2015 IEEE 21st International Conference on Embedded and Real-Time Computing Systems and Applications

Bi-Level Deadline Scaling for Admission Control in Mixed-Criticality Systems

Alejandro Masrur, Dirk Müller, and Matthias Werner
Department of Computer Science
TU Chemnitz, Germany

***Abstract*—In some cases, tasks may be allowed to migrate from one processor to another, e.g., due to hardware failures or for workload balancing. If a mixed-criticality setting is considered, it is necessary to decide whether new tasks with different levels of criticality may be accepted by a processor without compromising the already running tasks. Since this decision has to be taken on-line, there is a need for fast but yet accurate schedulability tests for mixed-criticality systems. In this paper, we consider that the EDF-VD algorithm is used to schedule tasks on the different processors. EDF-VD assigns *virtual deadlines* to high-criticality tasks, i.e., it uniformly downscales their *real deadlines*, to account for a potential increase in their execution demand. A *deadline scaling factor* is hence computed for the whole processor. However, in the case where the increase in computation demand strongly differs from one task to another, scaling deadlines uniformly makes EDF-VD incur pessimism. Of course, a scaling factor can be computed for every single task; however, this leads to a considerably more complex algorithm which cannot be used in an on-line setting. As a result, we propose an intermediate solution by introducing a *bi-level* deadline scaling. This way, high-criticality tasks that experience a small increase of workload are assigned one scaling factor, whereas tasks with a large increase of workload are assigned a separate scaling factor. Our experiments show that the proposed approach dominates the original EDF-VD algorithm while it does not increase complexity allowing for constant-time admission control in mixed-criticality systems.**

I. Introduction

With the aim of reducing complexity and costs, in many industrial domains such as automotive systems, there is a trend towards consolidating software functions onto fewer ECUs (electronic control units) or processors. Hence applications with different levels of criticality start being executed on the same processors. As a result, there is a need for techniques that allow designing such mixed-criticality (MC) systems and also complying with today's certification requirements.

In this paper, we study the problem of admission control in MC settings. We consider that a task may migrate from its source to a target processor, for example, in response to hardware failures or for workload balancing in an adaptive system. Clearly, in order not to affect the already running tasks, it needs to be decided whether the new task can be accepted (or not) on the target processor. This decision process is referred to as admission control. An admission control algorithm has an on-line nature and should ideally have a constant running time to accept/reject a new task. In other words, it should allow for fast decisions independent of the number of tasks currently in the system.

We assume that *earliest deadline first with virtual deadlines* (EDF-VD) is used to schedule a mix of high-criticality (HI) and low-criticality (LO) tasks on each processor [1] [2]. In principle, EDF-VD is based on the following two observations. First, techniques for *worst-case execution time* (WCET) computation are very conservative leading to a large overestimation of the real WCET [3] [4]. Moreover, a slight underestimation referred to as *optimistic* WCET, e.g., obtained by probabilistic or statistical methods [5] [6], is often sufficient to guarantee a correct operation in *most* of the cases. Second, in MC settings, the LO tasks do not require any guarantees and might be discarded in overload situations.

EDF-VD distinguishes between two operation modes: HI and LO mode. In LO mode, HI tasks require executing for no longer than their optimistic WCETs and are scheduled together with the LO tasks. The system enters the HI mode when one or more HI tasks require executing for their conservative WCETs (i.e., for longer than their optimistic WCETs). Here, EDF-VD discards all LO tasks in order to accommodate the increase in HI execution demand.

In HI mode, HI tasks need to be scheduled within their *real deadlines*, while these are assigned *virtual deadlines* in LO mode. Virtual deadlines are obtained by uniformly *downscaling* their real deadlines by a factor $x \in (0, 1)$. In practice, however, it is oftentimes the case that HI tasks strongly differ from one another. Some HI tasks might have conservative WCETs which are close to their optimistic WCETs. Some other HI tasks might have conservative WCETs which are many times greater than their optimistic WCETs [7]. When such HI tasks need to be jointly scheduled, uniformly downscaling deadlines leads to pessimism.

Alternatively, we can compute a *deadline scaling factor* for each individual HI task [8]. However, whereas the original EDF-VD allows for a linear-complexity schedulability test in the number of tasks (i.e., a constant complexity in accepting/rejecting a new task), the latter method results at best in an algorithm with pseudo-polynomial complexity [8]. Since it is difficult to precisely determine the running time of a pseudo-polynomial-time algorithm, this is of limited utility for admission control.

To overcome this problem, we propose an intermediate solution consisting of a *bi-level* deadline scaling. HI tasks with a small increase from optimistic to conservative WCET are assigned a scaling factor x, whereas tasks with a big increase in their execution demand are treated with a special scaling factor y. The proposed scheme significantly outperforms the original EDF-VD algorithm, while it does not increase complexity allowing for a more accurate constant-time admission control[1].

A. Contributions

The contributions of this paper are fourfold:

[1]Note that our approach can further be extended to more than two scaling factors. This does not affect the complexity, but it increases the effective running time of the admission control algorithm by a non-negligible time constant depending on the number of scaling factors.

- We present a new schedulability test for EDF-VD. This schedulability test is equivalent to the one in [2]; however, in contrast to it, our test can be easily extended to more than one deadline scaling factor.
- To better accommodate large and small execution time variations in HI mode, we propose a bi-level deadline scaling consisting of two scaling factors and derive schedulability conditions.
- We present two variants of the proposed technique allowing for more accurate constant-time admission control than approaches from the literature. The first variant computes two *independent* scaling factors; the second variant computes two *related* scaling factors.
- We analyze the admission control problem in the context of MC systems and derive a number of requirements that need to be met in order that new tasks can be accepted without affecting the schedulability of already running tasks.

B. Structure of the Paper

The rest of this paper is structured as follows. Related work is discussed in Section II. Next, Section III explains the task model and assumptions used. We shortly revisit the algorithm *wost-case reservations* (WCR) in Section IV, which can also be used for constant-time admission control in MC settings. Section V discusses EDF-VD in detail and Section VI presents our alternative schedulability test for EDF-VD, which we extend for bi-level deadline scaling in Section VII. The admission control problem in the context of MC systems is discussed in Section VIII. Section IX presents experimental results and Section X concludes the paper.

II. Related Work

MC scheduling was first proposed under this name by Vestal [9]. Baruah *et al.* later analyzed per-task priority assignments and the resulting response times [10]. In [1], Baruah *et al.* proposed the EDF-VD algorithm to schedule a mix of HI and LO tasks. As stated above, EDF-VD introduces two operation modes and uses a *priority-promotion* scheme by scaling deadlines of HI tasks. A speed-up factor for EDF-VD was first obtained in [1] as $(\sqrt{5}+1)/2$. Later this speed-up factor was improved to $4/3$ [2].

For multiprocessors, a partitioned and a global scheduling approach both based on EDF-VD were proposed in [11]. According to this work, partitioned behaves better than global scheduling in the context of MC systems. Further, in [12], Pathan studies global MC scheduling with task-level fixed priorities and gives a schedulability test based on response time analysis for more than two criticality levels.

A more flexible approach with per-task deadline scaling was presented by Ekberg and Yi [8] [13]. However, this algorithm has exponential complexity in its original form and pseudo-polynomial complexity when approximated by a heuristic. Therefore, it cannot be used in on-line settings where scaling factors might need to be recomputed.

Recently, other improvements to EDF-VD were proposed. In [14], Su and Zhu used an *elastic* task model [15] [16] to improve resource utilization in MC systems. In [17], Zhao *et al.* applied *preemption thresholds* [18] in MC scheduling in order to better utilize the processing unit. Although [14] and [18] improve resource-awareness in MC scheduling, they cannot efficiently manage task sets with small and large variations of execution demand in HI mode. A complete overview of mixed-criticality systems is given in [19].

The approach presented in this paper aims to improve MC scheduling for precisely that case. Note that the concepts in this paper may also be combined with those in [14] and [18].

III. Models and Assumptions

We consider a multi-processor system where a set of MC tasks are scheduled in a partitioned manner. That is, tasks are *assigned* to specific processors. In normal operation, they cannot be preempted and resumed on different processors. However, tasks may be explicitly migrated/re-assigned to another processor, e.g., in response to hardware failures or for workload balancing.

We basically adopt the task model originally proposed in [1]. We denote by τ the set of n independent sporadic tasks τ_i that run on a given processor under preemptive uniprocessor scheduling – to simplify we omit indexes identifying different processors. The minimum separation between any two jobs or instances of a τ_i is denoted by T_i and we assume implicit deadlines, i.e., $\forall i : D_i = T_i$ where D_i is a task's relative deadline. There is no self-suspension, and context-switch overheads are assumed to be negligible on the different processors.

In this paper, we are concerned with dual-criticality systems with two levels of criticality, namely LO and HI. The *criticality* of a task τ_i is denoted by $\chi_i \in \{LO, HI\}$. A LO task is associated with its WCET C_i^{LO}. Opposed to this, a HI task is characterized by its optimistic WCET estimate C_i^{LO} and its conservative WCET estimate C_i^{HI}, clearly being $C_i^{LO} < C_i^{HI}$. We define Δu_i as the increase in utilization from LO to HI mode by an individual HI task:

$$\Delta u_i = \frac{C_i^{HI} - C_i^{LO}}{T_i}. \tag{1}$$

Basically, the system operates in two modes denoted by m: LO and HI mode. In LO mode, HI tasks execute for no longer than C_i^{LO}, whereas these might require executing for up to C_i^{HI} in HI mode. Initially, we assume the system to be in LO mode where all LO and HI tasks need to meet their deadlines. As soon as a job of a HI task executes for longer than its C_i^{LO}, the system switches to HI mode where only the HI tasks are allowed to execute.

In the following, we define utilization parameters $U_\chi^m := \sum_{\chi_i=\chi} \frac{C_i^m}{T_i}$ where again $\chi, m \in \{LO, HI\}$. Note that, among the four potential criticality-to-mode combinations, only U_{LO}^{LO}, U_{HI}^{LO} and U_{HI}^{HI} are defined. U_{LO}^{HI} does not exist since LO tasks are dropped and, hence, do not run in HI mode.

IV. Worst-Case Reservations

WCR is the most intuitive approach to schedule MC systems, which can be used for constant-time admission control. However, it is also the most pessimistic. Under WCR, the optimistic C_i^{LO} of HI tasks is replaced by the conservative C_i^{HI}. If this simplified system is schedulable under EDF, the original MC system is also schedulable. Hence WCR allows successfully scheduling a set of MC tasks if (2) holds [20]:

$$U_{LO}^{LO} + U_{HI}^{HI} \leq 1. \tag{2}$$

The WCR approach assumes that HI tasks always require executing for C_i^{HI}. This is pessimistic since, in LO mode, HI tasks only require C_i^{LO} amount of execution.

V. The EDF-VD Algorithm

EDF-VD [1] is an extension of the EDF algorithm [20] to MC systems. Based on EDF-VD, we can also implement constant-time admission control. Its basic idea is to promote HI jobs in LO mode by shortening their deadlines so as to *reserve* processor capacity for the HI mode. That is, $D'_i = xT_i$ with $x \in (0, 1)$ for all i where $\chi_i = HI$. D'_i is referred to as *virtual deadline* and is used instead of D_i – the *real deadline* – to schedule HI tasks in LO mode. The parameter x is the so-called *deadline scaling factor*. There is no deadline scaling for LO tasks such that they are scheduled using their D_i. In HI mode, HI tasks start being scheduled according to their real deadlines D_i whereas LO tasks are discarded. In both LO and HI mode, tasks are scheduled under the EDF algorithm.

From the above description, in order that EDF-VD be schedulable, the LO and HI tasks need to be schedulable with their corresponding C_i^{LO} under EDF in LO mode. Similarly, in HI mode, the HI tasks also need to be schedulable with their corresponding C_i^{HI} under EDF. As a result, the following two schedulability conditions are necessary:

$$U_{LO}^{LO} + U_{HI}^{LO} \leq 1, \tag{3}$$
$$U_{HI}^{HI} \leq 1. \tag{4}$$

In [2], Baruah *et al.* also obtained a sufficient schedulability condition for EDF-VD in the form of a utilization bound: $\max\big(U_{LO}^{LO} + U_{HI}^{LO}, U_{HI}^{HI}\big) \leq 3/4$. They also proposed a more accurate schedulability test based on whether a scaling factor x can be obtained or not [2]. To this end, a *lower* and an *upper* bound on x are computed:

$$\frac{U_{HI}^{LO}}{1 - U_{LO}^{LO}} \leq x, \tag{5}$$
$$x \leq \frac{1 - U_{HI}^{HI}}{U_{LO}^{LO}}. \tag{6}$$

If the value of x obtained with (5) is less than or equal to the value obtained with (6), then it is possible to find a valid x for the considered system and EDF-VD is schedulable.

VI. New Schedulability Test for EDF-VD

In this section, we discuss a new schedulability test for EDF-VD. This consists of an alternative upper bound on x, which was previously presented in [21] and which we formalize in Theorem 1. In contrast to [2], the proposed test can be easily extended to more than one deadline scaling factor and constitutes the basis of our bi-level deadline scaling.

Theorem 1: A set τ of MC tasks as defined in Section III is schedulable under EDF-VD, if – apart from (3), (4), and (5) – the following condition holds for $0 < x < 1$:

$$\frac{\Delta U}{1 - x} \leq 1, \tag{7}$$

where ΔU is given by $\sum_{\chi_i = HI} \Delta u_i$ and Δu_i is defined in (1).

Proof: Since (5) holds for $0 < x < 1$, we obtain the following expression:

$$U_{LO}^{LO} + \frac{U_{HI}^{LO}}{x} \leq 1, \tag{8}$$

which means that all LO and all HI tasks can be scheduled in LO mode, where HI tasks are scheduled within their virtual deadlines.

Let us now consider that the system *switches* to HI mode at time t'. Note that (4) guarantees schedulability from the point in time t'' onwards, at which the processor first idles after t'. As a result, if a deadline is missed, this can only happen in the interval $[t', t'']$.

If all jobs would run for *at most* C_i^{LO} in $[t', t'']$, note that (8) guarantees that they finish executing *at latest* by their corresponding *virtual* deadlines and, hence, no deadline would be missed in $[t', t'']$. Recall that only jobs of HI tasks run from t' onwards.

Let us consider an arbitrary job of an arbitrary task τ_j in $[t', t'']$. Further, let this job be released at time t_j with $t' \leq t_j \leq t''$. As stated above, if all jobs run for at most C_i^{LO} in $[t', t'']$, this τ_j's job will finish executing at latest by $t_j + xT_j$. This implies that all higher-priority jobs, i.e., jobs whose (absolute) real deadlines are less than or equal to $t_j + T_j$, must have finished executing prior to $t_j + xT_j$.

In the worst case, any increase in execution demand by τ_j's or its higher-priority jobs will have to be scheduled in $[t_j + xT_j, t_j + T_j]$, i.e., within an interval of length $T_j - xT_j$ starting from $t_j + xT_j$. Again, all jobs with higher priority than this τ_j's job will have finished executing prior to $t_j + xT_j$, if they run for at most C_i^{LO}. As a consequence, the problem of guaranteeing schedulability in $[t', t'']$ reduces to proving that the worst-case increase in execution demand by each job in $[t', t'']$ of each τ_j can be accommodated within an interval of length $T_j - xT_j$ starting from the corresponding absolute virtual deadline.

Towards this, we can model the worst-case increase in execution demand by any τ_i in $[t', t'']$ as a task $\tilde{\tau}_i$ with execution demand $\tilde{C}_i = C_i^{HI} - C_i^{LO}$, period $\tilde{T}_i = T_i$, a relative deadline $\tilde{D}_i = T_i - xT_i$, and initial phase or release time $\tilde{\phi}_i \in [0, T_i]$ – clearly, at time t', not necessarily all $\tilde{\phi}_i$ are zero. We can use the concept of *demand bound function* [22] to compute the worst-case increase in execution demand in an interval of length $t - t'$. We can guarantee schedulability in $[t', t'']$, if the demand bound function for all $\tilde{\tau}_i$ is always less than or equal to the length of the interval $t - t'$ with $t' \leq t \leq t''$:

$$\sum_{\chi_i = HI} \max\left(0, \left\lfloor \frac{t - t' - \tilde{D}_i - \tilde{\phi}_i}{\tilde{T}_i} \right\rfloor + 1\right) \cdot \tilde{C}_i \leq t - t'. \tag{9}$$

However, since we cannot know the exact values of $\tilde{\phi}_i$, we opt for a safe approximation of (9). This consists in applying the *density test* [23] on the task set of all $\tilde{\tau}_i$:

$$\sum_{\chi_i = HI} \frac{\tilde{C}_i}{\tilde{D}_i} = \frac{\Delta U}{1 - x} \leq 1. \tag{10}$$

Note that, if (10) holds, (9) also holds for any $t - t'$ and any values of $\tilde{\phi}_i$. This implies that we can always accommodate the worst-case increase in execution demand by any τ_i's job and its higher-priority jobs within an interval of length $T_i - xT_i$. The theorem follows. □

The following lemma proves that the schedulability test given in Theorem 1 is equivalent to that of Baruah *et al.* [2].

Lemma 1: If a set τ of MC tasks as defined in Section III is schedulable by the schedulability test of Baruah *et al.* [2], it is also schedulable by Theorem 1 and vice versa.

Proof: For τ to be schedulable by Baruah *et al.*, the value of x obtained with (5) has to be less than or equal to the value obtained with (6). Similarly, if the system is schedulable by Theorem 1, the value of x obtained with (5) has to be less than or equal to the value obtained with (7).

The proof of equivalence consists in showing that (5) is always less than or equal to (7), if it is also less than or equal to (6) and vice versa:

$$\begin{aligned} & \frac{U_{HI}^{LO}}{1-U_{LO}^{LO}} \leq 1 - U_{HI}^{HI} + U_{HI}^{LO} \\ \Leftrightarrow\quad & \frac{U_{HI}^{LO} - U_{HI}^{LO}(1-U_{LO}^{LO})}{1-U_{LO}^{LO}} \leq 1 - U_{HI}^{HI} \\ \Leftrightarrow\quad & \frac{U_{HI}^{LO}U_{LO}^{LO}}{1-U_{LO}^{LO}} \leq 1 - U_{HI}^{HI} \\ \Leftrightarrow\quad & \frac{U_{HI}^{LO}}{1-U_{LO}^{LO}} \leq \frac{1-U_{HI}^{HI}}{U_{LO}^{LO}}. \end{aligned}$$

As a result, both these schedulability tests are proven to be equivalent to one another and the lemma follows. □

VII. Bi-level Deadline Scaling

EDF-VD outperforms the WCR approach. However, it does not consider the case where HI tasks with small and large variations of execution demand need to be jointly scheduled. In this section, to overcome this problem, we propose a bi-level deadline scaling scheme. We introduce an additional *independent* scaling factor y to EDF-VD and then consider the case of two related scaling factors. For both these cases, we derived schedulability tests that allow for a constant-time admission control as discussed in Section VIII.

A. Two Scaling Factors

We divide the set of HI tasks into two disjoint subsets $\check{\tau}$ and $\hat{\tau}$ on the considered processor such that the following holds for all $\tau_i \in \check{\tau}$ and all $\tau_j \in \hat{\tau}$:

$$\Delta u_i < \Delta u_j. \tag{11}$$

In other words, the increase in utilization produced by tasks in $\hat{\tau}$ is greater than that of tasks in $\check{\tau}$. For classifying tasks either into $\check{\tau}$ or $\hat{\tau}$, a threshold has to be defined such that (11) always holds. Later we perform a set of experiments to evaluate how to properly choose this threshold.

We rewrite (3) and (4) to consider the new notation resulting from these two subsets of HI tasks:

$$U_{LO}^{LO} + \check{U}_{HI}^{LO} + \hat{U}_{HI}^{LO} \leq 1, \tag{12}$$
$$\check{U}_{HI}^{HI} + \hat{U}_{HI}^{HI} \leq 1, \tag{13}$$

where $\check{U}_{HI}^{LO}$ and $\hat{U}_{HI}^{LO}$ denote the utilization in LO mode of respectively $\check{\tau}$ and $\hat{\tau}$. Similarly, $\check{U}_{HI}^{HI}$ and $\hat{U}_{HI}^{HI}$ denote $\check{\tau}$'s and $\hat{\tau}$'s utilization in HI mode.

When switching from LO to HI mode, tasks in $\hat{\tau}$ have a higher computation demand than tasks in $\check{\tau}$. Hence we propose *reserving* more computation capacity in LO mode for $\hat{\tau}$ than for $\check{\tau}$. This is achieved by introducing a second scaling factor such that virtual deadlines in $\hat{\tau}$ are given by yT_i and in $\check{\tau}$ by xT_i, where y and x are the scaling factors. Note that $0 < y \leq x < 1$ must hold. Analogous to (8), in LO mode, we have the following condition:

$$U_{LO}^{LO} + \frac{\check{U}_{HI}^{LO}}{x} + \frac{\hat{U}_{HI}^{LO}}{y} \leq 1. \tag{14}$$

If (14) holds, it can be guaranteed that all HI tasks execute for C_i^{LO} time within their corresponding virtual deadlines. We reshape (14) to get a lower bound on y depending on x:

$$\frac{\hat{U}_{HI}^{LO}}{1 - U_{LO}^{LO} - \frac{\check{U}_{HI}^{LO}}{x}} \quad\leq\quad y. \tag{15}$$

Now, to obtain an upper bound on y, we first need to extend Theorem 1 to the case of two deadline scaling factors.

Theorem 2: Consider a set τ of MC tasks as defined in Section III, whose HI tasks are classified into two disjoint subsets $\check{\tau}$ and $\hat{\tau}$, for which (12) and (13) hold. The set τ is schedulable under EDF-VD, if x and y can be found such that (14) and the following condition hold:

$$\frac{\Delta\check{U}}{1-x} + \frac{\Delta\hat{U}}{1-y} \leq 1, \tag{16}$$

where $0 < y \leq x < 1$, $\Delta\check{U}$ is equal to $\sum_{i\in\check{\tau}} \Delta u_i$, $\Delta\hat{U}$ is given by $\sum_{i\in\hat{\tau}} \Delta u_i$, and Δu_i is defined as per (1).

Proof: If (14) holds for given values of x and y where $0 < y \leq x < 1$, all LO tasks and all HI tasks can be scheduled in LO mode. The HI tasks are scheduled within their virtual deadlines.

Let us now consider that the system *switches* to HI mode at time t'. Note that (13) guarantees schedulability from the point in time t'' onwards, at which the processor first idles after t'. As a result, if a deadline is missed, this can only happen in the interval $[t', t'']$.

If all jobs would run for *at most* C_i^{LO} in $[t', t'']$, note that (14) guarantees that they finish executing *at latest* by their corresponding *virtual* deadlines and, hence, no deadline would be missed in $[t', t'']$. Recall again that only jobs of HI tasks run from t' onwards.

Similar as for Theorem 1's proof, let us consider an arbitrary job of an arbitrary task τ_j released at time t_j with $t' \leq t_j \leq t''$. Further let τ_j belong to $\check{\tau}$. If all jobs run for at most C_i^{LO} in $[t', t'']$, this τ_j's job will finish executing at latest by $t_j + xT_j$. This implies that all higher-priority jobs, i.e., jobs whose (absolute) real deadlines are less than or equal to $t_j + T_j$, must have finished executing prior to $t_j + xT_j$.

In the worst case, any increase in execution demand by τ_j's or its higher-priority jobs will have to be scheduled in $[t_j + xT_j, t_j + T_j]$, i.e., within an interval of length $T_j - xT_j$ starting from $t_j + xT_j$. Again, all jobs with higher priority than this τ_j's job will have finished executing prior to $t_j + xT_j$, if they run for at most C_i^{LO}. Note that the above analysis also holds true, if τ_j belongs to $\hat{\tau}$. In the latter case, clearly, x needs to be replaced by y. The problem of guaranteeing schedulability in $[t', t'']$ reduces to proving that the worst-case increase in execution demand by each job in $[t', t'']$ of each τ_j can be accommodated within an interval of length $T_j - xT_j$ if $\tau_j \in \check{\tau}$ or an interval of length $T_j - yT_j$ if $\tau_j \in \hat{\tau}$, starting from the corresponding absolute virtual deadline.

We can model the worst-case increase in execution demand by any τ_i in $[t', t'']$ as a task $\tilde{\tau}_i$ with execution demand $\tilde{C}_i =$

Algorithm 1 Algorithm twoFactors

```
Require: $\tau$, $\hat{\tau}$, $\check{\tau}$, and $step$
 1: if $U_{LO}^{LO} + \hat{U}_{HI}^{LO} + \check{U}_{HI}^{LO} > 1$ then
 2:     Return ("not schedulable")
 3: else if $\hat{U}_{HI}^{HI} + \check{U}_{HI}^{HI} > 1$ then
 4:     Return ("not schedulable")
 5: end if
 6: $x = step$
 7: while $x < 1$ do
 8:     Compute $y_{min}$ by (15)
 9:     Compute $y_{max}$ by (18)
10:     if $y_{min} \leq y_{max}$ and $y_{min} > 0$ and $y_{max} < 1$ then
11:         Return ("schedulable")
12:     end if
13:     $x = x + step$
14: end while
15: Return ("not schedulable")
```

$C_i^{HI} - C_i^{LO}$, period $\tilde{T}_i = T_i$, a relative deadline $\tilde{D}_i = T_i - xT_i$ or $\tilde{D}_i = T_i - yT_i$ depending on whether τ_i belongs to $\check{\tau}$ or $\hat{\tau}$, and initial phase or release time $\tilde{\phi}_i \in [0, T_i]$. We can again use the concept of *demand bound function* [22] to compute the worst-case increase in execution demand in an interval of length $t - t'$. We can guarantee schedulability in $[t', t'']$, if the demand bound function for all $\tilde{\tau}_i$ is always less than or equal to the length of the interval $t - t'$ with $t' \leq t \leq t''$ – see again (9). Since we cannot know the exact values of $\tilde{\phi}_i$, we again apply the *density test* [23] on the task set of all $\tilde{\tau}_i$:

$$\sum_{\chi_i = HI} \frac{\tilde{C}_i}{\tilde{D}_i} = \sum_{i \in \check{\tau}} \frac{\Delta u_i}{1-x} + \sum_{i \in \hat{\tau}} \frac{\Delta u_i}{1-y} = \frac{\Delta \check{U}}{1-x} + \frac{\Delta \hat{U}}{1-y} \leq 1. \quad (17)$$

Note that, if (17) holds, we can always accommodate the worst-case increase in execution demand by any τ_i's job and its higher-priority jobs within an interval of length $T_i - xT_i$ if $\tau_i \in \check{\tau}$ or an interval of length $T_i - yT_i$ if $\tau_i \in \hat{\tau}$. As a result, the theorem follows. □

We can now reshape (16) to obtain an upper bound on y as a function of x:

$$\frac{1 - \frac{\Delta \check{U}}{1-x} - \Delta \hat{U}}{1 - \frac{\Delta \check{U}}{1-x}} \quad \geq \quad y. \quad (18)$$

The fact that $0 < x < 1$ holds can be used to test schedulability of the system with two scaling factors. To this end, we propose algorithm *twoFactors* – see Alg. 1. This algorithm increases x in discrete steps from a value close to 0 towards 1. For each step, y_{min} is computed using (15) and y_{max} is computed using (18). If y_{min} is less than or equal to y_{max}, then the task set is schedulable – x, y_{min} and y_{max} have valid values.

On the other hand, if x is increased up to 1 and $y_{min} \leq y_{max}$ does not hold for any of the steps, the task set is deemed unschedulable. Although Alg. 1 systematically searches the entire parameter space, note that its accuracy depends on the choice of $step$. For a given application, if the task set is schedulable with two scaling factors, a wrong choice of $step$ may lead to a false negative result.

Finally, note that we can proceed as for Theorem 2 and add a third or even more additional scaling factors up to n, i.e., one per task. Even though the resulting schedulability test is *safe*, we will still only have one equation and three or n unknowns, depending on whether three or n scaling factors are used. This does not increase the complexity, but the time constant of the resulting algorithm in an non-negligible manner.

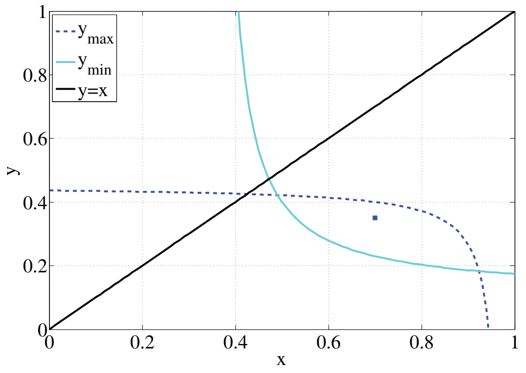

Fig. 1. Bounds (15) and (18) for the task set of Table I

Task	χ_i	T_i (ms)	C_i^{LO}(ms)	C_i^{HI}(ms)
τ_1	LO	10	1	-
τ_2	HI	20	2	13
τ_3	HI	40	13	14

TABLE I. Task parameters

An illustrative example. We consider a set of MC tasks consisting of one LO task τ_1 and two HI tasks τ_2 and τ_3 as shown in Table I. To test the schedulability under the original EDF-VD, we first obtain $U_{LO}^{LO} = 0.1$, $U_{HI}^{LO} = 0.425$ and $U_{HI}^{HI} = 1$. Then, the lower bound on x becomes $0.425/(1-0.1) \approx 0.472$ as per (5). Similarly, the upper bound on x is $(1-1)/0.1 = 0$ according to (6) and $1 - 1 + 0.425 = 0.425$ according to (7). Independently of whether (6) or (7) is used, the upper bound on x is below its lower bound rendering the set of solutions for x empty and, hence, the task set of Table I unschedulable.

Fig. 1 shows the outcome of twoFactors applied to this example. More precisely, lower (15) and upper (18) bounds on y are plotted as a function of x. The algorithm twoFactors returns *schedulable* as soon it finds a point between these two bounds, i.e., a valid combination of the two scaling factors x and y. Note that the line $y = x$ does not cross the solution space between y_{min} and y_{max} in Fig. 1. This illustrates the fact that a *uniform* deadline scaling – as originally proposed for EDF-VD – is unable to schedule task sets such as the one of Table I. Here, assigning τ_3 and τ_2 virtual deadlines equal to 28 and 7, for example, makes the above task set schedulable. This corresponds to $x = 0.7$ and $y = 0.35$ as indicated by the marker inside the solution space in Fig. 1.

Clearly, the above example could have been solved using the schedulability test of Ekberg and Yi [8] [13]. However, as mentioned above, this latter schedulability test has at best pseudo-polynomial complexity and is, hence, not suitable for admission control.

Comparison with the original EDF-VD. For $step \to 0$, we demonstrate that twoFactors *strictly dominates* the original EDF-VD algorithm [2], i.e., the case of uniform deadline scaling. Later, in Section IX-B, we show that $step = 0.01$ is a sufficiently good approximation of the limit case $step \to 0$.

Intuitively, the proposed deadline scaling based on two independent scaling factors is a more general case of the

Algorithm 2 Algorithm relFactors

```
Require: $\tau$, $\hat{\tau}$, $\check{\tau}$, and $\alpha$
 1: if $U_{LO}^{LO} + \hat{U}_{HI}^{LO} + \check{U}_{HI}^{LO} > 1$ then
 2:     Return (“not schedulable”)
 3: else if $\hat{U}_{HI}^{HI} + \check{U}_{HI}^{HI} > 1$ then
 4:     Return (“not schedulable”)
 5: end if
 6: Compute $x_{min}$ by (20)
 7: Compute $x_{max}$ by greatest positive real root $< 1$ of (22)
 8: if $x_{min} \le x_{max}$ and $x_{min} > 0$ and $x_{max} < 1$ then
 9:     Return (“schedulable”)
10: else
11:     Return (“not schedulable”)
12: end if
```

uniform deadline scaling by the original EDF-VD algorithm. The following lemma formalizes this statement.

Lemma 2: Eq. (15) and (18) reduce to (5) and (7) respectively for the case $y = x$.

Proof: We prove that Eq. (15) reduces to (5) for $y = x$:

$$\begin{aligned} & \frac{\hat{U}_{HI}^{LO}}{1 - U_{LO}^{LO} - \frac{\check{U}_{HI}^{LO}}{x}} \le x, \\ \Leftrightarrow \quad & \hat{U}_{HI}^{LO} \le (1 - U_{LO}^{LO})x - \check{U}_{HI}^{LO} \\ \Leftrightarrow \quad & \frac{\hat{U}_{HI}^{LO} + \check{U}_{HI}^{LO}}{1 - U_{LO}^{LO}} \le x. \end{aligned}$$

Since $\hat{U}_{HI}^{LO} + \check{U}_{HI}^{LO}$ is the sum of the LO utilizations of $\check{\tau}$ and $\hat{\tau}$, which are two disjoint subsets containing all HI tasks in τ, $\hat{U}_{HI}^{LO} + \check{U}_{HI}^{LO} = U_{HI}^{LO}$ holds. Similarly, we have that Eq. (18) reduces to (7) for $y = x$:

$$\begin{aligned} & \frac{1 - \frac{\Delta\check{U}}{1-x} - \Delta\hat{U}}{1 - \frac{\Delta\check{U}}{1-x}} \ge x \\ \Leftrightarrow \quad & 1 - \frac{\Delta\check{U}}{1-x} - \Delta\hat{U} \ge \left(1 - \frac{\Delta\check{U}}{1-x}\right)x \\ \Leftrightarrow \quad & -\frac{\Delta\check{U}}{1-x} + x\frac{\Delta\check{U}}{1-x} - \Delta\hat{U} \ge x - 1 \\ \Leftrightarrow \quad & -(x-1)\frac{\Delta\check{U} + \Delta\hat{U}}{1-x} \ge x - 1 \\ \Leftrightarrow \quad & \frac{\Delta\check{U} + \Delta\hat{U}}{1-x} \le 1. \end{aligned}$$

Since $\Delta\check{U} = \check{U}_{HI}^{HI} - \check{U}_{HI}^{LO}$, and $\Delta\hat{U} = \hat{U}_{HI}^{HI} - \hat{U}_{HI}^{LO}$, $\Delta U = \Delta\check{U} + \Delta\hat{U}$ holds and we finally obtain $\frac{\Delta U}{1-x} \le 1$. □

The above lemma shows that the algorithm twoFactors reduces to computing (5) and (7) for the special case $y = x$. By Lemma 1 and Theorem 1, we know that this is equivalent to the original schedulability test of [2]. This evidences that twoFactors is more general than and, hence, strictly dominates the original EDF-VD algorithm.

B. Related Scaling Factors

Although the technique presented above is safe, it requires iterating over x until valid values of y are obtained or the task set is rendered unschedulable. Clearly, the performance of the resulting twoFactors depends on the parameter $step$ by which x is incremented at every iteration – see Alg. 1.

In this section, we analyze a variant of this approach which does not require iterating over x. This consists in making $y = \alpha x$ where α is a real constant value and $0 < \alpha \le 1$ holds. As a consequence, (14) becomes:

$$U_{LO}^{LO} + \frac{\check{U}_{HI}^{LO}}{x} + \frac{\hat{U}_{HI}^{LO}}{\alpha x} \le 1, \tag{19}$$

which can be reshaped to obtain a lower bound for x:

$$\begin{aligned} & \frac{\check{U}_{HI}^{LO}}{x} + \frac{\hat{U}_{HI}^{LO}}{\alpha x} \le 1 - U_{LO}^{LO} \\ \Leftrightarrow \quad & \frac{\alpha\check{U}_{HI}^{LO} + \hat{U}_{HI}^{LO}}{\alpha x} \le 1 - U_{LO}^{LO} \\ \Leftrightarrow \quad & \frac{\alpha\check{U}_{HI}^{LO} + \hat{U}_{HI}^{LO}}{\alpha(1 - U_{LO}^{LO})} \le x. \end{aligned} \tag{20}$$

In a similar manner, replacing y by αx in (16) yields:

$$\frac{\Delta\check{U}}{1-x} + \frac{\Delta\hat{U}}{1-\alpha x} \le 1, \tag{21}$$

which again can be solved for x as follows:

$$\begin{aligned} & \frac{(1-\alpha x)\Delta\check{U} + (1-x)\Delta\hat{U}}{(1-x)(1-\alpha x)} \le 1, \\ \Leftrightarrow \quad & (1-\alpha x)\Delta\check{U} + (1-x)\Delta\hat{U} \le (1-x)(1-\alpha x) \\ \Leftrightarrow \quad & \Delta\check{U} - \alpha x\Delta\check{U} + \Delta\hat{U} - x\Delta\hat{U} \le 1 - x - \alpha x + \alpha x^2 \\ \Leftrightarrow \quad & -\alpha x^2 + (\alpha - \alpha\Delta\check{U} - \Delta\hat{U} + 1)x \le 1 - \Delta\check{U} - \Delta\hat{U}. \end{aligned} \tag{22}$$

Note that, for any schedulable task set, $\Delta\check{U} + \Delta\hat{U} \le 1$ must hold, i.e., the increase in utilization by HI tasks can never be more than 1 for the task set to be schedulable. (Note that if the latter does not hold, (13) will neither hold). In addition, since $0 < \alpha \le 1$ holds, the factor $\alpha - \alpha\Delta\check{U} - \Delta\hat{U} + 1$ in (22) is positive and greater than α. As a result, the closer x gets to 1, the greater $|\alpha x^2|$ and, hence, the harder it is to fulfill (22). For this reason, the greatest, positive, real root of (22) that is less than 1 constitutes an upper bound on x. If this root is also greater than the value given by (20), the set of MC tasks is schedulable.

Based on the previous analysis, we propose a second algorithm called *relFactors* – see Alg. 2. This algorithm computes x_{min} using (20) and x_{max} obtaining the greatest, positive, real root of (22). If x_{min} is less than or equal to x_{max}, then the task set is schedulable – x_{min} and x_{max} can be returned at this stage. If x_{min} is equal to or less than 0 or x_{max} is equal to or greater than 1, then the task set is rendered unschedulable.

Comparison with the original EDF-VD. The proposed relFactors is also more general than uniformly scaling deadlines as in the original EDF-VD. The following lemma formalizes the latter statement.

Lemma 3: Eq. (20) and (22) reduce to (5) and (7) respectively for the case $\alpha = 1$.

Proof: Considering that $U_{HI}^{LO} = \check{U}_{HI}^{LO} + \hat{U}_{HI}^{LO}$, it can be easily seen that (20) leads to (5) when replacing $\alpha = 1$. We now prove that (22) leads to (7) when replacing $\alpha = 1$. Towards this, let us further consider that $U_{HI}^{HI} = \check{U}_{HI}^{HI} + \hat{U}_{HI}^{HI}$. So, (22) can be reshaped to:

$$-x^2 + (1 - \Delta\check{U} - \Delta\hat{U} + 1)x \le 1 - \Delta\check{U} - \Delta\hat{U},$$

which has two roots: $1 - \Delta\check{U} - \Delta\hat{U}$ and 1. As stated above, we have to take the greatest positive root that is less than 1.

In this case, this is clearly $1 - \Delta\check{U} - \Delta\hat{U}$, which results in (7) considering that $\Delta U = \Delta\check{U} + \Delta\hat{U}$ holds. □

C. Complexity Analysis

Since U_{LO}^{LO}, $\check{U}_{HI}^{LO}$, $\check{U}_{HI}^{HI}$, $\hat{U}_{HI}^{LO}$ and $\hat{U}_{HI}^{HI}$ need to be computed for a set τ of n MC tasks, both twoFactors and relFactors – see Alg. 1 and Alg. 2 respectively – have linear complexity $\mathcal{O}(n)$. For twoFactors, note that the number of iterations of the while-loop is upper bounded by $\lfloor 1/step \rfloor$ which is a constant for a fixed value of $step$. How to choose this parameter is discussed later in the experimental evaluation.

Finally, note that twoFactors can be rewritten such that its while-loop has an $\mathcal{O}(\log n)$ instead of an $\mathcal{O}(n)$ complexity, e.g., applying *binary search*. The overall complexity is still $\mathcal{O}(n)$ due to the need of computing the utilization parameters, but its time-constant will be reduced in a considerable manner.

VIII. The Admission Control Problem

For a new task τ_{new} to be accepted or rejected in an already running MC system, a schedulability test needs to be performed. If the task passes the test, this results in a new deadline scaling factor – or two such factors. As a consequence, virtual deadlines of the HI tasks in the system will change, thus, altering scheduling conditions, which needs to be analyzed carefully.

In this section, we discuss a number of conditions to implement a constant-time admission control algorithm and guarantee a safe transition from τ to $\tau \cup \{\tau_{new}\}$, i.e., the set of MC tasks including τ_{new}.

Constant-time implementation. The state-of-the-art approaches, i.e., WCR and the original EDF-VD [2], as well as the proposed twoFactors and relFactors require computing utilization parameters. Note again that the algorithm of [8] and [13] cannot be used for constant-time admission control since it incurs pseudo-polynomial complexity.

WCR computes U_{LO}^{LO} and U_{HI}^{HI}, whereas the original EDF-VD additionally computes U_{HI}^{LO}. The proposed twoFactors and relFactors compute $\check{U}_{HI}^{LO}$, $\check{U}_{HI}^{HI}$, $\hat{U}_{HI}^{LO}$ and $\hat{U}_{HI}^{HI}$ instead of just U_{HI}^{LO} and U_{HI}^{HI}. Clearly, this requires a linear complexity; however, the admission control algorithm can keep the cumulated utilization values in memory. When a new task τ_{new} is added to the system, it just needs to update these values depending on χ_{new} – the criticality of the new task – which requires a constant instead of linear time.

If $\chi_{new} = HI$ and either the proposed twoFactors or relFactors are used, the admission control algorithm further computes $\Delta u_{new} = \frac{C_{new}^{HI} - C_{new}^{LO}}{T_{new}}$. If Δu_{new} is less than a preconfigured threshold – we discuss how to choose this threshold in our experimental evaluation, the admission control algorithm updates $\check{U}_{HI}^{LO}$ and $\check{U}_{HI}^{HI}$; otherwise it updates $\hat{U}_{HI}^{LO}$ and $\hat{U}_{HI}^{HI}$.

To allow for constant complexity, the EDF-VD scheduler should be implemented such that two conditions are met. First the deadline scaling factor x is kept in memory – for twoFactors x and y should be stored in memory. Second, the virtual deadline for any HI task in LO mode xT_i should be computed at its jobs' release times. In the case of twoFactors or relFactors, xT_i, yT_i or $\alpha x T_i$ should be computed at release time depending on whether the HI task belongs to τ or τ. This way, when a new task is accepted in the system, it suffices to update the scaling factors – no need to update virtual deadlines – leading to a constant-time complexity.

Safe admission control. In contrast to WCR, an admission control algorithm based on deadline scaling needs to be analyzed carefully. To the best of our knowledge, this is the first work dealing with this problem. In the following, due to space limitation, we consider the case of uniform deadline scaling. Note that the presented analysis extends to bi-level deadline scaling in a straightforward manner – as Theorem 2 easily extends to more than two scaling factors.

If a τ_{new} can be accepted in the system, a new deadline scaling factor x_{new} could be found for which $\tau \cup \{\tau_{new}\}$ is schedulable. We also know that τ is schedulable with the old scaling factor x_{old}. However, also the transition between τ and $\tau \cup \{\tau_{new}\}$ needs to be guaranteed schedulable. For this, all possible cases should be considered: (i) the system is *stable* in HI mode and the $\chi_{new} = LO$, (ii) the system is *stable* in HI mode and the $\chi_{new} = HI$, (iii) the system is in *stable* in LO mode and the $\chi_{new} = LO$, (iv) the system is in *stable* in LO mode and $\chi_{new} = HI$, and (v) the system is at a transition between LO and HI mode.

In **case (i)**, τ_{new} will not be able to run until the system returns to LO mode, while τ_{new} can start executing without delay after being accepted in **case (ii)**. The schedulability in case (ii) is guaranteed by (4). In case (i) and (ii), when the system returns to LO mode, x_{new} will further guarantee schedulability.

In **case (iii)**, we know from (5) that the following condition holds, i.e., the system with x_{old} is feasible:

$$U_{LO}^{LO} + \frac{U_{HI}^{LO}}{x_{old}} \leq 1, \tag{23}$$

and since τ_{new} can be accepted, $U_{LO}^{LO} + \frac{C_{new}^{LO}}{T_{new}} + \frac{U_{HI}^{LO}}{x_{new}} \leq 1$ must also hold leading to:

$$U_{LO}^{LO} + \frac{U_{HI}^{LO}}{x_{new}} \leq 1. \tag{24}$$

From the discussion above, note that once x_{new} has been computed, all *new* job releases of HI tasks will be *updated* with virtual deadlines $x_{new}T_i$. Hence, for some time, some of the HI tasks have *old* and some *new* virtual deadlines. We denote by $\bar{\tau}$ the set of HI tasks with $x_{old}T_i$ deadlines and by $\check{\tau}$ the set of HI tasks with $x_{new}T_i$ deadlines where $\bar{\tau}$ and $\check{\tau}$ are disjoint subsets containing the totality of the HI tasks in τ. Eqs. (23) and (24) can be rewritten as follows:

$$U_{LO}^{LO} + \frac{\bar{U}_{HI}^{LO} + \check{U}_{HI}^{LO}}{x_{old}} \leq 1, \tag{25}$$

$$U_{LO}^{LO} + \frac{\bar{U}_{HI}^{LO} + \check{U}_{HI}^{LO}}{x_{new}} \leq 1, \tag{26}$$

where $\bar{U}_{HI}^{LO}$ and $\check{U}_{HI}^{LO}$ are the LO utilizations of $\bar{\tau}$ and $\check{\tau}$ respectively. Since both (25) and (26) hold, $U_{LO}^{LO} + \frac{\bar{U}_{HI}^{LO}}{x_{old}} + \frac{\check{U}_{HI}^{LO}}{x_{new}} \leq 1$ also holds. As a result, tasks in τ will be schedulable with a combination of $x_{old}T_i$ and $x_{new}T_i$ deadlines provided that τ_{new} does not run. If τ_{new} starts running, only the $x_{new}T_i$ deadlines are guaranteed schedulable.

For this reason, the first release of τ_{new} has to be postponed until all HI tasks in the system have virtual deadlines equal to $x_{new}T_i$. Let us assume that the admission control algorithm accepts τ_{new} and, hence, computes x_{new} at time t. If d_{max} denotes the latest absolute virtual deadline of an active HI job at time t, τ_{new}'s first release will be delayed by a time equal to $d_{max} - t$. After d_{max}, note that all HI tasks will have virtual deadlines equal to $x_{new}T_i$ and, hence, it is safe to start executing τ_{new}. In this transition between τ and $\tau \cup \{\tau_{new}\}$, if the system remains in the LO mode, schedulability is guaranteed by (25) and (26).

Since the admission control algorithm has found x_{new}, switches to HI mode are safe after d_{max}. However, we need to analyze what happens if the system switches to HI mode between t and d_{max}. Here, from (7), we have that the following two conditions hold – since x_{old} and x_{new} are valid:

$$\frac{\Delta U}{1 - x_{old}} \leq 1, \tag{27}$$

$$\frac{\Delta U}{1 - x_{new}} \leq 1. \tag{28}$$

If the system switches to HI mode between t and d_{max}, some of the HI tasks will have *old* and some *new* virtual deadlines. Again, let us denote by $\bar{\tau}$ the set of HI tasks with $x_{old}T_i$ deadlines and by $\breve{\tau}$ the set of HI tasks with $x_{new}T_i$ deadlines. Since $\bar{\tau}$ and $\breve{\tau}$ are disjoint subsets containing all HI tasks, (27) and (28) can be rewritten as:

$$\frac{\Delta \bar{U}}{1 - x_{old}} + \frac{\Delta \breve{U}}{1 - x_{old}} \leq 1, \tag{29}$$

$$\frac{\Delta \bar{U}}{1 - x_{new}} + \frac{\Delta \breve{U}}{1 - x_{new}} \leq 1, \tag{30}$$

where $\Delta\bar{U} = \bar{U}_{HI}^{HI} - \bar{U}_{HI}^{LO}$ is the increase in utilization of $\bar{\tau}$, and $\Delta\breve{U} = \breve{U}_{HI}^{HI} - \breve{U}_{HI}^{LO}$ the increase in utilization of $\breve{\tau}$. Here (29) and (30) indicate that $\frac{\Delta\bar{U}}{1-x_{old}} + \frac{\Delta\breve{U}}{1-x_{new}} \leq 1$ also holds and, thus, switching to HI mode in $[t, d_{max}]$ is safe according to Theorem 2.

In **case (iv)**, we know that (23) holds. On the other hand, since $U_{LO}^{LO} + \frac{U_{HI}^{LO}}{x_{new}} + \frac{C_{new}^{LO}}{x_{new}T_{new}} \leq 1$ holds as τ_{new} can be accepted, (24) also holds. As for case (iii), the first release of τ_{new} in the system will be postponed until all HI tasks in the system have virtual deadlines equal to $x_{new}T_i$, i.e., until d_{max}, where d_{max} is the latest absolute virtual deadline of an active HI job at τ_{new}'s time of admission t.

Similar to case (iii), we need to analyze what happens if the system switches to HI mode between t and d_{max}. From (7), we know that (27) and the following condition hold guaranteeing safe switches to HI mode before t and after d_{max}:

$$\frac{\Delta U + \Delta u_{new}}{1 - x_{new}} \leq 1. \tag{31}$$

Let $\bar{\tau}$ represent the set of HI tasks with $x_{old}T_i$ deadlines and $\breve{\tau}$ the set of HI tasks with $x_{new}T_i$ deadlines in $[t, d_{max}]$. Again, these are disjoint subsets containing the totality of the HI tasks in τ. Considering that $C_{new}^{LO} < C_{new}^{HI}$ and that τ_{new} does not run in $[t, d_{max}]$, i.e., $\Delta u_{new} = 0$ in $[t, d_{max}]$, (27) and (31) can again be rewritten as (29) and (30) respectively. These indicate that $\frac{\Delta\bar{U}}{1-x_{old}} + \frac{\Delta\breve{U}}{1-x_{new}} \leq 1$ holds and, hence, switching to HI mode in $[t, d_{max}]$ is safe as per Theorem 2.

Finally, if the system switches to HI mode in $[t, d_{max}]$, τ_{new} must wait until the point in time at which the processor first idles, which is also the situation in **case (v)**. From the point in time at which the processor idles, schedulability is guaranteed by (4).

The MC system should clearly tolerate the delay incurred by the admission control algorithm. For example, in the case of workload balancing, τ_{new} will have to continue on its *old* processor until it can start running on the *new* processor. Similarly, if task migration is performed in response to hardware errors, these have to permit the above mentioned delay.

IX. Experimental Evaluation

We evaluate the performance of the proposed bi-level deadline scaling. The two variants, with independent scaling factors and with related scaling factors, are compared against the WCR and the original EDF-VD algorithm (i.e., with the algorithms from the literature that can be used for constant-time admission control).

A. Test Data

In this section, we explain how we obtained test data for our experiments. The description below is common to all curves presented in the paper. Details concerning a specific curve will be explained as it becomes necessary. We use the algorithm *UUniFast* [24] to generate task sets for a varying *LO utilization*, i.e., for a varying $U_{LO}^{LO} + \breve{U}_{HI}^{LO} + \hat{U}_{HI}^{LO}$. A total number of 250,000 different task sets were created for each of the curves shown below. In our experiments we consider task sets with 10 tasks respectively, where every time half of the tasks are LO and half HI tasks. Out of the HI tasks 80% belong to $\breve{\tau}$ and 20% to $\hat{\tau}$. That is, for 10 tasks per set, $|\breve{\tau}| = 4$ and $|\hat{\tau}| = 1$, i.e., only one task experiences a large increase of computation demand in HI mode.

B. Configuring twoFactors

As shown in Alg. 1, twoFactors requires a parameter called *step*. This variable determines the increase of x in each iteration of the while-loop. If *step* is small, twoFactors will require more time to run; however, its accuracy will be higher. On the other hand, if *step* is set to be large, twoFactors will run faster but it will have less accuracy.

In this section, we perform experiments to investigate what values are meaningful for *step*. For this purpose, let us consider Fig. 2. Test data was created as described in Section IX-A, where $\frac{\Delta u_i}{u_i}$ of tasks in $\breve{\tau}$ are uniformly generated in the interval $(0, 0.1]$. Here, Δu_i is defined in (1) and $u_i := \frac{C_i^{LO}}{T_i}$. That is, tasks in $\breve{\tau}$ experience up to 10% more execution demand in HI mode. Tasks in $\hat{\tau}$ have 3 times more execution demand in HI mode.

Clearly, for $step = 0.1$, twoFactors has less accuracy than for $step = 0.01$ or $step = 0.001$ as it can be observed in Fig. 2. However, for $step = 0.1$, the while-loop is executed 10 times – see Alg. 1, whereas for 0.01 and $step = 0.001$ it is executed 100 and 1,000 times respectively. Further there is almost no improvement from $step = 0.01$ to $step = 0.001$. Therefore, for our remaining experiments, we choose $step = 0.01$ for twoFactors. Note that this choice makes twoFactors be 100 time slower than the original EDF-VD. However, as discussed in Section VII-C, twoFactors can be speed up in a considerable manner by using *binary search*.

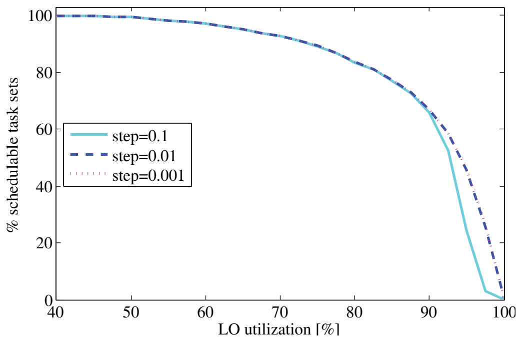

Fig. 2. twoFactors: different *step* for $n = 10$

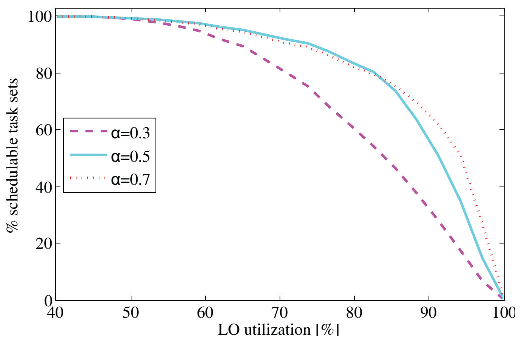

Fig. 3. relFactors: different α for $n = 10$

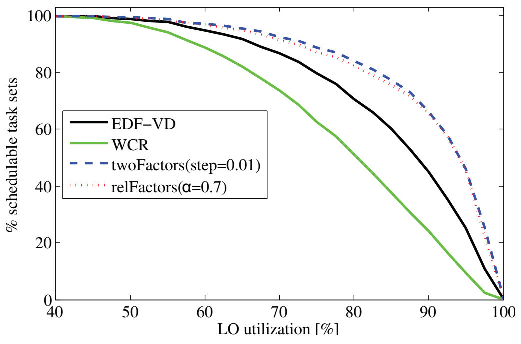

Fig. 4. Schedulability curves for $\frac{\Delta u_i}{u_i} = 3 \; \forall \, \tau_i \in \hat{\tau}$ and $n = 10$

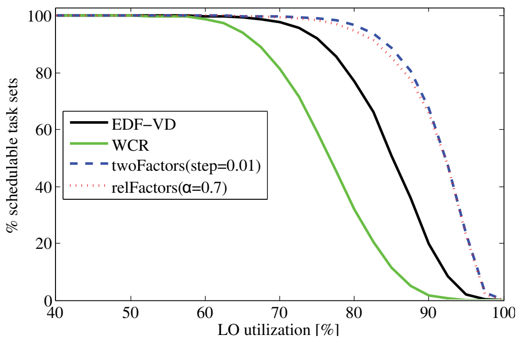

Fig. 5. Schedulability curves for $\frac{\Delta u_i}{u_i} = 3 \; \forall \, \tau_i \in \hat{\tau}$ and $n = 50$

C. *Configuring relFactors*

The algorithm relFactors requires a parameter named α which establishes the relation between the two scaling factors – see Alg. 2. As discussed previously, when α gets closer to 1, the performance of relFactors gets closer to that of the original EDF-VD algorithm.

In this section, we perform experiments to evaluate possible values of α. For this, let us consider Fig. 3. Again, test data was created as described in Section IX-A, where tasks in $\check{\tau}$ experience up to 10% more execution demand and tasks in $\hat{\tau}$ experience 3 times more execution demand in HI mode.

From Fig. 3, we can observe that $\alpha = 0.7$ delivers the best results. Probably, one would expect a better performance of relFactors for $\alpha = 0.3$, i.e., for a smaller α. However, a decreasing α leads to shorter virtual deadlines for tasks in $\hat{\tau}$ and, hence, these will stop being schedulable from a given point onwards. For our following experiments, we use $\alpha = 0.7$ obtained in an empirical manner. An optimum value of α clearly depends on task parameters and, hence, will vary from task set to task set remaining an open problem.

D. *Comparison with WCR and EDF-VD*

In this section we compare the proposed algorithms with WCR and EDF-VD with respect to schedulability. That is, for a number of task sets, we compare how many of them can be successfully scheduled by the different algorithms. In other words, algorithms with better schedulability curves will be able to accommodate more new tasks in an admission control setting.

Test data was created again as described in Section IX-A, where as before tasks in $\check{\tau}$ require in HI mode up to 10% additional execution demand. The additional execution demand of tasks in $\hat{\tau}$ is varied for the different curves.

In Fig. 4 and 5, we compare algorithms for the case where tasks in $\hat{\tau}$ require 3 times more execution demand in HI mode. The proposed algorithms twoFactors and relFactors both outperform EDF-VD, where twoFactors has a slightly better performance than relFactors. In a LO utilization interval from 80% to 95%, we can see that the proposed algorithms allow around 20% more schedulatility for $n = 10$ – see Fig. 4 – and around 40% more schedulability for $n = 50$ – see Fig. 5. As expected, WCR is the algorithm with the worst performance.

When considering that tasks in $\hat{\tau}$ double their execution demand in HI mode, i.e., their increase in execution demand gets closer to that of tasks in $\check{\tau}$, the improvement of both proposed algorithms over EDF-VD gets smaller. Considering twoFactors, From around 20% to 40% more schedulability for high LO utilizations in the case of Fig. 4 and 5, it reduces to around 10% more schedulable task sets for $n = 10$ in Fig. 6 and to 20% more schedulable task sets for $n = 50$ in Fig. 7. Note that relFactors presents almost no improvement with respect to EDF-VD in this case.

Overall, the experiments presented in this section show the usefulness of the proposed approach for the case where the HI execution demand of tasks in $\hat{\tau}$ strongly differs from that of tasks in $\check{\tau}$. If this is not the case, the proposed algorithms, particularly twoFactors, still introduce an improvement with respect to EDF-VD; however, this becomes less significant.

From the above exposition, the threshold on Δu_i – to classify tasks either into $\check{\tau}$ or $\hat{\tau}$ – should be chosen such that $\frac{\Delta u_i}{u_i} \geq 1$. That is, all HI tasks with a HI execution that is less than twice the LO execution demand should be classified

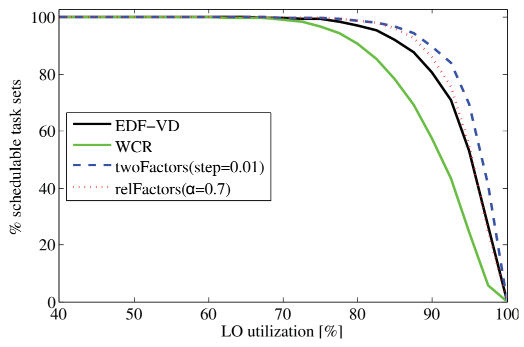

Fig. 6. Schedulability curves for $\frac{\Delta u_i}{u_i} = 1 \; \forall \tau_i \in \hat{\tau}$ and $n = 10$

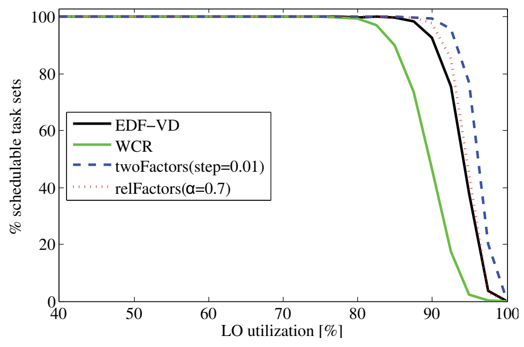

Fig. 7. Schedulability curves for $\frac{\Delta u_i}{u_i} = 1 \; \forall \tau_i \in \hat{\tau}$ and $n = 50$

into $\check{\tau}$; otherwise into $\hat{\tau}$. This constellation is not uncommon in practice, where the WCET of a task might be many times greater than possible *optimistic* estimates of it such as those based on the average execution time [7].

X. Concluding Remarks

In this paper, we proposed introducing a second deadline scaling factor to the original EDF-VD algorithm. This way, to achieve a more accurate schedulability test, tasks experiencing a large increase of computation in HI mode are treated separately from other tasks in the system.

After deriving schedulability conditions for this case, we proposed two algorithms *twoFactors* and *relFactors* and compared them with the state-of-the-art approaches having the same complexity, viz., WCR and the original EDF-VD algorithm. Our experiments show that the proposed technique strictly outperforms these known approaches which we also proved in an analytical comparison.

The two proposed algorithms can be used for constant-time admission control in MC settings, where new tasks need to be tested for admittance on an already running system. We further derived necessary conditions that any admission control algorithm needs to fulfill in order to be safe and, hence, used in MC systems.

Acknowledgments

The authors would like to thank the reviewers and Prof. Sanjoy Baruah for their valuable comments on this paper.

References

[1] S. Baruah, V. Bonifaci, G. D'Angelo, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie, "Mixed-criticality scheduling of sporadic task systems," in *Proc. of European Symposium on Algorithms (ESA)*, 2011.

[2] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie, "The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems," in *Proc. of Euromicro Conference on Real-Time Systems (ECRTS)*, 2012.

[3] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Müller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem - overview of methods and survey of tools," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, 2008.

[4] M. Lv, W. Yi, N. Guan, and G. Yu, "Combining abstract interpretation with model checking for timing analysis of multicore software," in *Proc. of Real-Time Systems Symposium (RTSS)*, 2010.

[5] L. Kosmidis, J. Abella, F. Wartel, E. Quinones, A. Colin, and F. Cazorla, "PUB: Path upper-bounding for measurement-based probabilistic timing analysis," in *Proc. of Euromicro Conference on Real-Time Systems (ECRTS)*, Jul. 2014.

[6] M. Slijepcevic, L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla, "Time-analysable non-partitioned shared caches for real-time multicore systems," in *Proc. of Design Automation Conference (DAC)*, Jun. 2014.

[7] A. Colin and S. Petters, "Experimental evaluation of code properties for WCET analysis," in *Proc. of Real-Time Systems Symposium (RTSS)*, 2003.

[8] P. Ekberg and W. Yi, "Bounding and shaping the demand of mixed-criticality sporadic tasks," in *Proc. of Euromicro Conference on Real-Time Systems (ECRTS)*, 2012.

[9] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Proc. of Real-Time Systems Symposium (RTSS)*, 2007.

[10] S. Baruah, A. Burns, and R. Davis, "Response-time analysis for mixed criticality systems," in *Proc. of Real-Time Systems Symposium (RTSS)*, 2011.

[11] S. Baruah, B. Chattopadhyay, H. Li, and I. Shin, "Mixed-criticality scheduling on multiprocessors," *Real-Time Systems (RTS)*, vol. 50, 2013.

[12] R. Pathan, "Schedulability analysis of mixed-criticality systems on multiprocessors," in *Proc. of Euromicro Conference on Real-Time Systems (ECRTS)*, 2012.

[13] P. Ekberg and W. Yi, "Bounding and shaping the demand of generalized mixed-criticality sporadic task systems," *Real-Time Systems (RTS)*, vol. 50, no. 1, 2014.

[14] H. Su and D. Zhu, "An elastic mixed-criticality task model and its scheduling algorithm," in *Proc. of Design, Automation and Test in Europe (DATE)*, 2013.

[15] T.-W. Kuo and A. K. Mok, "Load adjustment in adaptive real-time systems," in *Proc. of Real-Time Systems Symposium (RTSS)*, 1991.

[16] G. Buttazzo, G. Lipari, and L. Abeni, "Elastic task model for adaptive rate control," in *Proc. of Real-Time Systems Symposium (RTSS)*, 1998.

[17] Q. Zhao, Z. Gu, and H. Zeng, "PT-AMC: Integrating Preemption Thresholds into Mixed-Criticality Scheduling," in *Proc. of Design, Automation and Test in Europe (DATE)*, 2013, pp. 141–146.

[18] Y. Wang and M. Saksena, "Scheduling fixed-priority tasks with preemption threshold," in *Proc. of Real-Time Computing Systems and Applications (RTCSA)*, 1999.

[19] A. Burns and R. Davis, "Mixed criticality systems - a review," Department of Computer Science, University of York, Tech. Rep., 2015.

[20] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, 1973.

[21] D. Müller and A. Masrur, "The schedulability region of two-level mixed-criticality systems based on EDF-VD," in *Proc. of Design, Automation and Test in Europe (DATE)*, 2014.

[22] S. Baruah, A. Mok, and L. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *Proc. of Real-Time Systems Symposium (RTSS)*, Dec. 1990.

[23] J. W. S. Liu, *Real-Time Systems*. Prentice Hall, 2000.

[24] E. Bini and G. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems (RTS)*, vol. 30, no. 1-2, 2005.