fisher. The partitioned dynamic-priority scheduling of sporadic task systems. *Real-Time Systems*, 36(3):199–226, 2007.

[6] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. *Proceedings of the 11th IEEE Real-Time Systems Symposium*, December 1990.

[7] S. Baruah, L. Rosier, and R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor. *Real-Time Systems*, 2(4):301–324, November 1990.

[8] E. Bini and G. Buttazzo. Biasing effects in schedulability measures. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, June-July 2004.

[9] E. Bini and G. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[10] S. Chakraborty, S. Künzli, and L. Thiele. Approximate schedulability analysis. *Proceedings of the 23rd IEEE Real-Time Systems Symposium*, December 2002.

[11] M. Devi. An improved schedulability test for uniprocessor periodic task systems. *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, July 2003.

[12] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., San Francisco, USA, 1979.

[13] D. Johnson. *Near-Optimal Bin Packing Algorithms*. Massachusetts Institute of Technology (MIT), Department of Mathematics, Cambridge, USA, 1973. Ph.D. Thesis.

[14] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in hard real-time environments. *Journal of the Association for Computing Machinery*, 20(1):40–61, 1973.

[15] J. Liu. *Real-Time Systems*. Prentice Hall, 2000.

[16] J. López, J. Díaz, and D. García. Utilization bounds for EDF scheduling on real-time multiprocessor systems. *Real-Time Systems*, 28(1):39–68, 2004.

[17] J. López, M. García, J. Díaz, and D. García. Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems. *Proceedings of the 12th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 25–33, June 2000.

[18] A. Masrur, S. Drössler, and G. Färber. Improvements in polynomial-time feasibility testing for EDF. *Proceedings of the DATE 08 Conference*, March 2008.

[19] A. Ripoll, I. Crespo and A. Mok. Improvement in feasibility testing for real-time tasks. *Real-Time Systems*, 11(1):19–39, 1996.

[20] J. Stankovic, M. Spuri, K. Ramamritham, and G. Buttazzo. *Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms*. Kluwer, Dordrecht, The Netherlands, 1998.

[21] `http://ziyang.eecs.umich.edu/~dickrp/e3s/`.