Introducing Utilization Caps into Mixed-Criticality Scheduling

Mitra Mahdiani and Alejandro Masrur
Department of Computer Science
TU Chemnitz, Germany

***Abstract*—We are concerned with mixed-criticality systems where a set of low-criticality (LO) and high-criticality (HI) tasks share one processor and are scheduled under EDF-VD algorithm. EDF-VD implements two operation modes: LO and HI. In LO mode, one or more HI tasks may exceed their execution budgets, which then causes a change to HI mode in the system. In HI mode, HI tasks are assigned larger execution budgets at the cost of the LO tasks, which often need to be discarded. In some cases, however, we would like to allow some LO tasks to continue running on the processor in spite of switching to HI mode. To this end, we incorporate utilization caps into the original EDF-VD algorithm. The idea is to partition tasks on the processor, for example, according to functional dependencies, and assign them a portion the total utilization. EDF-VD then applies to each of these partitions individually and up to their corresponding utilization caps. If one HI task exceeds its execution budget in LO mode, this only affects the LO tasks in the same partition, but not LO tasks in other partitions which can continue running. We present a technique to optimally choose utilization caps for each partition and perform a large set of experiments based on synthetic data illustrating benefits of the proposed technique.**

I. Introduction

In safety-critical domains such as avionics and automotive systems, embedded software has to pass a strict certification process. This is usually regulated according to different *criticality levels* which are specified for the particular domains [1] [2]. Clearly, the certification of *high-criticality* (HI) tasks is more rigorous and well-regulated than that of *low-criticality* (LO) tasks and, hence, LO task are usually more error-prone than HI tasks.[1]

This is of concern when consolidating software functions onto fewer processing units, since functions or tasks with different levels of criticality start being executed on the same processor. It may happen that failures of any kind in a LO task affect one or more HI task. As a result, in particular, there is a need for methods and techniques that allow designing such mixed-criticality (MC) systems and, at the same time, complying with safety and certification requirements.

In this paper, 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 one processor [3]. In principle, EDF-VD introduces two operation modes: HI and LO. In LO mode, HI tasks run for no longer than their *optimistic* execution budgets and are scheduled together with the LO tasks. The system switches to HI mode when one or more HI tasks run for their conservative execution budgets.

[1]Although aerospace and automotive safety regulations define around five levels of criticality, for ease of exposition, we consider only two such levels in this paper. However, the proposed technique remains valid for multiple levels of criticality.

To accommodate the increase in HI execution demand, EDF-VD in its original form discards all LO tasks in HI mode. In practice, however, we would often require some LO tasks to continue running even if one or more HI tasks switch to HI mode. To this end, existing approaches from the literature are based on degrading LO tasks in HI mode [4], i.e., whatever computation capacity is left by HI tasks in HI mode is used to provide some reduced service for the LO tasks.

Another approach is based on servers that enforce temporal isolation being restricted to pre-configured execution budgets and replenishment periods [5]. The idea is that LO tasks are not affected by HI tasks running on different servers. The advantage of this latter approach is that LO tasks in the servers that are not affected by a switch to HI mode can continue running without being degraded.

Contributions. In this paper, we follow the above line of work and propose a technique that allows LO tasks to continue running without degradation (in spite of some HI tasks having switched to HI mode). Our technique extends EDF-VD by introducing utilization caps.

In principle, tasks are partitioned following some functional criteria. In particular, if a HI task switches to HI mode and, hence, it does not make sense that some LO task continues running, e.g., it becomes unnecessary or superfluous, then these two tasks should belong to the same partition or subset. Each of such partitions is assigned a utilization cap, i.e., a portion of the total processor utilization.

If one HI task switches to HI mode, then only those LO task within the same partition or subset will be discarded, whereas LO tasks in other partitions continue running. The main advantage over the sever-based approach is that there is no *starvation period*, i.e., the time interval between two runs/repetitions where no service is provided to tasks within the server.

In contrast to this, tasks in a partition run as long as they have not used up their assigned utilization, i.e., as long as they are below their utilization cap. This is a decisive property that allows reducing pessimism with respect to server-based approaches.

The proposed technique does not require modifying EDF-VD algorithm, which remains unchanged within a partition/subset of tasks. It hence facilitates a compositional design of MC systems. Our experimental evaluation based on synthetic data evidences the benefits of the proposed technique.

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 briefly revisit the algorithm EDF-VD in Section IV and

introduce our proposed extension to it in Section V. Section VI presents experimental results and Section VII wraps up and concludes the paper.

II. Related Work

In this section, we briefly revise the rich literature concerning MC systems — a complete overview can be found in [6]. The problem around MC systems was first addressed by Vestal in [7]. In particular, Vestal showed that the well-known deadline monotonic (DM) policy is not optimal for MC systems, where HI tasks may temporarily increase their execution demands. Since then, different approaches have been proposed for MC systems under both fixed and dynamic priorities.

For single processors, Baruah *et al.* analyzed different priority assignments and the resulting response times under fixed priorities [8]. In [9], Baruah *et al.* presented a further priority assignment with a partially better performance than those in [8]. Burns and Davis proposed another fixed-priority scheme where non-preemptive regions are added to tasks and showed that this leads to an even better performance than previously published schemes [10].

Under EDF, in [3], Baruah *et al.* proposed the EDF-VD algorithm to schedule a mix of HI and LO tasks. EDF-VD introduces two operation modes and uses a *priority-promotion* scheme by uniformly scaling deadlines of HI tasks. A speed-up factor for EDF-VD was first obtained in [3] as $(\sqrt{5}+1)/2$. Later this speed-up factor was improved to $4/3$ [11]. A more flexible approach with per-task deadline scaling was presented by Ekberg and Yi [12] [13]. Ekberg and Yi characterized the execution demand of MC systems under EDF by deriving demand bound functions for the LO and the HI mode.

Improvements to the original EDF-VD have also been proposed. In [14], Su and Zhu used an *elastic* task model [15] to improve resource utilization in MC systems. In [16], Zhao *et al.* applied *preemption thresholds* [17] to MC scheduling in order to better utilize the processor. Recently, Huang *et al.* proposed speeding up processors to account for increases in HI execution demand when switching to HI mode [18]. Huang *et al.* made use of Ekberg and Yi's demand bound functions to compute the necessary speed-up factors that guarantee meeting all deadlines.

For multiprocessors, a partitioned and a global scheduling approach both based on EDF-VD were proposed in [19]. According to this work, partitioned behaves better than global scheduling in the context of MC systems. Further, in [20], Pathan studies global MC scheduling under fixed priorities and gives a schedulability test based on response time analysis for more than two criticality levels. Another approach is proposed by Lee *et al.* with their MC-Fluid model [21], which has been improved later in [22]. The MC-Fluid model executes each task at a rate proportional to its utilization improving efficiency.

Whereas, in the above approaches, LO tasks are discarded to accommodate an increase in HI execution demand, Huang *et al.* rather proposed degrading LO tasks [4], for which they derive degraded timing guarantees. Recently, Ren and Phan proposed task grouping and a server-based approach to provide guarantees to both HI and LO tasks on multiprocessors [5]. This paper follows this line of work. Similar to [4] and [5], we propose not to discard LO tasks in case HI execution demand increases. As already mentioned, our technique allows LO tasks to continue running without degradation in contrast to [4], and is less pessimistic than a server-based approach as the one in [5], since it does not incur any starvation period.

III. Models and Assumptions

In this section, we discuss most of our notation. Note that further nomenclature will be introduced as it gets necessary along the paper. We consider a one-processor system where a set of MC tasks are scheduled and basically adopt the task model originally proposed in [3].

We denote by τ the set of n independent — with respect to timing — sporadic tasks that run under preemptive uniprocessor scheduling. The minimum separation between any two jobs or instances of a task 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.

As already stated, we are concerned with dual-criticality systems with two levels of criticality, namely LO and HI. The *criticality* of a task 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 $C_i^{LO} \leq C_i^{HI}$.

Tasks in τ are independent in the sense that they do not affect each other's execution apart from competing for resources. However, as mentioned previously, we consider that some functional dependency does exist for some tasks in the system. In particular, if a HI task switches to HI mode, some LO tasks may not need to run anymore, i.e., they become superfluous. On the other hand, it may be meaningful that some other LO tasks continue to run, if provided sufficient resources.

As a result, we assume that τ can be divided into a number of disjoint subsets $\tau_A, \tau_B, \ldots \tau_Z$, each of which comprises tasks that are functionally related in the described form. More specifically, each τ_X with $X \in \{A, B, \ldots Z\}$ contains a mix of HI and LO tasks: If a HI task within this τ_X switches to HI mode, all LO tasks in τ_X will be discarded; however, LO tasks in the remaining subsets $\tau_A, \tau_B, \ldots \tau_Z$ are allowed to continue running, since they do not functionally depend on any τ_X's tasks.

Basically, the system distinguishes two operation modes m for each subset $\tau_X \subset \tau$: 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, τ_X is in LO mode where all LO and HI tasks therein need to meet their deadlines. As soon as one job of a HI task executes for longer than its C_i^{LO}, τ_X switches to HI mode where only its HI tasks are allowed to execute. Again, this does not affects LO tasks in other subsets but only those in τ_X. In the following, we define utilization parameters such as:

$$U_\chi^m := \sum_{\chi_i = \chi} \frac{C_i^m}{T_i},$$

where again $\chi, m \in \{LO, HI\}$. For simplicity, we avoid an index identifying the subset τ_X in the above parameters.

U_χ^m indicates the processor utilization produced by tasks with criticality χ in mode m.

Finally, among the four potential criticality-to-mode combinations, note that only U_{LO}^{LO}, U_{HI}^{LO} and U_{HI}^{HI} are defined. U_{LO}^{HI} does not exist for a particular $\tau_X \subset \tau$, since LO tasks are dropped when a HI task in τ_X changes to HI mode and, hence, do not run in τ_X's HI mode.

IV. The EDF-VD Algorithm

EDF-VD [3] is an extension of the EDF algorithm [23] to MC systems. 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:[2]

$$U_{LO}^{LO} + U_{HI}^{LO} \leq 1, \quad (1)$$

$$U_{HI}^{HI} \leq 1. \quad (2)$$

In [11], Baruah *et al.* also obtained a sufficient schedulability condition for EDF-VD in the form of a utilization bound: $\max\left(U_{LO}^{LO} + U_{HI}^{LO}, U_{HI}^{HI}\right) \leq 3/4$. They also proposed a more accurate schedulability test based on whether a scaling factor x can be obtained or not [11]. To this end, a *lower* and an *upper* bound on x are computed:

$$\frac{U_{HI}^{LO}}{1 - U_{LO}^{LO}} \leq x, \quad (3)$$

$$x \leq \frac{1 - U_{HI}^{HI}}{U_{LO}^{LO}}. \quad (4)$$

If the value of x obtained with (3) is less than or equal to the value obtained with (4), then it is possible to find a valid x for the considered system and the task system is schedulable by EDF-VD.

V. Introducing Utilization Caps

In this section, we introduce utilization caps to EDF-VD. That is, instead of finding a value of x for the whole τ as previously, we find a value of x for each $\tau_X \subset \tau$, i.e., for each disjoint subset within τ, and limit the amount of utilization it can use. To this end, let us first reshape (3) and (4):

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

$$x \cdot U_{LO}^{LO} + U_{HI}^{HI} \leq 1.$$

Note that the left-hand sides of the above inequalities represent measures of how much utilization is used by the MC tasks on the processor. Since originally EDF-VD assumes that the full processor capacity is available, the task set is feasible or schedulable if those utilization measures are below 1.

We can limit the amount of utilization for any $\tau_X \subset \tau$ by reducing the right-hand side of the above expressions from 1 to U_L in $(0,1]$ as shown below:

$$U_{LO}^{LO} + \frac{U_{HI}^{LO}}{x} \leq U_L,$$

$$x \cdot U_{LO}^{LO} + U_{HI}^{HI} \leq U_L,$$

where we refer to U_L as utilization cap. Now, we can reshape these latter expressions to compute lower and upper bounds on the value of x for any $\tau_X \subset \tau$:

$$\frac{U_{HI}^{LO}}{U_L - U_{LO}^{LO}} \leq x, \quad (5)$$

$$x \leq \frac{U_L - U_{HI}^{HI}}{U_{LO}^{LO}}. \quad (6)$$

In order that $\tau_X \subset \tau$ is schedulable, we need to find a value of x in $(0,1)$, for which (5) and (6) hold. However, this defines a range of possible values, from which one can choose x to be, for example, in the middle of it:

$$x = \frac{U_{HI}^{LO}}{U_L - U_{LO}^{LO}} + \frac{U_L - U_{HI}^{HI}}{2 \cdot U_{LO}^{LO}} - \frac{U_{HI}^{LO}}{2\left(U_L - U_{LO}^{LO}\right)}. \quad (7)$$

A. Fixed utilization caps

Clearly, whether (5) and (6) hold depends on the value of U_L. Let us first assume this to be arbitrarily fixed by the designer as shown in Alg. 1. For example, the designer can decide to equally distribute the processor capacity among individual $\tau_X \subset \tau$. That is, if there are three such subsets in the systems, each of them would be using $1/3$ of the total utilization.

Schedulability test. Alg. 1 shows the schedulability test for EDF-VD with fixed utilization caps. This requires to know all disjoint subsets $\tau_A, \tau_B, \ldots \tau_Z$ included in τ, for which values of U_L are assumed to be specified by the designer.

For each such subset τ_X, the algorithm first computes the values of U_{LO}^{LO}, U_{HI}^{LO} and U_{HI}^{HI} and checks whether $U_{LO}^{LO} + U_{HI}^{LO} > 1$ or $U_{HI}^{HI} > 1$ hold or not — see lines 2 and 6. If these hold, it means that the current subset τ_X is not schedulable and the algorithm returns with an error.

If the above conditions are passed, the algorithm checks whether there exists a valid range of values for x in line 7. If this is the case, U_L of the current τ_X is sum to U — the total processor utilization — in line 8 and x is computed as per (7) in line 9.

If no valid range can be found for x, again an error is returned. Finally, τ is schedulable on one processor if $U <= 1$ holds in line 14, i.e., if the sum of all U_L — i.e., for each $\tau_X \subset \tau$ — is less than 1.

[2]For the equations in Section IV, note that τ is not divided into any partitions/subsets and that the utilization parameters are hence those obtained for all tasks in τ. In the following sections, utilization parameters are again defined for each subset $\tau_A, \tau_B, \ldots \tau_Z \subset \tau$, however, for the sake of simplicity, we omit identifying the different such subsets in the notation.

Algorithm 1 Schedulability test for fixed utilization caps

Require: $\tau = \tau_A \cup \tau_B \cup \ldots \tau_Z$

Require: U_L

1: **for each** $\tau_X \subset \tau$ **do**
2: Compute U_{LO}^{LO}, U_{HI}^{LO} and U_{HI}^{HI}
3: **if** $U_{LO}^{LO} + U_{HI}^{LO} > 1$ **then**
4: Return ("not schedulable")
5: **else if** $U_{HI}^{HI} > 1$ **then**
6: Return ("not schedulable")
7: **else if** $\frac{U_{HI}^{LO}}{U_L - U_{LO}^{LO}} \leq \frac{U_L - U_{HI}^{HI}}{U_{LO}^{LO}}$ **then**
8: $U = U + U_L$
9: Compute x
10: **else**
11: Return ("not schedulable")
12: **end if**
13: **end for**
14: **if** $U > 1$ **then**
15: Return ("not schedulable")
16: **else**
17: Return ("schedulable")
18: **end if**

Note that U_{LO}^{LO}, U_{HI}^{LO}, and U_{HI}^{HI} need to be computed for each subset $\tau_X \subset \tau$. This can be done in linear time, i.e., $\mathcal{O}(n)$, since each τ_X is a disjoint subset of τ and the total number of tasks in τ is given by n.

All other computations and checks in Alg. 1 can be performed in constant time, i.e., $\mathcal{O}(1)$. As a result, the overall complexity is $\mathcal{O}(n)$.

B. Optimized utilization caps

Although using fixed U_L is more straightforward, in some cases, we would like to find an optimum value of U_L for a given $\tau_X \subset \tau$ such that schedulability is guaranteed.

To this end, since the left-hand side of (5) needs to be less than or equal to the right-hand side of (6), we have:

$$\frac{U_{HI}^{LO}}{U_L - U_{LO}^{LO}} \leq \frac{U_L - U_{HI}^{HI}}{U_{LO}^{LO}},$$

and, reshaping to solve for U_L, we finally obtain:

$$0 \leq U_L^2 - \left(U_{LO}^{LO} + U_{HI}^{HI}\right) \cdot U_L + \left(U_{HI}^{HI} - U_{HI}^{LO}\right) \cdot U_{LO}^{LO}. \quad (8)$$

Now, for the system to be schedulable, either $\hat{U}_L$ or $\check{U}_L$ — i.e., any of the roots of (8) when equalized to zero — needs to be a real number in the interval $(0, 1]$:

$$\check{U}_L = \frac{\left(U_{LO}^{LO} + U_{HI}^{HI}\right)}{2} - \frac{\sqrt{\left(U_{LO}^{LO} + U_{HI}^{HI}\right)^2 - 4\left(U_{HI}^{HI} - U_{HI}^{LO}\right) \cdot U_{LO}^{LO}}}{2}, \quad (9)$$

$$\hat{U}_L = \frac{\left(U_{LO}^{LO} + U_{HI}^{HI}\right)}{2} + \frac{\sqrt{\left(U_{LO}^{LO} + U_{HI}^{HI}\right)^2 - 4\left(U_{HI}^{HI} - U_{HI}^{LO}\right) \cdot U_{LO}^{LO}}}{2}. \quad (10)$$

Note that the value of U_L given by one of these roots is a lower bound and that any other greater value that is less than 1 will also guarantee schedulability. On the other hand, if this is not the case, i.e., if neither of $\hat{U}_L$ or $\check{U}_L$ is a real number in the interval $(0, 1]$, the system is rendered unschedulable.

Note that finding a valid U_L implies that a valid x can be obtained as well, for which we again can use (7) as before.

Schedulability test. Alg. 2 shows the schedulability test for EDF-VD with utilization caps, where the values of U_L are optimized as per the above analysis. This algorithm also requires knowing all disjoint subsets $\tau_A, \tau_B, \ldots \tau_Z$ which τ consists of.

For each such subset τ_X, the algorithm first computes the values of U_{LO}^{LO}, U_{HI}^{LO} and U_{HI}^{HI} and checks whether $U_{LO}^{LO} + U_{HI}^{LO} > 1$ or $U_{HI}^{HI} > 1$ hold or not — see lines 3 and 5. If these hold, it means that the current subset τ_X is not schedulable and the algorithm returns an error.

If τ_X passes the above checks, $\check{U}_L$ and $\hat{U}_L$, i.e., the roots of (8), are computed in as per (9) and (10) in line 8. Clearly, if $\check{U}_L$ is a real number in $(0, 1]$, it is meaningful to use this value for U_L since $\check{U}_L < \hat{U}_L$ holds — see lines 9 and 10. That is, this is going to be the lowest possible value of U_L that renders the system schedulable.

On the other hand, if $\check{U}_L$ is not a valid value of U_L (in particular, if it is less than zero), $\hat{U}_L$ may still be a real number in $(0, 1]$ and, hence, this is checked next in lines 11 and 12. Note that, if $\check{U}_L$ is not a real number, $\check{U}_L$ is neither going to be real. As a result, it is not necessary to compute $\hat{U}_L$. The same happens if $\check{U}_L$ is greater than 1, i.e., $\hat{U}_L$ is also going to be greater than 1 and, hence, does not need to be computed.

If neither $\check{U}_L$ nor $\hat{U}_L$ are real numbers in $(0, 1]$, the subset τ_X and, hence, also τ are not schedulable — see lines 13 and 14. If a valid U_L was found for τ_X, this is summed to U, i.e., the total utilization on the processor, in line 16.

As mentioned above, if there exists a valid value of U_L, a valid x also exists for τ_X. However, this needs to be computed, e.g., as per (7) in line 17. Finally, if a U_L could be obtained for every disjoint subset τ_X in τ, τ is only feasible on one processor if $U <= 1$ holds, i.e., if the sum of all U_L is less than 1.

Finally, note that $\check{U}_L$ and $\hat{U}_L$ can be computed for each $\tau_X \subset \tau$ in constant time, i.e., $\mathcal{O}(1)$. As a consequence, similar to Alg. 1, the overall complexity of Alg. 2 is also linear in the form $\mathcal{O}(n)$.

VI. Experimental Evaluation

In this section we evaluate the benefits and drawbacks of the proposed approach consisting in introducing utilization caps to MC scheduling. In particular, we compare our approach as given in Alg. 1, i.e., with fixed utilization caps, to the original EDF-VD algorithm. The choice of Alg. 1 over Alg. 2 was taken to facilitate comparison based on synthetic data. Alg. 1

Algorithm 2 Schedulability test for optimum utilization caps

Require: $\tau = \tau_A \cup \tau_B \cup \ldots \tau_Z$

for each $\tau_X \subset \tau$ **do**
  Compute U_{LO}^{LO}, U_{HI}^{LO} and U_{HI}^{HI}
  if $U_{LO}^{LO} + U_{HI}^{LO} > 1$ **then**
    Return ("not schedulable")
  else if $U_{HI}^{HI} > 1$ **then**
    Return ("not schedulable")
  else
    Compute $\check{U}_L$ and $\hat{U}_L$
    if $\check{U}_L > 0$ **and** $\check{U}_L <= 1$ **then**
      $U_L = \check{U}_L$
    else if $\hat{U}_L > 0$ **and** $\hat{U}_L <= 1$ **then**
      $U_L = \hat{U}_L$
    else
      Return ("not schedulable")
    end if
    $U = U + U_L$
    Compute x
  end if
end for
if $U > 1$ **then**
  Return ("not schedulable")
else
  Return ("schedulable")
end if

makes it easier to systematically investigate how performance is affected by a decreasing utilization cap U_L.

In particular, we consider that the processor three cases $U_L = 1/2$, $U_L = 1/3$ and $U_L = 1/4$, i.e., where the total processor utilization is uniformly divided in 2, 3 and 4 portions respectively. As mentioned above, we assume that tasks in τ are partitioned or grouped according to some functional dependency. For the sake of comparison, however, we use a the well-known first fit decreasing (FFD) heuristic to build partitions or groups of tasks in this section. Thereby, tasks are sorted according to non-increasing utilization values — $\frac{C_i^{LO}}{T_i}$ for LO tasks or $\frac{C_i^{HI}}{T_i}$ for HI tasks.

The impact of (i) the total number of tasks, (ii) the number of HI tasks and (iii) a varying increase of HI execution demand for each HI task is investigated in Fig. 1 to Fig. 12. In each curve, the percentage of schedulable task sets that could be found by the different algorithms is shown on the y-axis for a varying $U_{LO}^{LO} + U_{HI}^{LO}$ on the x-axis. Each data-point was obtained by randomly generating 1000 task systems and testing each for schedulability according to the corresponding algorithms. To this end, we made use of the algorithm *UUniFast* [24] to generate valid utilization values.

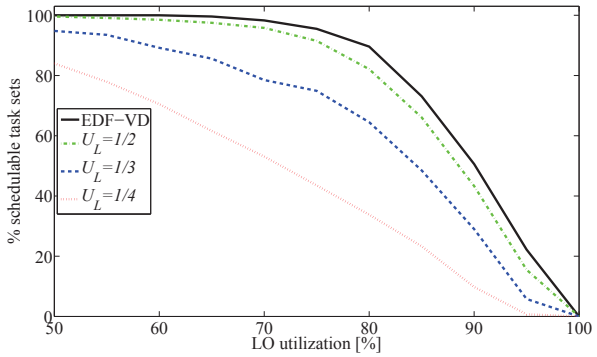

Fig. 1. Schedulability vs. LO utilization for 10 tasks, 50% of HI tasks, 100% increase of execution demand by HI tasks in HI mode

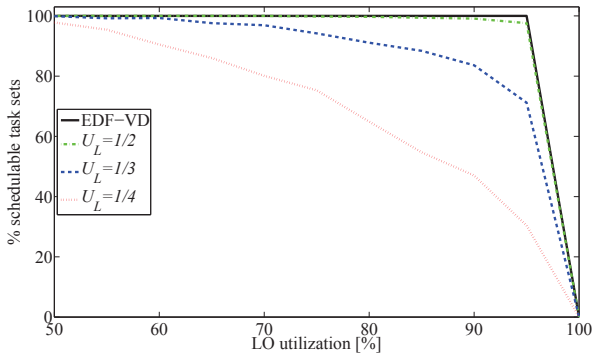

Fig. 2. Schedulability vs. LO utilization for 10 tasks, 10% of HI tasks, 100% increase of execution demand by HI tasks in HI mode

A. 10 task per task set

Fig. 1 to Fig. 4 show the results of our experiments for 10 tasks per set and different comparison conditions. We vary the number of HI tasks in each task set from 50%, i.e., 5 tasks, in Fig. 1 and 3 to 10%, i.e., 1 task, in Fig. 2 and 4. In addition, we vary the amount of execution demand from 100% in Fig. 1 and Fig. 2, i.e., HI tasks double their execution demand in HI mode, to 10% in Fig. 3 and Fig. 4, i.e., HI tasks have an increase of 10% more execution demand in HI mode.

As it can be seen, the smaller the value of U_L, the worse the algorithm's performance, i.e., less task sets will be rendered schedulable. In other words, $U_L = 1/3$ and $U_L = 1/4$ have the the worst performance compared to $U_L = 1/2$. On the other hand, $U_L = 1/2$ has a performance that is close to that of the original EDF-VD. This means that we can safeguard half of the LO tasks from being discarded in HI mode without significantly reducing the total *usable* utilization on the processor.

B. 20 task per task set

Fig. 5 to Fig. 8 show the results of our experiments for 20 tasks per set and different comparison conditions. We vary the number of HI tasks in each task set from 50%, i.e., 10 tasks, in Fig. 5 and 7 to 10%, i.e., 2 tasks, in Fig. 6 and 8. The amount of execution demand was varied from 100% in Fig. 5 and 6, i.e., HI tasks double their execution demand in HI mode, to 10% in Fig. 7 and Fig. 8, i.e., HI tasks have an increase of 10% more execution demand in HI mode.

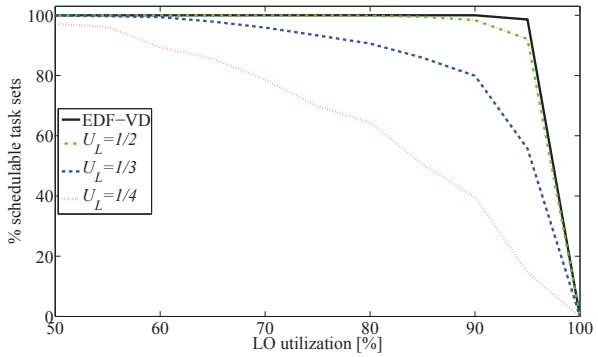

Fig. 3. Schedulability vs. LO utilization for 10 tasks, 50% of HI tasks, 10% increase of execution demand by HI tasks in HI mode

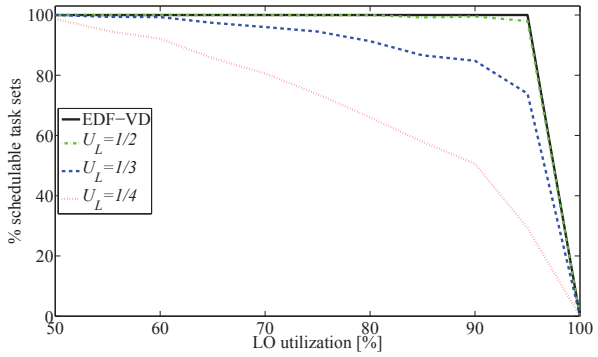

Fig. 4. Schedulability vs. LO utilization for 10 tasks, 10% of HI tasks, 10% increase of execution demand by HI tasks in HI mode

Again, the smaller the value of U_L, the worse the algorithm's performance in terms of schedulable task sets. This time, however, $U_L = 1/3$ and $U_L = 1/4$ are closer to the performance of $U_L = 1/2$ and of EDF-VD, being $U_L = 1/4$ still the one with the worst performance among all. The amount of *usable* utilization on the processor is not so drastically impacted as in the case of 10-task sets. The exception is Fig. 5, where the total HI execution demand is much higher than for Fig. 6 to Fig. 8.

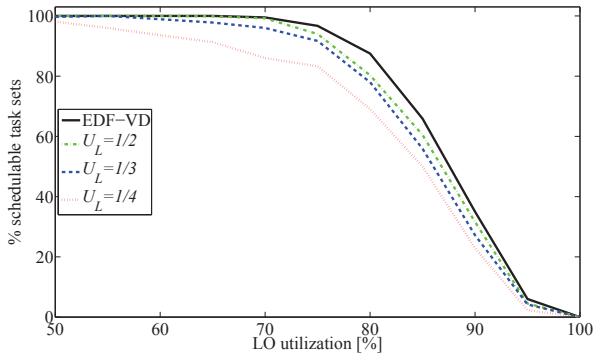

Fig. 5. Schedulability vs. LO utilization for 20 tasks, 50% of HI tasks, 100% increase of execution demand by HI tasks in HI mode

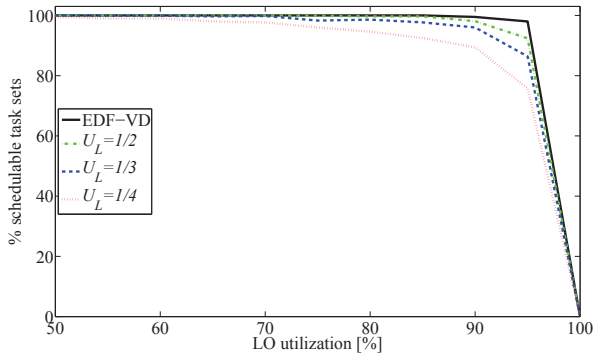

Fig. 6. Schedulability vs. LO utilization for 20 tasks, 10% of HI tasks, 100% increase of execution demand by HI tasks in HI mode

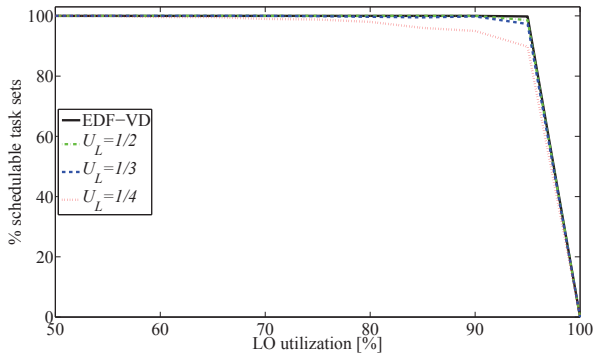

Fig. 7. Schedulability vs. LO utilization for 20 tasks, 50% of HI tasks, 10% increase of execution demand by HI tasks in HI mode

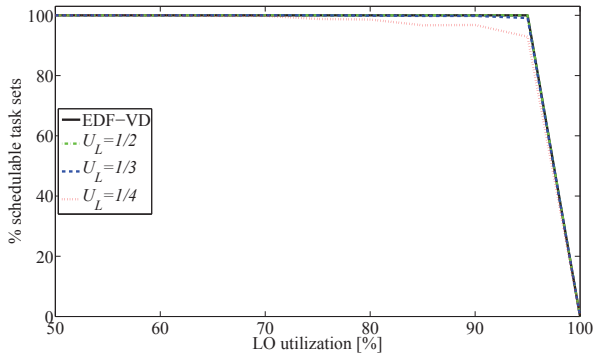

Fig. 8. Schedulability vs. LO utilization for 20 tasks, 10% of HI tasks, 10% increase of execution demand by HI tasks in HI mode

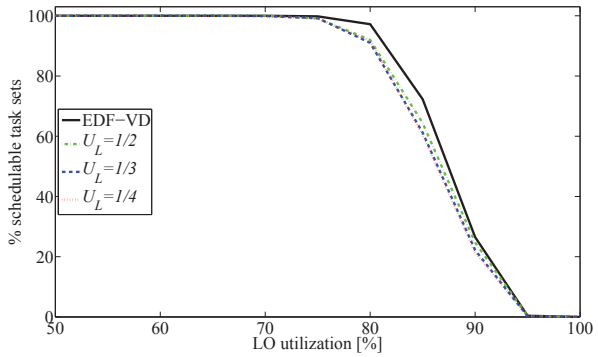

Fig. 9. Schedulability vs. LO utilization for 50 tasks, 50% of HI tasks, 100% increase of execution demand by HI tasks in HI mode

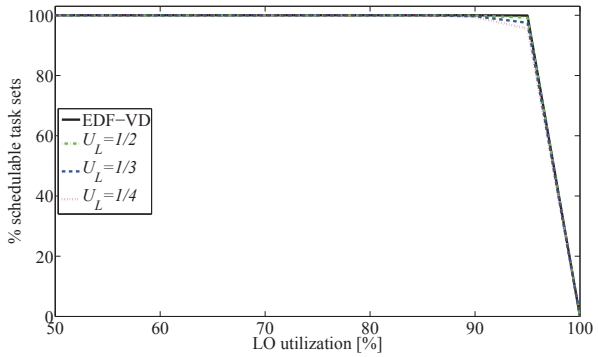

Fig. 10. Schedulability vs. LO utilization for 50 tasks, 10% of HI tasks, 100% increase of execution demand by HI tasks in HI mode

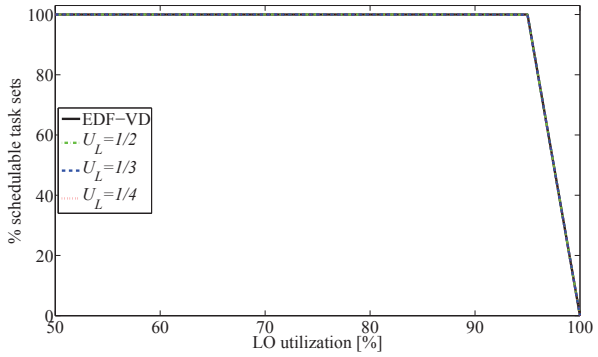

Fig. 11. Schedulability vs. LO utilization for 50 tasks, 50% of HI tasks, 10% increase of execution demand by HI tasks in HI mode

C. 50 task per task set

Similar to the case of 10-task and 20-task sets, Fig. 9 to Fig. 12 show the results of our experiments for 50 tasks per set and different comparison conditions. We vary the number of HI tasks in each task set from 50%, i.e., 25 tasks, in Fig. 9 and 11 to 10%, i.e., 5 tasks, in Fig. 10 and 12. The amount of execution demand was varied from 100% in Fig. 9 and 10, i.e., HI tasks have twice their LO execution demand in HI mode, to 10% in Fig. 11 and Fig. 12, i.e., HI tasks have an increase of 10% more execution demand in HI mode.

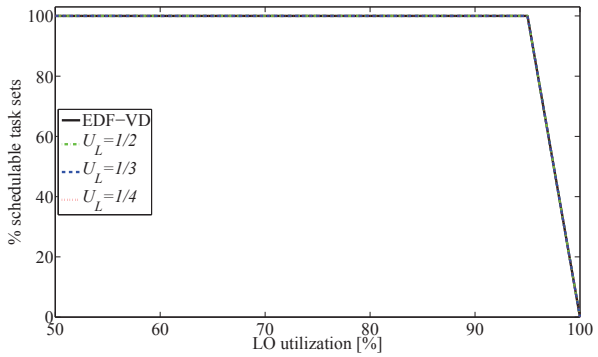

Fig. 12. Schedulability vs. LO utilization for 50 tasks, 10% of HI tasks, 10% increase of execution demand by HI tasks in HI mode

We can see that, this time, all algorithms behave almost the same. Only in Fig. 9, where the greatest HI execution demand is considered, they still have some difference in the number of task sets they render schedulable. In Fig. 10 to 12, algorithms' performance curves almost overlap fully showing a negligible difference with respect to EDF-VD.

VII. Concluding Remarks

In this paper, we proposed introducing *utilization caps* to the original EDF-VD algorithm. To this end, an MC task set is partitioned into disjoint subsets, each of which is assigned a portion of the total processor utilization. This approach is similar to using servers, i.e., virtual machines, however, in contrast to them, it has advantage of not incurring starvation periods or additional context switches — which can easily jeopardize performance.

EDF-VD is then applied to each such subset or partition independently. As a result, LO tasks within one partition are not affected by HI tasks from other partitions in case that the latter switch to HI mode. On the contrary, HI tasks can only cause the abortion of LO task within their own partition. This allows LO tasks in partitions not affected by HI mode to continue running without being degraded.

Clearly, partitions need to follow some functional criteria. For example, LO tasks that are not necessary anymore when a given HI task switches to HI mode should logically belong to the same partition as the corresponding HI task. Similarly, LO tasks which should not be affected by a given HI task should be put in a different partition.

As expected, there is a performance degradation with respect to EDF-VD. The smaller the utilization cap of one partition, the less performance in terms of schedulable task sets can be achieved. However, out experiments indicate that, on average, dividing the processor into two halves has almost no performance degradation with respect to EDF-VD (with the processor fully dedicated). This already allows for some LO tasks to be protected from switching to HI mode enabling for more design flexibility.

Acknowledgment

Mitra Mahdiani was funded by the German Academic Exchange Service (DAAD).

References

[1] R. Palin, D. Ward, I. Habli, and R. Rivett, "ISO 26262 safety cases: Compliance and assurance," in *Proc. of System Safety Conf.*, 2011.

[2] J. Rushby, "New challenges in the certification for aircraft software," in *Proc. of Conf. on Embedded Software (EMSOFT)*, 2011.

[3] 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.

[4] P. Huang, G. Giannopoulou, N. Stoimenov, and L. Thiele, "Service adaptions for mixed-criticality systems," Computer Engineering and Networks Laboratory, ETH Zurich, Tech. Rep., 2013.

[5] J. Ren and L. T. X. Phan, "Mixed-criticality scheduling on multiprocessors using task grouping," in *Proc. of Euromicro Conference on Real-Time Systems (ECRTS)*, July 2015.

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

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

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

[9] ——, "An extended fixed priority scheme for mixed criticality systems," in *Proc. of Workshop on Real-Time Mixed Criticality Systems (ReTiMics)*, Aug. 2013.

[10] A. Burns and R. Davis, "Adaptive mixed criticality scheduling with deferred preemption," in *Proc. of Real-Time Systems Symposium (RTSS)*, Dec. 2014.

[11] 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.

[12] 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.

[13] ——, "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] 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.

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

[18] P. Huang, P. Kumar, G. Giannopoulou, and L. Thiele, "Run and be safe: Mixed-criticality scheduling with temporary processor speedup," in *Proc. of Design, Automation and Test in Europe (DATE)*, March 2015.

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

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

[21] J. Lee, K.-M. Phan, X. Gu, A. Easwaran, I. Shin, and I. Lee, "MC-Fluid: Fluid model-based mixed-criticality scheduling on multiprocessors," in *Proc. of Real-Time Systems Symposium (RTSS)*, 2014.

[22] S. Baruah, A. Easwaran, and Z. Guo, "MC-Fluid: Simplified and optimally quantified," in *Proc. of Real-Time Systems Symposium (RTSS)*, 2015.

[23] 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.

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