Near-Optimal Constant-Time Admission Control for DM Tasks via Non-Uniform Approximations

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

***Abstract*—Admission control decisions involve determining whether a new task can be accepted by a running system such that the new task and the already running tasks all meet their deadlines. Since such decisions need to be taken on-line, there is a strong interest in developing fast and yet accurate algorithms for different setups. In this paper, we propose a constant-time admission control test for tasks that are scheduled under the Deadline Monotonic (DM) policy. The proposed test approximates the execution demand of DM tasks using a configurable number of linear segments. The more segments are used, the higher the running time of the test. However, a small number of segments normally suffice for a near-optimal admission control. The main innovation introduced by our test is that approximation segments are distributed in a non-uniform manner. We can concentrate more segments for approximating critical parts of the execution demand and reduce the number of segments where this does not change significantly. In particular, the tasks with shorter deadlines dominate the worst-case response time under DM and, hence, these should be approximated more accurately for a better performance of the algorithm. In contrast to other constant-time tests based on well-known techniques from the literature, our algorithm is remarkably less pessimistic and allows accepting a much greater number of tasks. We evaluate this through detailed experiments based on a large number of synthetic tasks and a case study.**

I. Introduction

In dynamic real-time systems, such as interactive computer games, virtual worlds, multimedia/communication servers, etc., a new task may arrive at any time. Such an arriving task needs then to be admitted or rejected on-line according to whether it can be feasibly scheduled or not. As a consequence, these systems require admission control tests with fast but also predictable running times.

In this paper, we propose an admission control test for real-time tasks scheduled under the Deadline Monotonic (DM) policy on a single processor. The test we propose has constant complexity, i.e., testing whether a new task can be feasibly scheduled does not depend on the number of tasks currently running in the system. The proposed test can also be combined with a bin-packing heuristic such as First Fit (FF) [1] to consider partitioned DM on identical processors. In this case, an arriving task is assigned once to a processor and remains its whole lifetime on that processor.

We consider the case where deadlines (d_i) are less than or equal to task periods or the minimum separation between any two consecutive jobs (p_i). In particular, deadlines less than periods are associated with demanding Quality of Service (QoS) constraints of many of the applications mentioned above. For example, in case of an interactive computer game over a network, 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).

As already mentioned, tasks are scheduled under the DM policy. DM assigns fixed priorities to tasks according to their deadlines: the shorter the deadline, the higher the priority assigned to the task. In contrast to dynamic priorities, fixed-priority scheduling policies are normally supported by commercial real-time operating systems and, hence, they are more relevant from a practical point of view. In addition, DM constitutes the optimal priority assignment for $d_i \leq p_i$ [2].

Exact pseudo-polynomial schedulability tests are already known for fixed priorities and $d_i \leq p_i$, e.g., [3], [4] and [5]. Although a high accuracy is always desirable, these tests are often not eligible for admission control. The reason for this is that the running time of such a pseudo-polynomial test depends not only on the number of tasks, but also on task parameters, like periods, execution times, etc. Hence, it is difficult to precisely bound such a running time in an on-line setting where the task set is constantly changing. Although the complexity of the admission control problem is the one of testing schedulability for only one new task in the system, an exact schedulability test still results in a pseudo-polynomial admission control test.

Additionally, all exact schedulability tests require tasks to be sorted according to decreasing priority, i.e., increasing deadlines under DM. Of course, it is possible to sort tasks on-line as they arrive. This way, when a new task arrives, we only need to add it to a sorted list. However, if a new task is added to the system, using an exact schedulability test also implies retesting all already accepted lower-priority tasks. This leads to additional delay and may be impractical, particularly, for a large number of tasks.

Polynomial-time approximation schemes have also been proposed [6], [7] for the known exact schedulability test [3], [4]. Nevertheless, these techniques also require tasks to be sorted according to priorities and have the same problem as the exact schedulability test. That is, if a new task is added to the system, all already admitted tasks with lower priority will have to be retested.

On the other hand, we can adapt utilization bounds obtained for Rate Monotonic (RM) to the case of DM and $d_i \leq p_i$. As discussed later, we can use, for example, the Liu and Layland bound [8] with very little modification. An admission control test based on this bound presents constant complexity $\mathcal{O}(1)$, however, it becomes extremely pessimistic as the number of accepted tasks grows.

The more recent *load test* [9] improves the accuracy when taking admission control decisions in constant time. This

test has a better performance than the other constant-time algorithms and allows reducing the number of rejections. However, the accuracy of this test is still poor when compared to the known exact test.

Our contributions: From the above discussion, we know that an admission control test incurs great pessimism in order to achieve constant complexity. However, a constant complexity results in fast and predictable running times and is desirable because of the on-line nature of the problem. To overcome this predicament, we introduce here a new admission control test that leads to a substantial improvement in the accuracy, while it retains constant complexity.

The test presented in this paper makes use of the concept of *loading factor*. The loading factor was originally defined for Earliest Deadline First (EDF) as the ratio between the maximum execution demand within a given time interval divided by the length of this interval [10]. Although this concept is still applicable here, we will have to adapt its definition to the case of DM.

Later we define the loading factor of a DM task as the ratio between its worst-case response time divided by its deadline. The proposed test computes an upper bound on the loading factor of all the already running tasks and the new one. Clearly, if this upper bound is less than or equal to 1, the worst-case response time of every task in the system does not exceed its respective deadline and, hence, the new task can be admitted.

To obtain such an upper bound on the loading factor, we need to compute the worst-case response time of each task in the system which is known to have pseudo-polynomial complexity [3], [4]. However, in this paper, we present an approximation technique that makes a constant complexity possible. The presented technique consists in partitioning the time line into non-overlapping intervals. Our test first accommodates an arriving task into one of these intervals according to its deadline. The resulting worst-case response time and, thus, the loading factor are then approximated by linear segments.

Our main observation in this paper is that the distribution of intervals over the time line has a big influence on the test's performance. As tasks with higher priority preempt the others, they have a bigger impact on the response time of the whole task set. As a consequence, approximating the execution demand of these tasks more accurately results in a better estimation of the worst-case response time of lower priority tasks. According to this, we use more segments to approximate the execution demand of tasks with shorter deadlines, since these have higher priority under DM. In a similar way, the execution demand of tasks with longer deadlines is approximated with fewer segments. These tasks have lower priority under DM and, consequently, less influence on the worst-case response time of other tasks.

This paper is organized as follows. After a survey of related work and a description of the task model and notation, we analyze the use of known techniques to design constant-time admission control tests in Section IV. The theoretical background for the proposed test is presented in Section V, whereas Section VI introduces and explains the proposed admission control test. In Section VII, we present a detailed comparison between the proposed admission control algorithm and well-known schedulability tests from the literature. To conclude this paper, Section VIII summarizes the most important aspects of our work.

II. Related Work

A number of sufficient tests have been proposed for RM. As shown later, these can be modified to derive admission control tests for the DM policy as well.

Liu and Layland [8] obtained a utilization bound for RM considering preemptive, independent, periodic real-time tasks and $d_i = p_i$. A better test for RM and $d_i = p_i$ was proposed independently by Liu [11] and by Bini et al. [12], [13]. Bini et al. called this test hyperbolic bound and proved that it improves the acceptance ratio over the utilization bound of Liu and Layland by a factor of $\sqrt{2}$ for a large number of tasks. A similar test was also proposed by Oh et al. [14] to be used in task allocation problems.

Other sufficient schedulability tests have also been proposed for RM and $d_i = p_i$. For example, Kuo and Mok [15] presented a bound that exploits the fact that 100% utilization is possible under RM when tasks have harmonic periods. Further, Burchard et al. [16] presented another utilization bound that varies not only with the number of tasks but also with a factor quantifying how close tasks are to having harmonic periods. For arbitrary deadlines, Lehoczky [17] proposed an RM utilization bound that depends on the number of tasks and on the ratio $\frac{d_i}{p_i}$ that is assumed to be the same for all tasks.

Joseph and Pandya [18] analyzed the problem of finding response times for fixed-priority tasks, from which they derived an exact pseudo-polynomial schedulability test. Further, Lehoczky et al. [3] presented an exact schedulability test also with pseudo-polynomial complexity for the case of RM and $d_i = p_i$. In [4], Audsley et al. improved Lehoczky's exact schedulability test by observing that tasks' worst-case response time can be found in an iterative manner. Further, Audsley et al. considered deadlines less than or equal to periods and other priority assignments. More recently, Bini and Buttazzo presented a tunable schedulability test [5] for $d_i \leq p_i$ and fixed priorities. Bini and Buttazzo's test allows configuring complexity versus acceptance ratio for the testing. In [6], Fisher and Baruah proposed a polynomial-time approximation scheme for the known exact schedulability test [3], [4]. Further, Bini and Baruah presented an efficient technique for estimating the worst-case response time of fixed-priority tasks in polynomial time [7].

As previously discussed, the exact tests as well as the known approximation techniques require tasks to be sorted according to priorities. If a new task is added to the system, all already accepted lower-priority tasks will have to be retested. In this case, the running time of the an admission control test increases as the number of accepted tasks grows in the system.

III. Task Model, Definitions and Notation

For the reason that we are concerned with the admission control problem, our task set changes dynamically at runtime. A new task may arrive to and a running task may leave the system 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}. Since tasks may run under a partitioned scheme on identical processors, we use $\mathbf{T}^l$ to denote any arbitrary subset of l tasks from $\mathbf{T}$ running on a given processor.

Once a task is accepted, it generates jobs sporadically as long as it remains active in the system. In our previous example consisting of a networked computer game, an arriving task stands for a connection request to the game server, which if admitted produces a sequence of sporadic packets or jobs. We further assume that tasks are independent and run fully preemptively under DM.

Each task T_i in $\mathbf{T}^l$ 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 separation between two consecutive activations/jobs of T_i. As already 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. The ratio $u_i = \frac{e_i}{p_i}$ is known as task utilization and the sum of all u_i is the processor utilization $U = \sum_{i=1}^{l} \frac{e_i}{p_i}$.

If the worst-case response time of a task $T_j \in \mathbf{T}^l$ is less than its deadline d_j, then T_j is schedulable under DM. This can be computed in the following manner [4]:

$$t^{(c+1)} = e_j + \sum_{T_i \in HP(j)} \left\lceil \frac{t^{(c)}}{p_i} \right\rceil e_i, \tag{1}$$

where $HP(j) \subset \mathbf{T}^l$ denotes the subset of tasks with higher priority than T_j (i.e., for every T_i in $HP(j)$, $d_i \leq d_j$ must hold under the DM policy). Thus, $\sum_{T_i} \lceil \frac{t^{(c)}}{p_i} \rceil e_i$ where $T_i \in HP(j)$ results in the execution demand of higher priority jobs.

Eq. (1) can be solved iteratively starting from $t^{(1)} = e_j$ and until $t^{(c+1)} = t^{(c)}$ is satisfied for some $c \geq 1$. For a processor utilization $U \leq 1$, $t^{(c+1)}$ converges to the worst-case response time of T_j which we denote by w_j. Clearly, if $w_j > d_j$ holds, T_j misses its deadline. As a result, the iterative computation of $t^{(c+1)}$ in Eq. (1) can be stopped if $t^{(c+1)}$ exceeds d_j.

Eq. (1) assumes the critical instant [8], i.e., that jobs of all tasks in $HP(j)$ are released together with T_j. In this paper, without loss of generality, the simultaneous release of jobs of all tasks is assumed to happen at time $t = 0$.

As stated above, the concept of loading factor was originally defined for EDF [10] as the ratio of the maximum execution demand in a given time interval divided by the length of this interval. In this paper, we adapt this concept to the case of the DM policy, for which we introduce the following definition.

Definition 1 *The loading factor of a task T_j scheduled under the Deadline Monotonic policy is given by the ratio between the worst-case response time divided by the deadline of T_j and will be denoted by: $\rho_j = \frac{w_j}{d_j}$.*

Clearly, T_j's worst-case response time w_j and, hence, its loading factor ρ_j depend on the execution demand of tasks with higher priority than T_j. However, for the sake of simplicity, the notation adopted in this paper (i.e., w_j and ρ_j) does not reflect this dependency.

IV. Constant-Time Tests for DM

The utilization bound of Liu and Layland [8] can be adapted to DM as follows:

$$\sum_{i=1}^{n} \frac{e_i}{d_i} \leq n(2^{1/n} - 1). \tag{2}$$

Note that here periods p_i have been replaced by the respective deadlines d_i to reflect the DM rather than the RM policy. The validity of Eq. (2) for DM follows from the validity of Liu and Layland's bound for RM and the fact that $d_i \leq p_i$ holds for all tasks.

Similarly, the hyperbolic bound [11], [12], [13] may be modified to (with p_i being replaced by d_i):

$$\prod_{i=1}^{n} \left(1 + \frac{e_i}{d_i}\right) \leq 2. \tag{3}$$

As we consider the case where tasks do not have harmonic periods, the test of Kuo and Mok [15] reduces to the hyperbolic bound. Further, although the utilization bounds of Burchard et al. [16] and of Lehoczky [17] can also be used in the above manner, our experiments with a large number of synthetic task sets show that the test given by Eq. (3) is the least pessimistic. That is, while all the mentioned tests are *safe*, Eq. (3) accepts more schedulable task sets.

All these tests can be used to perform a constant-time admission control for DM tasks, i.e., the testing time for a new task does not depend on the number of currently running tasks. In addition, the more recent load test can also be used to derive a constant-time admission control for DM [9]:

$$\sum_{i=1}^{n} \max\left(\frac{e_i}{d_i}, \frac{2e_i}{p_i + e_i}\right) \leq 1. \tag{4}$$

It can further be proven that the load test is less pessimistic than the adapted hyperbolic bound of Eq. (3) in the case that $d_i \leq \frac{p_i + e_i}{2}$ holds for all tasks [9]. Additionally, considering a uniform distribution of deadlines in $[e_i, p_i]$, our experimental results show that the load test yields a better accuracy than any other known constant-time test for admission control, particularly, as the number of accepted tasks increases in the system. However, the accuracy of the load test is rather poor when compared with the exact test.

V. Theoretical Background

The test proposed in this paper computes an upper bound on the loading factor of the task set $\mathbf{T}^l$ composed of all already running tasks and T_{new} on a given processor. By definition, if the maximum loading factor of $\mathbf{T}^l$ does not exceed 1, $\mathbf{T}^l$ is schedulable, i.e., adding T_{new} does not produce any deadline miss. As a result, T_{new} can be assigned to the considered processor.

The first lemma gives an upper bound on the loading factor of any task T_j that belongs to $\mathbf{T}^l$. We then apply this lemma to obtain an upper bound on the loading factor of the whole task set $\mathbf{T}^l$ as discussed later in this section.

Lemma 1 *Let $\mathbf{T}^l$ be a subset of $\mathbf{T}$. If any task $T_j \in \mathbf{T}^l$ is schedulable, its loading factor ρ_j is upper bounded as follows:*

$$\rho_j \leq \frac{e_j}{d_j} + \sum_{T_i \in HP(j)} \max\left(\frac{e_i}{d_i}, \frac{2e_i}{p_i + e_i}\right),$$

where $HP(j) \subset \mathbf{T}^l$ is the set of tasks with higher priority than T_j.

Proof: The following equation holds for w_j:

$$w_j = e_j + \sum_{T_i \in HP(j)} \left\lceil \frac{w_j}{p_i} \right\rceil e_i. \tag{5}$$

This results by replacing $t^{(c+1)}$ and $t^{(c)}$ by w_j in Eq. (1). Each term $\lceil \frac{w_j}{p_i} \rceil e_i$ stands for the execution demand within w_j of a higher-priority T_i. Since T_j's loading factor ρ_j is given by $\frac{w_j}{d_j}$, we have:

$$\rho_j = \frac{e_j}{d_j} + \sum_{T_i \in HP(j)} \frac{\left\lceil \frac{w_j}{p_i} \right\rceil e_i}{d_j}. \tag{6}$$

For $w_j \leq p_i$, $\frac{\lceil \frac{w_j}{p_i} \rceil e_i}{d_j} = \frac{e_i}{d_j}$ holds. As T_i has higher priority than T_j, $d_i \leq d_j$ also holds. As a result, $\frac{e_i}{d_j} \leq \frac{e_i}{d_i}$ is true.

On the other hand, if $\frac{\lceil \frac{w_j}{p_i} \rceil e_i}{d_j} = \frac{(k_i+1)e_i}{d_j}$ holds, w_j is strictly larger than $k_i p_i$ where $k_i \geq 1$ is an integer number. Apart from the k_i jobs released within $k_i p_i$ time units, one extra job of T_i interferes with T_j and, hence, $w_j \geq k_i p_i + e_i$ must hold since T_j cannot interrupt any higher-priority task. As $T_j \in \mathbf{T}^l$ is schedulable, $w_j \leq d_j$ is true and we have: $\frac{\lceil \frac{w_j}{p_i} \rceil e_i}{d_j} = \frac{(k_i+1)e_i}{d_j} \leq \frac{(k_i+1)e_i}{w_j} \leq \frac{(k_i+1)e_i}{k_i p_i + e_i}$. Now, we prove that $\frac{(k_i+1)e_i}{k_i p_i + e_i}$ is less than or equal to $\frac{2e_i}{p_i+e_i}$:

$$\begin{aligned} \frac{(k_i+1)e_i}{k_i p_i + e_i} &\leq \frac{2e_i}{p_i + e_i}, \\ k_i + 1 &\leq 2\left(\frac{(k_i-1)p_i}{p_i + e_i} + 1\right). \end{aligned} \tag{7}$$

Since $e_i \leq p_i$, $p_i + e_i \leq 2p_i$ holds. Hence, $2\left(\frac{(k_i-1)p_i}{2p_i} + 1\right) \leq 2\left(\frac{(k_i-1)p_i}{p_i+e_i} + 1\right)$ also holds. As $2\left(\frac{(k_i-1)p_i}{2p_i} + 1\right) = k_i + 1$, Eq. (7) is true. Hence, $\frac{\lceil \frac{w_j}{p_i} \rceil e_i}{d_j} \leq \frac{(k_i+1)e_i}{k_i p_i + e_i} \leq \frac{2e_i}{p_i+e_i}$ holds. We can see that, independently of the value of w_j, $\frac{\lceil \frac{w_j}{p_i} \rceil e_i}{d_j} \leq \max\left(\frac{e_i}{d_i}, \frac{2e_i}{p_i+e_i}\right)$ holds always. Finally, ρ_j in Eq. (6) is upper bounded by $\frac{e_j}{d_j} + \sum_{T_i} \max\left(\frac{e_i}{d_i}, \frac{2e_i}{p_i+e_i}\right)$ where T_i belongs to $HP(j)$. ∎

The following lemma proves that the bound given by Lemma 1 is *safe*. That is, if T_j is not schedulable, the bound of Lemma 1 is always greater than 1.

Lemma 2 *Let $\mathbf{T}^l$ be a subset of $\mathbf{T}$. If any task $T_j \in \mathbf{T}^l$ is not schedulable, the following must hold:*

$$\frac{e_j}{d_j} + \sum_{T_i \in HP(j)} \max\left(\frac{e_i}{d_i}, \frac{2e_i}{p_i + e_i}\right) > 1,$$

where $HP(j) \subset \mathbf{T}^l$ is the set of tasks with higher priority than T_j.

Proof: Since T_j is not schedulable, the first job of T_j misses its deadline at the critical instant. Let us denote by $\hat{w}_j$ the execution demand in $[0, d_j]$. Hence, if T_j is not schedulable, $\hat{w}_j$ is greater than d_j. Dividing $\hat{w}_j$ by d_j, we obtain:

$$\frac{\hat{w}_j}{d_j} = \frac{e_j}{d_j} + \sum_{T_i \in HP(j)} \frac{\lceil \frac{d_j}{p_i} \rceil e_i}{d_j} > 1. \tag{8}$$

For $d_j \leq p_i$, $\frac{\lceil \frac{d_j}{p_i} \rceil e_i}{d_j} = \frac{e_i}{d_j}$ holds. The condition $\frac{e_i}{d_j} \leq \frac{e_i}{d_i}$ also holds, since d_i is less than or equal to d_j under DM.

Further, if $d_j > p_i$, $\frac{\lceil \frac{d_j}{p_i} \rceil e_i}{d_j} = \frac{(k_i+1)e_i}{d_j}$ holds for an integer number $k_i \geq 1$. If $d_j \geq k_i p_i + e_i$ also holds, then from Eq. (8):

$$\frac{e_j}{d_j} + \sum_{T_i \in HP(j)} \frac{(k_i+1)e_i}{k_i p_i + e_i} \geq \frac{e_j}{d_j} + \sum_{T_i \in HP(j)} \frac{\lceil \frac{d_j}{p_i} \rceil e_i}{d_j} > 1.$$

We know from the proof of Lemma 1 that $\frac{(k_i+1)e_i}{k_i p_i + e_i} \leq \frac{2e_i}{p_i+e_i}$ holds and, hence, the lemma holds true in this case.

Now, we analyze the case where $d_j > p_i$ and $d_j < k_i p_i + e_i$ as shown in Figure 1. T_j is here the task with the lowest priority, so it can only run when no higher-priority task executes. This means that T_j finishes executing at $\hat{w}_j$ as depicted in Figure 1. Hence, $\hat{w}_j > k_i p_i + e_i$ holds and therefore $\frac{\hat{w}_j}{k_i p_i + e_i} > 1$ also holds:

$$\frac{\hat{w}_j}{k_i p_i + e_i} = \frac{e_j}{k_i p_i + e_i} + \sum_{T_i \in HP(j)} \frac{\lceil \frac{d_j}{p_i} \rceil e_i}{k_i p_i + e_i} > 1.$$

Since $d_j < k_i p_i + e_i$, $\frac{e_j}{d_j} > \frac{e_j}{k_i p_i + e_i}$ and we have: $\frac{e_j}{d_j} + \sum_{T_i} \frac{(k+1)e_i}{k_i p_i + e_i} > \frac{\hat{w}_j}{k_i p_i + e_i} > 1$ where $T_i \in HP(j)$. Again, from the proof of Lemma 1, we know that $\frac{(k_i+1)e_i}{k_i p_i + e_i} \leq \frac{2e_i}{p_i+e_i}$ holds. As a consequence, $\frac{e_j}{d_j} + \sum_{T_i} \max\left(\frac{e_i}{d_i}, \frac{2e_i}{p_i+e_i}\right) > 1$ holds always true and the lemma follows. ∎

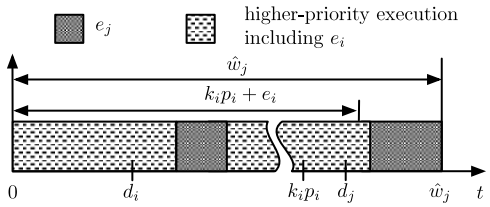

Figure 1. T_j finishes executing at $\hat{w}_j$

LEMMA 3 *Let $\mathbf{T}^l$ be a subset of $\mathbf{T}$. For any two tasks T_i and T_j that belong to $\mathbf{T}^l$, where T_i has higher priority than T_j, it always holds that* $\frac{\lceil \frac{d_j}{p_i} \rceil e_i}{d_j} \leq \max\left(\frac{e_i}{d_i}, \frac{2e_i}{p_i+e_i}\right)$.

Proof: This lemma follows immediately from the above discussion of Lemma 1 and Lemma 2. ■

Lemma 1 gives an upper bound on the loading factor ρ_j of a task T_j. Based on Lemma 1, it is possible to design an admission control test which is then equivalent to the *load test* of Eq. (4) [9]. However, as discussed before, this is too pessimistic compared to the exact test [3], [4].

In what follows, we consider a more accurate way of obtaining an upper bound on ρ_j. If T_j is schedulable $\rho_j = \frac{w_j}{d_j} \leq 1$ and, hence, $w_j \leq d_j$ holds. As a result, we have that $\rho_j \leq \frac{e_j}{d_j} + \sum_{T_i} \frac{\lceil \frac{d_j}{p_i} \rceil e_i}{d_j}$ also holds, where $T_i \in HP(j)$ and $HP(j) \subset \mathbf{T}^l$ is the set of tasks with higher priority than T_j. Further, we can apply Lemma 3 for one or more $T_i \in HP(j)$ and obtain an upper bound on ρ_j which we denote by $\hat{\rho}_j$. Clearly, for any $\hat{\rho}_j$, the following holds:

$$\frac{e_j}{d_j} + \sum_{T_i \in HP(j)} \frac{\left\lceil \frac{d_j}{p_i} \right\rceil e_i}{d_j} \leq \hat{\rho}_j \leq \frac{e_j}{d_j} + \sum_{T_i \in HP(j)} \max\left(\frac{e_i}{d_i}, \frac{2e_i}{p_i + e_i}\right). \quad (9)$$

If T_j is not schedulable, $\rho_j \geq \frac{e_j}{d_j} + \sum_{T_i} \frac{\left\lceil \frac{d_j}{p_i} \right\rceil e_i}{d_j} > 1$ holds. Hence, according to Eq. (9), $\hat{\rho}_j$ is also greater than 1, i.e., $\hat{\rho}_j$ is a *safe* bound.

Now, we are in the position of defining $\hat{\rho}^l(t_x)$ as the maximum $\hat{\rho}_j$ among all $T_j \in \mathbf{T}^l$ for which $d_j \in [t_x, t_{x+1})$:

$$\hat{\rho}^l(t_x) = \max_{T_j \in \mathbf{T}^l | t_x \leq d_j < t_{x+1}} (\hat{\rho}_j).$$

Here t_x and t_{x+1} are given points in time, where $t_x < t_{x+1}$ holds and x is a positive integer number.

The concept of $\hat{\rho}^l(t_x)$ gives an upper bound on the loading factor within $[t_x, t_{x+1})$. Hence, computing $\hat{\rho}^l(t_x)$ for consecutive intervals along the time line allows approximating the loading factor by pieces. We further denote by $\hat{\rho}^l$ the maximum among all piecewise approximations $\hat{\rho}^l(t_x)$ for $0 < t_x < \infty$, i.e., the maximum loading factor among all $T_j \in \mathbf{T}^l$:

$$\hat{\rho}^l = \max_{0 \leq t_x < \infty} \left(\hat{\rho}^l(t_x)\right).$$

Computing the maximum loading factor $\hat{\rho}^l$ by pieces $\hat{\rho}^l(t_x)$ increases the accuracy of the proposed approximation. The following two lemmas are concerned with the successive computation of $\hat{\rho}^l(t_x)$ as a new task arrives and constitute the basis of the proposed admission control test in this paper.

LEMMA 4 *For $\mathbf{T}^{l-1} \subset \mathbf{T}$, let $\hat{\rho}^{l-1}(t_x)$ be the maximum $\hat{\rho}_j$ (defined in Eq. (9)) among all T_j in $\mathbf{T}^{l-1}$, for which d_j is in the interval $[t_x, t_{x+1})$. If any task T_{new} is added to $\mathbf{T}^{l-1}$, where $t_x \leq d_{new} < t_{x+1}$, the loading factor in $[t_x, t_{x+1})$ of the resulting subset $\mathbf{T}^l$ is bounded above by:*

$$\hat{\rho}^l(t_x) = \hat{\rho}^{l-1}(t_x) + \max\left(\frac{e_{new}}{d_{new}}, \frac{2e_{new}}{p_{new} + e_{new}}\right).$$

Proof: Clearly, adding T_{new} with $t_x \leq d_{new} < t_{x+1}$ to $\mathbf{T}^{l-1}$ changes the loading factor of the system. Notice that, since the loading factor of any DM task is given by its worst-case response time over its deadline, T_{new} does not affect the loading factor of higher-priority tasks, i.e., tasks with shorter deadlines. As a consequence, T_{new} can only increase the loading factor for $t \geq d_{new}$. Further, let us assume that the resulting maximum loading factor within $[t_x, t_{x+1})$ occurs for a T_j with deadline d_j. Again, d_j is any possible deadline in $[d_{new}, t_{x+1})$. Denoting by w_j^{new} T_j's worst-case response time after adding T_{new} to the system, the following equation holds true: $w_j^{new} = e_j + \sum_{T_i} \lceil \frac{w_j^{new}}{p_i} \rceil e_i + \lceil \frac{w_j^{new}}{p_{new}} \rceil e_{new}$. Here, $T_i \in HP(j)$ and $HP(j)$ stands for all tasks in $\mathbf{T}^{l-1}$ with higher priority than T_j. Considering that T_j is schedulable (i.e., $w_j^{new} \leq d_j$), T_j's resulting loading factor ρ_j^{new} is bounded above: $\rho_j^{new} \leq \frac{e_j}{d_j} + \sum_{T_i} \frac{\lceil \frac{d_j}{p_i} \rceil e_i}{d_j} + \frac{\lceil \frac{w_j^{new}}{p_{new}} \rceil e_{new}}{d_j}$.

By definition, we can replace the first two terms of this inequality by $\hat{\rho}^{l-1}(t_x)$. Applying then Lemma 1, we obtain $\rho_j^{new} \leq \hat{\rho}^{l-1}(t_x) + \max\left(\frac{e_{new}}{d_{new}}, \frac{2e_{new}}{p_{new}+e_{new}}\right)$ and the lemma follows.

If T_j is not schedulable, notice that $\rho_j^{new} \geq \frac{e_j}{d_j} + \sum_{T_i} \frac{\lceil \frac{d_j}{p_i} \rceil e_i}{d_j} + \frac{\lceil \frac{w_j^{new}}{p_{new}} \rceil e_{new}}{d_j} > 1$ holds, so the bound $\hat{\rho}^{l-1}(t_x)$ given in this lemma is safe. ■

LEMMA 5 *For $\mathbf{T}^{l-1} \subset \mathbf{T}$, let $\hat{\rho}^{l-1}(t_x)$ be the maximum $\hat{\rho}_j$ (defined in Eq. (9)) among all T_j in $\mathbf{T}^{l-1}$, for which d_j is in the 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:*

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

where k is given by $\lceil \frac{t_x}{p_{new}} \rceil$ and $t_k = kp_{new}$.

Proof: Adding T_{new} to the system modifies the loading factor for $t \geq d_{new}$, since T_{new} has influence on the worst-case response times of lower-priority tasks, i.e., tasks with longer deadlines. Now, let us assume that the resulting maximum loading factor within $[t_x, t_{x+1})$, occurs for a T_j with d_j, where d_j is any possible deadline in $[t_x, t_{x+1})$. Denoting by ρ_j^{new} T_j's loading factor after adding T_{new}, the following equation holds true: $\rho_j^{new} = \frac{e_j}{d_j} + \sum_{T_i} \frac{\lceil \frac{w_j^{new}}{p_i} \rceil e_i}{d_j} + \frac{\lceil \frac{w_j^{new}}{p_{new}} \rceil e_{new}}{d_j}$ where

$T_i \in HP(j)$. Considering that T_j is schedulable (i.e., $w_j^{new} \leq d_j$), T_j's resulting loading factor ρ_j^{new} is bounded above:

$$\rho_j^{new} \leq \hat{\rho}^{l-1}(t_x) + \frac{\left\lceil \frac{d_j}{p_{new}} \right\rceil e_{new}}{d_j}. \quad (10)$$

As we are interested in finding $\hat{\rho}^l(t_x)$, i.e., an upper bound on the maximum loading factor in $[t_x, t_{x+1})$, we first consider the case where $d_j = t_x$ holds: $\rho_j^{new} \leq \hat{\rho}^{l-1}(t_x) + \frac{\lceil \frac{t_x}{p_{new}} \rceil e_{new}}{t_x}$, where $\lceil \frac{t_x}{p_{new}} \rceil$ was denoted by k in this lemma. Removing the ceiling function in Eq. (10) and reordering, we obtain:

$$\rho_j^{new} \leq \hat{\rho}^{l-1}(t_x) + \frac{e_{new}}{d_j} + \frac{e_{new}}{p_{new}}. \quad (11)$$

From Eq. (11), we know that the smallest possible d_j results in an upper bound on the loading factor. However, we do not need to compute Eq. (11) until $t_k = kp_{new}$ for $k = \lceil \frac{t_x}{p_{new}} \rceil$. This is because the loading factor for jobs in (t_x, t_k) cannot exceed the one at t_x. Replacing d_j by t_k in (11), we proceed to obtain: $\rho_j^{new} \leq \hat{\rho}^{l-1}(t_x) + \frac{e_{new} + t_k \frac{e_{new}}{p_{new}}}{t_k}$. Taking into account that t_k is equal to kp_{new}, we finally obtain: $\rho_j^{new} \leq \hat{\rho}^{l-1}(t_x) + \frac{(k+1)e_{new}}{kp_{new}}$.

Thus, $\hat{\rho}^l(t_x)$ (i.e., the resulting upper bound on the maximum loading factor in $[t_x, t_{x+1})$) is given by $\hat{\rho}^{l-1}(t_x) + \max\left(\frac{ke_{new}}{t_x}, \frac{(k+1)e_{new}}{t_k}\right)$.

On the other hand, if T_j is not schedulable after adding T_{new} (i.e., $w_j^{new} > d_j$), $e_j + \sum_{T_i} \lceil \frac{d_j}{p_i} \rceil e_i + \lceil \frac{d_j}{p_{new}} \rceil e_{new} > d_j$ holds where $T_i \in HP(j)$. As a result, the bound $\hat{\rho}^{l-1}(t_x)$ given in this lemma is safe. ■

VI. The Proposed Test

In this section, we introduce a test based on lemmas 4 and 5. 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 tasks and T_{new}. Now, if the maximum loading factor does not exceed 1, the new task T_{new} can be feasibly accommodated on the considered processor.

To calculate the maximum loading factor on a given processor, our algorithm partitions the time line into $b+1$ non-overlapping intervals where b is a positive integer number. This way, we can approximate the execution demand, the worst-case response time and, hence, the loading factor of tasks in $\mathbf{T}^l$. Clearly, the quality of the approximation depends on the number of intervals/segments we use. However, more segments result in a longer running time of the algorithm.

Figure 2 illustrates the approximation technique for uniform intervals and $b = 2$, i.e., the time axis is divided into three; t_1 and t_2 are the lower bounds of the second and third intervals respectively.

Accommodating an arriving task into one of these intervals according to its deadline d_{new} allows for more accuracy in computing the maximum loading factor generated by $\mathbf{T}^l$ on the processor. This procedure can be interpreted as restricted sorting of tasks according to deadlines such that we can retain constant complexity. The execution demand used for computing the worst-case response time is then approximated by $b+1$ slopes. These slopes are given by the maximum loading factors in each of the intervals. In Figure 2, we have three slopes: $\hat{\rho}^l(0)$ which is the maximum loading factor in $[0, t_1)$, $\hat{\rho}^l(t_1)$ in $[t_1, t_2)$ and $\hat{\rho}^l(t_2)$ in the interval $[t_2, \infty)$. The maximum between $\hat{\rho}^l(0)$, $\hat{\rho}^l(t_1)$ and $\hat{\rho}^l(t_2)$ gives an upper bound on the loading factor of all tasks executing on the processor.

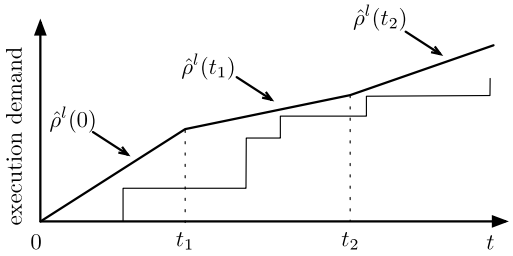

Figure 2. Approximation technique for the maximum loading factor

```
    compute t_len; //see explanation below

    if d_new ≥ t_b
     ρ̂^l(b+1) = ρ̂^l(b+1) + max(e_new/d_new, 2e_new/(p_new+e_new));
5:  else
     t_x = t_b;
     for i = 0 to b
      if t_x ≤ d_new
       ρ̂^l(b−i+1) = ρ̂^l(b−i+1)+max(e_new/d_new, 2e_new/(p_new+e_new));
10:    break;
      else
       k = ⌈t_x/p_new⌉;
       t_k = kp_new;
       ρ̂^l(b−i+1) = ρ̂^l(b−i+1)+max(ke_new/t_x, (k+1)e_new/t_k);
15:   end
      compute t_x; //see explanation below
     end
    end

20: if max_{1≤i≤b+1}(ρ̂^l(i)) > 1
     return("not schedulable");
    else
     return("schedulable");
    end
```
Figure 3. Admission control based on Lemma 4 and 5

Figure 3 shows the pseudo-code of the proposed test for admission control. The parameters b and t_b can be tuned to achieve a desired accuracy/running time. Here, b determines the number of intervals/segments used for the approximation and t_b is the lower bound of the last interval. We discuss how to select these parameters later.

Non-uniform approximations: The distribution of approximation segments along the time line can be adjusted to a particular application. In practice, it is not unusual to know a probability distribution according to which we can partition

the time line more efficiently. For example, if deadlines were normally distributed, almost 70% of all deadlines will be greater than $d_{mean}-\sigma$ and less than $d_{mean}+\sigma$. Here, d_{mean} is the mean and σ is the standard deviation of the normal distribution. As a result, using a constant number of approximation segments b, it is more meaningful to concentrate more of these segments within $[d_{mean}-\sigma, d_{mean}+\sigma]$ than distributing them uniformly along the time line.

In this paper, deadlines are considered to be uniformly distributed in $[0, d_{max}]$ with d_{max} being the maximum possible deadline. Further, we study two variants of the algorithm of Figure 3. The first variant consists in distributing intervals uniformly over the time line such that the first b intervals starting at $t = 0$ have the same length t_{len} and the last interval covers the remaining time axis up to infinity. In our second variant, intervals have non-uniform lengths. The first interval starts at $t = 0$ and is t_{len} time units long. The second interval here is $2t_{len}$ long, the third $3t_{len}$ and so on until the b-th interval which has a length of $b \times t_{len}$ time units. Here again, the last interval extends to infinity.

Whereas the use of uniform-length intervals is more intuitive and based on the fact that deadlines are assumed to have a uniform distribution, non-uniform-length intervals lead to a more accurate approximation as shown later. Using non-uniform-length intervals as suggested above allows us to increasingly concentrate more segments for approximating tasks with shorter deadlines. Since these tasks have higher priority under DM, they have bigger impact on the total execution demand of the whole task set and dominate the worst-case response times of tasks with larger deadlines. As a result, approximating the execution demand of tasks with shorter deadlines using more segments increases the accuracy of the proposed algorithm.

Computing t_{len}: The value of t_{len} computed at line 1 of Figure 3 clearly depends on the algorithm's variant. In the first variant, t_{len} is the length of the approximation intervals (which are here uniform). This variant uses b intervals of length t_{len}, so t_{len} is given by $\frac{t_b}{b}$.

The second variant uses intervals with non-uniform (i.e., increasing) lengths. Here, t_{len} is the length of the first interval starting at $t = 0$. As mentioned above, the second interval is twice as long as the first one, the third interval has three times the length of the first interval and so on. In this case, t_{len} can be easily obtained by $\frac{t_b}{c}$ with $c = \frac{b^2+b}{2}$.

Now, for an arriving T_{new}, if $d_{new} \geq t_b$ holds, we can apply Lemma 4 at line 4 to compute the maximum loading factor $\hat{\rho}^l(b+1) = \hat{\rho}^l(t_b)$, 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 all other intervals whose bounds are 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. For this purpose, we iterate over all intervals from line 7 to 17. The variable t_x adopts the value of the lower bounds of intervals starting from $t_x = t_b$ (i.e., from the last interval) until the condition $t_x \leq d_{new}$ is fulfilled. T_{new} is going to change the loading factor in all intervals for which $t_x > d_{new}$ holds. Lemma 5 can be used to compute the maximum loading factor in these intervals—see Figure 3 lines 11 to 15. The values k and t_k required by Lemma 5 are computed at lines 12 and 13 respectively, whereas $\mathbf{T}^l$'s maximum loading factor in the $(b-i+1)$-th interval is obtained at line 14 where i is the variable of the for-loop. In order to make the pseudo-code more intuitive, we use a slightly different nomenclature in Figure 3. Namely, $\hat{\rho}^l(t_x)$ of Lemma 5 is given by $\hat{\rho}^l(b-i+1)$, i.e., we use the index $b-i+1$ instead of the lower bound t_x to denote the loading factor of the $(b-i+1)$-th interval. Once the condition $t_x \leq d_{new}$ is satisfied, we apply Lemma 4 at line 9 to estimate the maximum loading factor in the corresponding interval and exit the loop at line 10.

Computing t_x: After each iteration, the next value of t_x is computed at line 16. In case of uniform intervals, this is $t_x = t_x - t_{len}$ and, for non-uniform-length intervals, this is $t_x = t_x - (b-i) \times t_{len}$ where $0 \leq i \leq b$ is the variable of the loop.

If the maximum loading factor in all the time intervals is not greater than 1 at line 20, $\mathbf{T}^l$ is schedulable on the processor and the new task T_{new} can be accepted (otherwise, this algorithm rejects T_{new}). Clearly, if T_{new} is rejected, its loading factor should not be added to the system's loading factor (this can be achieved using temporal variables to compute the loading factor with T_{new}, which values are only then committed if T_{new} is accepted).

A task leaving the system: The system's loading factor also changes when a task T_{old} leaves. In this case, the algorithm of Figure 3 needs little modification. We need again to compute the *loading factor component* of the leaving task: $\max\left(\frac{e_{old}}{d_{old}}, \frac{2e_{old}}{p_{old}+e_{old}}\right)$ or $\max\left(\frac{ke_{old}}{t_x}, \frac{(k+1)e_{old}}{t_k}\right)$ depending on the interval in which d_{old} is contained. Finally, instead of summing this as for T_{new} in Figure 3 (lines 4, 9 and 14), T_{old}'s loading factor component has to be subtracted from the maximum loading factor in the corresponding interval.

Complexity of the algorithm: The proposed admission control test has a constant complexity. That is, its running time for accepting/rejecting a new task does not depend on the number of tasks currently executing in the system. However, the running time of this algorithm increases linearly with the number of segments $b+1$ used to approximate the loading factor. As a result, the complexity of the proposed algorithm is given by $\mathcal{O}(b+1)$. As discussed above, b is a constant that can be adjusted to achieve a desired accuracy/running time ratio.

VII. Experimental Results

In this section, we present experimental results comparing the proposed admission control algorithm with approaches from the literature. The accuracy/acceptance ratio of tests is measured using 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 in a running system, the

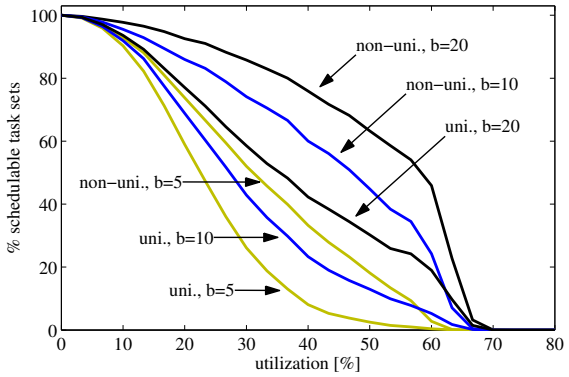

Figure 4. Schedulability versus utilization for 100 tasks: b

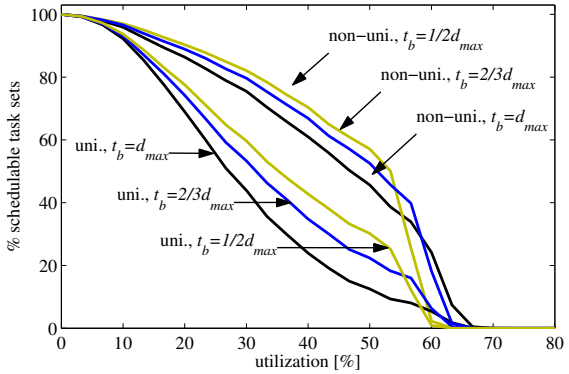

Figure 5. Schedulability versus utilization for 100 tasks: t_b

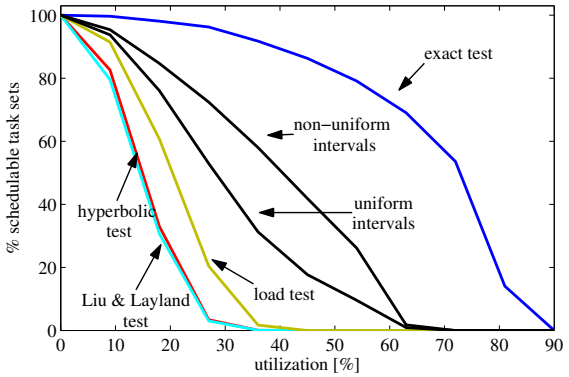

Figure 6. Schedulability vs. utilization for 50 tasks

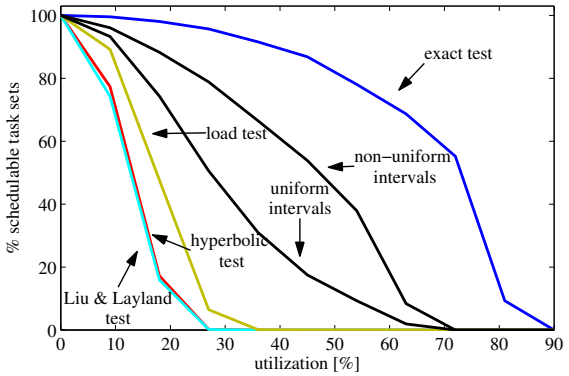

Figure 7. Schedulability vs. utilization for 100 tasks

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 l tasks is schedulable, it means that adding an l-th task to the set of $l-1$ tasks was possible.

Selecting b and t_b: The proposed test can be configured by means of b and t_b where $b+1$ is the number of intervals used for computing the maximum loading factor and t_b is the lower bound of the last interval. We analyze the previously considered two variants of this algorithm. The first variant has uniform intervals in $[0, t_b]$ while the second variant has non-uniform intervals whose lengths increase as we move from $t = 0$ towards t_b. Clearly, the values of b and t_b have influence on both these variants. Let us consider Figure 4 illustrating the performance of the two variants for sets of 100 tasks and different values of b.

As shown in Figure 4, a bigger value of b, i.e., considering more linear segments for the approximation, increases the accuracy/acceptance ratio of the algorithm. Additionally, for the same b, the variant featuring non-uniform-length intervals (denoted by *non-uni.*) outperforms the variant with uniform intervals (denoted by *uni.*). For comparing our test with those from the literature, we configure both variants with $b = \lfloor 0.1l \rfloor$ being l the number of tasks expected to be active at the same time in the system (e.g., if $l = 100$ as in Figure 4, we choose $b = 10$). This value of b results in an acceptable accuracy for the testing and, at the same time, it does not considerably increase the running time of the algorithm.

Figure 5 shows the influence of t_b on the accuracy of the proposed test. Again, the two variants are taken into account. Clearly, since t_b is the lower bound of the last interval, this should never be greater than d_{max}, where d_{max} is the longest possible deadline in the system. Assuming a constant b, if we select t_b to be smaller than d_{max}, we can concentrate more segments in the lower part of the time line, which increases the accuracy for approximating higher-priority tasks. On the other hand, as we do not know the exact values of the deadlines, we cannot be sure that concentrating more intervals in this region leads to the desired accuracy. Further, the smaller t_b, the more deadlines are going to fit in the last interval $[t_b, \infty)$, which again decreases the accuracy of the approximation. In general, considering synthetic task sets with a uniform distribution, a small t_b improves the accuracy for lower utilization ranges at the cost of a worse acceptance ratio for higher utilizations—see Figure 5. For our comparison, we choose $t_b = d_{max}$ for both variants since it allows a better accuracy at higher utilizations.

A. Synthetic Task Sets

We compare the proposed test with other constant-time tests based on approaches from the literature such as the hyperbolic bound of Eq. (3) and the Liu and Layland test of Eq. (2). Our experimental results with a large number of synthetic task sets have shown that the hyperbolic bound of Eq. (3) is less pessimistic than the other approaches from the literature [15], [16], [17] which we then do not include in the figures here.

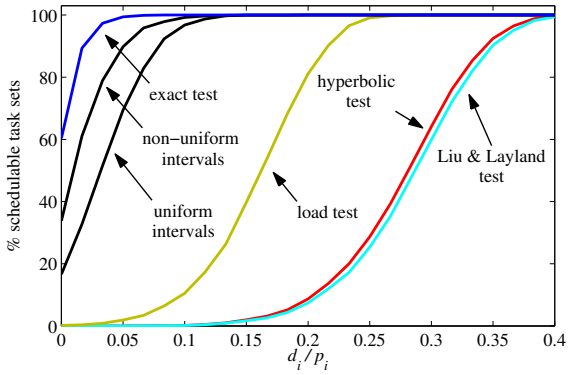

Figure 8. Schedulability vs. deadline/period ratio for 50 tasks

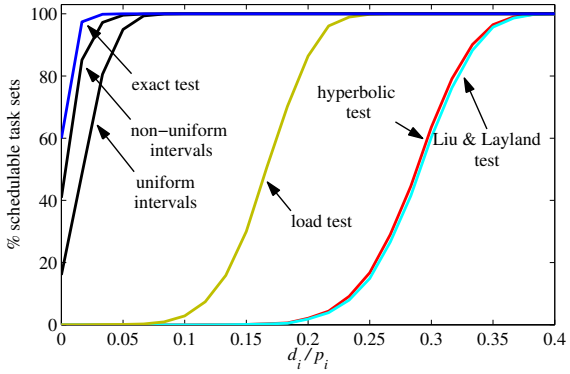

Figure 9. Schedulability vs. deadline/period ratio for 100 tasks

We also consider the more recent load test shown in Eq. (4) and the known exact test [3], [4] in this comparison. Note that the load test is equivalent to the proposed algorithm in this paper for which a $b = 0$ is selected (i.e., the case where only one approximation segment/interval is used). The exact test is the only one without constant complexity when used for admission control, however, it serves as a reference in this comparison.

Schedulability versus utilization: For schedulability curves with respect to the utilization, we first generated a random set of task utilizations u_i with *UUniFast* [19], [20]. Then, we obtained periods p_i using a uniform distribution in $[0, 1]$ and computed $e_i = u_i p_i$. The relative deadlines d_i were also generated using a uniform distribution in $[e_i, p_i]$. Additionally, we increased the utilization gradually in 24 uniform steps generating each time 10,000 different task sets.

Figure 6 shows the percentage of accepted task sets for the different algorithms and 50 tasks per set as the processor utilization grows. We use $b = 5$ and $t_b = d_{max}$ as stated above for the considered variants of the proposed algorithm. While the Liu and Layland, the hyperbolic and the load test become pessimistic for processor utilizations beyond 20%, the two variants of the proposed test have a better acceptance ratio for utilizations of up to 60%.

In the case of 100 tasks per set, we use $b = 10$ and $t_b = d_{max}$ just as explained before. Figure 7 illustrates the acceptance ratio of the different algorithms for 100 tasks per set. Again, the test of Liu and Layland, the hyperbolic and the load test are very pessimistic for processor utilizations over 20%. On the other hand, the two variants of the proposed test outperform the other constant-time algorithms for utilizations of up to 60%.

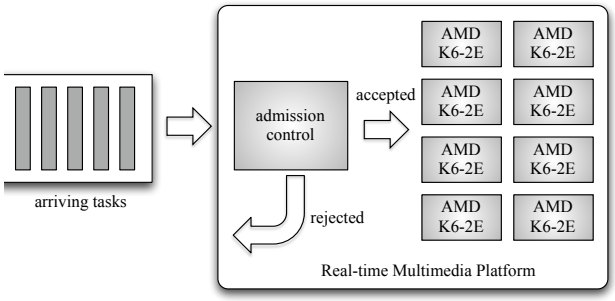

Figure 10. Setup for a real-time multimedia server

The propose algorithm with non-uniform intervals shows an accuracy that is closer to that of the exact test for utilizations in the range of $(0, 60\%)$. This test allows admitting around 60% more task sets compared with the other constant-time tests in the range 30% to 50% utilization (as shown in Figure 6 and Figure 7).

Schedulability versus $\frac{d_i}{p_i}$: For schedulability curves with respect to the deadline/period ratio $\frac{d_i}{p_i}$, we considered a processor utilization of 40% (i.e., $U = 0.4$) and used *UUniFast* [19], [20] to obtain a set of task utilizations u_i. Again, the periods were obtained with a uniform distribution in $[0, 1]$ and the execution times were computed just as before $e_i = u_i p_i$. The relative deadlines here are computed with the previously obtained value of p_i and according to the desired ratio $\frac{d_i}{p_i}$. The ratio $\frac{d_i}{p_i}$ was increased gradually in 24 uniform steps with each time 10,000 different task sets.

Figure 8 shows the percentage of accepted task sets for the algorithms and 50 tasks as $\frac{d_i}{p_i}$ grows from 0 to 0.4. Here, we use $b = 5$ and $t_b = d_{max}$ for the variants of the proposed algorithm. Figure 9 illustrates the acceptance ratio versus $\frac{d_i}{p_i}$ of the different algorithms for 100 tasks per set. For this, the variants of the proposed algorithm feature $b = 10$ and $t_b = d_{max}$.

In this comparison and for small values of $\frac{d_i}{p_i}$, the proposed algorithm shows (in both its variants) a similar performance to that of the exact test. The load test (i.e., the case $b = 0$) is pessimistic up to $\frac{d_i}{p_i} = 0.2$ and then it behaves similar to the exact test. On the other hand, the Liu and Layland and the hyperbolic test remain pessimistic for deadlines $d_i < 0.35 p_i$.

B. Case Study

In this section, we compare algorithms in the context of a case study. 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. The multimedia server in question features eight processors. Under normal operation, only four of the processors are in active mode, while the other four remain in

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 I
TASK POOL BASED ON E3S: PARAMETERS IN SECONDS

sleep mode and can be activated if necessary according to the computation demand. Our platform has identical AMD K6-2E processor operating at 400 Mhz. A partitioned DM scheduling is used, i.e., once a task is assigned to a processor, it remains on that processor (task migration is not allowed). Figure 10 illustrates our setup for the high-end multimedia server.

Here, the different algorithms are combined with the First Fit (FF) heuristic to achieve a feasible allocation of tasks to processors. This way, the Liu and Layland test, the hyperbolic and the load test are compared with the proposed admission control test. We consider the two variants of this algorithm with uniform and non-uniform intervals respectively. For both these variants, we set $b = 5$ and $t_b = d_{max}$ where d_{max} is the maximum deadline that we expect for an arriving task. As before, we include the known exact pseudo-polynomial test in this comparison. This latter is the only test with pseudo-polynomial complexity and acts here as a reference.

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 I 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 or reject it according to whether they can find a feasible allocation in one of the processors or not.

Figure 11 shows the number of tasks that the different admission control tests are able to accept when four processors are in active mode. The Liu and Layland and the hyperbolic test can accept around 10 incoming tasks after which they become pessimistic and start rejecting further requests. The load test allows accepting 10 more tasks than the previous tests. The variant of the proposed test consisting of uniform intervals can accept up to 45 before starting to reject almost all further requests. Our algorithm with non-uniform interval lengths admits approximately 55 tasks and is only 10 tasks below the acceptance curve of the exact test.

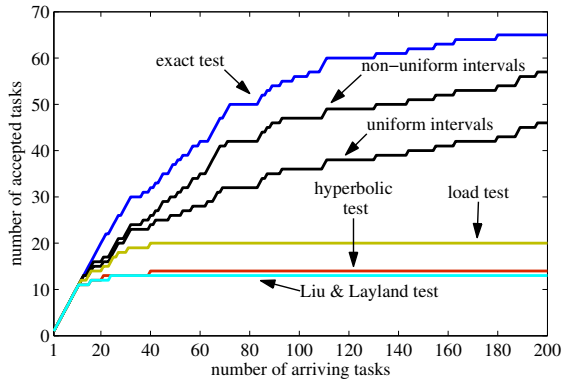

Figure 11. Accepted vs. arriving tasks: 4 processors

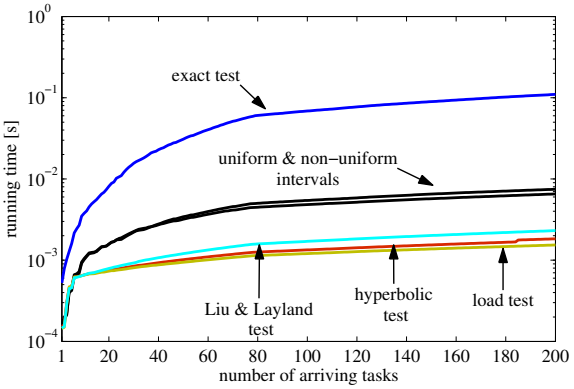

Figure 12. Running time vs. arriving tasks: 4 processors

Figure 12 shows the running times of algorithms as a function of the number of arriving tasks for the case of four active processors. The proposed test (for both uniform and non-uniform intervals) and the other constant-time tests require around 1 to 2 orders of magnitude less time than the solution based on the exact test. The proposed test is here approximately half an order of magnitude slower than the other constant-time tests.

The performance of the admission control algorithms is shown in Figure 13 for the case where eight processors are in active mode. Clearly, the number of accepted tasks increases because more computation capacity is available. However, the test of Liu and Layland and the hyperbolic bound accept only around 20 tasks. The load test accommodates 10 tasks more than these previous tests. Our proposed test with uniform intervals allows admitting around 80 tasks before getting pessimistic. On the other hand, the test with non-uniform intervals is able to accommodate approximately 100. This is around 20 requests less than the exact test, but, in contrast to the exact test, our test has constant complexity.

In Figure 14, the running times versus the number of arriving tasks are depicted for the case of eight processors in active mode. Again, the admission control based on the exact test still requires around 2 orders of magnitude additional running time than the constant-time algorithms, whereas the proposed test takes half an order of magnitude longer than the other constant-time tests.

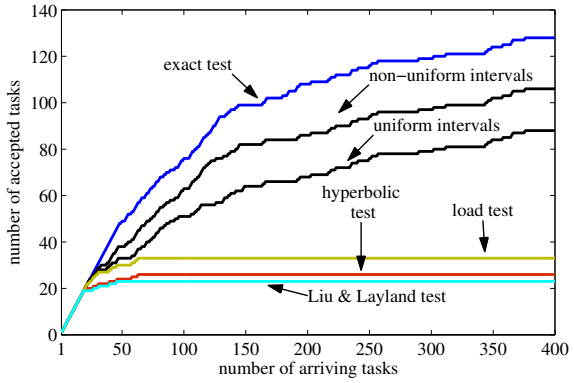

Figure 13. Accepted vs. arriving tasks: 8 processors

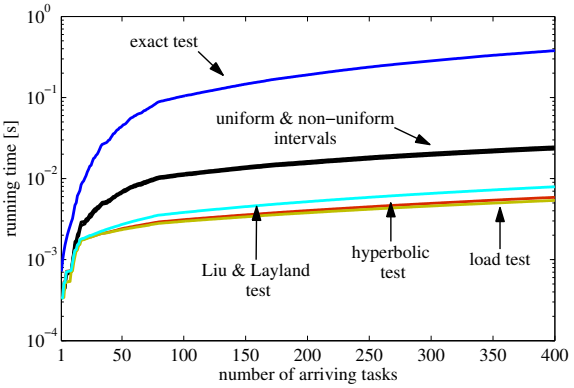

Figure 14. Running time vs. arriving task: 8 processors

VIII. Concluding Remarks

In this paper, we presented an admission control test with constant complexity for the Deadline Monotonic (DM) policy and the case $d_i \leq p_i$. Although we can adapt techniques from the literature to derive constant-time tests for this setup, the proposed test is less pessimistic and allows accepting considerably more tasks with the same complexity.

The test we propose computes the maximum loading factor generated by all already running tasks and the new task on a given processor. We defined the loading factor of a DM task as the ratio between the task's worst-case response time divided by its deadline. If the maximum loading factor on the considered processor does not exceed 1, the task set with the new task is schedulable and the new task can be accepted.

In addition, the proposed test partitions the time line into intervals, such that the maximum loading factor is approximated by linear segments. The number of intervals determines the number of segments and, hence, the quality of the approximation. However, more segments lead to a longer running time of the algorithm.

We analyzed two variants of this test. The first one is based on uniform intervals, i.e., b intervals of the same length in $[0, t_b]$ where the parameter t_b can be selected according to the expected deadlines. On the other hand, the second variant consists of non-uniform intervals with gradually increasing lengths in $[0, t_b]$. In both these variants, the $(b+1)$-th interval extends from t_b to infinity.

For a given b, our experiments show that the variant with non-uniform intervals yields a better acceptance ratio. This result demonstrates that the accuracy of our approximation technique does not only depend on the number of segments used, but also on their distribution along the time line. Hence, a better accuracy can be obtained through better distributing approximation segments. Finally, as future work, we plan to use previous information about task parameters such as probabilistic distributions to improve approximation techniques.

References

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

[2] J.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," in *Performance Evaluation*, vol. 2, 1982, pp. 237–250.

[3] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: exact characterization and average case behavior," in *Proceedings of the Real-Time Systems Symposium*, December 1989, pp. 166–171.

[4] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, vol. 8, no. 5, pp. 284–292, September 1993.

[5] E. Bini and G. Buttazzo, "Schedulability analysis of periodic fixed priority systems," *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1462–1473, November 2004.

[6] N. Fisher and S. Baruah, "A fully polynomial-time approximation scheme for feasibility analysis in static-priority systems with arbitrary relative deadlines," in *Proceedings of the 17th Euromicro Conference on Real-Time Systems*, July 2005, pp. 117–126.

[7] E. Bini and S. Baruah, "Efficient computation of response time bounds under fixed-priority scheduling," in *Proceedings of the 15th conference on Real-Time and Network Systems*, March 2007.

[8] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in hard real-time environments," *Journal of the Association for Computing Machinery*, vol. 20, no. 1, pp. 40–61, 1973.

[9] A. Masrur, S. Chakraborty, and G. Färber, "Constant-time admission control for deadline monotonic tasks," in *Proceedings of the DATE Conference on Design, Automation and Test in Europe*, March 2010, pp. 220–225.

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

[11] J. Liu, *Real-Time Systems*. New Jersey, USA: Prentice Hall, 2000.

[12] E. Bini, G. Buttazzo, and G. Buttazzo, "A hyperbolic bound for the rate monotonic algorithm," in *Proceedings of the 13th Euromicro Conference on Real-Time Systems*, June 2001.

[13] E. Bini and G. Buttazzo, "Rate monotonic analysis: the hyperbolic bound," *IEEE Transactions on Computers*, vol. 52, no. 7, pp. 933–942, July 2003.

[14] Y. Oh and S. Son, "Allocating fixed-priority periodic tasks on multiprocessor systems," *Real-Time Systems*, vol. 9, no. 3, pp. 207–239, 1995.

[15] T.-W. Kuo and A. Mok, "Load adjustment in adaptive real-time systems," in *Proceedings of the 12th IEEE Real-Time Systems Symposium*, December 1991, pp. 160–170.

[16] A. Burchard, J. Liebeherr, Y. Oh, and S. Son, "New strategies for assigning real-time tasks to multiprocessor systems," *IEEE Transactions on Computers*, vol. 44, no. 12, pp. 1429–1442, 1996.

[17] J. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," in *Proceedings of the 11th IEEE Real-Time Systems Symposium*, December 1990, pp. 201–209.

[18] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal*, vol. 29, no. 5, pp. 390–395, 1986.

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

[20] E. Bini and G. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2005.

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