TECHNISCHE UNIVERSITÄT
CHEMNITZ

Advanced Scheduling Techniques for Mixed-Criticality Systems

Dissertation
zur Erlangung des akademischen Grades

Dr.-Ing.

Frau M.Sc. Mitra Mahdiani
geboren am 30.April 1983 in Kordkoy (Iran)

Fakultät für Informatik
an der Technischen Universität Chemnitz

Acknowledgments

First and foremost, I would like to express my special appreciation and sincere gratitude to my advisor Professor Alejandro Masrur for the continuous support of my Ph.D study and research, for all his unfailing patience, motivation, enthusiasm, and immense knowledge. Conducting the academic study regarding such a difficult topic couldn't be as simple as he made this for me. His guidance helped me in all the time of research and writing of this thesis. I am very fortunate to have him as my supervisor, you have been a tremendous mentor for me. I would like to thank you for encouraging my research and for allowing me to grow as a research scientist. Your advice on both research as well as on my career have been priceless. Also, I would like to express my deepest gratitude to Professor Matthias Werner for accepting to be my second supervisor, thank you for your valuable time, co-operation, and generosity which set this work possible as it is till the end. Your support has been the most profitable experience for me.

To all my colleagues, I am deeply and forever grateful for the stimulating discussions, for their friendship and support, and their unconditional help whenever I needed it and, of course, for creating a cordial, pleasant and friendly working atmosphere, for all the good times, laughter and fun we have had in the last four years. I would also like to thank all of my friends who supported me in writing, and incented me to strive towards my goal.

On a more personal level, there are several people that deserve acknowledgment. Last but not least, I would like to thank my family for their continuous and unparalleled love, unconditional and endless support, both financially and emotionally throughout my doctorate, and my life in general. In particular, the patience and understanding shown by my mother and father during all these years is greatly appreciated. I am forever indebted to my dear parents for giving me the opportunities and experiences that have made me who I am. They selflessly encouraged me to explore new directions in life and seek my own destiny. This journey would not have been possible if weren't for them, and I dedicate this milestone to them.

Finally, I gratefully acknowledge the support by the DAAD for helping and providing the funding for the work at the beginning and during four and a half years of my Ph.D.

All of you have been there to support me for my Ph.D. Thanks for your encouragement!

Chemnitz, November 2019

To my parents.

"It Always Seems Impossible Until It Is Done."
— Nelson Mandela

Abstract

Typically, a real-time system consists of a controlling system (i.e., a computer) and a controlled system (i.e., the environment). Real-time systems are those systems where correctness depends on two aspects: i) the logical result of computation and, ii) the time in which results are produced. It is essential to guarantee meeting timing constraints for this kind of systems to operate correctly. Missing deadlines in many cases — in so-called hard real-time systems — is associated with economic loss or loss of human lives and must be avoided under all circumstances.

On the other hand, there is a trend towards consolidating software functions onto fewer processors in different domains such as automotive systems and avionics with the aim of reducing costs and complexity. Hence, applications with different levels of criticality that used to run in isolation now start sharing processors. As a result, there is a need for techniques that allow designing such mixed-criticality (MC) systems — i.e., real-time systems combining different levels of criticality — and, at the same time, complying with certification requirements in the different domains.

In this research, we study the problem of scheduling MC tasks under EDF (Earliest Deadline First) and propose new approaches to improve scheduling techniques. In particular, we consider that a mix of low-criticality (LO) and high-criticality (HI) tasks are scheduled on one processor. While LO tasks can be modeled by minimum inter-arrival time, deadline, and worst-case execution time (WCET), HI tasks are characterized by two WCET parameters: an optimistic and a conservative one.

Basically, the system operates in two modes: LO and HI mode. 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 more than their conservative execution budgets. In this case, LO tasks are immediately discarded so as to be able of accommodating the increase in HI execution demand. We propose an exact test for mixed-criticality EDF, which increases efficiency and reliability when compared with the existing approaches from the literature. On this basis, we further derive approximated tests with less complexity and, hence, a reduced running time that makes them more suitable for online checks.

List of Acronyms

ABS	Anti-lock Braking System
AMC	Adaptive Mixed Criticality
BF	Best Fit
CBEDF	Criticality-Based Earliest Deadline First
CO	Carry-Over execution demand
dbf	Demand Bound Function
DM	Deadline Monotonic
ECDF	Earliest Carry-over Deadline First
ECU	Electronic Control Unit
EDF	Earliest Deadline First
EDF-VD	Earliest Deadline First with Virtual Deadline
FCFS	First Come First Served
FF	First Fit
FFD	First Fit Decreasing
FIFO	First In First Out
FP	Fixed Priority
fpEDF	fixed-priority EDF
GA	Genetic Algorithm
HI	High criticality
HPA	Heuristic Priority Assignment
LO	Low criticality
LSF	Least Laxity First
LSTF	Least Slack Time First
MC	Mixed Criticality
MCS	Mixed-Criticality System
NP	Nondeterministic Polynomial
OCBP	Own Criticality-Based Priorities
OPA	Optimal Priority Assignment
PC	Personal Computer

QPA	Quick convergence Processor-demand Analysis
RM	Rate Monotonic
RR	Round Robin
RTA	Response-Time Analysis
SJF	Shortest Job First
SJN	Shortest Job Next
SRTF	Shortest Remaining Time First
TT	Time Triggered
WCET	Worst-Case Execution Time
WCRT	Worst-Case Response Time
WF	Worst Fit

Contents

Chapter 1.

Introduction

A computer-based system that is embedded within the larger system is called an embedded system. An embedded system periodically executes or controls a specific function and, in contrast to a personal computer (PC), is not designed to be programmed by the user [53] [95]. Nowadays, our surroundings are full of embedded systems such as mobile phones, pacemakers, hearing aids, refrigerators, toothbrushes, cars etc.

The importance of embedded systems is such that 98% of all manufactured microprocessors are used in embedded systems and only 2% of them are applied to general-purpose computers and devices [41] [93]. In 2008, the number of available embedded microprocessors was estimated about 30 microprocessors per person in developed countries [41]. It is interesting to notice that, based on further research, this number of processors will be increased to 100 processors per each person by early 2020, with more than 50 billions connected devices [1] [78].

According to the Embedded Systems Institute in the Netherlands (ESI) [2], there are six groups of embedded systems that are categorized as automotive systems, consumer electronics (such as domestic appliances or mobile devices), infrastructure systems (i.e., public or industrial systems with a priority on efficient resource management and control of safety-critical actions), avionics, professional systems (i.e., systems applied for work-related purposes such as medical equipment) and defense systems.

Regardless of their category, since embedded systems interact with their environment, they usually need to comply with real-time requirements or deadlines and are, hence, often regarded as real-time systems. A real-time system is one in which the correctness of the computations not only depends on their logic, but also on the time at which the result is produced. In other words, a system is real-time if its computations/tasks need to be completed on time. This makes scheduling an important aspect, for which there are already solutions in the literature.

On the other hand, most existing solutions for scheduling disregard the fact that not all deadlines are equally critical. Moreover, in many domains such as automotive systems, avionics and medical devices, embedded software has to pass a strict certification process according to different *criticality levels* [73] [85]. Clearly, the certification of *high-criticality* (HI) tasks is more rigorous and well-regulated than that of *low-criticality* (LO) tasks and, hence, LO tasks are usually more error-prone than HI tasks. Although aerospace and automotive safety regulations define around five levels of criticality, only two levels are considered in this dissertation for the ease of exposition. However, the proposed techniques remain valid for multiple levels of criticality.

The above is of particular concern when consolidating software functions onto fewer processing units, since functions or tasks with different levels of criticality start being executed on common hardware platforms. This allows for a reduction of costs and complexity, however, it leads to mixed-criticality (MC) systems that require careful design and analysis. It may happen that failures of any kind in a low-criticality (LO) tasks affect one or more high-criticality (HI) functions/tasks.

As a result, 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. For instanse, modern cars consist of 50 to 100 ECUs (Electronic Control Units) [68] with even more ECUs needed to perform more advanced and complex functionalities [35], further increasing costs and weight in the car. Moreover, due to complex network between those ECUs, guaranteeing deadlines becomes more difficult. In order to reduce the complexity and difficulty of scheduling, there is a trend towards replacing most ECUs with fewer but powerful processing units, i.e., towards consolidating software on fewer ECUs. However, this also implies that different tasks of different criticalities are integrated to share one processer [35], leading to mixed-criticality (MC) system.

The focus of this research is on scheduling techniques for mixed-criticality, real-time systems as described above. In particular, this thesis is concerned with a set of low-criticality (LO) and high-criticality (HI) tasks that share one processor and are scheduled under the Earliest Deadline First (EDF) algorithm. While LO tasks can be modeled by minimum inter-arrival time, deadline, and worst-case execution time (WCET), HI tasks are characterized by two WCET parameters: an optimistic and a conservative one. Conservative WCET parameters normally result from deterministic analysis and largely overestimate the actual WCET [98]. On the other hand, optimistic WCET parameters can be obtained by probabilistic or statistical methods [59] [89] and are often sufficient to guarantee a correct operation in the most cases.

Basically, the system implements two operation modes: LO and HI mode. In LO mode, HI tasks run for their optimistic WCETs and are scheduled within *virtual deadlines* together with all LO tasks. Virtual deadlines are given by $x_i \cdot D_i$ and are usually shorter than real deadlines D_i with $x_i \in (0, 1)$ being referred to as *deadline scaling factor*. A switch to HI mode occurs when one or more HI tasks require running for longer than its optimistic WCETs (but still less than its conservative one). HI tasks are then scheduled within their *real deadlines* and LO tasks are stopped from running in HI mode, i.e., LO tasks are immediately degraded or even discarded, which then allows accommodating this increase in HI execution demand.

In this setting, referred to as mixed-criticality EDF, it is necessary to guarantee schedulability of transitions between modes and not of only individual modes in isolation. This then reduces to finding valid x_i for each of the HI tasks in the system. So far, there have been different approaches to this, which are usually based on approximating the execution demand by MC task sets [8] [42] [40]. In particular, since a switch from LO to HI mode may occur at an arbitrary point in time, it remains difficult to accurately bound the execution demand by so-called *carry-over jobs*, i.e., HI jobs that have been released before but have not finished executing at the point in time of switching. As a result, the known demand bounds for mixed-criticality EDF tend to be pessimistic.

1.1. Motivation

As already mentioned, in safety-critical domains such as automotive systems, avionics, and medical engineering, there is increasingly important trend towards integrating functions with different levels of criticality onto a common hardware platform with the aim of reducing costs and complexity. This leads to mixed-criticality (MC) systems where applications with different levels of criticality start sharing processors. However, these require careful design and analysis, since it may happen that failures of any kind in a low-criticality (LO) task affect one or more high-criticality (HI) task. As a result, there is a need for techniques that allow designing such MC systems — i.e., real-time systems combining different levels of criticality — and, at the same time, complying with certification requirements in the different domains.

MC scheduling is challenging in terms of the ratio between accuracy and complexity. That is, existing approaches are either too pessimistic or they incur to much complexity [8] [40] [42]. As a consequence, the purpose of this thesis is to contribute to improving accuracy while keeping a low complexity in testing schedulability of MC systems based on the EDF algorithm. The contributions achieved by this work are stated in the following section.

1.2. Contributions

In this research, we study the problem of scheduling MC tasks under the EDF scheduling algorithm and propose new approaches to improve scheduling techniques. We present different approximated and exact tests for mixed-criticality EDF, which increase efficiency and reliability, and compare them with the existing approaches from the literature.

In particular, as mentioned above, we consider that a mix of low-criticality (LO) and high-criticality (HI) tasks are scheduled on one processor— however, a discussion for more levels of criticality is presented in the Appendix A.1. While LO tasks can be modeled by minimum inter-arrival time, deadline, and worst-case execution time (WCET), HI tasks are characterized by two WCET parameters: an optimistic and a conservative one.

The system implements two operation modes, LO and HI mode. In LO mode, HI tasks execute for no longer than their *optimistic* execution budgets and are scheduled together with the LO tasks. The system switches to HI mode, where all LO tasks are prevented from running, when one or more HI tasks run for longer than expected (run for their conservative execution budgets). In this setting, the contributions by this work can be summarized as follows:

- **Introducing utilization caps:** We introduce utilization caps to the original EDF-VD (Earliest Deadline First with Virtual Deadlines) algorithm [10], which is typically used for scheduling MC tasks. To this end, the 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.

- **Better exact schedulability test:** The approach based on utilization caps is very simple and easy-to-use. However, it results in a sufficient but not necessary test, i.e., it leads to a suboptimal use of processor resources. To counteract this, we propose a better, exact test for mixed-criticality EDF. Clearly, the proposed exact test comes at the cost of an increased complexity.

  The idea is to better bound execution demand under mixed-criticality EDF. To this end, we derive a separate demand bound function for transitions from LO to HI mode and prove its validity. This technique allows us to work around the computation of carry-over execution demand and, hence, to reduce the amount of pessimism in characterizing mixed-criticality EDF. We further proved that the proposed technique leads to a tighter bound on the execution demand under mixed criticality EDF in most relevant cases.

It is interesting to notice that the proposed technique reduces the problem of testing schedulability under mixed criticality EDF to testing schedulability of three almost unrelated task sets: the one in LO mode, the one in HI mode and the equivalent task set for transitions between LO and HI mode. This leads to a considerably simpler schedulability test and improves our understanding of this problem.

- **Better approximated schedulability tests:** Since testing schedulability for mixed-criticality EDF boils down to testing three separate task sets under standard EDF, we are able to extend and apply known approximation techniques from the literature (originally conceived for standard EDF).

  In particular, we extend the so-called Devi's test to be used in the context of MC systems. This extension of Devi's test is also sufficient but not necessary, however, it is more accurate than using utilization caps (our first approach) and is considerably faster than any exact test (including the one proposed in this thesis). This represents a good trade-off between accuracy and running time relevant in online settings, e.g., admission control.

1.3. Structure of this Thesis

This thesis is basically structured into three main chapters dealing with the most important contributions apart from some accessory chapters as detailed next.

Chapter 2 briefly introduces concepts, models and assumptions required for the rest of this dissertation. In turn, Chapter 3 is concerned with related work and provides a detailed literature review regarding the scheduling of mixed-criticality systems. It especially covers existing approaches that form the background of presented methods including a discussion about their advantages, disadvantages, and differences.

In Chapter 4, 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 of the total utilization. EDF-VD then applies to each of these partitions individually and up to their corresponding utilization caps. As briefly explained above, if one HI task exceeds its execution budget in LO mode, this only affects the LO tasks in the same partition and LO tasks in other partitions can continue running. We present a technique to optimally choose utilization caps for each partition.

Chapter 5 presents a technique that works around the computation of carry-over execution demand and results in a more accurate bound on execution demand under mixed-criticality EDF. In principle, the proposed technique consists in separating the schedulability analysis of *stable* HI mode from that of the *transition* between modes and deriving a separate demand bound function for the latter case. The proposed technique results not only in a considerably simpler, but also tighter bound on execution demand under mixed-criticality EDF, in particular, as the number of HI tasks increases.

A transition from LO to HI mode is feasible, if an equivalent task set derived from the original is schedulable under plain EDF. On this basis, as illustrated in Chapter 6, we can apply approximation techniques such as, e.g., the well-known Devi's test to derive further tests that trade off accuracy versus complexity/runtime.

In Chapter 7, we evaluate the benefits and drawbacks of the proposed approaches and perform a large set of experiments based on synthetic data illustrating the performance by the proposed techniques and comparing them to the most prominent approaches from the literature. The intention is to show how the different algorithms behave with respect to each other and not to provide any absolute performance metrics as detailed later.

Chapter 8 finally concludes this thesis by discussing and summarizing the main contributions, putting them into perspective, and presenting concluding remarks as well as potential future work.

Chapter 2.

Concepts, Models and Assumptions

In this chapter, we first introduce important concepts that are necessary to understand the remainder of this thesis. Based on this concepts, we then discuss models used and assumptions made in this work.

2.1. Real-Time Systems

Generally, real-time systems are those whose correctness depends on two aspects: i) the logical result of computations and, ii) the time in which results are produced. A real-time system consists of a controlling system (i.e., a computer or processor) and a controlled system (i.e., environment). Real-time systems play a crucial role in our everyday lives and in industry, since many complex systems depend on computer control (partly or entirely). Some of the application domains that require real-time computing are automotive systems, avionics, multimedia systems, virtual/augmented reality, etc.

A real-time systems requires a correct scheduling policy to provide a schedule that guarantees the execution of all tasks (or jobs) within their timing constraints where the worst-case execution times (WCETs) and timing constraints of tasks are the important concepts of scheduling. The WCET of a task (or job) is the maximum length of time (or an upper bound on the execution time) the task or job needs to execute on a specific hardware platform [31].

A timing constraint on a task is known as its deadline. A deadline with respect to the task arrival time is called a relative deadline. An absolute deadline is a deadline which is specified with respect to time zero. Typically, based on the consequences of a missed deadline, real-time systems are divided into two types: hard and soft real-time systems. This classification depends, particularly, on the functional criticality of tasks (jobs), usefulness of late results, and deterministic or probabilistic nature of the constraints [31].

A hard real-time computer system or a safety-critical computer system must meet its timing constraint or deadline, otherwise, disastrous consequences may occur, i.e., any missed deadline may cause a harmful system failure. A few systems have this requirement such as avionics, nuclear systems, medical applications like pacemakers and defense applications among others [58] [67].

A real-time computer system is called soft real-time, if no hard deadline exists (i.e., it can miss some deadlines). In other words, a missed deadline in a soft real-time system does not result in the system failure or serious harm, but rather causes a performance degradation. Examples of soft real-time systerms are video games, audio/video streaming, web browsers, etc. [31] [58] [67].

2.1.1. Tasks Models

Real-time systems consist of a set of tasks to be scheduled that can be classified basically into sporadic, periodic and aperiodic tasks, based on the type of the tasks.

Sporadic Task Model

A piece of code that executes repeatedly is a sporadic task. A sporadic task generates an infinite sequence of jobs with a minimum inter-arrival time T_i (informally also refered to as period), has a worst-case execution time C_i and, relative deadline D_i. In a sporadic task, each job arrives at an unpredictable time, but at least T_i time units from the last job of the same task [25].

A sporadic task system τ contains a set of such sporadic tasks which can generate many distinct job sequences due to the various inter-arrival times. In this thesis, our systems are modeled as a set of sporadic tasks.

Periodic Task Model

In periodic task systems, jobs are always released at exact points in time such that the inter-arrival time T_i (i.e., the period) between any two consecutive jobs is constant. In contrast to sporadic tasks, a set of periodic tasks generates a much reduced number of distinct job sequences.

Aperiodic Task Model

Aperiodic tasks are those that have no period of repetition, i.e., they generate only one job and that at an arbitrary point in time. Whereas periodic tasks are an idealization of sporadic tasks and these latter are related to control applications (which need to periodically sample data and compute outputs), aperiodic tasks are related to exceptional or isolated events that are not intended to repeat over time.

2.2. Scheduling Policies

Scheduling for real-time systems has been a very active research area over the last several decades. As a result, there exist already multiple approaches that we try to survey in this section.

Each unit of work that is scheduled and executed by the system is called a job and a set of related jobs which provide some system function is called a task [67]. When a set of tasks has to be executed simultaneously on a processor — i.e., tasks that can overlap in time — the processor has to be assigned to various tasks according to a predefined criterion which is called a scheduling policy [31].

A scheduling algorithm is the set of rules that determines the order in which tasks are executed. Scheduling algorithms of real time tasks can be divided into two categories: i) preemptive scheduling and ii) non-preemptive scheduling. In preemptive scheduling the low priority running task can be interrupted at any time for a short period of time (even though the running task has not yet completed) to assign the processor to another task with higher priority. Algorithms based on preemptive scheduling are: Round Robin (RR), Shortest Remaining Time First (SRTF) and priority-based algorithms such as Rate Monotonic (RM), etc. [31].

In a non-preemptive scheduling algorithm, the running task is executed until completion. It cannot be suspended by a new task, in other words, we cannot preempt a running task, i.e., take control of the CPU or processor to run some other processes. Non-preemptive scheduling algorithms include: First In First Out (FIFO), First Come First Served (FCFS), Shortest Job Next (SJN), which is also known as Shortest Job First (SJF), but also priority-based algorithms can be non-preemptive [31] [74].

A feasible schedule is one in which all tasks can be accomplished according to a set of timing constraints. A set of tasks system is said to be schedulable, if at least one scheduling algorithm exists that can produce a feasible schedule [31] [74] [90].

In turn, priority-based scheduling algorithms can be categorized into static and dynamic algorithms. A scheduling algorithm is called static, if a feasible schedule is computed off-line. In other words, in static algorithms, the scheduling decisions based on fixed parameters (e.g., deadlines, precedence constraints and maximum execution time) are assigned to tasks before their activation, where the tasks priorities are fixed and known a priori, and never change at run-time. Such algorithms are also known as fixed-priority (FP) algorithms [67] [58].

A well-known fixed-priority algorithm is the Rate Monotonic (RM) [66], in which priorities are assigned to tasks based on their periods: the shorter the period, the higher the priority of a task. Another well-known fixed-priority algorithm is the Deadline Monotonic (DM). In this latter, priorities are allocated to tasks according their relative deadlines: the shorter the relative deadline, the higher the priority [67] [58].

In contrast, a scheduling algorithm is called dynamic (or on-line), if a feasible schedule is (or scheduling decisions are) determined at run time, that is, systems with dynamic priorities can change at run-time based on varying properties. In other words, in a dynamic scheduling algorithm, the priorities are determined during the execution of the system. Dynamic priority scheduling algorithms are flexible and allow adapting to evolving task scenarios. On the other hand, it is much difficult to verify the correctness of a schedule based on such algorithms [56] [67].

Examples of dynamic priority scheduling algorithms include the Earliest Deadline First (EDF) algorithm and the Least Slack Time First (LSTF) algorithm, which is also known as Least Laxity First (LSF). Under EDF, the task with the next upcoming absolute deadline has the highest priority, in other words, the earlier the deadline of a task, the higher the priority assigned to that task. The absolute deadline of a job is the release time of the job plus its relative deadline [58] [74].

Similarly, under LSTF, priorities are assigned based on the slack time of a process. The task's slack time is the difference between the current time value to the deadline of a task, i.e., this is the remaining time a task has to complete execution. In this scheduling algorithm, processes with the smallest slack time will be executed first, i.e., the smaller the slack time of a task, the higher the priority value assigned to a task [58] [74].

2.2.1. Feasibility versus Schedulability

Scheduling is concerned with allocating resources to tasks in a timely manner in such a way that specific constraints like communication, synchronization, timing, etc., are met. A successful scheduling algorithm or policy selects which task executes at each time instant resulting in a schedulable or valid schedule.

The difference between feasibility and schedulability is very subtle such that, most of the time, these two terms are used interchangeably. To understand this, let us first introduce the notion of optimal scheduling algorithm on a single-processor preemptive system. That is a scheduling algorithm that always finds a valid (i.e., a scheulable) schedule, if the task set is feasible on one processor. This implies also that there are algorithms that are non-optimal and may not find a valid schedule for some set of preemptive tasks, although this is feasible on one processor. The most prominent example of an optimal scheduling algorithm is EDF [37]. Similarly, an example of a non-optimal scheduling algorithm is RM [66].

Now, a schedule is feasible when jobs (generated by tasks) execute for their specified worst-case execution times and within their timing constraints. Further a task set is called feasible when there exists a feasible schedule, i.e., there exists a scheduling algorithm that generates a feasible schedule for the task set. Similarly, we say that a task set is schedulable by a given scheduling algorithm, if that particular scheduling algorithms is able to generate a feasible schedule for the task set in question.

As a consequence, it may happen that a task set is feasible and, hence, schedulable on one processor by an optimal scheduling algorithms like EDF, but not schedulable by an non-optimal scheduling algorithms like RM.

2.2.2. Schedulability Test

A schedulabililty test is concerned with verifying that no task (for hard real-time systems) or a minimum number of tasks (for soft real-time systems) misses their deadlines under a specified scheduling algorithm [88]. In other words, if the schedulability test is successful, tasks and, hence, their jobs are always able to meet their deadlines.

In real-time systems, various schedulability tests (i.e., exact or necessary-and-sufficient tests as well as approximate or sufficient tests) are used to verify meeting time constraints. Thereby, the complexity is generally high in exact schedulability tests making them inappropriate for on-line admission control with a large number of tasks (or dynamic workload). On the contrary, a sufficient schedulability test with low complexity can be used with this context, clearly, at the cost of a less accuracy [88].

Even if there is only one common resource, the complexity of an exact schedulability test is of NP-complete [46], therefore, it is not computationally tractable in almost all cases, but particularly when tasks exhibit dependencies among them.

Clearly, this makes exact schedulability test unsuitable for online testing. In this case, however, sufficient tests can be used instead. In real-time systems, the simplest sufficient tests are utilization-based which have polynomial complexity [101]. The schedulability of a given task set cannot be conclusively determined by a sufficient schedulability test, if the task set does not pass the test, however, passing the test means that the task set is definitely schedulable.

2.3. Mixed-Criticality Systems

As discussed above, many embedded systems are real-time systems consisting of a controlling system (i.e., a computer) and a controlled system (i.e., the environment) where correctness depends on logical results and the time in which results are produced. Systems such as digital audio players, a vehicle's cruise control and anti-lock braking system (ABS) are examples of real-time systems [7].

Further, if the failure in a system jeopardizes the environment or human life, the system is called safety-critical (e.g pacemakers or aircraft control systems) [58]. A determination of the level of assurance against failure which is required for a system component is called criticality [28]. In the later years, with the trend towards consolidating software onto fewer processors or ECUs, functionalities with different levels of criticality start coexisting on the same hardware resources. This has led to a mix of safety-critical and non-safety-critical, i.e., to mixed-criticality (MC) systems. Since different levels of criticality need to comply with different certification regulations as per existing standards [28] [29] [52], it is necessary to guarantee that more error-prone tasks/functions with low criticality do not affect those with higher criticality. The challenge lies in finding scheduling techniques that allow guaranteeing safety in MC systems, but also make an efficient use of resources.

Mixed-criticality systems consist of two or more different criticality levels. If the tasks are critical, they are classified as high-criticality (HI) tasks, and less critical tasks are called low-criticality (LO) tasks. This is a classic model for mixed-criticality systems is known as dual-criticality. This model is typically adopted in the majority of the research works in this area, since it reduces the problem to its essentials and can then be easily extended to more general cases [32].

Another model that has established over the last few years is based on criticality modes [96]. That is, as depicted in Fig. 2.1, the system switsches between LO and HI mode assuming a dual-criticality setting — in more general settings with more than two levels of criticaltiy, there will be a mode for each criticality level. In LO mode, HI and LO tasks are scheduled successfully, since HI tasks have a lesser execution demand. In HI mode, HI tasks have higher execution demand and, hence, LO tasks need to either be degraded or discarded to accommodate such additional workload.

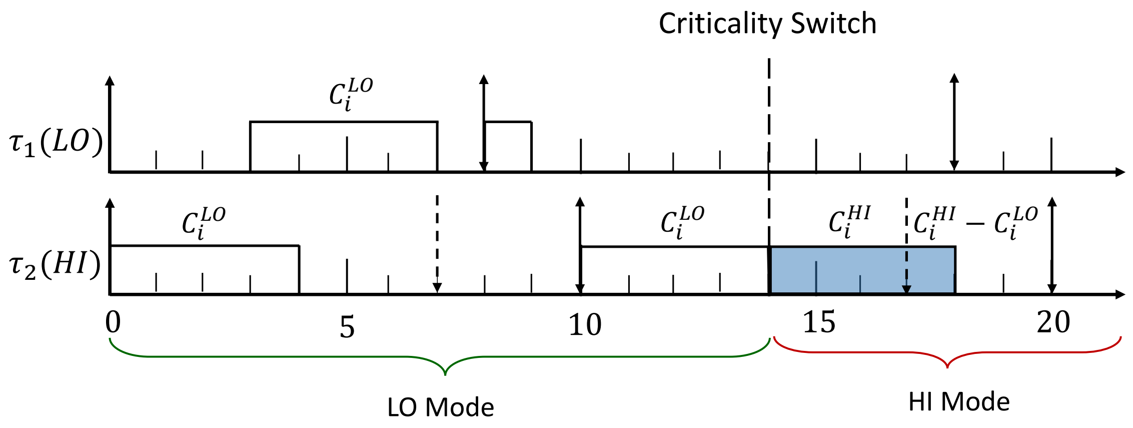
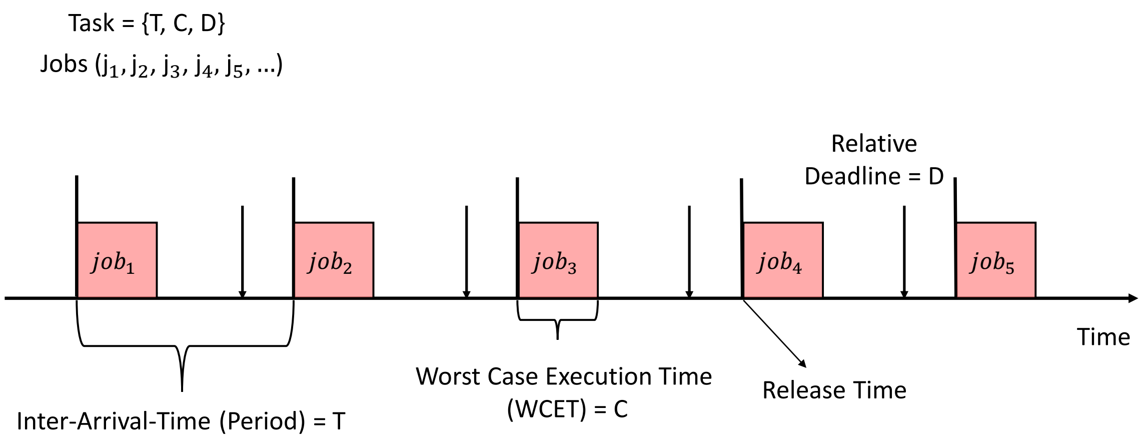


Figure 2.1.: Mode switch in mixed-criticality systems

2.4. Basic Nomenclature

In this section, we discuss most of our notation. Note that further nomenclature will be introduced as it gets necessary along this dissertation. Similar to [40] and [42], we basically adopt the task model originally proposed in [10] [96] and consider a one-processor system where a set of MC tasks are scheduled.

As depicted in Fig. 2.2, we denote by τ the set of n independent — with respect to timing — preemptable and sporadic tasks that run on one processor under preemptive EDF scheduling. Each individual task τ_i in τ is characterized by its minimum inter-release time T_i, i.e., the minimum distance between two consecutive jobs or instances of a task, and by its relative deadline D_i where a task with deadline lower or equal to its period is called a *constrained* deadline task system, i.e., $\forall i : D_i \leq T_i$ and the deadlines equal to the period are assumed as *implicit* deadlines, i.e., $\forall i : D_i = T_i$. Further, we assume that tasks do not self-suspend and that overhead by context switches is either already included in C_i or can be neglected on the different processors.



Task = {T, C, D}
Jobs (j₁, j₂, j₃, j₄, j₅, ...)
Relative Deadline = D
job₁
job₂
job₃
job₄
job₅
Time
Worst Case Execution Time (WCET) = C
Release Time
Inter-Arrival-Time (Period) = T

Figure 2.2.: Parameters and timing requirements in scheduling real-time systems

Similar to most approaches in the literature, we are concerned with dual-criticality systems with two levels of criticality, namely LO and HI.[1] The *criticality* of a task i is denoted by χ_i with:

$$\chi_i \in \{LO, HI\}.$$

A LO task is associated with only one WCET value/estimate denoted by 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} with:[2]

$$C_i^{LO} < C_i^{HI} \leq D_i \leq T_i.$$

Tasks in τ are independent in the sense that they do not affect each other's execution apart from competing for resources. However, we consider that some functional dependency may 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_i with $i \in \{A, B, \ldots Z\}$ contains a mix of HI and LO tasks: If a HI task

[1]See the appendix for an extension to more than two levels of criticality.
[2]Note that either real or integer numbers can be used for C_i^{LO}, C_i^{HI}, D_i and T_i.

within this τ_i switches to HI mode, all LO tasks in τ_i will be discarded; however, LO tasks in the remaining subsets $\tau_A, \tau_B, \dots \tau_Z$ may be allowed to continue running, since they do not functionally depend on any τ_i's tasks.

Basically, the system distinguishes two operation modes denoted by m for each subset $\tau_i \subset \tau$: 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_i 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}, the system switches to HI mode where only the HI tasks are allowed to run – LO tasks are immediately discarded. Similar to context switches, we assume that the overhead by mode switches has been accounted for in C_i^{HI}.

We denote the *utilization* by LO and HI tasks in the LO and HI mode respectively as follows:

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

where again χ and m can assume values in $\{LO, HI\}$. U_χ^m indicates the processor utilization produced by tasks with criticality χ in mode m. 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/is effectively zero for a particular $\tau_i \subset \tau$, since LO tasks are dropped when a HI task in τ_i changes to HI mode and, hence, do not run in τ_i's HI mode.

Finally, in contrast to [42] and [40], we are not constrained to integer numbers, but rather use real numbers for all above parameters which gives us more flexibility in modeling MC workloads.

2.5. The Earliest Deadline First Algorithm

The already mentioned Earliest Deadline First (EDF) is a well-known dynamic-priority scheduling algorithm that schedules jobs of tasks according to their absolute deadlines and constitutes the basis of the proposed approaches in this thesis.

Whenever a scheduling event occurs (task finishes, new task released, etc.), the job or process closest to its deadline will be the next job to be scheduled. As discussed above, EDF is an optimal scheduling algorithm on one processor.

That is, the utilization bound of EDF is 100% when scheduling periodic tasks with deadlines equal to periods. Thus, the schedulability test for EDF boils down to a simple upper bound on the total utilization U and is obtained as follows:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1,$$

where C_i is the WCET of a task and T_i represents its period of repetition. Again, tasks' deadlines D_i must be equal to their corresponding peroids.

Note that, if a task set is feasible, it will be schedulable by EDF due to its optimality. In addition, EDF does not specifically make any assumption on periodicity of tasks, so it is independent of periodicity of the tasks and therefore it can be used to schedule periodic as well as aperiodic tasks [31].

In the following, an example of timing data of task set scheduled by EDF is illustrated in Table 2.1. The utilization of this task set is:

$$U = \left(\frac{1}{8} + \frac{2}{5} + \frac{4}{10}\right) = \left(\frac{37}{40}\right) = 0.925 = 92.5\%$$

Now, since this is less than 100%, the task set is surely feasible and, hence, schedulable by EDF.

Task (τ_i)	**Execution Time** (C_i)	**Deadline** (D_i)	**Period** (T_i)
τ_1	1	8	8
τ_2	2	5	5
τ_3	4	10	10

Table 2.1.: Timing data of task set

2.5.1. EDF-VD

Earliest Deadline First with Virtual Deadlines (EDF-VD) is an adaption of EDF to MC systems [10]. Its basic idea is also 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:[3]

$$U_{LO}^{LO} + U_{HI}^{LO} \quad \leq \quad 1, \tag{2.5.1}$$

[3]For the equations in Section 2.5.1, 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.

$$U_{HI}^{HI} \leq 1. \tag{2.5.2}$$

In [8], Baruah *et al.* also obtained a sufficient schedulability condition for EDF-VD in the form of a utilization bound: $\max\bigl(U_{LO}^{LO} + U_{HI}^{LO}, U_{HI}^{HI}\bigr) \leq 3/4$. They also proposed a more accurate schedulability test based on whether a scaling factor x can be obtained or not [8]. To this end, a *lower* and an *upper* bound on x are computed:

$$\frac{U_{HI}^{LO}}{1 - U_{LO}^{LO}} \leq x, \tag{2.5.3}$$

$$x \leq \frac{1 - U_{HI}^{HI}}{U_{LO}^{LO}}. \tag{2.5.4}$$

If the value of x obtained with (2.5.3) is less than or equal to the value obtained with (2.5.4), then it is possible to find a valid x for the considered system and the task system is schedulable by EDF-VD.

Finally, note that we denote by EDF-VD the original, utilization-based algorithm from [10], which we described in this section. Later, for comparison purposes, we denote by DEDF-VD (i.e., Density EDF-VD) a density-based variant of the original algorithm. In DEDF-VD, the period T_i is replaced by the deadline D_i in the above equations, i.e., instead of computing utilization, we compute density. This extension is necessary to deal with task sets where $D_i \leq T_i$ holds as the ones intended in this thesis, which the original EDF-VD algorithm does not consider.

2.5.2. Mixed-Criticality EDF

In contrast to EDF-VD, where deadlines are assumed to be equal to periods, we now allow a deadline D_i with $D_i \leq T_i$. In addition, we now assign a *virtual deadline* equal to $x_i \cdot D_i$ with $x_i \in (0, 1)$ to all τ_i with $\chi_i = HI$.

Similar to EDF-VD, this virtual deadline is used instead of D_i — the *real* deadline — to schedule HI tasks in LO mode. The parameter x_i is now a *per-task* deadline scaling factor. There is no deadline scaling for LO tasks such that they are scheduled (only in LO mode) using their D_i.

When the system switches to HI mode, HI tasks start being scheduled according to their real deadlines D_i whereas LO tasks are discarded immediately. Clearly, whereas schedulability of separated modes can be easily tested, i.e., when the system is stable in either LO or HI mode, it is difficult to test schedulability of transitions between modes. In particular, careful analysis is required when the system switches from LO to HI mode.

In this work, similar to other approaches from the literature, transitions from HI back to LO mode are disregarded. The reason is that, in contrast to changes from LO to HI, a change from HI to LO mode can be programmed or postponed to a suitable point in time, e.g., at which the processor idles after all HI tasks have run again for their optimistic WCETs, and does not require further analysis.

2.5.3. Demand Bound Function

The concept of demand bound function proposed by Baruah *et al.* is a successful approach to analyze the schedulability of real-time task systems [17], where demand bound function $\text{dbf}_i(\tau_i, t)$ is defined as the maximum possible execution demand of a task τ_i in any time interval of a given size t [43].

For many task models in the normal systems (or non-mixed-criticality systems), there are approaches to accurately compute the demand bound functions. As an example, for a standard sporadic tasks, the demand bound function for a given t can be computed in pseudo-polynomial time [17].

Demand Bounds for MC Tasks

The extension of the demand bound function to mixed-criticality systems was demonstrated by Ekberg and Yi in [43]. The idea is to introduce two demand bound functions for each mode, dbf_{LO} and dbf_{HI}, i.e., for the low and high-criticality modes respectively.

In low mode, tasks τ_i behave similar to the normal sporadic tasks, and all of their jobs are guaranteed to execute for up to their C_i^{LO}, otherwise the system switches to high mode. To this end, the typical approach can be used for computing demand bound functions for sporadic tasks [17]. In contrast to this, with dbf_{HI}, it is required to consider high-criticality *carry-over jobs* that are active at the time of switching to high-criticality mode, where low-criticality jobs are discarded [43]. The detailed description of demand bound function are provided in Chapter 5.

Chapter 3.

Related Work

In recent years, mixed-criticality (MC) scheduling has been attracting a lot of attention from both the research community [96] and the industry [7]. Up to date, the real-time community was mainly concerned with providing enough timing guarantees for a task set with different criticality levels. Scheduling MC tasks depends on different factors as depicted in Fig. 2.1. In this chapter, we briefly revise the rich literature concerning scheduling MC tasks on one processor and, in particular, under EDF — a complete overview can be found in [29].

3.1. Uniprocessor Scheduling

Scheduling mixed-criticality (MC) tasks is proven to be NP-hard even under the uniprocessor platforms [21]. Different scheduling techniques such as fixed-priority scheduling [10] [12] [50] [63] and dynamic-priority scheduling in particular, based on EDF [9] [49] [100] have been studied over the last few decades to provide efficient, approximate algorithms for scheduling MC tasks on uniprocessor platforms.

The problem around MC systems was first addressed by Vestal in [96] who proposed modeling HI tasks with multiple WCET parameters to account for potential increases in execution demand. 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.* later analyzed different priority assignments and the resulting response times under fixed priorities [12]. In [13], Baruah *et al.* presented a further priority assignment with a partially better performance than those in [12]. 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 [27].

As already stated, a mixed-criticality task model and the algorithm to schedule MC tasks was proposed by Vestal in [96], where, priorities are assigned according to DM, i.e., the shorter relative deadlines, the higher priority.

Fixed-priority (FP) scheduling algorithms have been studied in different works in order to schedule MC systems. Baruah *et al.* presented one such scheduling approach on uniprocessors with two levels of criticality, low criticality (LO) and high criticality (HI) [13]. Based on their approach, if the system is in LO mode, priorities are determined using the LO-mode WCETs. When the system mode changes to HI mode, the LO tasks are discarded and HI tasks use another priority assignment. They also showed a method for a fixed an FP assignment of periodic tasks with more than two criticality levels [12].

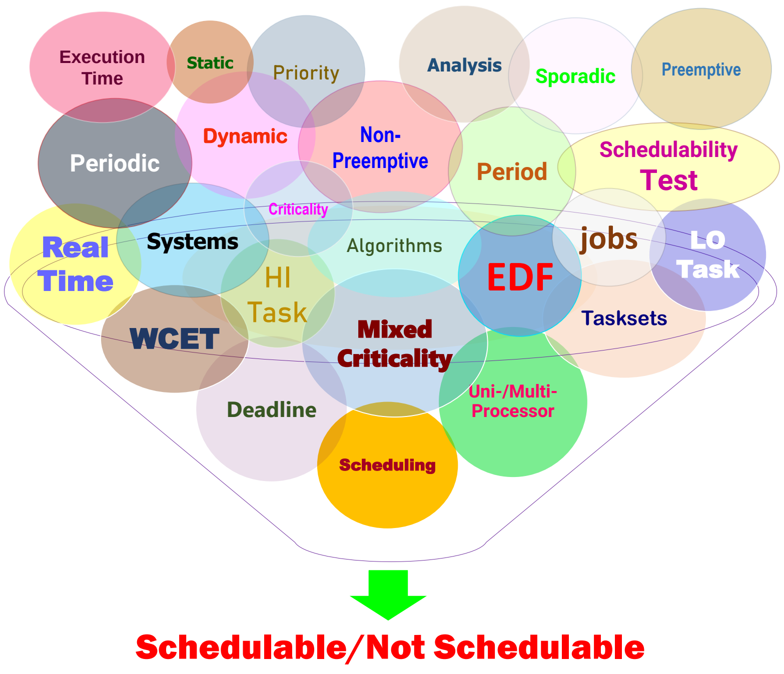


Figure 3.1.: Important Factors for Mixed-Criticality Scheduling

FP scheduling of MC systems on a uniprocessor platform was studied by Chen *et al.* in a more general way [33]. In that work, different priority sequences are used in various levels of execution. Based on that, in [12], a novel priority assignment scheme by so-called heuristic priority assignment (HPA) was presented based on Audsley's optimal priority assignment (OPA) [3] [4].

3.1.1. Uniprocessor Scheduling Based on EDF

A more complete analysis for EDF-based scheduled systems was presented by Ekberg and Yi, and Guan *et al.* in [42] and [50] to bound and shape the demand in a dual-criticality setting with sporadic tasks. The idea of their approach is to assign two relative deadlines to each HI task. One deadline is defined as *real* deadline of the task, and the other is a shorter artificial deadline meant to guarantee schedulability when switching between LO and HI mode. They showed an improvement over previous approaches [51] and presented a generalization of the model to more than two criticality levels [43].

In [40], Easwaran proposed tighter analysis for two criticality levels, however it is not clear whether the method will work for more than two levels or not. Yao *et al.* provided further improvements [100] by using genetic algorithm (GA) and Quick convergence Processor-demand Analysis (QPA) [101] (an improved schedulability test for EDF), to make better artificial (shorter) deadlines.

Alternative analyses for scheduling MC systems under EDF are addressed respectively in [65] and [86]. In [65], Lipari and Buttazzo presented a reservation-based approach, in which, sufficient budget is reserved for the HI tasks. Based on that, a set of LO tasks can be guaranteed when HI tasks use only of their LO-mode requirements. Again, in the latter work, only two levels of criticality are considered. Further, in [86], Santinelli *et al.* used of multiple demand-bound curves. Their proposed technique derives sensitivity analysis which can use to trade off between resource usage and schedulability.

A speed-up factor for EDF-VD was first obtained in [10] as $(\sqrt{5}+1)/2$. Later this speed-up factor was improved to 4/3 [8]. A more flexible approach referred to as GREEDY with per-task deadline scaling was presented by Ekberg and Yi for the case of two criticality levels [42]. Ekberg and Yi characterized the execution demand of MC systems under EDF by deriving demand bound functions for the LO and the HI mode. Later, they extended this work to the case of more than two criticality levels [43]. In [40], Easwaran presented a similar technique called ECDF also for the case of two criticality levels and showed that it strictly dominates the GREEDY approach that of Ekberg and Yi.

Recently, Huang *et al.* proposed speeding up processors to account for increases in HI execution demand when switching to HI mode [55]. 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. It should be noted that the proposed techniques of this paper can also be combined with that of [55] to compute more accurate speed-up factors.

Baruah *et al.* further proposed extensions to EDF-VD, where, in particular, a per-task deadline scaling is used [9]. However, they also concluded that the speed-up factor of 4/3 cannot be improved [9]. Similarly, Müller presented a more general per-task deadline scaling technique that allows improving schedulability [71]. He proposed a new and explicit per-task deadline scaling schedulability test for MC task sets on a uniprocessor which his approach relies on the runtime dispatching scheme as given for EDF-VD, only the virtual deadlines of the HI tasks are no longer a result of a uniform scaling by a per-task-set scaling factor [71].

Improvements to the original EDF-VD have also been proposed by other authors. In [91] [92], Su and Zhu used an *elastic* task model [30] [60] to improve resource utilization in MC systems, in which the task period can change. In their work, a minimum level of service which is defined maximum period T_i^{max} was proposed for each LO task τ_i. When all LO and HI tasks use their T^{max}, C^{LO} and C^{HI} values, the system must be schedulable. Therefore, for certain parameter sets, they illustrated that their approach outperform the EDF-VD algorithm. In [102], Zhao *et al.* applied *preemption thresholds* [97] to MC scheduling in order to better utilize the processing unit. In [69], a technique consisting of two scaling factors is proposed for a admission control in MC systems.

3.2. Multiprocessor Scheduling

On multiprocessors, partitioned fixed-priority MC task sets based on Vestal's RTA [96] were studied comparatively with different task allocation and priority assignment algorithms by Kelly *et al.* [57]. In their work, different partitioning approaches such as first-fit (FF), best-fit (BF) and worst-fit (WF) for the task allocation algorithms were considered with respect to decreasing utilization and decreasing criticality Later, in [76], Pathan showed that Audsley's priority assignment algorithm is also applicable to multiprocessors and can use for arbitrary criticality levels.

Baruah *et al.* extended Audsley's OPA [3] [4] considering non-recurrent jobs of MC system resulting in an algorithm called Own Criticality-Based Priorities (OCBP) for an arbitrary number of criticality levels [16]. A generalization of the OCBP algorithm to sporadic task systems, i.e., with recurrent jobs, was presented by Li and Baruah [63].

Dorin *et al.* formalized Vestal's approach and proved that the use of Audsley's OPA [3] [4] is optimal for traditional fixed-priority MC systems [39]. They also extended the model, included release jitter and showed how sensitivity analysis could be applied [39]. Based on Vestal's approach, the priorities of LO and HI tasks are allowed to be interleaved [11] [28].

The idea is to monitor execution time of HI tasks and to allocate more processing time (which is taken away from LO tasks), if necessary. For this, again, LO tasks need to be degraded or discarded [11] [28]. Further, this approach was extended to make better use of the processor and, hence, be able to provide better scheduling guarantees [12] and [26].

As already mentioned, to be able to allocate more processing time for HI tasks, the system implements different operation modes. In principle, there is a mode per criticality level to which the system switches depending on whether tasks of the corresponding criticality increase their execution demand. In the above works, since they are based on a dual criticality, there are only two modes: LO and HI mode [11] [12].

On the other hand, there are other settings which have problems similar to mixed-criticality scheduling. For example, mode-change protocols focus on response-time analysis and resource allocation techniques for the case where a task set is changed at run-time and the pending/partially-processed jobs are required to be completed, transferred or discarded [62] [82] [79] [80]. The literature on mode-change protocols already considers the important problem that a system can be schedulable in each mode in isolation, but not schedulable during a mode change, which is also true for systems that change criticality levels [6] [44] [77] [82] [87] [94].

3.2.1. Multiprocessor Scheduling Based on EDF

Mixed-criticality systems with EDF scheduling was first considered by Baruah and Vestal [18]. Later, a slack-based mixed criticality was presented by Park and Kim [75] for EDF scheduled dual-criticality jobs, which is called Criticality-Based Earliest Deadline First (CBEDF). The schedule is divided into intervals according to the deadlines of the jobs. In each interval, two slack values are determined by the algorithm: the empty slack and the remaining slack. To this end, a combination of off-line and on-line analysis is used to run LO jobs in the generated slack and HI jobs as late as possible. Moreover, an older protocol developed by Chetto and Chetto [34] to run soft real-time tasks in the gaps left by hard real-time tasks is used in [75], where LO tasks are treated as soft real-time and HI tasks as hard real-time.

Furthermore, Mollison *et al.* presented a two-level hierarchical scheduler for multi-processor systems [70] in which they consider periodic tasks with five criticality levels from A to E. When tasks are in criticality level A, they are scheduled and statically assigned to a processor using a table-driven scheduling. Tasks with criticality level B (the next lower criticality level) are partitioned on a specific processor and scheduled by EDF, i.e., tasks are scheduled using partitioned EDF. In criticality levels C and respectively D, tasks are scheduled by applying a global EDF scheduler, when there are no activated schedulers with higher criticality level. Finally, in the criticality level E, tasks will be scheduled by best effort. As a consequence, temporal isolation between different criticality levels is provided by the proposed structure [70].

Under EDF schedule, in [10], Baruah *et al.* proposed the Earliest Deadline First with Virtual Deadlines (EDF-VD) algorithm to schedule a mix of HI and LO tasks. As already discussed, EDF-VD introduces two operation modes and uses a *priority-promotion* scheme by uniformly scaling deadlines of HI tasks.

The idea of EDF-VD is to meet the dynamic guarantees on mixed-criticality task sets by shortening the deadline of high-criticality tasks in normal mode such that they can meet their real deadlines in critical mode [10]. Baruah *et al.* further provided a more strict bound on the schedulability test of EDF-VD [8].

Many works have been conducted on EDF-VD scheduling and algorithms based on it. The scheduling of implicit-deadline sporadic tasks on uniform multiprocessor platforms have been considered in [14] by Baruah *et al.* Apart from investigating EDF-VD in this context, Baruah et al. proposed the EDF-based global scheduling algorithm which is called fpEDF [8] for multiprocessors.

In a later work, Baruah proposed an extension to the MC task model including criticality-specific values for period and deadline as well as WCET [20], i.e., scheduling tasks in MC systems based on a three-parameter model for which he extended the EDF-VD scheduling algorithm.

Furthermore, the issue of MC scheduling on partitioned multiprocessor using three different scheduling approaches ECDF, EDF-VD and Adaptive MC (AMC) is considered in [81] by Ramanathan and Easwaran to decrease the maximum difference between the overall LO and HI utilization devoted to each processor.

The importance and increasing demand for multiprocessor platforms has brought attention to MC scheduling algorithms for such platforms as well [19] [47] [48] [64]. Various techniques of MC scheduling are introduced in different studies by extending existing standard multiprocessor scheduling algorithms [5] [72] [84] [99] such as fluid-based MC model [61], global fixed-priority scheduling algorithms [19] [76], hierarchical servers [47] [48], and a semi-partitioned scheme [5].

For multiprocessors, a partitioned and a global scheduling approach both based on EDF-VD were proposed in [14]. According to this work, partitioned behaves better than global scheduling in the context of MC systems. Further, in [76], 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. Another approach is proposed by Lee *et al.* with their MC-Fluid model [61], which has been improved later in [15]. 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 [54], 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 [83].

Similar to [54] and [83], with our proposed utilization caps, we follow the idea of not discarding LO tasks in case HI execution demand increases. Our technique allows LO tasks to continue running without degradation in contrast to [54], and is less pessimistic than a server-based approach as the one in [83], since it does not incur any starvation period. Moreover, with the aim of improving the known demand bound functions for mixed-criticality EDF, we also propose separating the analysis of transitions from stable HI mode, which we show to be more accurate than the ones by Ekberg and Yi and of Easwaran for the most relevant cases. We further demonstrate that the proposed demand bound functions can be easily approximated with known techniques from the literature, in particular, Devi's test. This way, we derive schedulability tests for MC systems under EDF that trade off accuracy versus performance as discussed later in detail.

Chapter 4.

Introducing Utilization Caps

In this chapter, as already mentioned, we are concerned with mixed-criticality (MC) systems where a set of low-criticality (LO) and high-criticality (HI) tasks share one processor and are scheduled under the 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 of 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.

The main advantage over the server-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 the EDF-VD algorithm, which remains unchanged within a partition/subset of tasks. It hence facilitates a compositional design of MC systems. In a later chapter, we also show experimental evaluation based on synthetic data that evidences the benefits of the proposed technique.

4.1. Introducing Utilization Caps

In this section, we introduce utilization caps to EDF-VD as discussed in Section 2.5.1. That is, instead of finding a value of x for the whole task set τ as originally, we find a scaling factor 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 the original EDF-VD's equations (2.5.3) and (2.5.4) as follows:

$$\begin{aligned} U_{LO}^{LO} + \frac{U_{HI}^{LO}}{x} &\leq 1, \\ x \cdot U_{LO}^{LO} + U_{HI}^{HI} &\leq 1. \end{aligned}$$

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:

$$\begin{aligned} U_{LO}^{LO} + \frac{U_{HI}^{LO}}{x} &\leq U_L, \\ x \cdot U_{LO}^{LO} + U_{HI}^{HI} &\leq U_L, \end{aligned}$$

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 $\tau_X \subset \tau$:

$$\frac{U_{HI}^{LO}}{U_L - U_{LO}^{LO}} \leq x, \tag{4.1.1}$$

$$x \leq \frac{U_L - U_{HI}^{HI}}{U_{LO}^{LO}}. \tag{4.1.2}$$

In order that $\tau_X \subset \tau$ is schedulable, we need to find a value of x in $(0,1)$, for which (4.1.1) and (4.1.2) hold. However, this defines a range of possible values, from which one can choose x to be, for example, in the middle of this range:

$$x = \frac{U_{HI}^{LO}}{U_L - U_{LO}^{LO}} + \frac{U_L - U_{HI}^{HI}}{2 \cdot U_{LO}^{LO}} - \frac{U_{HI}^{LO}}{2\left(U_L - U_{LO}^{LO}\right)}. \tag{4.1.3}$$

4.1.1. Fixed utilization caps

Clearly, whether (4.1.1) and (4.1.2) 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,

Algorithm 1 Schedulability test for fixed utilization caps

```
Require: $\tau = \tau_A \cup \tau_B \cup \dots \tau_Z$
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
```

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 (4.1.3) 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.

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 $\mathcal{O}(n)$.

4.1.2. 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 (4.1.1) needs to be less than or equal to the right-hand side of (4.1.2), 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:

$$0 \leq U_L^2 - \left(U_{LO}^{LO} + U_{HI}^{HI}\right) \cdot U_L + \left(U_{HI}^{HI} - U_{HI}^{LO}\right) \cdot U_{LO}^{LO}. \tag{4.1.4}$$

Now, for the system to be schedulable, either $\hat{U}_L$ or $\check{U}_L$ — i.e., any of the roots of (4.1.4) when equalized to zero — needs to be a real number in the interval $(0, 1]$:

$$\check{U}_L = \frac{\left(U_{LO}^{LO} + U_{HI}^{HI}\right)}{2} - \frac{\sqrt{\left(U_{LO}^{LO} + U_{HI}^{HI}\right)^2 - 4\left(U_{HI}^{HI} - U_{HI}^{LO}\right) \cdot U_{LO}^{LO}}}{2}, \tag{4.1.5}$$

$$\hat{U}_L = \frac{\left(U_{LO}^{LO} + U_{HI}^{HI}\right)}{2} + \frac{\sqrt{\left(U_{LO}^{LO} + U_{HI}^{HI}\right)^2 - 4\left(U_{HI}^{HI} - U_{HI}^{LO}\right) \cdot U_{LO}^{LO}}}{2}. \tag{4.1.6}$$

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 (4.1.3) 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 (4.1.4), are computed in as per (4.1.5) and (4.1.6) 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.

Algorithm 2 Schedulability test for optimum utilization caps

```
Require: $\tau = \tau_A \cup \tau_B \cup \ldots \tau_Z$
 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
 8:         Compute $\check{U}_L$ and $\hat{U}_L$
 9:         if $\check{U}_L > 0$ and $\check{U}_L <= 1$ then
10:             $U_L = \check{U}_L$
11:         else if $\hat{U}_L > 0$ and $\hat{U}_L <= 1$ then
12:             $U_L = \hat{U}_L$
13:         else
14:             Return ("not schedulable")
15:         end if
16:         $U = U + U_L$
17:         Compute $x$
18:     end if
19: end for
20: if $U > 1$ then
21:     Return ("not schedulable")
22: else
23:     Return ("schedulable")
24: end if
```

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 (4.1.3) 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 $\mathcal{O}(n)$.

4.2. Findings of this Chapter

In this chapter, 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.

On the other hand, a performance decay with respect to EDF-VD can be expected. The smaller the utilization cap of one partition, the less performance in terms of schedulable task sets can be achieved. However, our experiments indicate later 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 at least some HI tasks switching to HI mode, which enables for more design flexibility.

Chapter 5.

Bounding Execution Demand under Mixed-Criticality EDF

In this chapter, we are also concerned with MC systems with a mix of LO and HI tasks that are scheduled under EDF on one processor. Again, the system operates in two modes: LO and HI mode. In LO mode, one HI task may exceed its execution budget, which then causes a change to HI mode in the system. HI tasks are assigned larger execution budgets in HI mode at the cost of the LO tasks — which are assumed to be discarded.

Since these mode changes may happen at arbitrary points in time, it is difficult to find an accurate bound on the amount of *carry-over* execution demand. That is the execution demand by HI jobs that were released before, but did not finish executing at the point of changing to HI mode. As a consequence, the resulting characterization of the overall execution demand becomes pessimistic.

To overcome this problem, a technique is proposed that works around the computation of carry-over execution demand and results in a more accurate bound on execution demand under mixed-criticality EDF. In principle, the proposed technique consists in separating the schedulability analysis of *stable* HI mode from that of the *transition* between modes and deriving a separate demand bound function for the latter case. The proposed technique results not only in a considerably simpler, but also tighter bound on execution demand under mixed-criticality EDF, in particular, as the number of HI tasks increases.

Mixed-Criticality EDF. A common approach when scheduling MC systems is to shorten the deadlines of HI jobs in LO mode. This way, processor capacity can be *reserved* for a potential switch to HI mode — where HI tasks require more execution demand. In other words, we assign a *virtual deadline* equal to $x_i \cdot D_i$ with $x_i \in (0, 1]$ to all τ_i where $\chi_i = HI$.

This virtual deadline is used instead of D_i — the *real* deadline — to schedule HI tasks in LO mode. The parameter x_i is the so-called *deadline scaling factor.* There is no deadline scaling for LO tasks such that they are scheduled (only in LO mode) using their D_i.

When the system switches to HI mode, HI tasks start being scheduled according to their real deadlines D_i whereas LO tasks are discarded immediately. In this chapter, we consider that tasks are scheduled under EDF in both modes and refer to this scheme as *mixed-criticality* EDF.

Clearly, whereas schedulability of separated modes can be easily tested, i.e., when the system is stable in either LO or HI mode, it is difficult to test schedulability of transitions between modes. In particular, careful analysis is required when the system switches from LO to HI mode.

In this work, similar to other approaches from the literature, transitions from HI back to LO mode are disregarded. The reason is that, in contrast to changes from LO to HI, a change from HI to LO mode can be programmed or postponed to a suitable point in time, e.g., at which the processor idles after all HI tasks have run again for their optimistic WCETs, and does not require further analysis.

5.1. Bounding Execution Demand

In this section, we introduce the proposed technique. Basically, similar to [42] and [40], we characterize the execution demand of τ by applying the concept of *demand bound function* [17]. In [42] and [40], a demand bound function was derived for each mode in the system, i.e., one for the LO mode and one for the HI mode.

In contrast to this, as mentioned above, we derive a third demand bound function for the transition between modes. This allows working around the computation of carry-over execution demand, reducing the amount of pessimism and, hence, relaxing schedulability conditions in HI mode as we illustrate later.

Schedulability in LO mode. In LO mode, LO tasks need to be scheduled together with HI tasks, while the latter are assigned virtual deadlines $x_i \cdot D_i$. As a consequence, the demand bound function $\mathrm{dbf}_{LO}(t)$ in LO mode is given by:

$$\begin{aligned}\mathrm{dbf}_{LO}(t) \quad &= \quad \sum_{\chi_i = LO} \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right) C_i^{LO} \\ &\quad + \sum_{\chi_i = HI} \left(\left\lfloor \frac{t - x_i \cdot D_i}{T_i} \right\rfloor + 1 \right) C_i^{LO}. \end{aligned} \tag{5.1.1}$$

Here $t \geq 0$ is a real number representing time, i.e., $\mathrm{dbf}_{LO}(t)$ returns a τ's maximum execution demand in LO mode in an interval of length t. Note that $\mathrm{dbf}_{LO}(t)$ is always greater than or equal to zero, since $D_i \leq T_i$ holds for all τ_i and x_i has values in $(0, 1]$.

The system is schedulable in LO mode, if $\mathrm{dbf}_{LO}(t) \leq t$ holds for all possible t until the processor first idles [17], i.e., until a point in time $\hat{t}_{LO}$ by which $\mathrm{dbf}_{LO}(\hat{t}_{LO}) = \hat{t}_{LO}$ holds:

$$\hat{t}_{LO} \quad = \quad \sum_{\chi_i = LO} \left(\left\lfloor \frac{\hat{t}_{LO} - D_i}{T_i} \right\rfloor + 1 \right) C_i^{LO} + \sum_{\chi_i = HI} \left(\left\lfloor \frac{\hat{t}_{LO} - x_i \cdot D_i}{T_i} \right\rfloor + 1 \right) C_i^{LO},$$

Following the technique in [17], we can remove the floor function:

$$\hat{t}_{LO} \quad \leq \quad \sum_{\chi_i = LO} \left(\frac{\hat{t}_{LO} - D_i}{T_i} + 1 \right) C_i^{LO} + \sum_{\chi_i = HI} \left(\frac{\hat{t}_{LO} - x_i \cdot D_i}{T_i} + 1 \right) C_i^{LO},$$

and reshape to:

$$\hat{t}_{LO} \quad \leq \quad \hat{t}_{LO} \left(\sum_{\chi_i = LO} \frac{C_i^{LO}}{T_i} + \sum_{\chi_i = HI} \frac{C_i^{LO}}{T_i} \right) + \sum_{\chi_i = LO} (T_i - D_i) \frac{C_i^{LO}}{T_i} + \sum_{\chi_i = HI} (T_i - x_i \cdot D_i) \frac{C_i^{LO}}{T_i}.$$

Finally, considering that $U_{LO}^{LO} = \sum_{\chi_i=LO} \frac{C_i^{LO}}{T_i}$ and $U_{HI}^{LO} = \sum_{\chi_i=HI} \frac{C_i^{LO}}{T_i}$, we obtain an upper bound on $\hat{t}_{LO}$:

$$\hat{t}_{LO} \quad \leq \quad \frac{\sum\limits_{\chi_i=LO} (T_i - D_i)\frac{C_i^{LO}}{T_i}}{1 - U_{LO}^{LO} - U_{HI}^{LO}} + \frac{\sum\limits_{\chi_i=HI} (T_i - x_i \cdot D_i)\frac{C_i^{LO}}{T_i}}{1 - U_{LO}^{LO} - U_{HI}^{LO}}. \tag{5.1.2}$$

The bound in (5.1.2) depends on the values of x_i, which need to be computed and are not known a priori. On the other hand, to resolve this dependency, note that this bound maximizes for all $x_i = 0$ which then leads to the following:

$$\hat{t}_{LO} \quad \leq \quad \frac{\sum\limits_{\chi_i=LO} (T_i - D_i)\frac{C_i^{LO}}{T_i}}{1 - U_{LO}^{LO} - U_{HI}^{LO}} + \frac{\sum\limits_{\chi_i=HI} C_i^{LO}}{1 - U_{LO}^{LO} - U_{HI}^{LO}}, \tag{5.1.3}$$

Clearly, $U_{LO}^{LO} + U_{HI}^{LO}$ — the utilization in LO mode — must be strictly less than one in order that (5.1.2) and (5.1.3) return valid and finite values.

Schedulability in HI mode. In HI mode, again, LO tasks do not run and HI tasks run for their corresponding C_i^{HI} leading to the following demand bound function:

$$\mathrm{dbf}_{HI}(t) = \sum_{\chi_i=HI} \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right) C_i^{HI}, \tag{5.1.4}$$

where again $t \geq 0$ is a real number representing time, i.e., $\mathrm{dbf}_{HI}(t)$ returns the maximum execution demand in a time interval of length t.

The system is schedulable in *stable* HI mode, if $\mathrm{dbf}_{HI}(t) \leq t$ for all possible t until the processor first idles, i.e., until a point in time $\hat{t}_{HI}$ is reached where $\mathrm{dbf}_{HI}(\hat{t}_{HI}) = \hat{t}_{HI}$. We again can remove the floor function in (5.1.4) to obtain an upper bound on $\hat{t}_{HI}$:

$$\hat{t}_{HI} \leq \frac{\sum\limits_{\chi_i=HI} (T_i - D_i)\frac{C_i^{HI}}{T_i}}{1 - U_{HI}^{HI}}. \tag{5.1.5}$$

U_{HI}^{HI} — the utilization in HI mode — must be strictly less than one in order that (5.1.5) returns a valid and finite upper bound on $\hat{t}_{HI}$.

Schedulability in the transition from LO to HI mode. The transition from LO to HI mode may happen at an arbitrary point in time when one HI job exceeds its LO execution budget C_i^{LO}. Unfinished LO jobs are discarded at that time; however, the problem arises with HI jobs that are released but have not finished executing their C_i^{LO}, i.e., carry-over jobs. Since it is difficult to accurately bound the execution demand by carry-over jobs, usually, pessimistic assumptions need to be taken.

The following theorem is a generalization of a theorem in [69] and allows us to work around carry-over jobs constituting the main contribution by this work. In other words, this theorem allows us to guarantee schedulability without computing carry-over execution demand at the point of switching from LO to HI mode, which considerably reduces the amount of pessimism.

Theorem 1. *Given a set τ of MC tasks, let us assume that the following two conditions hold: (i) $dbf_{LO}(t) \leq t$ and (ii) $dbf_{HI}(t) \leq t$ hold for $0 < t \leq \hat{t}_{LO}$ and $0 < t \leq \hat{t}_{HI}$ respectively, i.e., τ is schedulable in LO and stable HI mode. The transition from LO to HI mode is schedulable under mixed-criticality EDF, if $dbf_{SW}(t) \leq t$ also holds for $0 < t \leq \hat{t}_{SW}$, where $dbf_{SW}(t)$ is given by:*

$$dbf_{SW}(t) = \sum_{\chi_i = HI} \left(\left\lfloor \frac{t - \Delta D_i}{T_i} \right\rfloor + 1 \right) \Delta C_i, \tag{5.1.6}$$

with $\Delta D_i = D_i - x_i \cdot D_i$, $\Delta C_i = C_i^{HI} - C_i^{LO}$, and $\hat{t}_{SW}$ is upper bounded by the following expression:

$$\hat{t}_{SW} \leq \frac{\sum\limits_{\chi_i = HI} \Delta C_i}{1 - U_{HI}^{SW}}, \tag{5.1.7}$$

where U_{HI}^{SW} is given by $\sum\limits_{\chi_i = HI} \frac{\Delta C_i}{T_i}$.

Proof. Let us consider that the system *switches* to HI mode at time t' and that the processor idles for the first time thereafter at t'' with $t' < t''$, i.e., all jobs released prior to t'' finish executing at latest by t''. Clearly, jobs that are released after t'' are guaranteed schedulable by assumption (ii).

Let us now assume that a deadline is missed for the first time at t_{miss} by a job of any τ_i that we denote τ_{miss}. Clearly, t_{miss} must be in the interval $[t', t'']$ and the following must hold for $\delta_{miss} = t_{miss} - t'$:

$$\delta_{miss} < CO + \sum_{\chi_i = HI} \left(\left\lfloor \frac{\delta_{miss} - \phi_i - D_i}{T_i} \right\rfloor + 1 \right) C_i^{HI},$$

where $\phi_i = r_i' - t'$ is the *phase* of a τ_i at t', i.e., the release time r_i' of its first job after t' minus t'. Note that ϕ_i is in the interval $[0, T_i)$. In addition, CO denotes the *carry-over* execution demand at t'. This is the amount of execution in $[t', t_{miss}]$ by HI jobs that are released prior to t', but have not finished executing at t'.

As already discussed, it is difficult to determine CO in an accurate manner. Hence, to work around CO, let us first divide each τ_i, whose jobs have both release times and deadlines in $[t', t_{miss}]$, into two subtasks. The first subtask — denoted by τ_i^{LO} — is released for the first time at ϕ_i and requires executing C_i^{LO} within $x_i \cdot D_i$ every T_i time, i.e., this represents τ_i's execution demand in LO mode. The second subtask — denoted by τ_i^{SW} — is released for the first time at $\phi_i' = \phi_i + x_i \cdot D_i$ and requires executing $\Delta C_i = C_i^{HI} - C_i^{LO}$ within $\Delta D_i = D_i - x_i \cdot D_i$ every T_i time, i.e., this presents τ_i's increase in execution demand incurred in HI mode. Note that, in spite of this modification, the total amount of execution demand in $[t', t_{miss}]$ does not change, i.e., a deadline is still missed at t_{miss} as per assumption, and we can reshape the above inequality to:

$$\begin{aligned}\delta_{miss} < CO + \sum_{\chi_i=HI} \left(\left\lfloor \frac{\delta_{miss} - \phi_i - x_i \cdot D_i}{T_i} \right\rfloor + 1 \right) C_i^{LO} \\ + \sum_{\chi_i=HI} \left(\left\lfloor \frac{\delta_{miss} - \phi_i' - \Delta D_i}{T_i} \right\rfloor + 1 \right) \Delta C_i. \end{aligned} \tag{5.1.8}$$

Note that t_{miss} coincides with the deadline of the corresponding job of τ_{miss}^{SW}, which misses its deadline. (Recall that τ_{miss} is now divided into the subtasks τ_{miss}^{LO} and τ_{miss}^{SW}.) Now, there are two possible cases to consider in order to prove this theorem: The set of *only* subtasks τ_i^{SW} is either (1) unschedulable or (2) schedulable in isolation.

Case (1): This is a rather trivial case. If the set of only τ_i^{SW} is unschedulable in isolation, i.e., when scheduled alone on a single processor, $\text{dbf}_{SW}(t) > t$ must hold for some t in $[0, \hat{t}_{SW}]$ with $\text{dbf}_{SW}(t)$ given as per (5.1.6). As a result, we will be able to detect a deadline miss in the transition between LO and HI mode by only testing the set of all τ_i^{SW}.

To this end, we need to find an upper bound on $\hat{t}_{SW}$ making $\text{dbf}_{SW}(\hat{t}_{SW}) = \hat{t}_{SW}$ and removing the floor function as before:

$$\hat{t}_{SW} \leq \frac{\sum\limits_{\chi_i=HI} (T_i - \Delta D_i)\frac{\Delta C_i}{T_i}}{1 - U_{HI}^{SW}}.$$

Here, $U_{HI}^{SW} = \sum_{\chi_i=HI} \frac{\Delta C_i}{T_i}$ is the utilization of the set of only τ_i^{SW}. Since $U_{HI}^{SW} < 1$ holds by assumption (ii), the above inequality returns a valid bound on $\hat{t}_{SW}$. This depends on $\Delta D_i = D_i - x_i \cdot D_i$ and, therefore, on the values of x_i, which we do not know in advance. However, we can make $x_i = 1$ for all i leading to the upper bound in (5.1.7). The theorem follows.

Case (2): If $\text{dbf}_{SW}(t) \leq t$ holds for all t in $[0, \hat{t}_{SW}]$, we show that assuming that a deadline is missed at t_{miss} leads to a contradiction.

We have assumed that a deadline is missed for the first time at t_{miss}, hence, all previous jobs in $[t', t_{miss})$ can actually finish executing in time. Since now τ_{miss} is divided into the subtasks τ_{miss}^{LO} and τ_{miss}^{SW}, the τ_{miss}^{LO}'s job with a deadline equal to $t_{miss} - \Delta D_{miss}$ must push *carry-over* τ_i^{SW}'s jobs (i.e., those that are released prior to and have not finished executing at $t_{miss} - \Delta D_{miss}$ and that have deadlines prior to t_{miss}) by at least Δ_{miss} (being Δ_{miss} the amount of the deadline miss at t_{miss}). Otherwise, no deadline can be missed at t_{miss}, since again $\text{dbf}_{SW}(t) \leq t$ is assumed to hold for all t. In addition, note that the processor does not idle in $[t_{miss} - \Delta D_{miss}, t_{miss}]$.

Case (2.a): Let us assume that there is only one carry-over job of an arbitrary τ_i^{SW} and that $\Delta D_i \leq \Delta D_{miss}$ holds. Note that, after moving this job forward to force its release time to coincide at $t_{miss} - \Delta D_{miss}$, τ_{miss}^{SW}'s job continues to miss its deadline by at least Δ_{miss} at t_{miss}, since the deadline of this carry-over τ_i^{SW}'s job remains within the interval

$[t_{miss} - \Delta D_{miss}, t_{miss}]$. As a result, the amount of execution demand in $[t_{miss} - \Delta D_{miss}, t_{miss}]$ remains the same or even increases after this displacement — see Fig. 5.1.

The above analysis leads to a contradiction, since τ_{miss}^{SW}'s job and its carry-over τ_i^{SW}'s job are now released in synchrony at $t_{miss} - \Delta D_{miss}$. Consequently, τ_{miss}^{LO} cannot push any additional execution demand into $[t_{miss} - \Delta D_{miss}, t_{miss}]$ and, hence, if a deadline is missed at t_{miss}, $\mathrm{dbf}_{SW}(t) \leq t$ cannot hold for all t.

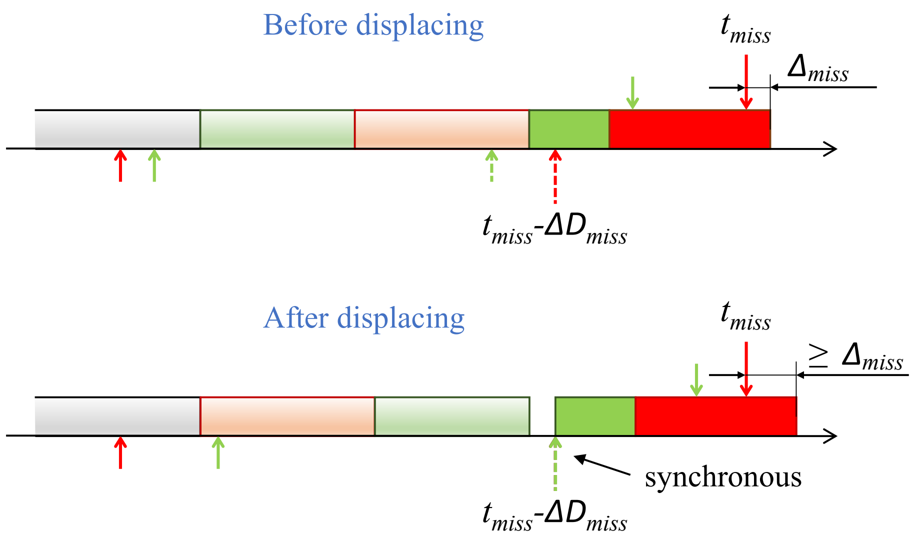


Figure 5.1.: Illustration of Case (2.a) with $\Delta D_i \leq \Delta D_{miss}$: The carry-over τ_i^{SW}'s job is illustrated in green and τ_{miss}^{SW} is illustrated in red. The light green and red shading represent τ_i^{LO} and τ_{miss}^{LO} respectively, whereas a light gray shading represents any previous higher-priority execution (i.e., of jobs with shorter deadlines). Solid upward arrows stand for the release times of τ_i^{LO} and τ_{miss}^{LO}, while dashed upward arrows represent the release times of τ_i^{SW} and τ_{miss}^{SW} (which are also the (virtual) deadlines of τ_i^{LO} and τ_{miss}^{LO}). Solid downward arrows stand for the (real) deadlines of τ_i^{SW} and τ_{miss}^{SW}.

Case (2.b): Let us now assume that there is again only one carry-over job of an arbitrary τ_i^{SW}, however, $\Delta D_i > \Delta D_{miss}$ holds this time. Note that we can displace this carry-over τ_i^{SW}'s job forward until its deadline occurs at $t_{miss} + \varepsilon$, where ε is an infinitesimally small number greater than zero. As a result, this τ_i^{SW}'s job starts missing its deadline by an amount equal to at least $\Delta_{miss} - \varepsilon$, since at least the original execution demand in $[t_{miss} - \Delta D_{miss}, t_{miss}]$ starts being executed in $[t_{miss} - \Delta D_{miss}, t_{miss} + \varepsilon]$ — see Fig. 5.2.

It is easy to see that we can now apply the analysis of Case (2.a) where the carry-over τ_i^{SW}'s job of this case misses its deadline at $t_{miss} + \varepsilon$ (by an amount equal to $\Delta_{miss} - \varepsilon$) and the τ_{miss}^{SW}'s job of this case becomes the carry-over job in Case (2.a).

As a result, Case (2.b) also leads to a contradiction, i.e., if a deadline is missed at t_{miss}, $\mathrm{dbf}_{SW}(t) \leq t$ cannot hold for all t.

Clearly, there can be several carry-over jobs whose execution demands are pushed (at least partially) by τ_{miss}^{LO} into $[t_{miss} - \Delta D_{miss}, t_{miss}]$, however, the total amount of execution demand pushed by τ_{miss}^{LO} remains Δ_{miss}. In this latter case, again, it is easy to see that we can apply the analysis of Case (2.a) and Case (2.b) to each individual carry-over job. Consequently, if a deadline is missed at t_{miss}, $\mathrm{dbf}_{SW}(t) > t$ must hold for some t and the theorem follows.

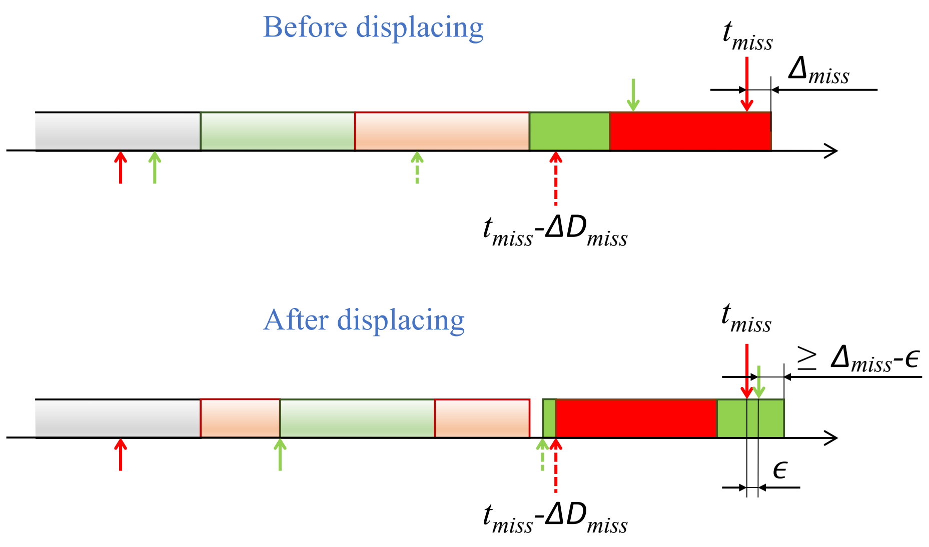

Before displacing
t_{miss}
Δ_{miss}
t_{miss}-ΔD_{miss}
After displacing
t_{miss}
$\geq \Delta_{miss}$-ϵ
ϵ
t_{miss}-ΔD_{miss}

Figure 5.2.: Illustration of Case (2.b) with $\Delta D_i > \Delta D_{miss}$: The carry-over τ_i^{SW}'s job is illustrated in green and τ_{miss}^{SW} is illustrated in red. The light green and red shading represent τ_i^{LO} and τ_{miss}^{LO} respectively, whereas a light gray shading represents any previous higher-priority execution (i.e., of jobs with shorter deadlines). Solid upward arrows stand for the release times of τ_i^{LO} and τ_{miss}^{LO}, while dashed upward arrows represent the release times of τ_i^{SW} and τ_{miss}^{SW} (which are also the (virtual) deadlines of τ_i^{LO} and τ_{miss}^{LO}). Solid downward arrows stand for the (real) deadlines of τ_i^{SW} and τ_{miss}^{SW}.

$\square$

Theorem 1 allows characterizing the *additional* execution demand in the transitions from LO to HI mode in a more accurate manner. Based on it, we test whether deadlines are met or not in $[t', t'']$, i.e., from the time t' of switching to HI mode to the time t'' at which the processor first idles after switching. Next we perform an analytical comparison with the known approaches from the literature.

5.2. Analytical Comparison

In this section, we compare the proposed demand bound functions for mixed-criticality EDF with those used by Ekberg and Yi in the GREEDY algorithm [42] and by Easwaran in the ECDF algorithm [40]. We show, for most cases, that the proposed ones result in tighter bounds on the execution demand than the other mentioned approaches.

5.2.1. The GREEDY Algorithm

In LO mode, note that $\mathrm{dbf}_{LO}(t)$ in (5.1.1) is identical to that of Ekberg and Yi — denoted by $\mathrm{gdbf}_{LO}(t)$ in this work. That is, $\mathrm{dbf}_{LO}(t) = \mathrm{gdbf}_{LO}(t)$ for all $0 < t \leq \hat{t}_{LO}$.

In HI mode, Ekberg and Yi proposed a demand bound function — denoted $\mathrm{gdbf}_{HI}(t)$ in this work — which is given by the following expression [42]:

$$\mathrm{gdbf}_{HI}(t) \quad = \quad \sum_{\chi_i = HI} \left(\left\lfloor \frac{t - \Delta D_i}{T_i} \right\rfloor + 1 \right) C_i^{HI} - \sum_{\chi_i = HI} \mathrm{done}_i(t), \tag{5.2.1}$$

with $\Delta D_i = D_i - x_i \cdot D_i$ and $\mathrm{done}_i(t)$ is given by:

$$\mathrm{done}_i(t) = \begin{cases} \max\left(0, C_i^{LO} - \mathrm{mod}\left(\frac{t}{T_i}\right) + \Delta D_i\right), \\ \qquad\qquad \text{if } D_i > \mathrm{mod}\left(\frac{t}{T_i}\right) \geq \Delta D_i, \\ 0, \quad \text{otherwise.} \end{cases}$$

Note that $\mathrm{gdbf}_{HI}(t)$ bounds the execution demand in HI mode taking transitions into account. In our case, as discussed above, we derive different bounds on the execution demand at transitions and in stable HI mode, viz., $\mathrm{dbf}_{SW}(t)$ and $\mathrm{dbf}_{HI}(t)$ respectively.

Now, since $\mathrm{done}_i(t) \leq C_i^{LO}$ holds for all valid values of t and x_i, $\mathrm{dbf}_{SW}(t) \leq \mathrm{gdbf}_{HI}(t)$ also holds for all t. In other words, if the transition to HI mode is safe by $\mathrm{gdbf}_{HI}(t)$, it will also be safe by the proposed $\mathrm{dbf}_{SW}(t)$. However, this does not hold the other way around, i.e., $\mathrm{dbf}_{SW}(t)$ results in a tighter bound on the execution demand at transitions from LO to HI mode than $\mathrm{gdbf}_{HI}(t)$.

On the other hand, if transitions are safe, it is guaranteed that no deadlines are missed after switching to HI mode at a t' and until the processor first idles at a t''. From t'' onwards, it is easy to see that our proposed $\mathrm{dbf}_{HI}(t)$ is sufficient and necessary. That is, if $\mathrm{dbf}_{HI}(t) \leq t$ does not hold for some t with $0 \leq t \leq \hat{t}_{HI}$, then the system is not feasible, i.e., it will be neither be feasible by $\mathrm{gdbf}_{HI}(t)$.

5.2.2. The ECDF Algorithm

Similar to the case of the GREEDY algorithm, note that $\mathrm{dbf}_{LO}(t)$ in (5.1.1) is identical to that used in the ECDF algorithm in LO mode. We denote this latter by $\mathrm{edbf}_{LO}(t)$ in this study. That is, $\mathrm{dbf}_{LO}(t) = \mathrm{edbf}_{LO}(t)$ for all $0 < t \leq \hat{t}_{LO}$.

In HI mode, the ECDF algorithm uses a demand bound function — denoted $\mathrm{edbf}_{HI}(t)$ in this work — which is given by the following expression [40]:

$$\begin{aligned} \mathrm{edbf}_{HI}(t_1, t_2) \quad = \quad & \min\left(t_1, \sum_{j=1}^{3} \mathrm{dbf}_{L_j}(t_1, t_2)\right) \\ & + \sum_{\substack{\chi_i = HI \\ \text{and case 2 or 3}}} \mathrm{dbf}_i^{HI}(t_1, t_2) \\ & + \sum_{\substack{\chi_i = HI \\ \text{and case 2}}} \left(CO(t_2 - t_1) + \Delta C_i\right), \end{aligned} \tag{5.2.2}$$

where ΔC_i is defined as $C_i^{HI} - C_i^{LO}$. In addition, $0 \leq t_2 \leq \hat{t}_{HI}$ and $0 \leq t_1 \leq t_2 - \min\limits_{\chi_i = HI} (\Delta D_i)$ hold with again $\Delta D_i = D_i - x_i \cdot D_i$.

Here t_1 represents the point in time at which the system switches to HI mode (i.e., $t_1 = t'$ in this Chapter's notation) and t_2 is the point in time at which a deadline is potentially missed (i.e., $t_2 = t_{miss} \leq t''$ in this work). Note that $\mathrm{dbf}_i^{HI}(\cdot)$ is a τ_i's contribution to $\mathrm{dbf}_{HI}(\cdot)$ shown in (5.1.4). HI tasks in (5.2.2) are classified into three cases: case 1 which plays a role in computing $\mathrm{dbf}_{L_j}(\cdot)$, case 2, and case 3. For details on how to compute $\mathrm{dbf}_{L_j}(\cdot)$ for $1 \leq j \leq 3$ and how to compute $CO(\cdot)$ we refer to [40].

The system is schedulable in HI mode, if $\mathrm{edbf}_{HI}(t_1, t_2) \leq t_2$ holds. (Here again no distinction is made between transition and stable HI mode.) Let us now consider that t_1 is less than or equal to $\sum_{j=1}^{3} \mathrm{dbf}_{L_j}(t_1, t_2)$, such that the schedulability condition by ECDF now becomes:

$$\sum_{\substack{\chi_i = HI \\ \text{and case 2 or 3}}} \mathrm{dbf}_i^{HI}(t_1, t_2) + \sum_{\substack{\chi_i = HI \\ \text{and case 2}}} (CO(t_2 - t_1) + \Delta C_i) \leq t_2 - t_1. \tag{5.2.3}$$

Easwaran proved that the left-hand side of the above condition is equal to $\mathrm{gdbf}_{HI}(t_2 - t_1)$ [40]. As a consequence, the proposed $\mathrm{dbf}_{SW}(t)$ results in a tighter bound than (5.2.3), since $\mathrm{dbf}_{SW}(t) \leq \mathrm{gdbf}_{HI}(t)$ holds for all t — see again the above Section 5.2.1.

In the case where t_1 is greater than $\sum_{j=1}^{3} \mathrm{dbf}_{L_j}(t_1, t_2)$, the analytical comparison between $\mathrm{edbf}_{HI}(\cdot)$ and $\mathrm{dbf}_{SW}(\cdot)$ becomes difficult. This is the case where some amount of the execution demand given by $\mathrm{edbf}_{HI}(\cdot)$ starts being executed before t_1, at $t_1 - \sum_{j=1}^{3} \mathrm{dbf}_{L_j}(t_1, t_2)$ to be more precise. Whether a proof of dominance exists (in either way) remains an open problem.

At least, from the above discussion, we can assert that the proposed $\mathrm{dbf}_{SW}(\cdot)$ is tighter for the more stringent case, i.e., when none of the execution demand by $\mathrm{edbf}_{HI}(\cdot)$ can be executed before t_1. Our experiments, based on a large number of synthetic task sets, present evidence that this also holds on average, i.e., $\mathrm{dbf}_{SW}(\cdot)$ usually results in tighter bounds, particularly, when the number of HI task increases.

5.3. Finding Valid x_i

In this section, we propose an algorithm to find valid values of x_i for each HI task in τ. Clearly, this is closely related to the technique used to *tighten* deadlines in LO mode. In this study, we do not aim to improve deadline tightening. The contribution is rather a new technique for bounding demand execution, which can be combined with existing deadline tightening techniques, e.g., from [42] or [40].

The proposed algorithm shown in Alg. 3 essentially tests τ's schedulability in the LO mode (line 1), and in HI mode (line 2). If τ is schedulable in LO mode, the function testLO() returns a vector X_{LW} with the minimum values of x_i that could be found to be valid. If this vector is not empty, i.e., valid x_i values could be found, and τ is schedulable in HI mode, Alg. 3 tests schedulability at the transitions from LO to HI mode (line 3).

Algorithm 3 Schedulability test for mixed-criticality EDF

```
Require: $\tau$
Require: $\tau_{HI}$ /* subset of HI tasks */
1: X$_{LW}$=testLO($\tau$)
2: if testHI($\tau_{HI}$)='Passed' and X$_{LW} \neq \emptyset$ then
3:     X$_{UP}$=testSW($\tau_{HI}$)
4:     if X$_{UP} \neq \emptyset$ and X$_{LW} \leq \mathbf{1} -$ X$_{UP}$ then
5:         Return ('Passed')
6:     else
7:         Return ('Not passed')
8:     end if
9: end if
```

Further, if the set of HI tasks in τ — denoted by τ_{HI} — is schedulable at transitions from LO to HI, the function testSW() returns a vector X$_{UP}$ with the minimum values of $1 - x_i$ that are also valid. That is, if X$_{UP}$ is neither empty, the whole τ will be schedulable under mixed-criticality EDF provided that X$_{LW} \leq \mathbf{1} -$ X$_{UP}$ holds (line 4). Here, **1** denotes a unity vector (where all elements are equal to one). That is, for each element in the vectors X$_{LW}$ and X$_{UP}$, the following condition has to hold:

$$
\begin{gathered}
\mathrm{X}_{LW}(i) \leq x_i, \\
\mathrm{X}_{UP}(i) \leq 1 - x_i, \\
\implies x_i \leq 1 - \mathrm{X}_{UP}(i).
\end{gathered}
$$

As already mentioned, the functions testLO() and testSW() — shown in Alg. 4 and Alg. 5 — test schedulability in LO mode and at the transitions from LO to HI mode. These two functions are very similar — apart from testLO() dealing with the whole τ and testSW() with the subset τ_{HI} — and return lower bounds on x_i and on $1 - x_i$ respectively. Thus, the following discussion of testLO() also applies to testSW().

Basically, testLO() computes $\mathrm{dbf}_{LO}(t)$ for all $0 \leq t \leq \hat{t}_{LO}$ starting from $x_i = 1$ for all HI tasks. However, at least the first deadline of each task should be checked, since we need to compute each x_i. That is, we need to compute $\mathrm{dbf}_{LO}(t)$ at least until $D_{max} = \max\limits_{\forall i}(D_i)$ — see line 3 in Alg. 4. If the current t corresponds to a deadline of a HI task (lines 10 to 14), its (relative) virtual deadline $x_i \cdot D_i$ is adjusted such that its absolute deadline $r_i + x_i \cdot D_i$ is equal to $\mathrm{dbf}_{LO}(t)$ (i.e., the total execution demand at t).

Note that the execution demand of jobs with prior deadlines to t is contained in $\mathrm{dbf}_{LO}(t)$. As a result, the computed x_i in line 12 can never compromise schedulability of these previous jobs. In addition, the currently computed x_i can only replace a previously computed x_i, if it is greater than this latter (lines 11 to 13). This deadline tightening reduces the number of possibilities for x_i, but it also reduces the complexity of the algorithm.

Algorithm 4 Function testLO()

```
Require: $\tau$
 1: Compute $\hat{t}_{LO}$ by (5.1.3)
 2: $\mathrm{X}_{LW} = \mathbf{1}$
 3: while $t \leq \max(D_{max}, \hat{t}_{LO})$ do
 4:   if $\mathrm{dbf}_{LO}(t) > t$ then
 5:     if $\chi_i$=$LO$ or $\mathrm{dbf}_{LO}(t) - r_i > D_i$ then
 6:       $\mathrm{X}_{LW} = \emptyset$
 7:       Return
 8:     end if
 9:   end if
10:   if $\chi_i$=$HI$ then
11:     if Computed($i$)='false' or $\mathrm{X}_{LW}(i) < \frac{\mathrm{dbf}_{LO}(t) - r_i}{D_i}$ then
12:       $\mathrm{X}_{LW}(i) = \frac{\mathrm{dbf}_{LO}(t) - r_i}{D_i}$ /* $r_i$ = job's release time */
13:     end if
14:   end if
15:   $(t, i)$=getNextDeadline()
16: end while
17: Return
```

Computed(i) in line 11 returns 'false', if no x_i has been computed yet for the current i. The function getNextDeadline() in line 15 returns the point in time t at which the next deadline occurs and the index i of the task corresponding to that deadline. Clearly, this function has to take the computed values of x_i into account.

The function testLO() succeeds if it finishes testing $\mathrm{dbf}_{LO}(t)$ for $0 \leq t \leq \max(D_{max}, \hat{t}_{LO})$ [1] and it could find a value of x_i in $(0, 1]$ for each HI task in τ. On the other hand, testLO() fails, if $\mathrm{dbf}_{LO}(t) > t$ holds for some t and either t corresponds to a deadline of a LO task — whose deadline cannot be adjusted by the used tightening technique — or the resulting x_i becomes greater than 1 (lines 4 to 8).

Analogous to testLO(), testSW() computes $\mathrm{dbf}_{SW}(t)$ for all deadlines in the interval $0 \leq t \leq \max(D_{max}, \hat{t}_{SW})$ starting from $1 - x_i = 1$ for all HI tasks – recall that deadlines in $\mathrm{dbf}_{SW}(t)$ are equal to $(1 - x_i) \cdot D_i$. Otherwise, as mentioned above, testLO() and testSW() are very similar and, hence, the above explanation for testLO() also applies to testSW(). Finally, the function testHI() in Alg. 3 is the known schedulability test for EDF from the literature [17] and, hence, does not require further discussion.

[1] As values of x_i start being known, we can update this bound in order to reduce runtime by testLO(). However, we did not implement this behavior in the current version.

Algorithm 5 Function testSW()

```
Require: $\tau_{HI}$
1:  Compute $\hat{t}_{SW}$ by (5.1.7)
2:  $\mathrm{X}_{UP} = \mathbf{1}$
3:  while $t \leq \max(D_{max}, \hat{t}_{SW})$ do
4:     if $\mathrm{dbf}_{SW}(t) > t$ and $\mathrm{dbf}_{SW}(t) - r_i > D_i$ then
5:        $\mathrm{X}_{UP} = \emptyset$
6:        Return
7:     end if
8:     if Computed($i$)='false' or $\mathrm{X}_{UP}(i) < \frac{\mathrm{dbf}_{SW}(t) - r_i}{D_i}$ then
9:        $\mathrm{X}_{UP}(i) = \frac{\mathrm{dbf}_{SW}(t) - r_i}{D_i}$ /* $r_i$ = job's release time */
10:    end if
11:    $(t, i)$=getNextDeadline()
12: end while
13: Return
```

5.4. Findings of this Chapter

In this chapter, we studied the problem of mixed-criticality scheduling under EDF, where a mix of low-criticality (LO) and high-criticality (HI) tasks share the processor. Similar to the literature, we characterize the execution demand of a mixed-criticality task set by deriving demand bound functions in the different operation modes, viz., HI and LO mode.

However, it was shown that treating the transitions from LO to HI mode separately from the stable HI mode allows working around carry-over jobs and, therefore, reducing pessimism in estimating the execution demand under mixed-criticality EDF. Carry-over jobs are those HI jobs that start prior to, but have not yet finished executing at the moment of switching from LO mode to HI mode.

It is interesting to notice that the proposed technique reduces the problem of testing schedulability under mixed-criticality EDF to testing schedulability of three *almost* unrelated task sets: the one in LO mode, the one in HI mode and the equivalent task set for transitions between LO and HI mode. This leads to a considerably simpler schedulability test and improves our understanding of this problem.

Chapter 6.

Approximating Execution Demand Bounds

In the previous chapter, we proposed a technique that works around the computation of carry-over execution demand, which originates due to switching from LO to HI mode at arbitrary points in time. The proposed technique separates the schedulability analysis of the transition between LO and HI mode from that of *stable* HI mode allowing us to guarantee schedulability in a transition from LO to HI mode, if an equivalent task set derived from the original is schedulable under plain EDF. Since the proposed approach is based on standard demand bound functions for EDF, we can apply approximation techniques such as, e.g., the well-known Devi's test to derive further tests that trade off accuracy versus complexity/runtime.

6.1. Applying Approximation Techniques

Basically, there are different two possible ways of applying, Devi's test [38] to the demand bound functions of Section 5.1. This leads either to a per-task deadline scaling or a uniform deadline scaling as discussed next.

In this section, we apply approximation techniques, particularly, Devi's test [38] to derive two variants of the proposed approach trading off accuracy for the sake of lesser complexity/runtime. The first variant computes one deadline scaling factor per HI task, similar to the proposed approach, but with less accuracy to reduce complexity. In contrast to it, the second variant computes only one deadline scaling factor for all HI tasks requiring, in principle, less computation. However, as discussed later in detail, this does not necessarily lead to a lower complexity.

6.2. Devi's Test

In the following, we make use of approximation techniques such as the well-known Devi's test [38] to derive further schedulability tests for MC systems under EDF that trade off accuracy versus performance (complexity/runtime). We apply Devi's test to derive two approximated variants of our proposed approach, namely, based on per-task and on uniform deadline scaling, that is, using Devi's test to compute a scaling factor for each individual HI task and a uniform deadline scaling factor for all HI tasks in τ.

Devi proposed a schedulability test of periodic task sets. Assume that $\tau = \{\tau_1, \tau_2, \dots \tau_n\}$ is the task systems of n preemptable, asynchronous and periodic tasks with arbitrary relative deadlines, sorted in order of non-decreasing relative deadlines. τ is schedulable under an optimal scheduling algorithm such as EDF if the following inequality holds for every possible k [38]:

$$\forall k : 1 \leq k \leq n :: \sum_{i=1}^{k} \frac{C_i}{T_i} + \frac{1}{D_k} \sum_{i=1}^{k} \left(\frac{T_i - min\,(T_i, D_i)}{T_i} \right) C_i \leq 1 \tag{6.2.1}$$

where each task contains an execution time C_i, a relative deadline D_i and a period T_i with the considered case $\forall i : D_i < T_i$. This test has a complexity of $\mathcal{O}(n \log n)$ because it requires sorted task sets.

6.2.1. Per-task deadline scaling

We can now derive the first approximated variant of our proposed approach from Chapter 5 computing a scaling factor for each individual HI task in τ. To this end, we use Devi's test to analyze each mode and the transition between them.

Schedulability in LO mode. When applying Devi's test, the demand bound function in (5.1.1) reduces to the following condition:

$$\sum_{i=1}^{k} \frac{C_i^{LO}}{T_i} + \frac{1}{x_k \cdot D_k} \left(\sum_{\substack{i=1 \\ \chi_i = LO}}^{k} \frac{T_i - D_i}{T_i} C_i^{LO} + \sum_{\substack{i=1 \\ \chi_i = HI}}^{k} \frac{T_i - x_i \cdot D_i}{T_i} C_i^{LO} \right) \leq 1, \tag{6.2.2}$$

where $1 \leq k \leq |\tau|$ and $|\tau|$ denotes the total number of tasks in τ. Tasks in (6.2.2) need to be sorted in order of non-decreasing real deadlines D_i for $\chi_i = LO$ or virtual deadlines $x_i \cdot D_i$ for $\chi_i = HI$. Note that the order of tasks might change depending on the values of x_i. As a result, (6.2.2) might need to be recomputed for the corresponding tasks, if their relative order changes.

Further, in (6.2.2), we have considered that the current τ_k (i.e., for the current value of k) is a HI task. As a result, a deadline scaling factor x_k needs to be computed. If τ_k is a LO task, the second term of (6.2.2) is divided by D_k instead of $x_k \cdot D_k$, since LO tasks are scheduled within their real deadline and require no x_k to be computed. However, clearly, (6.2.2) still needs to hold for all previously selected x_i.

Schedulability in stable HI mode. Applying Devi's test to the demand bound function in (5.1.4) now results in the following condition:

$$\sum_{\substack{i=1 \\ \chi_i = HI}}^{k} \frac{C_i^{HI}}{T_i} + \frac{1}{D_k} \left(\sum_{\substack{i=1 \\ \chi_i = HI}}^{k} \frac{T_i - D_i}{T_i} C_i^{HI} \right) \leq 1, \tag{6.2.3}$$

where again $1 \leq k \leq |\tau|$ holds. Note that only HI tasks are considered in (6.2.3), which need to be sorted in order of non-decreasing D_i.

Schedulability in the transition from LO to HI mode. The demand bound function in (5.1.6) reduces to the following condition after applying Devi's test:

$$\sum_{\substack{i=1\\ \chi_i=HI}}^{k} \frac{\Delta C_i}{T_i} + \frac{1}{(1-x_k)D_k}\left(\sum_{\substack{i=1\\ \chi_i=HI}}^{k} \frac{T_i - \Delta D_i}{T_i}\Delta C_i\right) \leq 1, \tag{6.2.4}$$

with $1 \leq k \leq |\tau|$ as before, $\Delta C_i = C_i^{HI} - C_i^{LO}$, and $\Delta D_i = (1-x_i)\cdot D_i$. Similar to stable HI mode, only HI tasks are considered in (6.2.4). However, this time, tasks are sorted in the order of non-decreasing ΔD_i instead. Similar to LO mode, the order of tasks might change depending on the values of x_i. In this case, (6.2.4) needs to be recomputed for all tasks whose relative order changes for a newly computed x_k.

Finding deadline scaling factors. As already discussed, (6.2.2) needs to hold for all k in $1 \leq k \leq |\tau|$, which is tested in an iterative manner. Clearly, x_k only needs to be computed for $\chi_k = HI$. To this end, let us first reshape (6.2.2) to the following:

$$\begin{aligned}\sum_{i=1}^{k} \frac{C_i^{LO}}{T_i} + \frac{1}{x_k \cdot D_k}&\left(\sum_{\substack{i=1\\ \chi_i=LO}}^{k} \frac{T_i - D_i}{T_i}C_i^{LO}\right.\\ +\sum_{\substack{i=1\\ \chi_i=HI}}^{k-1} \frac{T_i - x_i\cdot D_i}{T_i}C_i^{LO} &+ \left.\frac{T_k - x_k\cdot D_k}{T_k}C_k^{LO}\right) \quad\leq\quad 1,\end{aligned}$$

which we can then solve for x_k leading to:

$$\frac{\sum\limits_{\substack{i=1\\ \chi_i=LO}}^{k} \frac{T_i-D_i}{T_i}C_i^{LO} + \sum\limits_{\substack{i=1\\ \chi_i=HI}}^{k-1} \frac{T_i-x_i\cdot D_i}{T_i}C_i^{LO} + C_k^{LO}}{D_k\left(1-\sum_{i=1}^{k}\frac{C_i^{LO}}{T_i}\right) + D_k\frac{C_k^{LO}}{T_k}} \leq x_k.$$

Note that we can change the upper limit of the first summation in the numerator to $k-1$, since $\chi_k = HI$ holds. Further, reshaping the denominator, we finally obtain:

$$\frac{\sum\limits_{\substack{i=1\\ \chi_i=LO}}^{k-1} \frac{T_i-D_i}{T_i}C_i^{LO} + \sum\limits_{\substack{i=1\\ \chi_i=HI}}^{k-1} \frac{T_i-x_i\cdot D_i}{T_i}C_i^{LO} + C_k^{LO}}{D_k\left(1-\sum_{i=1}^{k-1}\frac{C_i^{LO}}{T_i}\right)} \leq x_k. \tag{6.2.5}$$

As already mentioned, we might need to recompute (6.2.2) for all tasks whose relative order changes depending on the value of x_k. To avoid this complication, we derive the following lower bound for x_k:

$$\frac{D^{LO}_{k-1}}{D_k} \leq x_k, \tag{6.2.6}$$

where $D^{LO}_{k-1} = D_{k-1}$ or $D^{LO}_{k-1} = x_{k-1} \cdot D_{k-1}$ depending on whether $\chi_{k-1} = LO$ or $\chi_{k-1} = HI$ respectively. In words, (6.2.6) guarantees that the selected x_k does not change the order of tasks in LO mode to avoid having to recompute (6.2.2), clearly, at the cost of a lesser accuracy.

Now, we can obtain an upper bound on x_k reshaping (6.2.4) to:

$$\sum_{\substack{i=1\\ \chi_i=HI}}^{k} \frac{\Delta C_i}{T_i} + \frac{1}{(1-x_k)\cdot D_k}\left(\sum_{\substack{i=1\\ \chi_i=HI}}^{k-1} \frac{T_i - \Delta D_i}{T_i}\Delta C_i + \frac{T_k - (1-x_k)\cdot D_k}{T_k}\Delta C_k\right) \leq 1.$$

Solving for $1 - x_k$ and reshaping as before, we obtain:

$$1 - \frac{\sum\limits_{\substack{i=1\\ \chi_i=HI}}^{k-1} \frac{T_i-\Delta D_i}{T_i}\Delta C_i + \Delta C_k}{D_k\left(1 - \sum_{i=1}^{k-1}\frac{\Delta C_i}{T_i}\right)} \geq x_k. \tag{6.2.7}$$

Similar to before, to avoid having to recompute (6.2.4), we need to prevent that the order of tasks changes, for which we derive the following upper bound on x_k:

$$1 - \frac{(1-x_{k-1})\cdot D_{k-1}}{D_k} \geq x_k. \tag{6.2.8}$$

Finally, a system is feasible under mixed-criticality EDF, if (6.2.2) holds for all k and (6.2.3) holds for all k with $\chi_k = HI$. In addition, we need to find a valid x_k for every τ_k with $\chi_k = HI$. That is, x_k must be greater than or equal to the maximum between (6.2.5) and (6.2.6). Simultaneously, x_k must be less than or equal to the minimum between (6.2.7) and (6.2.8).

6.2.2. Uniform deadline scaling

We can also apply Devi's test to derive a second approximated variant of our proposed approach, computing only one deadline scaling factor for all HI tasks in τ. In principle, this requires less computation than our first approximated variant. However, in contrast to what is expected, this second variant does not result in a lower complexity, since it cannot prevent the order of tasks from changing as discussed below.

Schedulability in LO mode. Similar to before, we apply Devi's test to the demand bound function in (5.1.1), but considering this time a uniform deadline scaling factor denoted by x. This results in:

$$\sum_{i=1}^{k} \frac{C_i^{LO}}{T_i} + \frac{1}{x \cdot D_k} \left(\sum_{\substack{i=1 \\ \chi_i = LO}}^{k} \frac{T_i - D_i}{T_i} C_i^{LO} + \sum_{\substack{i=1 \\ \chi_i = HI}}^{k} \frac{T_i - x \cdot D_i}{T_i} C_i^{LO} \right) \leq 1, \tag{6.2.9}$$

where $1 \leq k \leq |\tau|$ and $|\tau|$ denote the total number of tasks in τ. Tasks in (6.2.9) need to be sorted in order of non-decreasing real deadlines D_i for $\chi_i = LO$ or virtual deadlines $x \cdot D_i$ for $\chi_i = HI$. Although the relative order of HI tasks (among themselves) never changes, the order of LO tasks with respect to HI tasks might still change for some x. If that happens, (6.2.9) needs to be recomputed for the corresponding tasks.

In (6.2.9), note that the current τ_k is considered to be a HI task, hence, the second term of (6.2.9) is divided by $x \cdot D_k$. If the current τ_k is a LO, this term will be divided by only D_k leading to:

$$\sum_{i=1}^{k} \frac{C_i^{LO}}{T_i} + \frac{1}{D_k} \left(\sum_{\substack{i=1 \\ \chi_i = LO}}^{k} \frac{T_i - D_i}{T_i} C_i^{LO} + \sum_{\substack{i=1 \\ \chi_i = HI}}^{k} \frac{T_i - x \cdot D_i}{T_i} C_i^{LO} \right) \leq 1. \tag{6.2.10}$$

Schedulability in stable HI mode. When considering a uniform deadline scaling factor x, we still obtain (6.2.3) as a result of applying Devi's test to (5.1.5). Hence, this case requires no further discussion.

Schedulability in the transition from LO to HI mode. The demand bound function in (5.1.7) reduces to the following condition after applying Devi's test for a uniform deadline scaling factor x:

$$\sum_{\substack{i=1 \\ \chi_i = HI}}^{k} \frac{\Delta C_i}{T_i} + \frac{1}{(1-x) \cdot D_k} \left(\sum_{\substack{i=1 \\ \chi_i = HI}}^{k} \frac{T_i - (1-x) \cdot D_i}{T_i} \Delta C_i \right) \leq 1, \tag{6.2.11}$$

with $1 \leq k \leq |\tau|$, and ΔC_i is defined as before. Only HI task are considered in (6.2.11) and tasks are sorted in the order of non-decreasing $(1-x) \cdot D_i$. This time, however, the order of tasks cannot change for different values of x, since x affects all deadlines the same.

Finding a uniform deadline scaling factor. We can obtain two lower bounds on x depending on whether we operate on (6.2.9) or (6.2.10). From (6.2.9), i.e., for $\chi_k = HI$, we obtain:

$$\frac{\sum\limits_{\substack{i=1\\ \chi_i=LO}}^{k} \frac{T_i-D_i}{T_i} C_i^{LO} + \sum\limits_{\substack{i=1\\ \chi_i=HI}}^{k} C_i^{LO}}{D_k\left(1-\sum_{i=1}^{k}\frac{C_i^{LO}}{T_i}\right) + \sum\limits_{\substack{i=1\\ \chi_i=HI}}^{k} D_i \frac{C_i^{LO}}{T_i}} \leq x. \tag{6.2.12}$$

Similarly, we obtain the following lower bound on x by operating on (6.2.10), i.e., for the case $\chi_k = LO$:

$$\frac{\sum\limits_{\substack{i=1\\ \chi_i=LO}}^{k} \frac{T_i-D_i}{T_i} C_i^{LO} + \sum\limits_{\substack{i=1\\ \chi_i=HI}}^{k} C_i^{LO} - D_k\left(1-\sum_{i=1}^{k}\frac{C_i^{LO}}{T_i}\right)}{\sum\limits_{\substack{i=1\\ \chi_i=HI}}^{k} D_i \frac{C_i^{LO}}{T_i}} \leq x. \tag{6.2.13}$$

Note that the order between some HI and LO tasks might change for a given x, requiring us to recompute (6.2.9). As mentioned above, in contrast to the per-task deadline scaling, it becomes difficult to derive an additional lower bound on x that prevents this from happening. We can, of course, select an x for the current τ_k such that $D_{k-1}^{LO} \leq x \cdot D_k$ holds, where D_{k-1}^{LO} represents τ_{k-1}'s deadline in LO mode (independent of whether this is a LO or HI task). However, this is not sufficient, since a new value of x also affects all previously tested HI tasks, whose deadlines might become smaller than some deadline of a LO task.

An upper bound on x can be obtained from (6.2.11), which leads to:

$$x \leq 1 - \frac{\sum\limits_{\substack{i=1\\ \chi_i=HI}}^{k} \Delta C_i}{D_k\left(1-\sum\limits_{\substack{i=1\\ \chi_i=HI}}^{k}\frac{\Delta C_i}{T_i}\right) + \sum\limits_{\substack{i=1\\ \chi_i=HI}}^{k} D_i \frac{\Delta C_i}{T_i}}. \tag{6.2.14}$$

According to this second variant of our proposed approach, the system is feasible under mixed-criticality EDF, if (6.2.9) holds for all k and (6.2.3) holds for all k in $1 \leq k \leq |\tau|$ with $\chi_k = HI$. In addition, none of the values of x obtained with (6.2.14) for $1 \leq k \leq |\tau|$ with $\chi_k = HI$ should be less than any value obtained either by (6.2.12) or by (6.2.13) for $1 \leq k \leq |\tau|$.

6.2.3. Complexity

Similar to GREEDY [42] and ECDF [40], the proposed approach of Section 5.1 is based on computing demand bound functions and, hence, has a pseudo-polynomial complexity $\mathcal{O}(K_n \cdot n)$ for task sets with a total utilization that is strictly less than 1 [17]. Note that n

represents the number of tasks in the given task set and K_n is a *factor* that depends on task parameters and, therefore, on n. In contract to this, EDF-VD [8] has a linear complexity $\mathcal{O}(n)$.

The approximated variant in Section 6.2.1, based on computing per-task deadline scaling factors, has a polynomial complexity $\mathcal{O}(n \cdot \log n)$, when the bounds in (6.2.6) and (6.2.8) are considered. As explained above, these prevent that the order of tasks changes for a newly computed deadline scaling factor. If (6.2.6) and (6.2.8) are not considered, the order of tasks might change requiring us to retest some of the (previously tested) tasks and, hence, leading to a quadratic complexity $\mathcal{O}(n^2)$ instead.

On the other hand, the complexity of our second approximated variant of Section 6.2.2 is quadratic $\mathcal{O}(n^2)$, since, in the general case, i.e., with at least one LO task in the system, it cannot guarantee that the order of tasks does not change. As a result, retesting cannot be avoided.

6.3. Findings of this Chapter

In this chapter, we proposed applying approximaiton techniques to the demand bound functions derived in Chapter 5. The rationale behind this is to reduce the complexity of the resulting algorithms for the sake of a reduced runtime. In contrast to that of Chapter 5, the approximated algorithms of this chaper with either a per-task or a uniform deadline scaling are more suitable for online tests in a MC setting. On the other hand, such faster tests are also more benefitial for design-space exploration, where they are repeatedly run and can considereably reduce the overall time required.

Chapter 7.

Evaluation and Results

This chapter evaluates our proposed approaches from Chapter 4, Chapter 5 and Chapter 6, and compares them with other state-of-the-art approaches from the literature. We perform a simulation-based evaluation of the three proposed thechniques on the basis of synthetic data.

7.1. Mixed-Criticality EDF

In contrast to EDF-VD, where deadlines are assumed to be equal to periods, we now allow a deadline D_i with $D_i \leq T_i$. In addition, we now assign a *virtual deadline* equal to $x_i \cdot D_i$ with $x_i \in (0, 1)$ to all τ_i with $\chi_i = HI$.

Similar to EDF-VD, this virtual deadline is used instead of D_i — the *real* deadline — to schedule HI tasks in LO mode. The parameter x_i is now a *per-task* deadline scaling factor. There is no deadline scaling for LO tasks such that they are scheduled (only in LO mode) using their D_i.

When the system switches to HI mode, HI tasks start being scheduled according to their real deadlines D_i whereas LO tasks are discarded immediately. Clearly, whereas schedulability of separated modes can be easily tested, i.e., when the system is stable in either LO or HI mode, it is difficult to test schedulability of transitions between modes. In particular, careful analysis is required when the system switches from LO to HI mode.

In this work, similar to other approaches from the literature, transitions from HI back to LO mode are disregarded. The reason is that, in contrast to changes from LO to HI, a change from HI to LO mode can be programmed or postponed to a suitable point in time, e.g., at which the processor idles after all HI tasks have run again for their optimistic WCETs, and does not require further analysis.

7.2. Obtaining Test Data

This section briefly describes how test data was created for this comparison. This description is common to all curves presented next, however, we distinguish between the case $D_i = T_i$ and $D_i \leq T_i$. Details concerning a specific curve will be given as it becomes necessary.

7.2.1. The Case $D_i = T_i$

For comparing the different algorithms in the case where $D_i = T_i$ for all tasks, we generate a large number of task sets. In particular, each data-point of the curves shown below 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, i.e., $\frac{C_i^{LO}}{T_i}$, for each task in a set.

We then use a log-uniform distribution approach proposed by Emberson *et al.* [45] to generate the task periods T_i with a factor of 1000 difference between the minimum and maximum possible task period. This represents a range of task periods from $1ms$ to $1000ms$ as found in most hard real-time applications such as automotive and aerospace. The log-uniform distribution generates an equal number of tasks in each time band (e.g., $1-10ms$, $10-100ms$ etc.), thus providing reasonable correspondence with real systems.

We later use the obtained values of $\frac{C_i^{LO}}{T_i}$ and T_i to compute a corresponding value of C_i^{LO}. Further, given a percentage of HI tasks, which we vary in our experiments, we assumed that HI tasks experience a random increase in execution demand in HI mode of either 10% or 100% more of their respective $\frac{C_i^{LO}}{T_i}$. With this, we finally obtained the values of C_i^{HI} that match that increase.

7.2.2. The Case $D_i \leq T_i$

We again used the algorithm *UUniFast* [23] [24] to generate sets with different numbers of tasks for a varying *LO utilization*, i.e., for a varying $U_{LO}^{LO} + U_{HI}^{LO}$. For each curve, a total number of 5,000 different task sets were created.

For a given value of LO utilization, UUniFast returns a vector of (individual) task utilizations as explained before. We then generate periods T_i randomly based on the log-uniform distribution [45] and use the task utilization to obtain the values of C_i^{LO} similar to the previous case. Now, given a percentage of HI tasks, we assumed again that HI tasks experience a random increase in execution demand in HI mode of either 10% or 100% more of their $\frac{C_i^{LO}}{T_i}$ to obtain the values of C_i^{HI}. Finally, we randomly selected D_i in $[C_i^{HI}, T_i]$ for HI tasks and in $[C_i^{LO}, T_i]$ for LO tasks.

7.3. Weighted schedulability

In the following, we make use of the concept of *weighted schedulability* [22] [36] to analyze performance by the different algorithms. That is, for a schedulability test y whose accuracy on testing a task set τ is a function of parameter p, e.g., the number of tasks per set, etc., its weighted schedulability $W_y(p)$ is given by:

$$\mathrm{W}_y(p) \quad = \quad \frac{\sum\limits_{\forall\tau} U(\tau) \cdot \mathrm{S}_y(\tau, p)}{\sum\limits_{\forall\tau} U(\tau)}, \tag{7.3.1}$$

where U(τ) is the utilization of a given τ and $S_y(\tau, p)$ is y's binary result (1 if schedulable and 0 if not) for a task set τ with parameter value p. In other words, individual schedulability results by y are weighted according to the utilization of the task sets tested, putting more emphasis on higher-utilization ones.

We created weighted schedulability curves varying following parameters: (i) total number of tasks, (ii) percentage of HI tasks, (iii) increase of HI execution demand and (iv) range of task periods. Every time we varied one of these parameters, we generated 1000 different task sets for each LO utilization value between 0 and 100% at steps of 10%, i.e., a total of 10,000 task sets per marker on the shown curves. To this end, we proceeded as described previously to obtain task parameters.

7.4. Algorithms in this Comparison

In this section, we introduce and briefly explain the algorithms to which we compare the proposed ones in this evaluation.

7.4.1. The EDF-VD and DEDF-VD Algorithms

As discussed above, EDF with Virtual Deadlines (EDF-VD) assumes that deadlines are equal to periods $D_i = T_i\ \forall i$. Under EDF-VD, virtual deadlines are obtained by $x \cdot D_i$, where $x \in (0, 1)$ is a uniform deadline scaling factor for all HI tasks [10].

Based on this, the LO and HI tasks are schedulable with their corresponding C_i^{LO} under EDF in LO mode. Similarly, in HI mode, the HI tasks also need to be schedulable under EDF, but with their corresponding C_i^{HI} instead, leading to schedulability conditions as shown in Section 2.5.1.

Now, since we are concerned with the case $D_i \leq T_i$ for all tasks, we need to adapt EDF-VD for this case. We call the resulting algorithm Density EDF-VD or DEDF-VD. This latter is based on the concept of density. $\frac{C_i^{LO}}{D_i}$ and $\frac{C_i^{LO}}{xD_i}$ then characterize LO and HI tasks in LO mode, whereas $\frac{C_i^{HI}}{D_i}$ characterizes HI tasks in HI mode. This all leads to:

$$\delta_{LO}^{LO} = \sum_{\chi_i = LO} \frac{C_i^{LO}}{D_i},$$

$$\delta_{HI}^{LO} = \sum_{\chi_i = HI} \frac{C_i^{LO}}{x \cdot D_i},$$

and

$$\delta_{HI}^{HI} = \sum_{\chi_i = HI} \frac{C_i^{HI}}{D_i}.$$

As a result, DEDF-VD's schedulability conditions now turn to the following:

$$\delta_{LO}^{LO} + \delta_{LO}^{HI} \leq 1, \tag{7.4.1}$$

$$\delta_{HI}^{HI} \leq 1, \tag{7.4.2}$$

Note that these two conditions do not ensure schedulability in the transition phase. Hence, they are indeed only necessary, but not sufficient conditions.

In [8], 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 feasible lower and upper bounds can be obtained on x [8]:

$$\frac{\delta_{LO}^{HI}}{1-\delta_{LO}^{LO}} \;\leq\; x, \tag{7.4.3}$$

$$x \;\leq\; \frac{1-\delta_{HI}^{HI}}{\delta_{LO}^{LO}}. \tag{7.4.4}$$

Similar to EDF-VD, if the value of x obtained with (7.4.3) is less than or equal to the value obtained with (7.4.4), then it is possible to find a valid x for the considered system rendering it schedulable under DEDF-VD.

7.4.2. The GREEDY algorithm

In LO mode, note that $\mathrm{dbf}_{LO}(t)$ in (5.1.1) is identical to that of Ekberg and Yi — denoted by $\mathrm{gdbf}_{LO}(t)$ in this work. That is, $\mathrm{dbf}_{LO}(t) = \mathrm{gdbf}_{LO}(t)$ for all $0 < t \leq \hat{t}_{LO}$.

In HI mode, Ekberg and Yi proposed a demand bound function — denoted $\mathrm{gdbf}_{HI}(t)$ in this study — which is given by the following expression [42]:

$$\mathrm{gdbf}_{HI}(t) \;=\; \sum_{\chi_i=HI}\left(\left\lfloor\frac{t-\Delta D_i}{T_i}\right\rfloor+1\right)C_i^{HI} - \sum_{\chi_i=HI}\mathrm{done}_i(t), \tag{7.4.5}$$

with $\Delta D_i = D_i - x_i \cdot D_i$ and $\mathrm{done}_i(t)$ is given by:

$$\mathrm{done}_i(t) = \begin{cases}\max\left(0, C_i^{LO} - \mathrm{mod}\!\left(\frac{t}{T_i}\right) + \Delta D_i\right), & \text{if } D_i > \mathrm{mod}\!\left(\frac{t}{T_i}\right) \geq \Delta D_i,\\ 0, & \text{otherwise.}\end{cases}$$

Note that $\mathrm{gdbf}_{HI}(t)$ bounds the execution demand in HI mode taking transitions into account. In our case, as discussed above, we derive different bounds on the execution demand at transitions and in stable HI mode, viz., $\mathrm{dbf}_{SW}(t)$ and $\mathrm{dbf}_{HI}(t)$ respectively.

7.4.3. The ECDF algorithm

Similar to the case of the GREEDY algorithm, note that $\mathrm{dbf}_{LO}(t)$ in (5.1.1) is identical to that used in the ECDF algorithm in LO mode. We denote this latter by $\mathrm{edbf}_{LO}(t)$ in this work. That is, $\mathrm{dbf}_{LO}(t) = \mathrm{edbf}_{LO}(t)$ for all $0 < t \leq \hat{t}_{LO}$.

In HI mode, the ECDF algorithm uses a demand bound function — denoted $\mathrm{edbf}_{HI}(t)$ in this work — which is given by the following expression [40]:

$$\begin{aligned}\mathrm{edbf}_{HI}(t_1,t_2) &= \min\left(t_1, \sum_{j=1}^{3}\mathrm{dbf}_{L_j}(t_1,t_2)\right) \\ &+ \sum_{\substack{\chi_i=HI \\ \text{and case 2 or 3}}} \mathrm{dbf}_i^{HI}(t_1,t_2) \\ &+ \sum_{\substack{\chi_i=HI \\ \text{and case 2}}} \left(CO(t_2-t_1)+\Delta C_i\right), \qquad (7.4.6)\end{aligned}$$

where ΔC_i is defined as $C_i^{HI} - C_i^{LO}$. In addition, $0 \leq t_2 \leq \hat{t}_{HI}$ and $0 \leq t_1 \leq t_2 - \min_{\chi_i=HI}(\Delta D_i)$ hold with again $\Delta D_i = D_i - x_i \cdot D_i$.

Here t_1 represents the point in time at which the system switches to HI mode (i.e., $t_1 = t'$ in our notation) and t_2 is the point in time at which a deadline is potentially missed (i.e., $t_2 = t_{miss} \leq t''$ in this work). Note that $\mathrm{dbf}_i^{HI}(\cdot)$ is a τ_i's contribution to $\mathrm{dbf}_{HI}(\cdot)$ shown in (5.1.4). HI tasks in (7.4.6) are classified into three cases: case 1 which plays a role in computing $\mathrm{dbf}_{L_j}(\cdot)$, case 2, and case 3. For details on how to compute $\mathrm{dbf}_{L_j}(\cdot)$ for $1 \leq j \leq 3$ and how to compute $CO(\cdot)$ we refer to [40].

7.5. Evaluation of Utilization Caps

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 in the case where $D_i = T_i$ holds for all tasks. The choice of Alg. 1 over Alg. 2 was taken to facilitate comparison based on synthetic data. Alg. 1 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 discussed in Chapter4, we assume that tasks in τ are partitioned or grouped according to some functional dependency. For the sake of comparison, however, we use 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. 7.1 to Fig. 7.12. 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.

7.5.1. 10 tasks per task set

Fig. 7.1 to Fig. 7.4 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. 7.1 and 7.3 to 10%, i.e., 1 task, in Fig. 7.2 and 7.4. In addition, we vary the amount of execution demand from 100% in Fig. 7.1 and Fig. 7.2, i.e., HI tasks double their execution demand in HI mode, to 10% in Fig. 7.3 and Fig. 7.4, i.e., HI tasks have an increase of 10% more execution demand in HI mode.

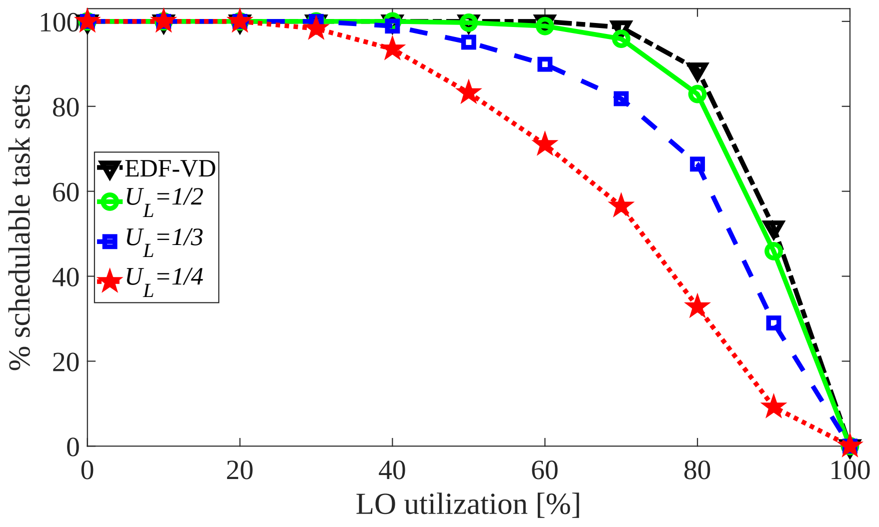


Figure 7.1.: Schedulability vs. LO utilization for 10 tasks, 50% of HI tasks, 100% increase of execution demand in HI mode

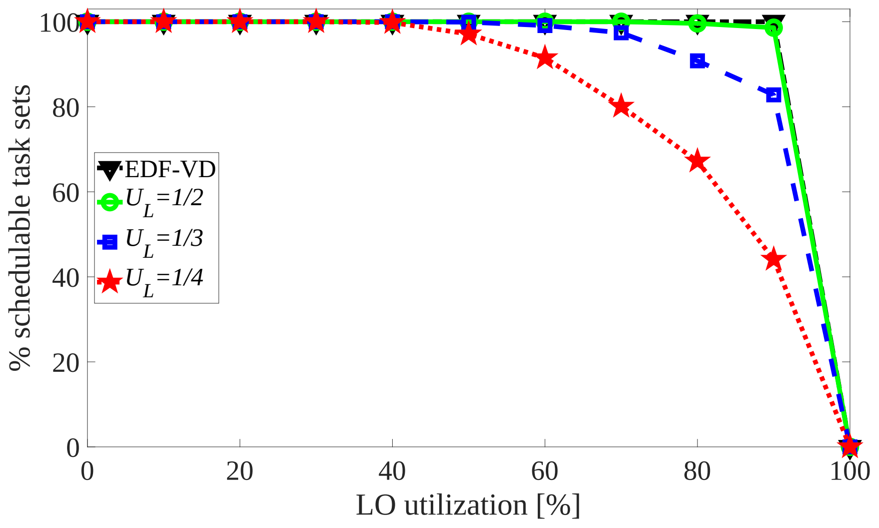


Figure 7.2.: Schedulability vs. LO utilization for 10 tasks, 10% of HI tasks, 100% increase of execution demand in HI mode

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.

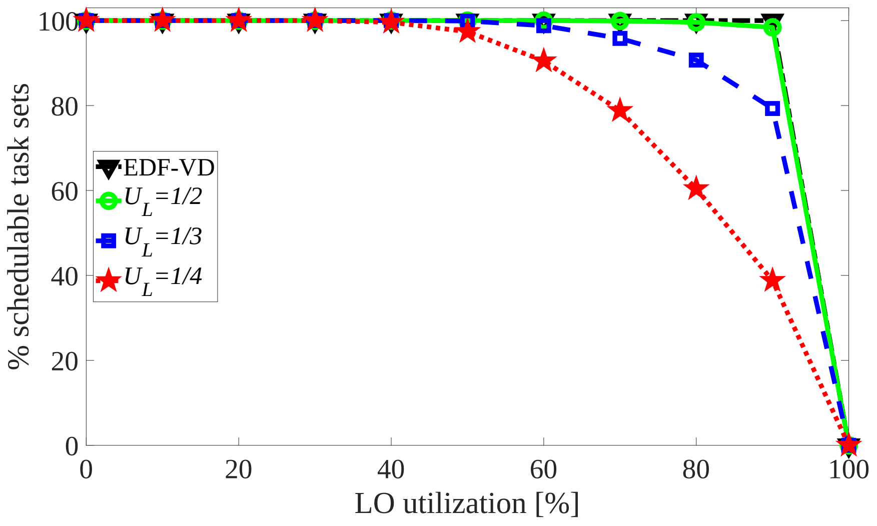


Figure 7.3.: Schedulability vs. LO utilization for 10 tasks, 50% of HI tasks, 10% increase of execution demand in HI mode

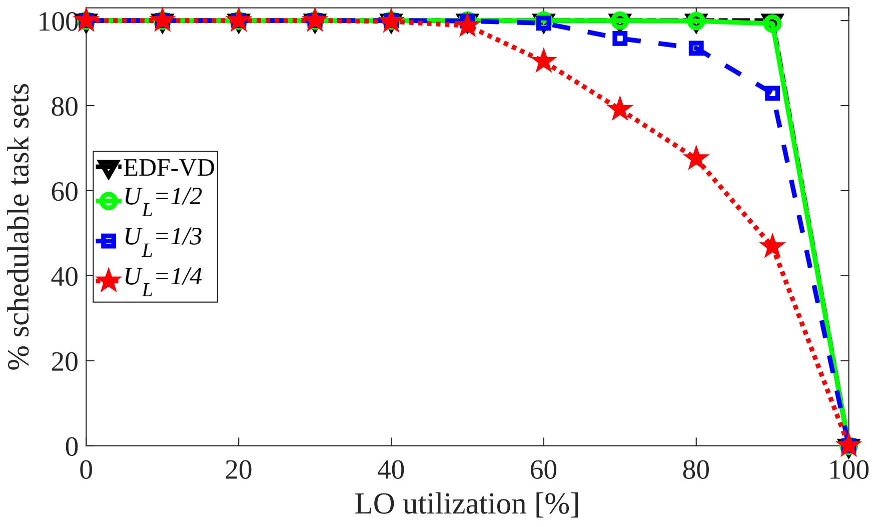


Figure 7.4.: Schedulability vs. LO utilization for 10 tasks, 10% of HI tasks, 10% increase of execution demand in HI mode

7.5.2. 20 tasks per task set

Fig. 7.5 to Fig. 7.8 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. 7.5 and 7.7 to 10%, i.e., 2 tasks, in Fig. 7.6 and 7.8. The amount of execution demand was varied from 100% in Fig. 7.5 and 7.6, i.e., HI tasks double their execution demand in HI mode, to 10% in Fig. 7.7 and Fig. 7.8, i.e., HI tasks have an increase of 10% more execution demand in HI mode.

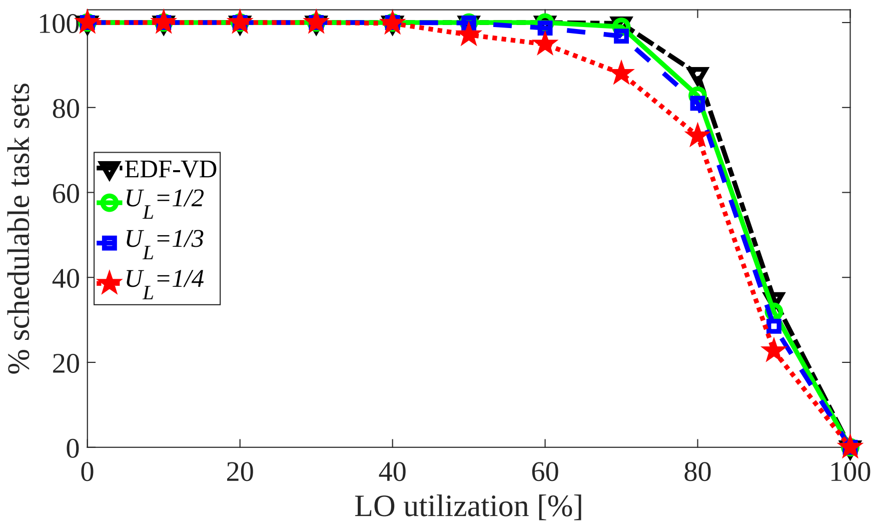


Figure 7.5.: Schedulability vs. LO utilization for 20 tasks, 50% of HI tasks, 100% increase of execution demand in HI mode

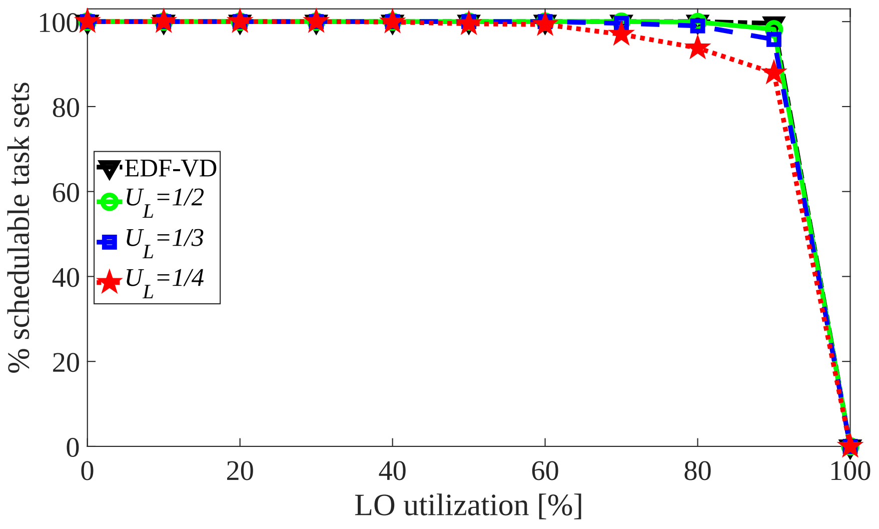


Figure 7.6.: Schedulability vs. LO utilization for 20 tasks, 10% of HI tasks, 100% increase of execution demand in HI mode

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. 7.5, where the total HI execution demand is much higher than for Fig. 7.6 to Fig. 7.8.

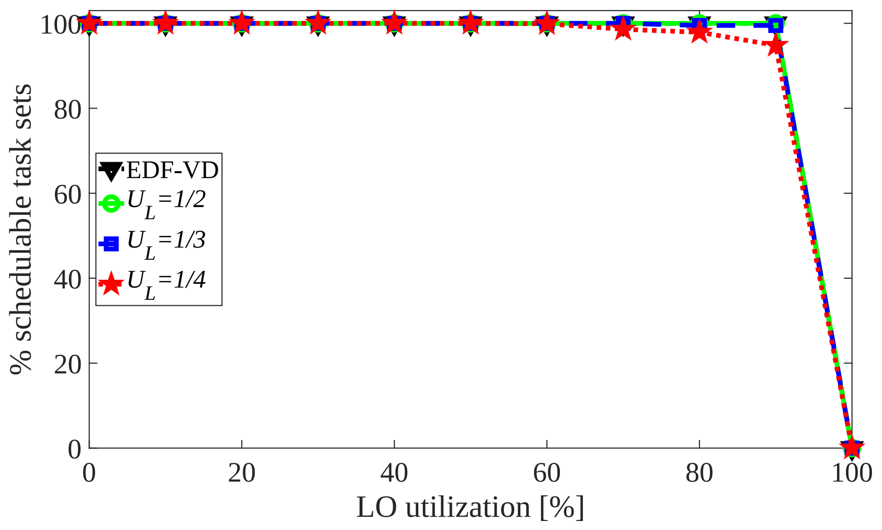


Figure 7.7.: Schedulability vs. LO utilization for 20 tasks, 50% of HI tasks, 10% increase of execution demand in HI mode

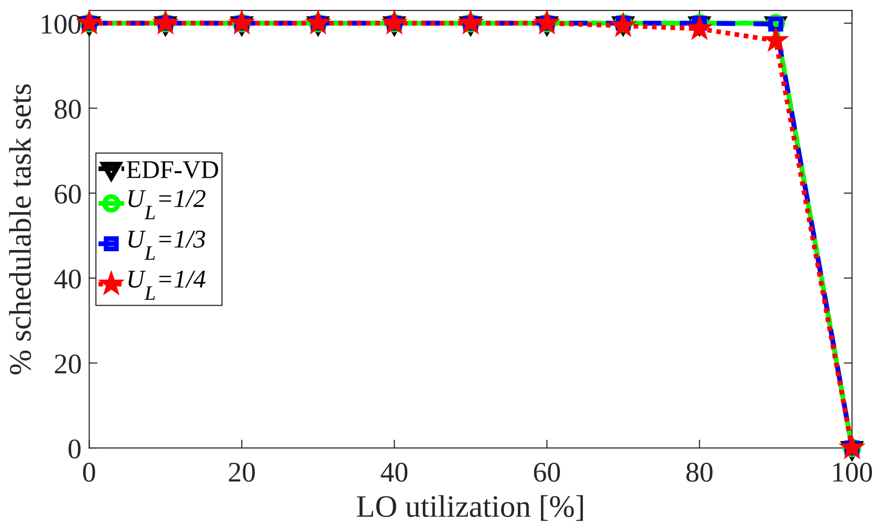


Figure 7.8.: Schedulability vs. LO utilization for 20 tasks, 10% of HI tasks, 10% increase of execution demand in HI mode

7.5.3. 50 tasks per task set

Similar to the case of 10-task and 20-task sets, Fig. 7.9 to Fig. 7.12 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. 7.9 and 7.11 to 10%, i.e., 5 tasks, in Fig. 7.10 and 7.12. The amount of execution demand was varied from 100% in Fig. 7.9 and 7.10, i.e., HI tasks have twice their LO execution demand in HI mode, to 10% in Fig. 7.11 and Fig. 7.12, i.e., HI tasks have an increase of 10% more execution demand in HI mode.

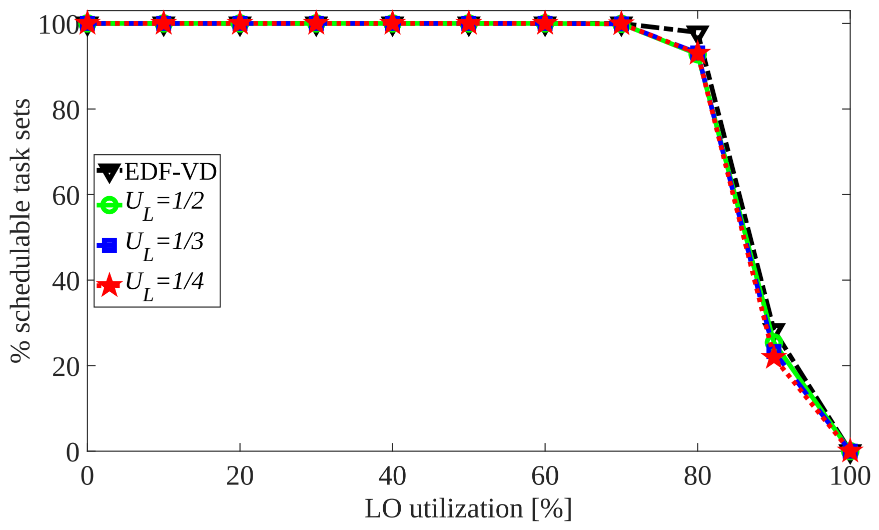


Figure 7.9.: Schedulability vs. LO utilization for 50 tasks, 50% of HI tasks, 100% increase of execution demand in HI mode

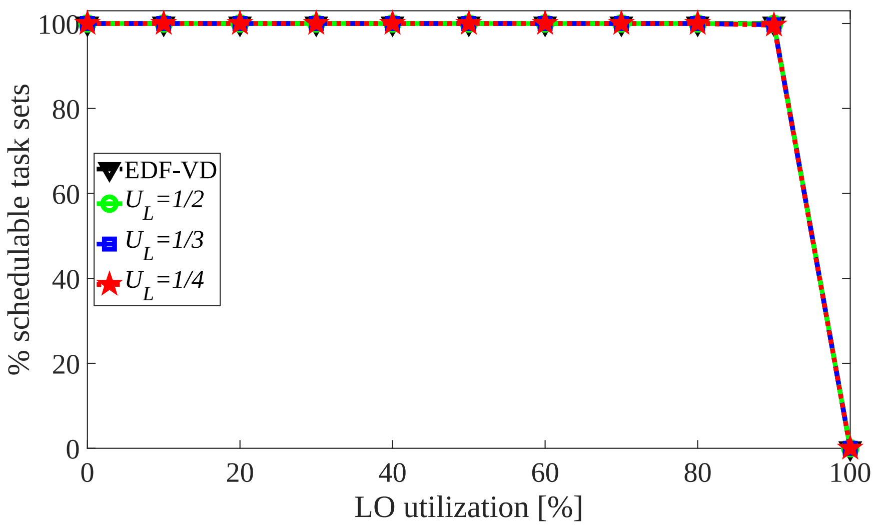


Figure 7.10.: Schedulability vs. LO utilization for 50 tasks, 10% of HI tasks, 100% increase of execution demand in HI mode

We can see that, this time, all algorithms behave almost the same. Only in Fig. 7.9, where the greatest HI execution demand is considered, they still have some difference in the number of task sets they render schedulable. In Fig. 7.10 to 7.12, algorithms' performance curves almost overlap fully showing a negligible difference with respect to EDF-VD.

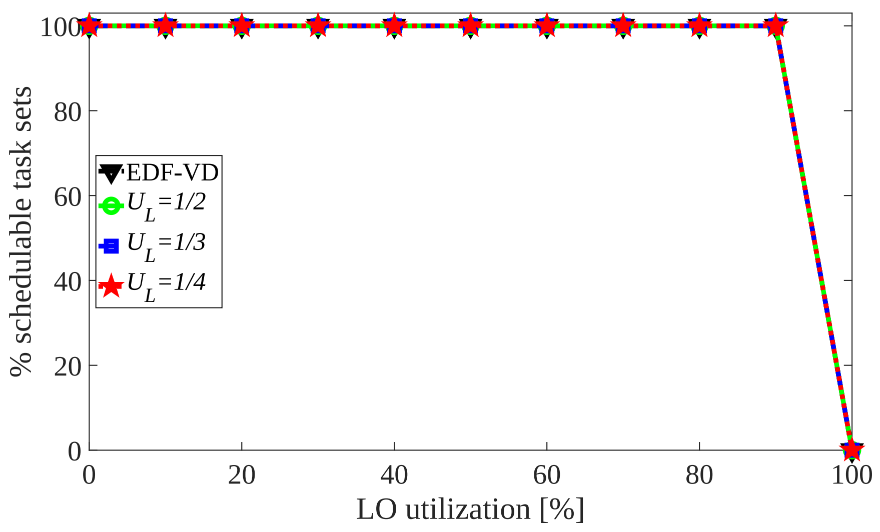


Figure 7.11.: Schedulability vs. LO utilization for 50 tasks, 50% of HI tasks, 10% increase of execution demand in HI mode

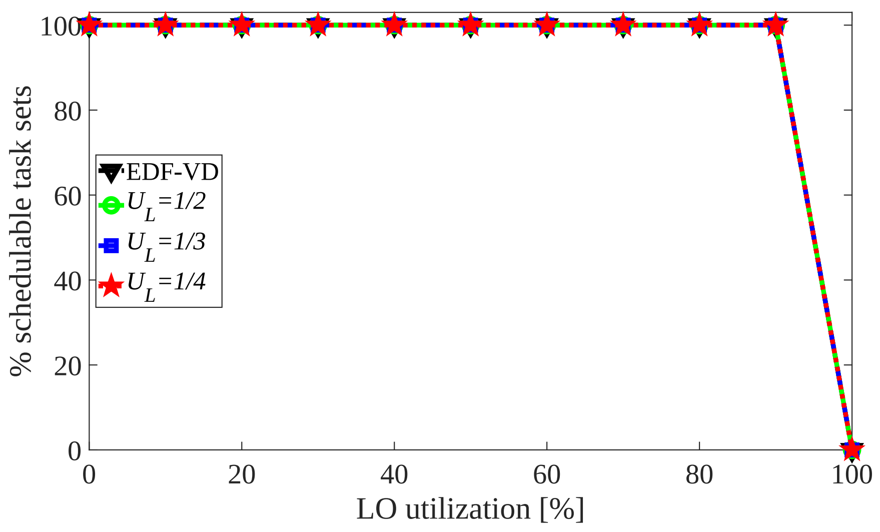


Figure 7.12.: Schedulability vs. LO utilization for 50 tasks, 10% of HI tasks, 10% increase of execution demand in HI mode

7.5.4. Comparison of runtime

In this section, we compare runtime versus utilization and versus number of tasks by our approach as given in Alg. 1, i.e., with fixed utilization caps, to the original EDF-VD algorithm. Again, the purpose is to show relative behavior rather than providing absolute timing values. To this end, all algorithms were implemented in Matlab and executed on a computer featuring an Intel Core i5 processor.

Fig. 7.13 compares average runtime versus utilization for a fixed number of task ($n = 10$) and increasing utilization considering 10,000 different task sets. As it can be observed, $U_L = 1/3$

and $U_L = 1/4$ are around two or more times faster than $U_L = 1/2$, however, they are also almost one or two times slower than EDF-VD.

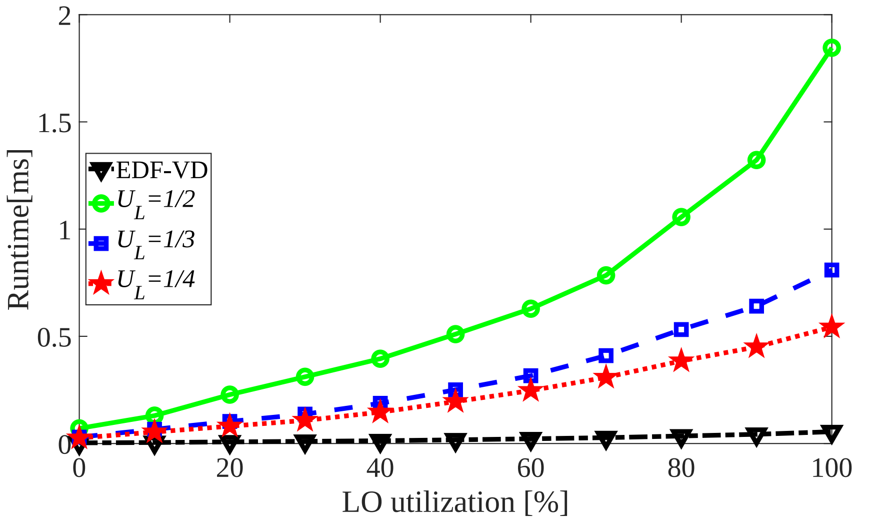


Figure 7.13.: Runtime vs. utilization for $n = 10$ and 50% of HI tasks with 100% increase of HI execution demand

Fig. 7.14 depicts runtime as the number of tasks grows. Again, $U_L = 1/3$ and $U_L = 1/4$ remain approximately two or more times faster than $U_L = 1/2$ as the number of tasks grows, but they are also almost two or more times slower than EDF-VD. In other words, the values of $U_L = 1/3$ and $U_L = 1/4$ provide a comparatively good performance in terms of schedulability at a lesser cost in terms of runtime.

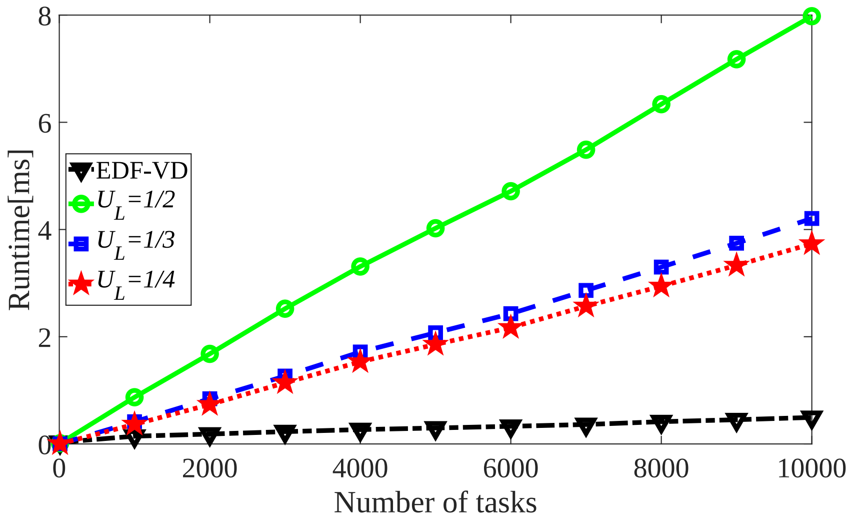


Figure 7.14.: Runtime vs. number of tasks for 50% of HI tasks with 100% increase of HI execution demand

7.6. Evaluation of Execution Demand Bounds

In this section, we evaluate the proposed technique from Chapter 5 based on synthetic data and compare it to the most prominent approaches from the literature. Again, the intention of this section is to show how the different algorithms roughly behave with respect to each other.

In particular, we compare the proposed technique in form of Alg. 3 (denoted by *Proposed* in the next curves) with DEDF-VD [8], with the GREEDY algorithm by Ekberg and Yi [42], and with ECDF by Easwaran [40]. It should be noted that the proposed Alg. 3 as well as GREEDY and ECDF essentially perform two inter-related functions: (i) selection of x_i parameters by some deadline tightening technique, (ii) schedulability test for a given set of x_i. Clearly, the more accurate the schedulability test is, the better the selection of x_i is and vice versa.

As discussed above, the aim of this work is not to improve the deadline tightening, but to propose a new technique to bound execution demand of mixed-criticality EDF. As a result, the proposed Alg. 3 makes use of a rather rudimentary (though less complex) tightening technique compared to those of the GREEDY and the ECDF algorithms. However, on average, experimental results evidence that benefits overcome drawbacks by the proposed Alg. 3.

Finally, as already stated, we had to modify EDF-VD — denoted by DEDF-VD in this section — to consider the deadlines D_i of tasks instead of their inter-arrival times or periods T_i, i.e., to consider the tasks' densities instead of their utilizations, to account for the case of constrained deadlines $D_i \leq T_i$ and be meaningfully compared with the other algorithms in this section.

7.6.1. Comparison for sets of 10 tasks

Fig. 7.15 to Fig. 7.20 show the results of our experiments for 10 tasks and a varying number of HI tasks. In Fig. 7.15, Fig. 7.16 and Fig. 7.17, only one HI task was considered (10% of n) for an increasing amount of HI execution demand.

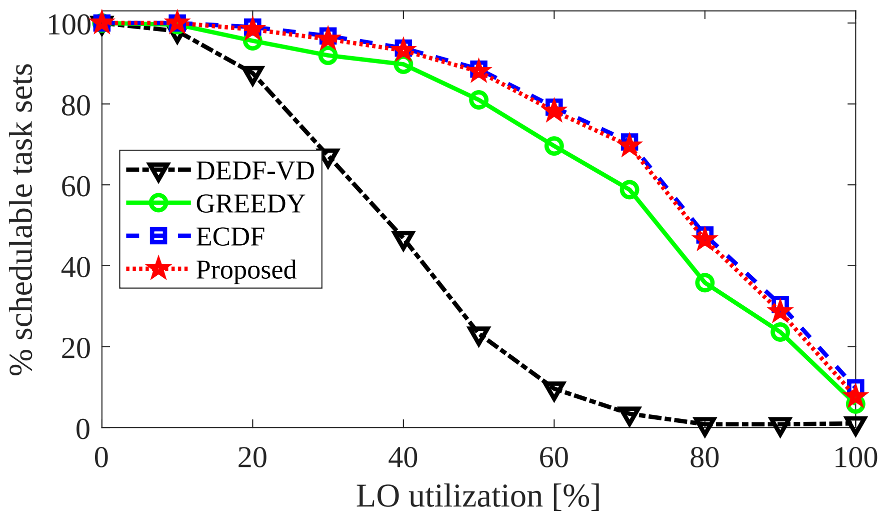


Figure 7.15.: Schedulability vs. LO utilization for n = 10 and 10% of HI tasks with 10% increase of HI execution demand

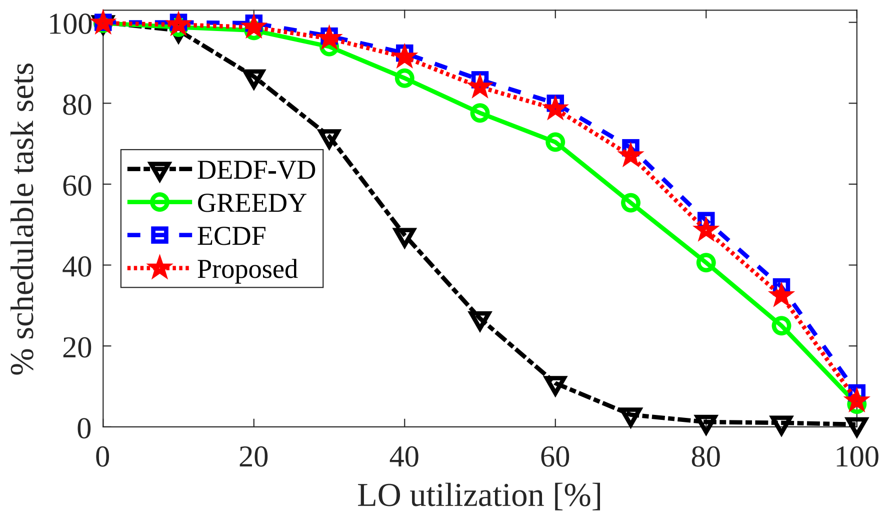


Figure 7.16.: Schedulability vs. LO utilization for $n = 10$ and 10% of HI tasks with 50% increase of HI execution demand

As we can see, in this case, the *Proposed* algorithm and ECDF behave almost the same with ECDF being slightly better for an increase in HI execution demand that goes from 10% in Fig. 7.15 to 100% in Fig. 7.17, i.e., the HI task doubles its execution demand in HI mode. The GREEDY algorithm is outperformed by the *Proposed* algorithm and by ECDF by around 10%, i.e., the *Proposed* algorithm and ECDF constantly find around 10% more task sets that are feasible under mixed-criticality EDF than the GREEDY algorithm.

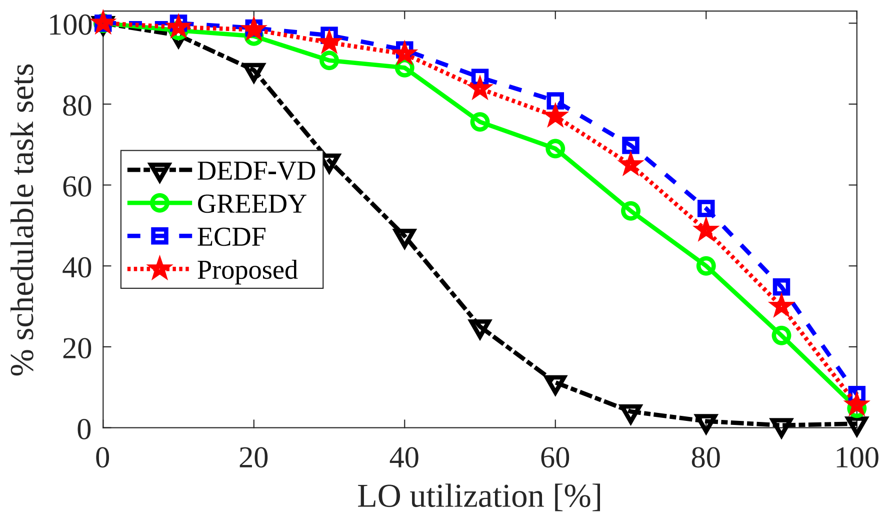


Figure 7.17.: Schedulability vs. LO utilization for $n = 10$ and 10% of HI tasks with 100% increase of HI execution demand

In Fig. 7.18, Fig. 7.19 and Fig. 7.20, three HI tasks were considered (30% of n) again for an increasing amount of HI execution demand. In this case, the *Proposed* algorithm and ECDF behave almost the same; however, the *Proposed* algorithm is most of the time slightly better than ECDF for an increase in HI execution demand that goes from 10% in Fig. 7.18 to 100% in Fig. 7.20, i.e., the three HI tasks double their execution demand in HI mode.

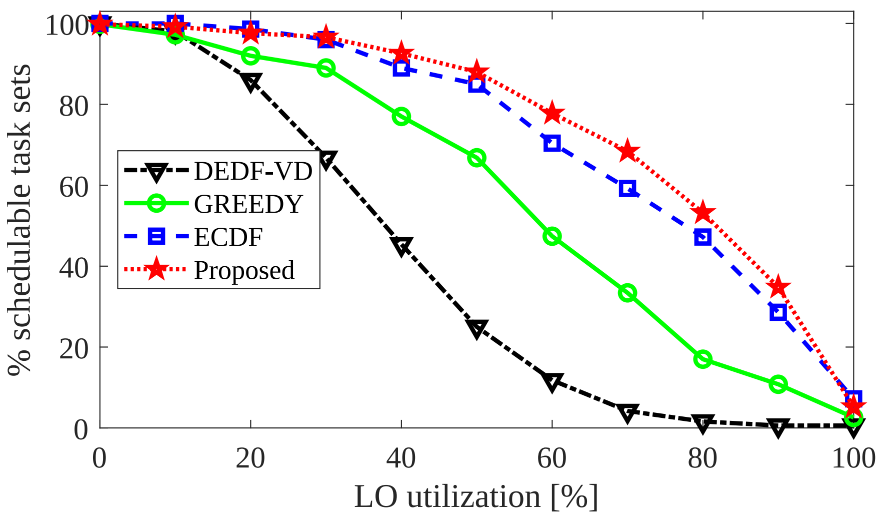


Figure 7.18.: Schedulability vs. LO utilization for $n = 10$ and 30% of HI tasks with 10% increase of HI execution demand

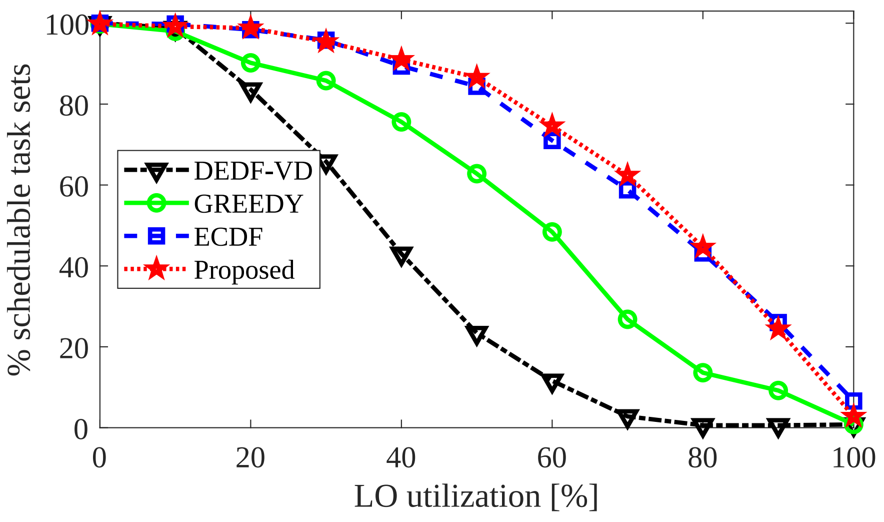


Figure 7.19.: Schedulability vs. LO utilization for $n = 10$ and 30% of HI tasks with 50% increase of HI execution demand

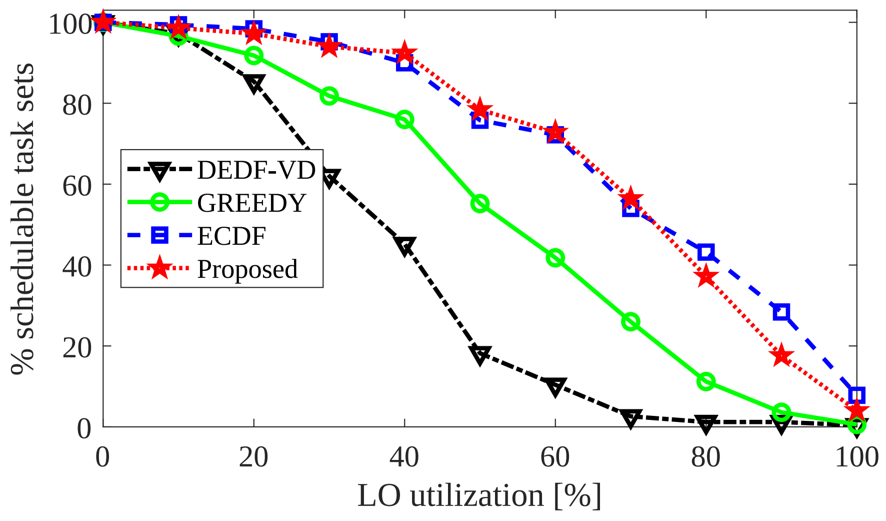


Figure 7.20.: Schedulability vs. LO utilization for $n = 10$ and 30% of HI tasks with 100% increase of HI execution demand

The GREEDY algorithm is outperformed by the *Proposed* and ECDF by around 20% this time, i.e., the *Proposed* algorithm and ECDF constantly find around 20% more task sets that are feasible under mixed-criticality EDF.

Even though there is not much difference between the *Proposed* algorithm and ECDF, it should be noted that ECDF has a more sophisticated deadline tightening technique than Alg. 3. Tightening deadlines in a more efficient manner will certainly improve Alg. 3's performance.

7.6.2. Comparison for sets of 20 tasks

Fig. 7.21 to Fig. 7.26 show the results of our experiments for 20 tasks and a varying number of HI tasks. In Fig. 7.21, Fig. 7.22 and Fig. 7.23, two HI tasks were considered (10% of n) for an increasing amount of HI execution demand.

Here, it can be seen that the *Proposed* algorithm and ECDF behave almost the same with ECDF being again slightly better for an increase in HI execution demand that goes from 10% in Fig. 7.21 to 100% in Fig. 7.23. The GREEDY algorithm is again outperformed by the *Proposed* and the ECDF algorithm by around 10%, i.e., these latter find around 10% more task sets that are feasible under mixed-criticality EDF.

In Fig. 7.24, Fig. 7.25 and Fig. 7.26, six HI tasks were considered (30% of n) for an increasing amount of HI execution demand. This time the *Proposed* algorithm considerably outperforms ECDF by around 10% to 20% more accepted task sets. The GREEDY algorithm is still outperformed by ECDF and the *Proposed* one.

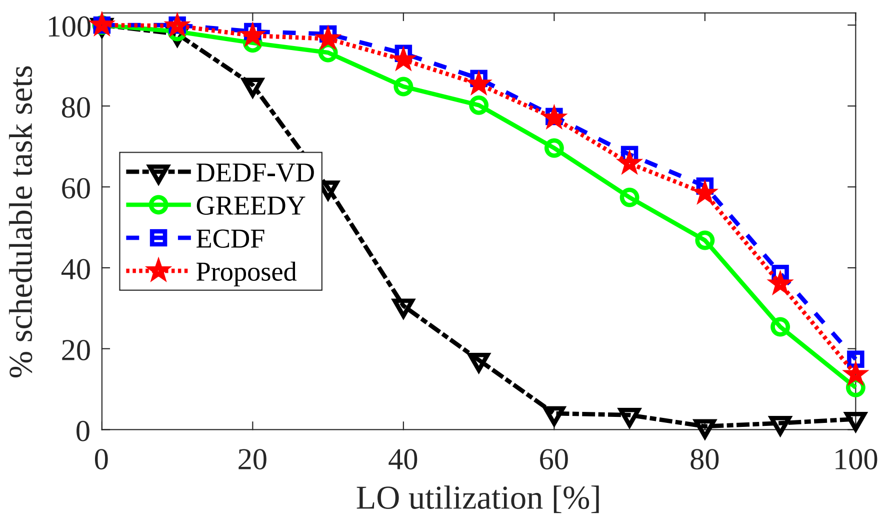


Figure 7.21.: Schedulability vs. LO utilization for $n = 20$ and 10% of HI tasks with 10% increase of HI execution demand

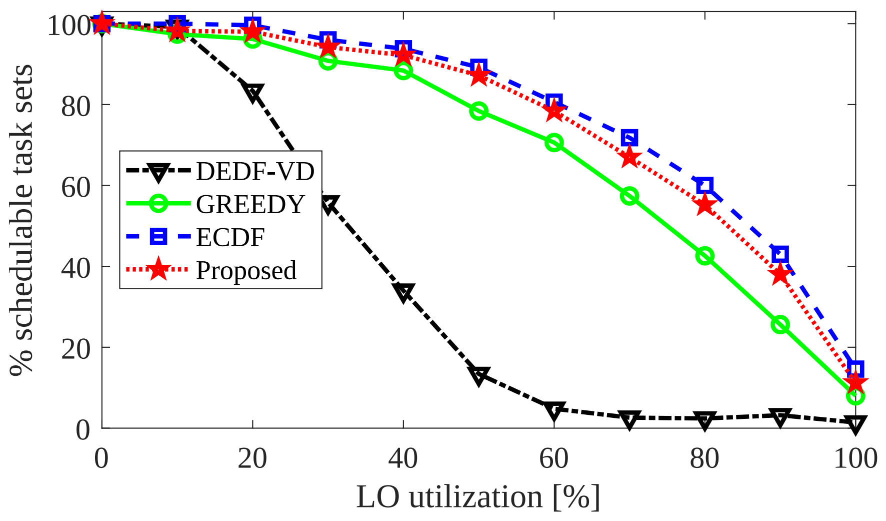


Figure 7.22.: Schedulability vs. LO utilization for $n = 20$ and 10% of HI tasks with 50% increase of HI execution demand

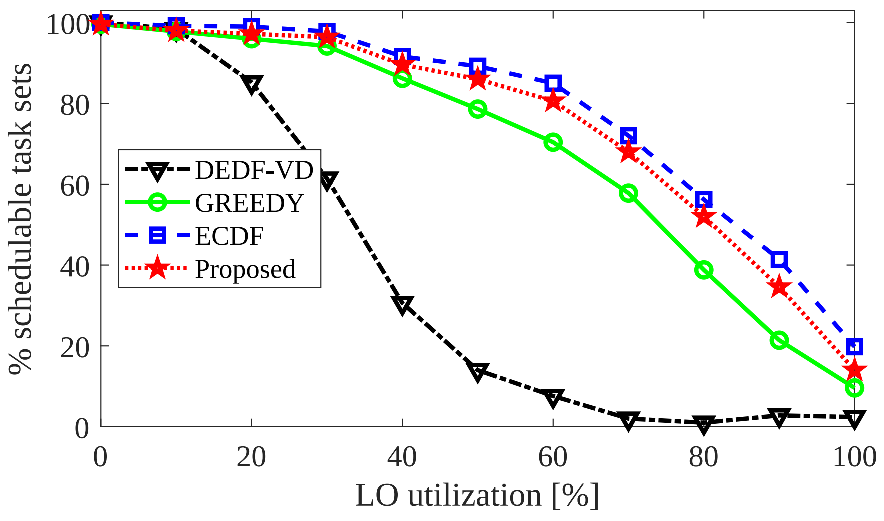


Figure 7.23.: Schedulability vs. LO utilization for $n = 20$ and 10% of HI tasks with 100% increase of HI execution demand

The performance of the *Proposed* algorithm is the highest for the experiments in Fig. 7.24, i.e., where there are a relatively big number of HI tasks, each of which experiences a relatively small increase in execution demand in HI mode. Again, the performance of the *Proposed* Alg. 3 can be further improved by using a more sophisticated deadline tightening scheme.

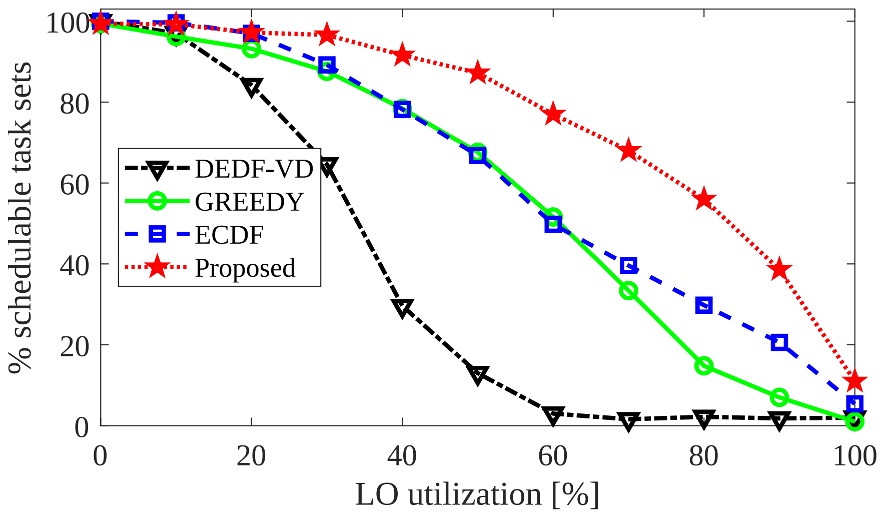


Figure 7.24.: Schedulability vs. LO utilization for $n = 20$ and 30% of HI tasks with 10% increase of HI execution demand

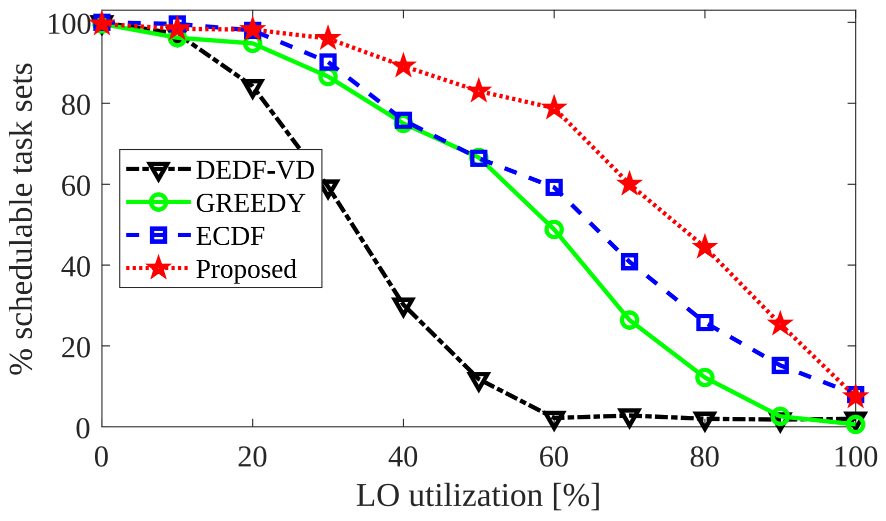


Figure 7.25.: Schedulability vs. LO utilization for $n = 20$ and 30% of HI tasks with 50% increase of HI execution demand

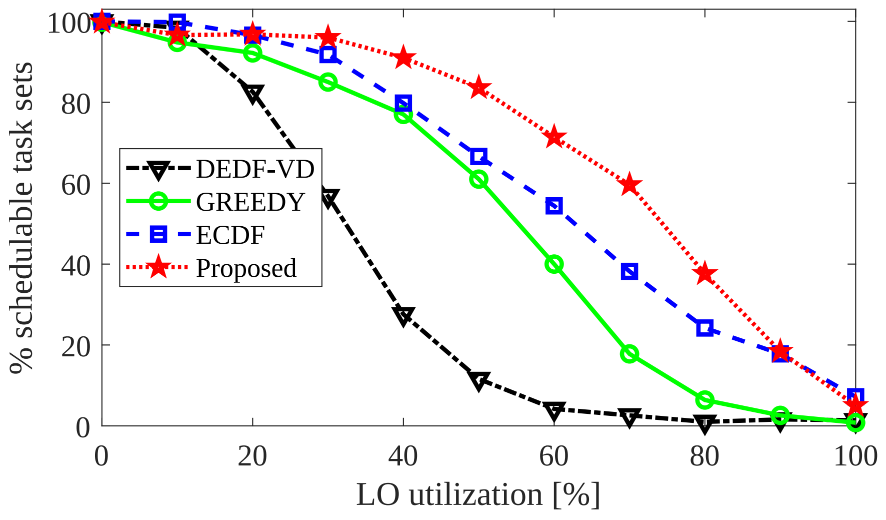


Figure 7.26.: Schedulability vs. LO utilization for $n = 20$ and 30% of HI tasks with 100% increase of HI execution demand

7.7. Evaluation of Approximation Techniques

In this section, we compare the proposed approach of Section 5.1 and its approximated variant based on per-task deadline scaling of Section 6.2.1 with DEDF-VD as explained above, with the GREEDY algorithm by Ekberg and Yi [42], and with ECDF by Easwaran [40].

To this end, we evaluate the impact of different factors on performance by the different algorithms: (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.

In addition, we include our first approximated variant from Section 6.2.1 in this comparison. As discussed in Section 6.2.3, this is the one with the lowest complexity and helps illustrating how much performance can be attained with the proposed technique at the least possible cost.

7.7.1. Schedulability curves

Fig. 7.27 shows schedulability curves, i.e., the percentage of feasible task sets by the above algorithms, versus LO utilization. For every increase in LO utilization, a total number of 1000 different sets of 20 tasks each were randomly generated — 10,000 task sets in total. We made use of UUniFast [24] to generate individual task utilizations.

Further, we used the log-uniform distribution proposed by Emberson *et al.* [45] to create the task periods T_i in the range of $1ms$ to $1000ms$. The log-uniform distribution guarantees that task periods are equally spread into the time bands $1 - 10ms$, $10 - 100ms$, etc.

With T_i and the task utilization, we obtained the values of C_i^{LO}. We assumed that 30% of the tasks are HI tasks, i.e., 6 out of 20 tasks. Further, for each HI task, we randomly selected an increase in HI execution demand of at most 50% of $\frac{C_i^{LO}}{T_i}$. With this, we then obtained the values of C_i^{HI}. Deadlines D_i are constrained and chosen from a uniform distribution in the range $[C_i^{HI}, T_i]$ for HI tasks and in $[C_i^{LO}, T_i]$ for LO tasks.

As depicted in Fig. 7.27, expectedly, the percentage of schedulable task sets decreases with an increasing LO utilization. On the other hand, whereas all algorithms perform similarly for a LO utilization below 50%, they exhibit different behaviors for higher LO utilizations. In particular, the proposed approach outperforms ECDF by around 10% to 20% more accepted task sets in the range of 60% to 100% LO utilization. Interestingly, in spite of having a lesser complexity, our approximated variant shows a similar performance to the GREEDY algorithm.

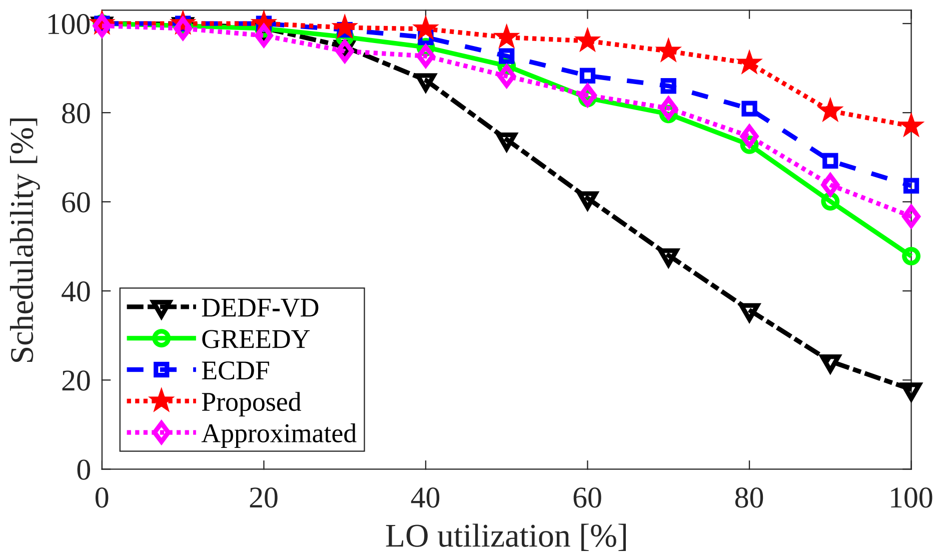


Figure 7.27.: Schedulability vs. LO utilization for $|\tau| = 20$, 30% HI tasks and 50% increase of HI execution demand

7.7.2. Weighted schedulability

Next, we make use of the concept of *weighted schedulability* [22] [36] to analyze the performance by the above algorithms. That is, for a schedulability test y whose accuracy on testing a task set τ is a function of parameter p, its weighted schedulability $W_y(p)$ is given by:

$$\mathrm{W}_y(p) \quad = \quad \frac{\sum\limits_{\forall\tau} U(\tau)\cdot \mathrm{S}_y(\tau,p)}{\sum\limits_{\forall\tau} U(\tau)}, \tag{7.7.1}$$

where $\mathrm{U}(\tau)$ is the utilization of a given τ and $S_y(\tau,p)$ is y's binary result (1 if schedulable and 0 if not) for a task set τ with parameter value p. In other words, individual schedulability results by y are weighted according to the utilization of the task sets tested, putting more emphasis on higher-utilization ones.

We created weighted schedulability curves varying following parameters: (i) total number of tasks, (ii) percentage of HI tasks, (iii) increase of HI execution demand and (iv) range of task periods. Every time we varied one of these parameters, we generated 1000 different task sets for each LO utilization value between 0 and 100% at steps of 10%, i.e., a total of 10,000 task sets per marker on the shown curves. To this end, we proceeded as described previously to obtain task parameters.

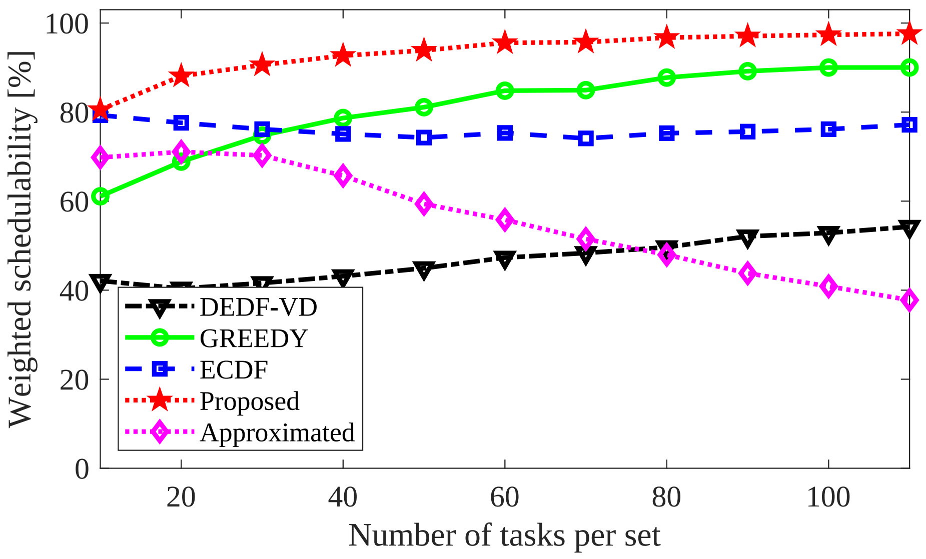


Figure 7.28.: Weighted schedulability vs. total number of tasks for 30% HI tasks and 50% increase of HI execution demand

Fig. 7.28 shows weighted schedulability curves for a varying total number of tasks where we selected the number of HI tasks to be equal to 30% of the total (i.e., it also varies proportionally) and the increase of HI execution demand to be 50% of the LO execution demand for each HI task. The proposed approach outperforms all other algorithms by around 10% to 20% depending on the total number of tasks. In general, the more the tasks, the better the proposed algorithm performs with respect to the others. It is interesting to note that

ECDF performs better than GREEDY up to 30 tasks per set, after which GREEDY performs better. Further, ECDF seems to stagnate at around 80% weighted schedulability, in contrast to GREEDY and the proposed approach.

Our approximated variant has a good performance up to around 40 tasks, after which it decays pronouncedly, even becoming worse than EDF-VD from 80 tasks onwards. The reason is that this algorithm is based on Devi's test, which requires tasks to be sorted by deadline. Since the order of tasks may change with every new deadline scaling factor, some tasks may need to be retested as explained in Section 6.2.1. Clearly, the more tasks there are, the more likely it is that their order changes (when scaling one deadline). To avoid this and reduce complexity, we introduced conditions (6.2.6) and (6.2.8), which truncate the valid range of deadline scaling factors. This has the downside, however, that the number of wrongly rejected task sets increases disproportionally as the total number of tasks grows.

Weighted schedulability curves for a varying percentage of HI tasks are shown in Fig. 7.29. This time, we selected the total number of tasks to be 20, whereas the increase of HI execution demand continues to be 50% as in the previous case. We can see that the performance of all approaches decreases with an increasing number of HI tasks. Up to around 20% HI tasks (i.e., 4 out of 20), the proposed approach and ECDF behave similarly. However, ECDF's performance then decreases rapidly, becoming worse than GREEDY and even DEDF-VD at 50% and 60% HI tasks respectively.

At around 80% HI tasks (16 out of 20 tasks), the proposed approach still allows for around 80% schedulable task sets independent of the LO utilization, whereas all other algorithms are at or below 40% schedulable task sets. This evidences the effectiveness of the proposed approach over GREEDY and ECDF for general cases. In particular, GREEDY and ECDF are based on estimating the worst-case contributions by carry-over jobs at the moment of switching from LO to HI mode. This inevitably becomes pessimistic as the number of carry-over jobs grows, which directly depends on the number of HI tasks.

In the case of our approximated variant, again, conditions (6.2.6) and (6.2.8) start dominating in Fig. 7.29. Even though the total number of tasks remains constant, these two conditions are evaluated for each HI task. As a result, if the number of HI tasks increases, they start playing a bigger role and, hence, accentuating the decrease in performance.

In Fig. 7.30, we further present weighted schedulability curves for a varying increase of HI execution demand. We again selected the total number of tasks to be 20 and the number of HI tasks is set to 30%. In this case, the behavior of algorithms slightly worsens for a growing HI execution demand with exception of ECDF, whose behavior slightly improves. In spite of this, the proposed algorithm outperforms all others by around 10% more schedulable task sets in the range of 10% to 80% increase in HI execution demand. Interestingly, our approximated variant also shows a good performance in this range, which is even better than that of the GREEDY algorithm up to 60% increase in HI execution demand.

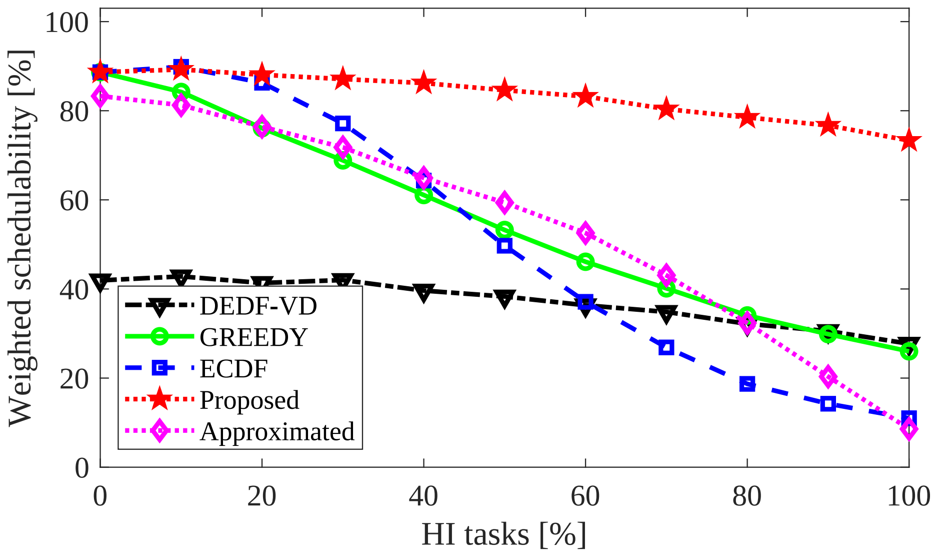


Figure 7.29.: Weighted schedulability vs. percentage of HI tasks for $|\tau| = 20$ and 50% increase of HI execution demand

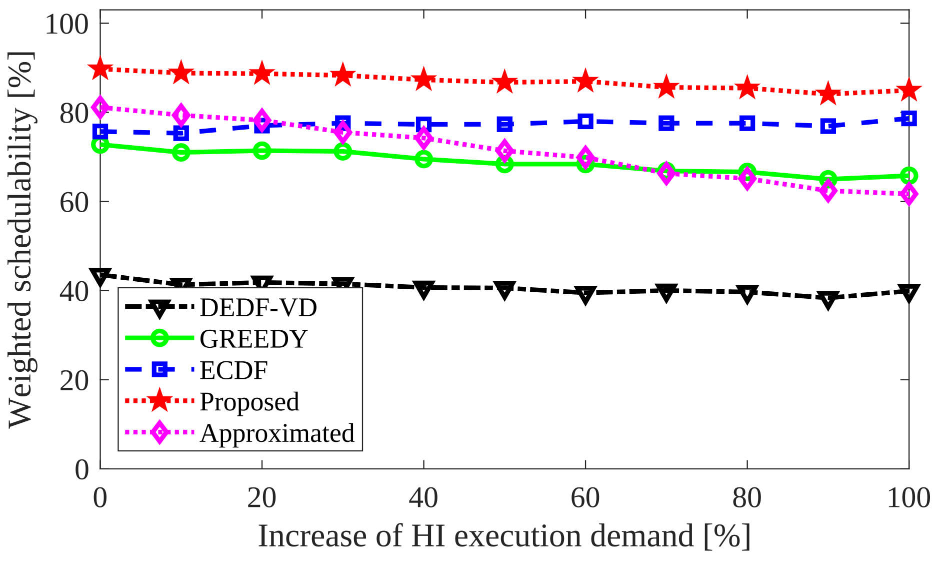


Figure 7.30.: Weighted schedulability vs. increase of HI execution demand for $|\tau| = 20$ and 30% HI tasks

Last, Fig. 7.31 shows weighted schedulability curves for a varying range of task periods with the total number of tasks being again 20, out of which 30% are HI tasks with a 50% increase in HI execution demand. In this case, the performance of all algorithms rapidly goes down for an increasing range of task periods. The proposed approach outperforms ECDF by 10% to 20% more schedulable task sets when the minimum and the maximum task period are 3 to 4 orders of magnitude apart. Note that our approximated variant outperforms GREEDY for period ranges of 2.5 orders of magnitude upwards and has a comparable performance to that of ECDF between 3.5 to 4 orders of magnitude.

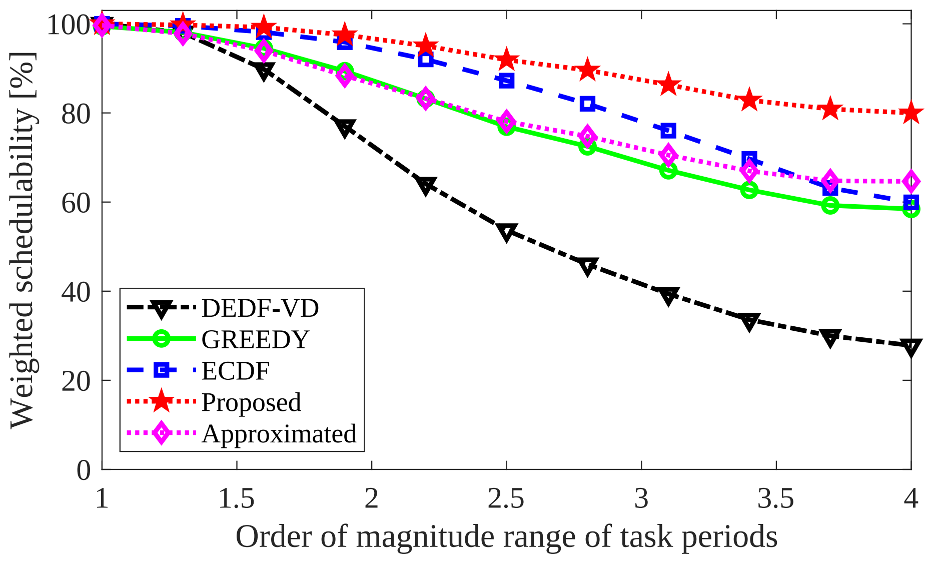


Figure 7.31.: Weighted schedulability vs. range of task periods for $|\tau| = 20$, 30% HI tasks and 50% increase of HI execution demand

7.7.3. Comparison of runtime

Fig. 7.32, Fig. 7.33 to Fig. 7.34 show a comparison of runtime versus LO utilization, total number of tasks and range of task periods respectively. However, note that we have implemented the different algorithms in Matlab and, hence, they can be further optimized, potentially changing their behavior with respect to runtime.

Now, ECDF is around one to two orders of magnitude faster than GREEDY and the proposed approach depending on LO utilization as shown in Fig. 7.32. Our approximated variant is around one order of magnitude slower than DEDF-VD and around one to two orders of magnitude faster than ECDF. This behavior remains almost unchanged as the number of tasks increases towards 100 tasks per set — see Fig. 7.33. Here we maintained the percentage of HI tasks and the increase of HI execution demand equal to 30% and 50% respectively. Only for 10 tasks per set, GREEDY and the proposed algorithm are as fast as ECDF.

Fig. 7.34, shows runtime curves for an increasing range of task periods. We again kept the total number of tasks at 20, the percentage of HI tasks at 30% and the increase of HI execution demand at 50%. As expected, our approximated variant and DEDF-VD have a constant runtime for an increasing range of periods, since both have polynomial complexity. All other algorithms experience an increasing runtime for greater period ranges. ECDF continues to be around one order of magnitude faster than GREEDY and the proposed algorithm. However, this difference reduces as the range of task periods grows. At 4 orders of magnitude between the minimum and the maximum period, all these algorithms show the same runtime.

It should be noted that Fig. 7.33 and Fig. 7.34 are independent of LO utilization. For each marker on these curves, we generated 1000 different task sets for each LO utilization value

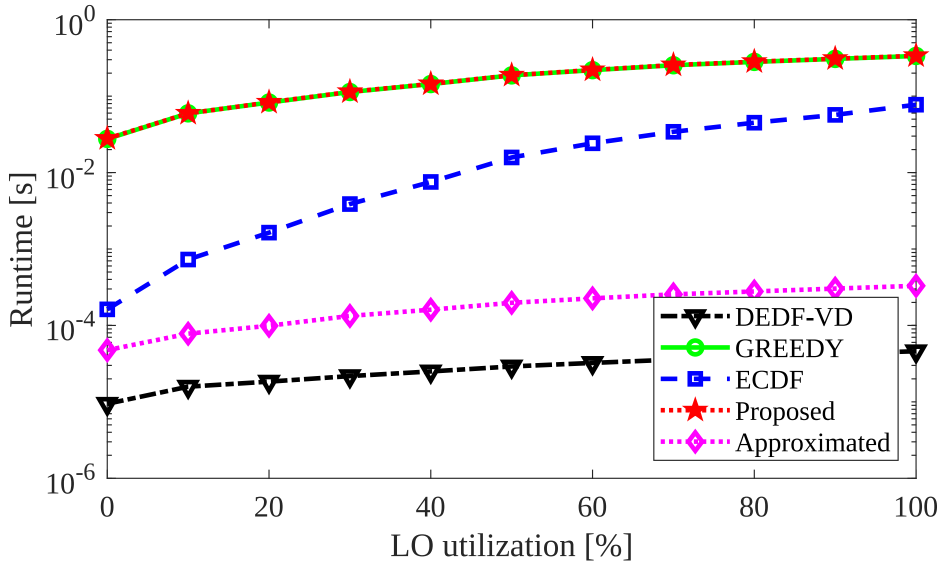


Figure 7.32.: Runtime vs. LO utilization for $|\tau| = 20$, 30% HI tasks and 50% increase of HI execution demand

between 0 and 100% at steps of 10%, i.e., a total of 10,000 task sets per marker.

7.8. Summary

This chapter evaluated the proposed approaches containing utilization caps, the proposed demand bound function for transitions from LO to HI mode and our approximation technique based on it. We particularly, compared them to state-of-the-art solutions from the literature. To this end, this chapter is organized into three main sections.

In Section 7.5, we evaluated the benefits and drawbacks of the proposed approach consisting in introducing utilization caps to MC scheduling. This technique allows LO tasks to continue running without degradation (in spite of some HI tasks having switched to HI mode).

We compared our approach as given in Alg. 1, i.e., with fixed utilization caps, to the original EDF-VD algorithm. Our results illustrated that our algorithm's performance curves almost overlap fully showing a negligible difference with respect to EDF-VD. However, 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, our experiments indicated that, on average, dividing the processor into two halves has almost no performance degradation with respect to EDF-VD (with the processor fully dedicated).

The second Section 7.6 evaluated our technique to better bounding the execution demand under mixed-criticality EDF, which is based on separating the schedulability analysis of transitions to HI mode from that of stable HI mode. This allows working around the computation of carry-over execution demand. We showed how the different algorithms roughly behave with respect to each other. In particular, we compared the proposed technique in form of Alg. 3 with DEDF-VD [8], with the GREEDY algorithm by Ekberg and Yi [42], and with ECDF by Easwaran [40].

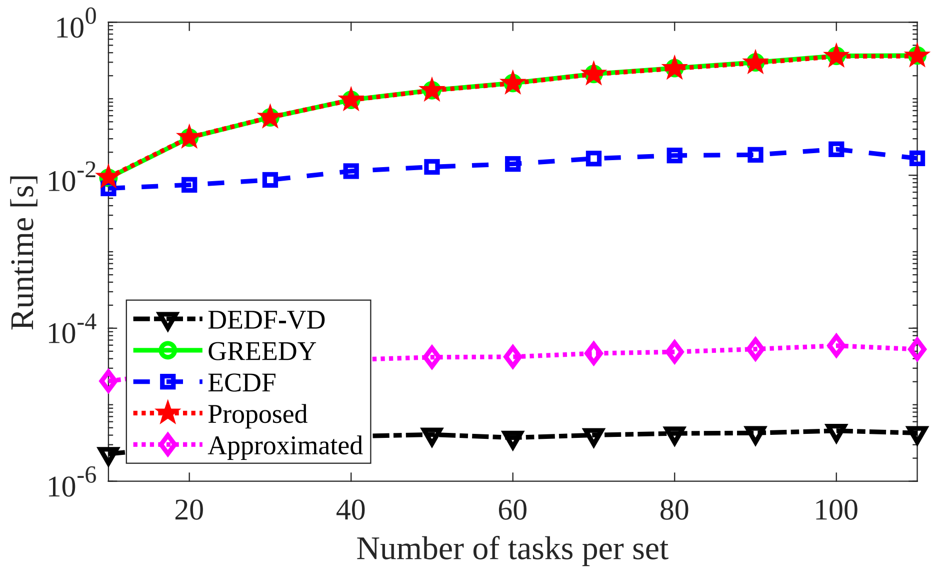


Figure 7.33.: Runtime vs. total number of tasks for 30% HI tasks and 50% increase of HI execution demand

Our experiments indicated that the *Proposed* algorithm and ECDF behaved almost the same with ECDF being slightly better when the increase in HI execution demand is distributed among a few tasks. On the other hand, the *Proposed* algorithm considerably outperformed ECDF by around 10% to 20% more accepted task sets when the increase in HI execution demand is due to a greater number of tasks. The GREEDY algorithm was outperformed by the *Proposed* and the ECDF algorithm by around 10%, i.e., these latter found around 10% more tasks sets that are feasible under mixed-criticality EDF.

Overall, the proposed technique results not only in a considerably simpler, but also tighter bound on execution demand under mixed-criticality EDF, in particular, as the number of HI tasks increases.

Section 7.7 presented evaluation results between the proposed approach of Section 5.1 and its approximated variant based on uniform scaling of Section 6.1 as well as DEDF-VD the GREEDY algorithm [42], and ECDF [40].

Our proposed approximation technique such consists in applying the well-known Devi's test to the demand bound functions obtained in Chapter 5 with the aim of trading off accuracy versus complexity/runtime. Based on our experiments, this approach allows considerably reducing the runtime of the resulting algorithms while maintaining a very good performance in terms of schedulability ratio.

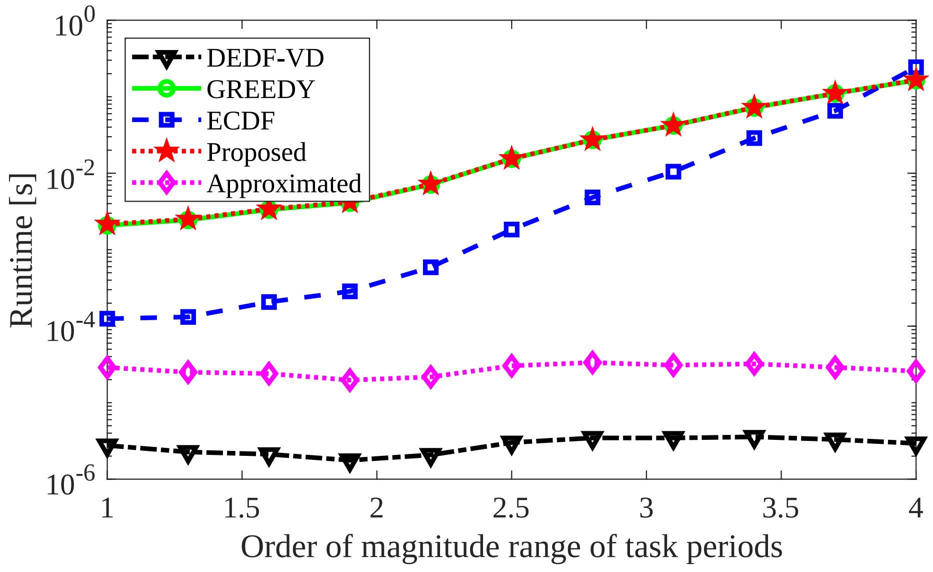


Figure 7.34.: Runtime vs. range of task periods for $|\tau| = 20$, 30% HI tasks and 50% increase of HI execution demand

Chapter 8.

Conclusion and Future Work

This chapter provides concluding remarks and discusses contributions in the context of mixed-criticality real-time embedded systems. Finally, a brief outlook of possible future extensions and modifications to the proposed approaches is further given.

In this work, new techniques for the scheduling of Mixed-Criticality (MC) systems based on EDF were investigated. Thereby, we considered a dual-criticality system consisting of a mix of high-criticality (HI) and low-criticality (LO) tasks.

Basically, the system implements two operation modes: LO and HI mode. In LO mode, HI tasks run for their optimistic WCETs and are scheduled within *virtual deadlines* together with all LO tasks. Virtual deadlines are given by $x_i \cdot D_i$ and are usually shorter than real deadlines D_i with $x_i \in (0, 1)$ being referred to as *deadline scaling factor*.

A switch to HI mode occurs when one or more HI tasks require running for longer than its optimistic WCETs (but still less than its conservative one). HI tasks are then scheduled within their *real deadlines* and LO tasks are stopped from running in HI mode, i.e., LO tasks are immediately discarded, which then allows accommodating this increase in HI execution demand. In this context, following findings by this work can be highlighted:

- We proposed introducing utilization caps to the original EDF-VD algorithm, which allows LO tasks to continue running without degradation (in spite of some HI tasks having switched to HI mode).

  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 partition 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 tasks within the same partition or subset will be discarded, whereas LO tasks in other partitions continue running. The main advantage over the server-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.

- Whereas utilization caps can be basically used for $D_i = T_i$, we also proposed a technique to better bound execution demand for the case $D_i \leq T_i$. Especially, we derived a separate demand bound function for transitions from LO to HI mode and proved its validity. This technique allows us to work around the computation of carry-over execution demand and, hence, to reduce the amount of pessimism in characterizing mixed-criticality EDF. We further proved that the proposed technique leads to a tighter bound on the execution demand under mixed criticality EDF for most relevant cases.

  As discussed before, it is interesting to notice that the proposed technique reduces the problem of testing schedulability under mixed criticality EDF to testing schedulability of three almost unrelated task sets: the one in LO mode, the one in HI mode and the equivalent task set for transitions between LO and HI mode. This leads to a considerably simpler schedulability test while improving our understanding of this problem.

- Since we showed that testing schedulability for mixed-criticality EDF reduces to testing three separate task sets under standard EDF scheduling, we are now able to extend and apply known approximation techniques from the literature (originally conceived for standard EDF).

  In particular, we extended the so-called Devi's test to be used in the context of MC systems. This extension of Devi's test is also sufficient but not necessary, however, it is more accurate than using utilization caps (our first approach) and is considerably faster than any exact test (including the one proposed in this thesis). This represents a good trade-off between accuracy in testing schedulability for MC systems and running time of the test (relevant in online settings where testing needs to be done while the system is running, e.g., admission control).

  Further, we conducted a large set of experiments on synthetic data illustrating the effectiveness of the proposed approach and of its approximated variant in terms of (weighted) schedulability and runtime compared to the most prominent approaches from the literature. This is particularly notable as the number of HI tasks increases.

  Which algorithm should be used depends very much on the context. If we are testing offline, we believe, there is no harm in using all three algorithms (GREEDY, ECDF and the proposed test). If tests are to be performed online, it is probably better to used the approximated variant of the proposed test — with a $\mathcal{O}(n \cdot \log n)$ rather than pseudo-polynomial complexity.

  Finally, in the Appendix A.1, we show how to extend the proposed approach to more than two levels of criticality considering ordered (i.e., where criticality levels cannot be skipped) and unordered modes switches.

In the following, Table 8.1 and Table 8.2 summarize the results of our experimental evaluation. These tables covered a simulation-based evaluation examining the performance of proposed scheduling algorithms and comparing them to other scheduling algorithms from the literature.

Comparision Criteria ($D_i = T_i$) **/ Algorithms**	**EDF-VD**	$U_L = 1/2$	$U_L = 1/3$	$U_L = 1/4$
10 **Tasks,** 10% **to** 50% **HI tasks,** 10% **more HI execution demand**	✓✓✓	✓✓	-	-
10 **Tasks,** 10% **to** 50% **HI tasks,** 100% **more HI execution demand**	✓✓✓	✓✓	-	-
20 **Tasks,** 10% **to** 50% **HI tasks,** 10% **more HI execution demand**	✓✓✓	✓✓	✓✓	✓
20 **Tasks,** 10% **to** 50% **HI tasks,** 100% **more HI execution demand**	✓✓✓	✓✓	✓	-
50 **Tasks,** 10% **to** 50% **HI tasks,** 10% **more HI execution demand**	✓✓✓	✓✓✓	✓✓✓	✓✓✓
50 **Tasks,** 10% **to** 50% **HI tasks,** 100% **more HI execution demand**	✓✓✓	✓✓✓	✓✓	✓
Runtime, 10 **Tasks,** 50% **HI tasks,** 100% **more HI execution demand**	✓✓✓	-	✓	✓✓

Table 8.1.: Comparison of the different algorithms. '-' means poor performance, '✓' medium performance, '✓✓' good performance and '✓✓✓' excellent performance.

As it can be seen in Table 8.1, $U_L = 1/4$ has the worst performance and dividing the processor into two halves 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 reducing the total usable utilization on the processor. It can be observed that as the number of tasks and HI tasks grow, all algorithms behave almost the same where HI tasks have an increase of 10% more execution demand in HI mode. On the other hand, in terms of runtime comparison, $U_L = 1/4$ is faster than $U_L = 1/2$ as the number of tasks grows, however, it is also slower than EDF-VD.

Table 8.2 illustrates that the *Proposed* algorithm and ECDF behave almost the same for an increase in HI execution demand that goes from 10% to 100%. It can be seen that DEDF-VD shows the worst performance among all algorithms. On the other hand, the Approximated algorithm has a performance close to that of GREEDY. From these results, it is observed that the Proposed algorithm outperforms all approaches as the number of HI tasks per task set grows. The Approximated algorithm also has a good performance close to GREEDY algorithm. On the other hand, in terms of runtime comparison, the *Approximated* and the *Proposed* algorithm are faster than ECDF and GREEDY, however, they are also slower than DEDF-VD.

Comparision Criteria ($D_i \leq T_i$) / Algorithms	**DEDF-VD**	**GREEDY**	**ECDF**	**Proposed**	**Approximated**
10 **Tasks,** 10% **HI tasks,** 10% **to** 100% **more HI execution demand**	-	✓✓	✓✓✓	✓✓✓	✓
10 **Tasks,** 30% **HI tasks,** 10% **to** 100% **more HI execution demand**	-	✓✓	✓✓✓	✓✓✓	✓
20 **Tasks,** 10% **HI tasks,** 10% **to** 100% **more HI execution demand**	-	✓✓	✓✓✓	✓✓✓	✓✓
20 **Tasks,** 30% **HI tasks,** 10% **to** 100% **more HI execution demand**	-	✓	✓✓	✓✓✓	✓
Runtime, 20 **Tasks,** 30% **HI tasks,** 50% **more HI execution demand**	✓✓✓	-	✓	-	✓✓

Table 8.2.: Comparison of the different algorithms. '-' means poor performance, '✓' medium performance, '✓✓' good performance and '✓✓✓' excellent performance.

8.1. Outlook/Future Perspectives

To conclude this chapter (and this work), this section briefly discusses possible future extensions to scheduling MC systems based on EDF. Regarding the scheduling policies presented in Chapter 4, we would like to develop a technique in which LO tasks are not affected by switching to HI mode and continue running in the same partition. To this end, an interesting approach is to provide a tardiness bound for LO task that allows LO tasks in a partition to incur some amount of deadline miss when switching to HI mode. This allows us to safeguard some of the LO tasks from being discarded in HI mode enabling for more design flexibility.

In addition, the performance of the proposed Alg. 3 in Chapter 5 can be further improved by using a more sophisticated deadline tightening scheme. In this thesis, however, the contribution was rather a new technique for bounding demand execution, which can be combined with existing deadline tightening techniques, e.g., from [42] or [40], presumably achieving even better results.

Finally, although aerospace and automotive safety regulations define around five levels of criticality, for ease of exposition, we considered only two such levels in this thesis. As a result, a further extension is to adapt the proposed techniques from Chapter 4, Chapter 5 and Chapter 6 to multiple levels of criticality.

Bibliography

[1] More than 50 billion connected devices. In *Ericsson, white paper 284 23-3149 Uen*, 2011.

[2] Strategic research agenda 2012. Technical report, The Embedded Systems Institute (ESI), Eindhoven, 2012.

[3] N. C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical Report YCS 164, University of York, York, England, UK, 1991.

[4] N. C. Audsley. On priority assignment in fixed priority scheduling. *Information Processing Letters*, 79(1):39–44, 2001.

[5] M. A. Awan, K. Bletsas, P. F. Souto, and E. Tovar. Semi-partitioned mixed-criticality scheduling. In *Architecture of Computing Systems - ARCS 2017*, pages 205–218. Springer International Publishing, 2017.

[6] A. Azim and S. Fischmeister. Efficient mode changes in multi-mode systems. In *2016 IEEE 34th International Conference on Computer Design (ICCD)*, pages 592–599, 2016.

[7] J. Barhorst, T. Belote, P. Binns, J. H. nd James Paunicka, P. Sarathy, J. Scoredos, P. Stanfill, D. Stuart, and R. Urzi. A research agenda for mixed-criticality systems. In *In Cyber-Physical Systems Week*, 2009.

[8] 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.

[9] S. Baruah, V. Bonifaci, G. D'angelo, H. Li, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie. Preemptive uniprocessor scheduling of mixed-criticality sporadic task systems. *Journal of the ACM (JACM)*, 62(2), 2015.

[10] 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.

[11] S. Baruah and A. Burns. Implementing mixed criticality systems in Ada. In A. Romanovsky and T. Vardanega, editors, *Ada-Europe'11 Proceedings of the 16th Ada-Europe International Conference on Reliable Software Technologies*, pages 174–188. Springer, 2011.

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

[13] S. Baruah, A. Burns, and R. Davis. An extended fixed priority scheme for mixed criticality systems. In *Proc. of Workshop on Real-Time Mixed Criticality Systems (ReTiMics)*, Aug. 2013.

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

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

[16] S. Baruah, H. Li, and L. Stougie. Towards the design of certifiable mixed-criticality systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE*, pages 13–22, 2010.

[17] 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.

[18] S. Baruah and S. Vestal. Schedulability analysis of sporadic tasks with multiple criticality specifications. In *ECRTS '08 Proceedings of the 2008 Euromicro Conference on Real-Time Systems*, pages 147–155, 2008.

[19] S. K. Baruah. Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors. *Computers, IEEE Transactions on*, 53(6):781 – 784, June 2004.

[20] S. K. Baruah. Schedulability analysis for a general model of mixed-criticality recurrent real-time tasks. In *2016 IEEE Real-Time Systems Symposium (RTSS)*, 2016.

[21] S. K. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie. Scheduling real-time mixed-criticality jobs. 2012.

[22] A. Bastoni, B. B. Brandenburg, and J. H. Anderson. Cache-related preemption and migration delays: Empirical approximation and impact on schedulability. In *Proc. of Sixth Int'l Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, pages 33–44, July 2010.

[23] E. Bini and G. Buttazzo. Biasing effects in schedulability measures. In *Proc. of Euromicro Conference on Real-Time Systems (ECRTS)*, 2004.

[24] E. Bini and G. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems (RTS)*, 30(1-2), 2005.

[25] V. Bonifaci and A. Marchetti-Spaccamela. Feasibility analysis of sporadic real-time multiprocessor task systems. In *Proceeding ESA'10 Proceedings of the 18th annual European conference on Algorithms: Part II*, pages 230–241, 2010.

[26] A. Burns and S. Baruah. Timing faults and mixed criticality systems. In C. B. Jones and J. L. Lloyd, editors, *Dependable and Historic Computing. Lecture Notes in Computer Science*, pages 146–166. Springer, 2011.

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

[28] A. Burns and R. I. Davis. Mixed criticality systems - a review. Technical report, Department of Computer Science, University of York, York, UK, 2018.

[29] A. Burns and R. I. Davis. A survey of research into mixed criticality systems. *ACM Computing Surveys (CSUR)*, 2018.

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

[31] G. C. Buttazzo. *Hard RealTime Computing Systems: Predictable Scheduling Algorithms and Applications (Third Edition).* Springer, third edition, 2011.

[32] H. Chai, G. Zhang, J. Sun, A. Vajdi, J. Hua, and J. Zhou. A review of recent techniques in mixed-criticality systems. *Journal of Circuits, Systems and Computers*, 28(7):25, 2018.

[33] Y. Chen, K. G. Shin, and H. Xiong. Generalizing fixed-priority scheduling for better schedulability in mixed-criticality systems. *Information Processing Letters*, 116(8):508–512, 2016.

[34] H. Chetto and M. Chetto. Some results of the earliest deadline scheduling algorithm. *IEEE Transactions on Software Engineering*, 15:1261–1269, 1989.

[35] G. K. C. S. L. M. H. S. Christian Buckl, Alexander Camek and A. Knoll. The software car: Building ICT architectures for future electric vehicles. In *2012 IEEE International Electric Vehicle Conference*, pages 1–8, 2012.

[36] R. I. Davis. On the evaluation of schedulability tests for real-time scheduling algorithms. In *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2016.

[37] M. L. Dertouzos. Control robotics: The procedural control of physical processes. In *IFIP Congress*, pages 807–813, 1974.

[38] U. C. Devi. An improved schedulability test for uniprocessor periodic task systems. In *Proc. of Euromicro Conference on Real-Time Systems (ECRTS)*, 2003.

[39] F. Dorin, P. Richard, M. Richard, and J. Goossens. Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities. *Real-Time Systems (RTS)*, 46:305–331, 2010.

[40] A. Easwaran. Demand-based scheduling of mixed-criticality sporadic tasks on one processor. In *Proc. of Real-Time Systems Symposium (RTSS)*, Dec. 2013.

[41] C. Ebert and C. Jones. Embedded software: Facts, figures, and future. *Computer*, (42):42–42, 2009.

[42] 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.

[43] P. Ekberg and W. Yi. Bounding and shaping the demand of generalized mixed-criticality sporadic task systems. *Real-Time Systems (RTS)*, 50(1), 2014.

[44] P. Emberson and I. Bate. Minimising task migration and priority changes in mode transitions. In *RTAS '07 Proceedings of the 13th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS)*, pages 158–167, 2007.

[45] P. Emberson, R. Stafford, and R. I. Davis. Techniques for the synthesis of multiprocessor tasksets. In G. Lipari and T. Cucinotta, editors, *Proc. of 1st Int'l Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010) In conjunction with the 22nd Euromicro Conference on Real-Time Systems (ECRTS10)*, pages 6–11, Brussels, Belgium, July 2010.

[46] M. R. Garey and D. S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal of Computing*, 4:397–411, 1975.

[47] R. Gratia, T. Robert, and L. Pautet. Generalized mixed-criticality scheduling based on RUN. In *Proceedings of the 23rd International Conference on Real Time and Networks Systems (RTNS)*, 2015.

[48] R. Gratia, T. Robert, and L. Pautet. Scheduling of mixed-criticality systems with RUN. In *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, pages 1–8, 2015.

[49] X. Gu and A. Easwaran. Efficient schedulability test for dynamic-priority scheduling of mixed-criticality real-time systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 17:1–24, 2017.

[50] N. Guan, P. Ekberg, M. Stigge, and W. Yi. Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems. In *Proceedings of the 32nd IEEE Real-Time Systems Symposium, (RTSS)*, pages 13–23, 2011.

[51] N. Guan, P. Ekberg, M. Stigge, and W. Yi. Improving the scheduling of certifiable mixed criticality sporadic task systems. Technical report, Uppsala University, 2013.

[52] Z. Guo and S. Baruah. *Mixed-Criticality Real-Time Systems*, pages 1–20. Springer, Berlin, Heidelberg, 2018.

[53] S. Heath. *Embedded Systems Design*. Newnes, 2 edition, 2002.

[54] P. Huang, G. Giannopoulou, N. Stoimenov, and L. Thiele. Service adaptions for mixed-criticality systems. Technical report, Computer Engineering and Networks Laboratory, ETH Zurich, 2013.

[55] 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.

[56] L. L. Jean-franois Hermant and N. Rivierre. Real-time fixed and dynamic priority driven scheduling algorithms: Theory and experience. *REFLECS - Distributed Real-Time Fault-Tolerant Computing Systems*, 1996.

[57] O. R. Kelly, H. Aydin, and B. Zhao. On partitioned scheduling of fixed-priority mixed-criticality task sets. In *TRUSTCOM '11 Proceedings of the 2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 1051–1059, 2011.

[58] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer, second edition, 2011.

[59] 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)*, July 2014.

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

[61] 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.

[62] H. Li. *Scheduling Mixed-Criticality Real-Time Systems*. PhD thesis, 2013.

[63] H. Li and S. K. Baruah. An algorithm for scheduling certifiable mixed-criticality sporadic task systems. In *Proceedings of the 31st IEEE Real-Time Systems Symposium, (RTSS)*, 2010.

[64] H. Li and S. K. Baruah. Global mixed-criticality scheduling on multiprocessors. In *2012 24th Euromicro Conference on Real-Time Systems (ECRTS)*, 2012.

[65] G. Lipari and G. C. Buttazzo. Resource reservation for mixed criticality systems. In *Proc. of the Workshop on Real-Time Systems: the past, the present, and the future*, 2013.

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

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

[68] A. P. Manfred Broy, Ingolf H. Kruger and C. Salzmann. Engineering automotive software. *Proceedings of the IEEE*, 95(2):356–373, 2007.

[69] A. Masrur, D. Müller, and M. Werner. Bi-level deadline scaling for admission control in mixed-criticality systems. In *Proc. of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Aug. 2015.

[70] M. S. Mollison, J. P. Erickson, J. H. Anderson, S. K. Baruah, and J. A. Scoredos. Mixed-criticality real-time scheduling for multicore systems. In *CIT '10 Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology (CIT)*, pages 1864–1871, 2010.

[71] D. Müller. SS01 — explicit per-task deadline scaling for uniprocessor scheduling of job-level static mixed-criticality systems. In *Proc. of Symposium on Industrial Embedded Systems (SIES)*, 2016.

[72] D. d. Niz, K. Lakshmanan, and R. Rajkumar. On the scheduling of mixed-criticality real-time task sets. In *RTSS '09: Proceedings of the 2009 30th IEEE Real-Time Systems Symposium*, pages 291–300, Washington, DC, USA, Dec. 2009. IEEE Computer Society.

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

[74] F. Panzieri and R. Davoli. Real time systems: A tutorial. In *Lecture Notes in Computer Science*, volume 729, 2005.

[75] T. Park and S. Kim. Dynamic scheduling algorithm and its schedulability analysis for certifiable dual-criticality systems. In *2011 Proceedings of the Ninth ACM International Conference on Embedded Software (EMSOFT)*.

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

[77] P. Pedro and A. Burns. Schedulability analysis for mode changes in flexible real-time systems. In *Proceeding. 10th EUROMICRO Workshop on Real-Time Systems (Cat. No.98EX168)*, pages 172–179, 1998.

[78] A. Pétrissans, S. Krawczyk, G. Cattaneo, N. Feeney, L. Veronesi, and C. Meunier. Design of future embedded systems toward system of systems: trends and challenges. Technical report, European Commission, 2012.

[79] L. T. X. Phan, S. Chakraborty, and I. Lee. Timing analysis of mixed time/event-triggered multi-mode systems. In *RTSS '09 Proceedings of the 2009 30th IEEE Real-Time Systems Symposium*, pages 271–280, 2009.

[80] L. T. X. Phan, I. Lee, and O. Sokolsky. A semantic framework for mode change protocols. In *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 91–100, 2011.

[81] S. Ramanathan and A. Easwaran. Utilization difference based partitioned scheduling of mixed-criticality systems. In *DATE '17 Proceedings of the Conference on Design, Automation and Test in Europe*, pages 238–243, 2017.

[82] J. Real and A. Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Systems*, 26(2):161–197, 2004.

[83] 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.

[84] P. Rodriguez, L. George, Y. Abdeddaim, and J. Goossens. Multi-criteria evaluation of partitioned EDF-VD for mixed-criticality systems upon identical processors. In *1st International Workshop on Mixed Criticality Systems (WMC)*, 2013.

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

[86] L. Santinelli, D. Doose, G. Durrieu, F. Boniol, C. Lesire-Cabaniols, and C. Grand. Schedulability analysis for mixed critical cyber physical systems. In *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, pages 297–303, 2018.

[87] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham. Mode change protocols for priority-driven preemptive scheduling. *Journal of Real-Time Systems (RTS)*, 1:243–264, 1989.

[88] J. Singh and S. P. Singh. Schedulability test for soft real-time systems under multiprocessor environment by using an earliest deadline first scheduling algorithm. *CoRR*, page 14, 2012.

[89] 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)*, June 2014.

[90] J. A. Stankovic and K. Ramamritham. *Tutorial on Hard Real-Time Systems.* IEEE Computer Society Press, Los Alamitos, CA, USA, 1987.

[91] 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.

[92] H. Su, D. Zhu, and D. Mossé. Scheduling algorithms for elastic mixed-criticality tasks in multicore systems. In *2013 IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2013.

[93] D. Tămaş-Selicean, P. Pop, and J. Madsen. Design of mixed-criticality applications on distributed real-time systems. *Technical University of Denmark*, 2014.

[94] K. Tindell, A. Burns, and A. J. Wellings. Mode changes in priority preemptively scheduled systems. In *[1992] Proceedings Real-Time Systems Symposium (RTSS)*, pages 100–109, 1992.

[95] F. Vahid and T. Givargis. *Embedded System Design: A Unified Hardware/Software Introduction.* John Wiley & Sons, 2002.

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

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

[98] 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)*, 7, 2008.

[99] H. Xu and A. Burns. Semi-partitioned model for dual-core mixed criticality system. In *23rd International Conference on Real-Time Networks and Systems (RTNS)*, 2015.

[100] C. Yao, L. Qiao, L. Zheng, and X. Huagang. Efficient schedulability analysis for mixed-criticality systems under deadline-based scheduling. *Chinese Journal of Aeronautics*, 24:856–866, 2014.

[101] F. Zhang and A. Burns. Schedulability analysis for real-time systems with EDF scheduling. *IEEE Transactions on Computers*, 58:1250–1258, 2009.

[102] 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)*, pages 141–146, 2013.

Appendix A.

Introduction

A.1. Multiple Levels of Criticality

In practice, usually more than two levels of criticality are common, e.g., four different *automotive safety integrity levels* (ASIL) are defined in the ISO 26262 standards. To account for this, we illustrate how to apply the proposed approach to add a third level of criticality between LO and HI: the mid-criticality (MI) level. Note that additional levels of criticality can also be added in an straightforward manner based on the presented analysis.

Just as before, the system implements a mode of operation per criticality level resulting now in three modes: LO, MI, and HI mode. Further, tasks are classified according to their criticality χ_i into LO, MI and HI tasks. LO tasks only run in LO mode and are discarded in MI and HI mode. MI tasks run in LO and MI mode, but are discarded in HI mode, whereas HI tasks run in all modes.

All tasks are defined by their minimum inter-release times T_i and their relative deadlines D_i. LO task are characterized by their WCET parameter C_i^{LO}, whereas MI tasks have a C_i^{LO} and a C_i^{MI} parameter, which denote their WCET in LO and MI mode respectively. HI tasks now have three WCET parameters, i.e., C_i^{LO}, C_i^{MI}, and C_i^{HI} where $C_i^{LO} < C_i^{MI} < C_i^{HI} \leq D_i \leq T_i$ holds.

A.1.1. Ordered mode switches

We first consider *ordered* mode switches. That is, the system switches from LO to MI, if either a MI or a HI task runs for more than its C_i^{LO} in LO mode, and from MI to HI mode, if a HI task runs for more than its C_i^{MI} in MI mode. Note that there is no direct transition from LO and HI mode, i.e., the system first switches to MI and then to HI mode. The necessary extensions for *unordered* mode switches, i.e., when the system switches from LO directly to HI mode, are discussed below. Next, for ease of exposition, we first analyze schedulability in MI mode, then in HI mode, and last in LO mode.

Schedulability in stable MI mode. In MI mode, LO tasks are discarded and MI tasks are scheduled together with HI tasks. MI tasks are scheduled within their real deadline, however, HI tasks are assigned virtual deadlines $y_i \cdot D_i$. Here, $y_i \in (0, 1]$ denotes the per-task scaling factor in MI mode. Both MI and HI run for their corresponding C_i^{MI} leading to the following demand bound function — which resembles (5.1.1):

$$\begin{aligned}\mathrm{dbf}_{MI}(t) \quad &= \sum_{\chi_i = MI} \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right) C_i^{MI} \\ &+ \sum_{\chi_i = HI} \left(\left\lfloor \frac{t - y_i \cdot D_i}{T_i} \right\rfloor + 1 \right) C_i^{MI}. \end{aligned} \tag{A.1.1}$$

Now, proceeding as before, we obtain an upper bound on $\hat{t}_{MI}$, i.e., the point in time until which we need to check feasibility, i.e., that $\mathrm{dbf}_{MI}(t) < t$ holds. With $U_{MI}^{MI} = \sum_{\chi_i=MI} \frac{C_i^{MI}}{T_i}$, $U_{HI}^{MI} = \sum_{\chi_i=HI} \frac{C_i^{MI}}{T_i}$ and letting y_i tend to 0, we obtain:

$$\hat{t}_{MI} \quad \leq \quad \frac{\sum_{\chi_i=MI} (T_i - D_i)\frac{C_i^{MI}}{T_i}}{1 - U_{MI}^{MI} - U_{HI}^{MI}} + \frac{\sum_{\chi_i=HI} C_i^{MI}}{1 - U_{MI}^{MI} - U_{HI}^{MI}}. \tag{A.1.2}$$

Schedulability in stable HI mode. Only HI tasks are allowed to run and they are scheduled within their real deadlines in HI mode. Hence, $\mathrm{dbf}_{HI}(t)$ is given by (5.1.4) and the upper bound on $\hat{t}_{HI}$ is given by (5.1.5), which requires no further discussion.

Schedulability in the transition from MI to HI mode. We can apply Theorem 1 to obtain the demand bound function at transitions from MI to HI mode. Note that this also resembles (5.1.6):

$$\mathrm{dbf}_{SW1}(t) \quad = \quad \sum_{\chi_i=HI} \left(\left\lfloor \frac{t - \Delta D_i^{SW1}}{T_i} \right\rfloor + 1 \right) \Delta C_i^{SW1}, \tag{A.1.3}$$

with $\Delta D_i^{SW1} = D_i - y_i \cdot D_i$, and $\Delta C_i^{SW1} = C_i^{HI} - C_i^{MI}$. Similarly, we proceed to obtain an upper bound on $\hat{t}_{SW1}$, i.e., the point in time until which $\mathrm{dbf}_{SW1}(t) < t$ needs to be checked. The resulting expression resembles (5.1.7) with U_{HI}^{SW1} given by $\sum_{\chi_i=HI} \frac{\Delta C_i^{SW1}}{T_i}$:

$$\hat{t}_{SW1} \leq \frac{\sum_{\chi_i=HI} \Delta C_i^{SW1}}{1 - U_{HI}^{SW1}}. \tag{A.1.4}$$

Schedulability in LO mode. In LO mode, LO tasks need to be scheduled together with MI and HI tasks. MI tasks are assigned virtual deadlines $x_i \cdot D_i$, while HI tasks are assigned virtual deadlines $x_i \cdot y_i \cdot D_i$. That is, their virtual deadlines in MI mode (i.e., $y_i \cdot D_i$) are again scaled by $x_i \in (0, 1]$. As a consequence, the resulting demand bound function $\mathrm{dbf}_{LO}(t)$ in LO mode is given by:

$$\begin{aligned} \mathrm{dbf}_{LO}(t) \quad = \quad & \sum_{\chi_i=LO} \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right) C_i^{LO} \\ & + \sum_{\chi_i=MI} \left(\left\lfloor \frac{t - x_i \cdot D_i}{T_i} \right\rfloor + 1 \right) C_i^{LO}, \\ & + \sum_{\chi_i=HI} \left(\left\lfloor \frac{t - x_i \cdot y_i \cdot D_i}{T_i} \right\rfloor + 1 \right) C_i^{LO}. \end{aligned} \tag{A.1.5}$$

We can proceed as before to obtain an upper bound on $\hat{t}_{LO}$. That is, the point in time until which we need to check that $\mathrm{dbf}_{LO}(t) < t$ holds. In addition to U_{LO}^{LO} and U_{HI}^{LO}, considering $U_{MI}^{LO} = \sum_{\chi_i = MI} \frac{C_i^{LO}}{T_i}$ and letting x_i tend to 0, we obtain:

$$\hat{t}_{LO} \leq \frac{\sum_{\chi_i=LO} (T_i - D_i)\frac{C_i^{LO}}{T_i}}{1 - U_{LO}^{LO} - U_{MI}^{LO} - U_{HI}^{LO}} + \frac{\sum_{\chi_i = MI \vee HI} C_i^{LO}}{1 - U_{LO}^{LO} - U_{MI}^{LO} - U_{HI}^{LO}}, \tag{A.1.6}$$

where it should be noted that the second summation on the right-hand side applies to both MI and HI tasks in the system.

Schedulability in the transition from LO to MI mode. We can again apply Theorem 1 to obtain the demand bound function for transitions from LO to MI mode:

$$\begin{aligned} \mathrm{dbf}_{SW2}(t) &= \sum_{\chi_i=MI} \left(\left\lfloor \frac{t - \Delta D_i^{SW2}}{T_i} \right\rfloor + 1 \right) \Delta C_i^{SW2} \\ &\quad + \sum_{\chi_i=HI} \left(\left\lfloor \frac{t - \hat{\Delta} D_i^{SW2}}{T_i} \right\rfloor + 1 \right) \Delta C_i^{SW2}, \end{aligned} \tag{A.1.7}$$

with $\Delta D_i^{SW2} = D_i - x_i \cdot D_i$, $\hat{\Delta} D_i^{SW2} = D_i - x_i \cdot y_i \cdot D_i$, and $\Delta C_i^{SW2} = C_i^{MI} - C_i^{LO}$. Similarly, we proceed to obtain an upper bound on $\hat{t}_{SW2}$, i.e., the point in time until which $\mathrm{dbf}_{SW2}(t) < t$ needs to be checked:

$$\hat{t}_{SW2} \leq \frac{\sum_{\chi_i = MI \vee HI} \Delta C_i^{SW2}}{1 - U_{MI}^{SW2} - U_{HI}^{SW2}}, \tag{A.1.8}$$

where U_{MI}^{SW2} is given by $\sum_{\chi_i=MI} \frac{\Delta C_i^{SW2}}{T_i}$ and U_{HI}^{SW2} is given by $\sum_{\chi_i=HI} \frac{\Delta C_i^{SW2}}{T_i}$.

Finding deadline scaling factors. In contrast to the case of two levels of criticality, we now have to compute two deadline scaling factors y_i and x_i. We can still use the proposed approach from Section 5.1, but in an iterative manner. That is, we first use the proposed approach to obtain y_i, i.e., the deadline scaling factor in MI mode. Once we have the values of y_i, we can apply this approach again to find x_i, i.e., the deadline scaling factor in LO mode.

A.1.2. Unordered mode switches

If we were to allow for *unordered* mode switches, i.e., from LO directly to HI mode in the above setting with three levels of criticality, we need to consider it separately. To this end, we assume that a subset of the HI tasks cause a direct transition to HI mode (instead of MI mode as assumed so far) when running for more than C_i^{LO} in LO mode.[1] As a result, in LO mode, we now have:

$$\begin{aligned}\mathrm{dbf}_{LO}(t) \quad = \quad & \sum_{\chi_i = LO} \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right) C_i^{LO} \\ & + \sum_{\chi_i = MI} \left(\left\lfloor \frac{t - x_i \cdot D_i}{T_i} \right\rfloor + 1 \right) C_i^{LO}, \\ & + \sum_{\chi_i = HI} \left(\left\lfloor \frac{t - z_i \cdot D_i}{T_i} \right\rfloor + 1 \right) C_i^{LO}, \qquad \text{(A.1.9)}\end{aligned}$$

where z_i is a deadline scaling factor that guarantees schedulability for the direct transition from LO to HI mode. In HI mode, again, $\mathrm{dbf}_{HI}(t)$ given by (5.1.4) continues to be valid. As a result, with all x_i obtained as discussed for the case of ordered mode switches, we can compute each z_i in (A.1.9) also based on the approach from Section 5.1.

Finally, to guarantee safety independent of whether the system switches to MI or HI mode, HI task will now have to be scheduled in LO mode using the minimum between $x_i \cdot y_i$ that covers ordered transitions and z_i that accounts for the unordered case. For more than three levels of criticality, note that all possible unordered mode switches will have to be analyzed as shown here to determine suitable deadline scaling factors for the tasks involved.

[1] In principle, any HI task can be allowed to switch either to MI or to HI mode too. However, in this case, we will need to extend our task model such that mode switches can be triggered independent of the tasks' execution budgets.

Appendix B.

Evaluation and Results

B.1. Uniform Distribution for Task Periods

In Chapter 7 we used the log-uniform distribution proposed by Emberson *et al.* [45] to generate task periods in $[1, T_{max}]$, where T_{max} was set to 1000 in the default case. The log-uniform distribution equally spreads task periods into the time bands $1 - 10$, $10 - 100$, etc. and, hence, the resulting task sets have an equal number of tasks in each such bands.

In contrast to this, a uniform distribution tends to concentrate task periods in the middle of $[1, T_{max}]$, resulting in task sets where most tasks have periods of the same order of magnitude around 500 for $T_{max} = 1000$. Task sets generated this way lead to different performance by algorithms as shown below in Fig. B.1 for schedulability and in Fig. B.2 to Fig. B.5 for weighted schedulability. In particular, the algorithms' behavior changes with respect to runtime as shown in Fig. B.6 to Fig. B.8.

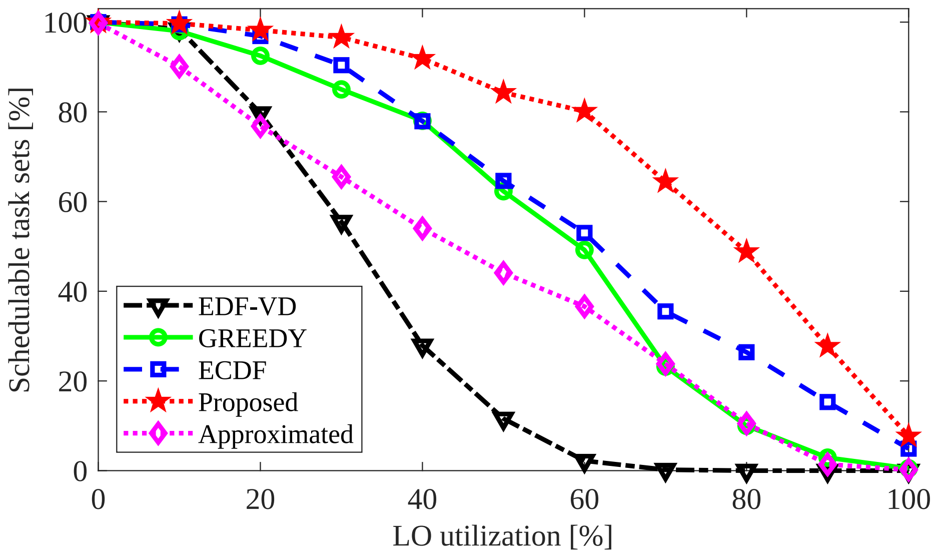


Figure B.1.: Schedulability vs. LO utilization for $|\tau| = 20$, 30% HI tasks and 50% increase of HI execution demand — uniform distribution of task periods

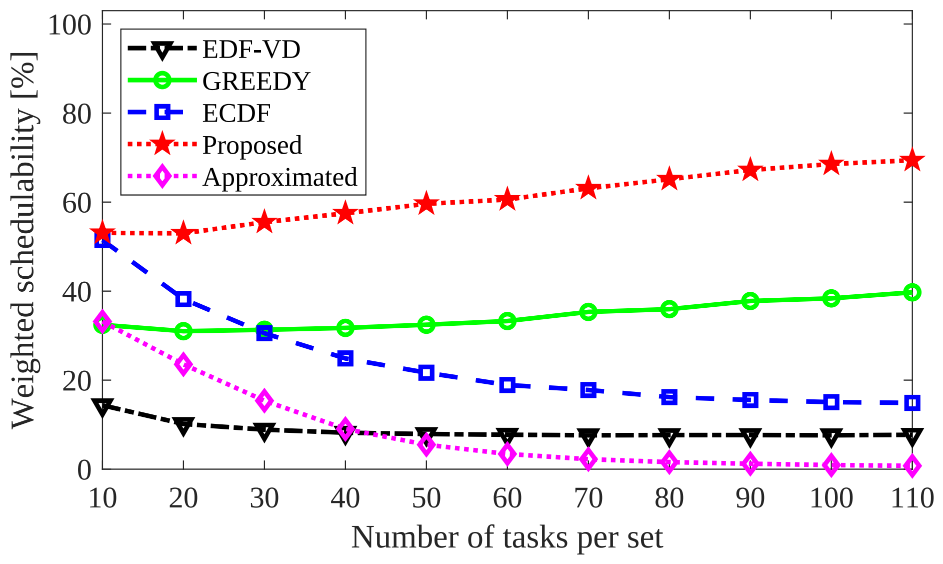


Figure B.2.: Weighted schedulability vs. total number of tasks for 30% HI tasks and 50% increase of HI execution demand — uniform distribution of task periods

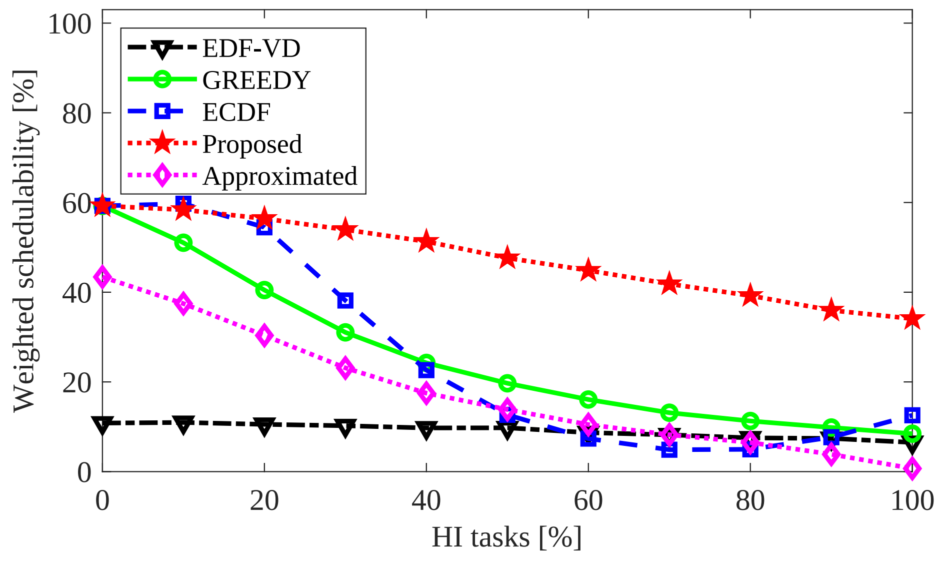


Figure B.3.: Weighted schedulability vs. percentage of HI tasks for $|\tau| = 20$ and 50% increase of HI execution demand — uniform distribution of task periods

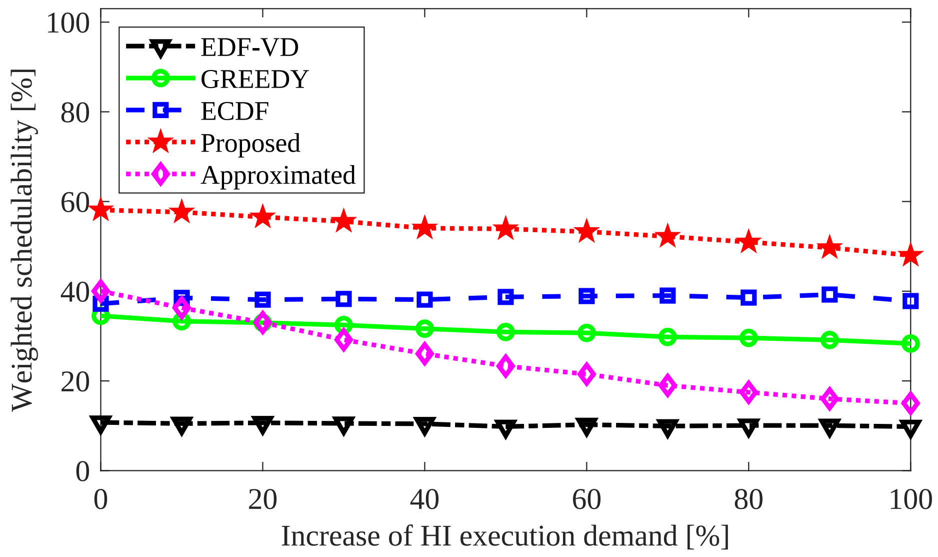


Figure B.4.: Weighted schedulability vs. increase of HI execution demand for $|\tau| = 20$ and 30% HI tasks — uniform distribution of task periods

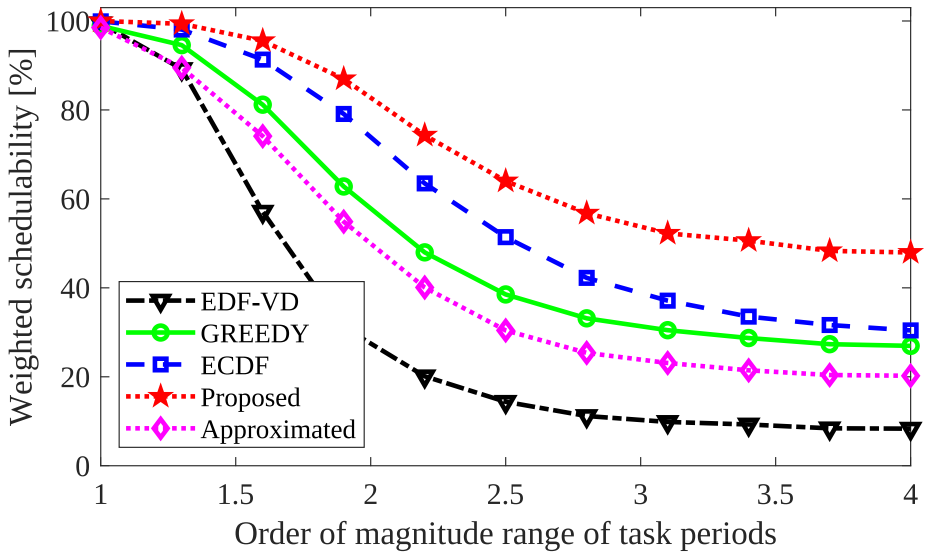


Figure B.5.: Weighted schedulability vs. range of task periods for $|\tau| = 20$, 30% HI tasks and 50% increase of HI execution demand — uniform distribution of task periods

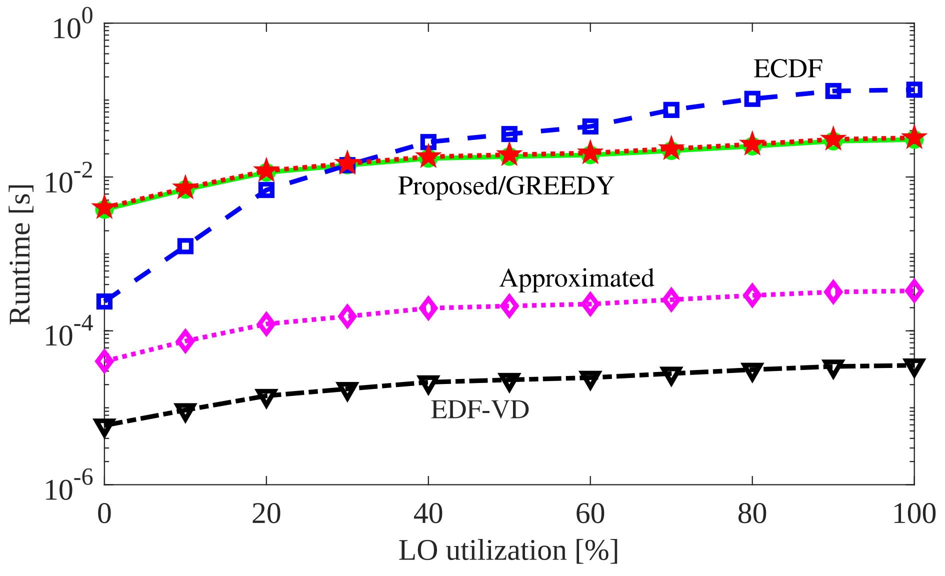


Figure B.6.: Runtime vs. LO utilization for $|\tau| = 20$, 30% HI tasks and 50% increase of HI execution demand — uniform distribution of task periods

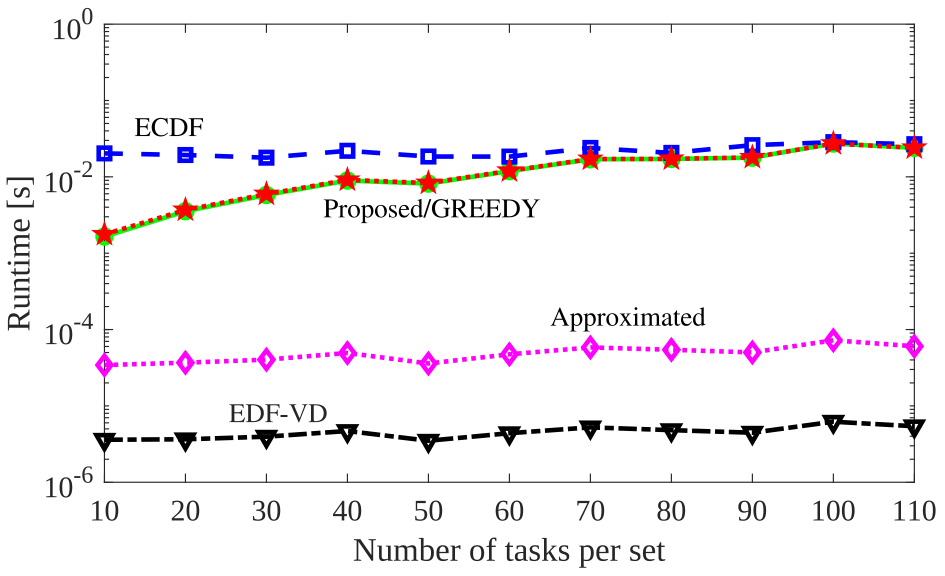


Figure B.7.: Runtime vs. total number of tasks for 30% HI tasks and 50% increase of HI execution demand — uniform distribution of task periods

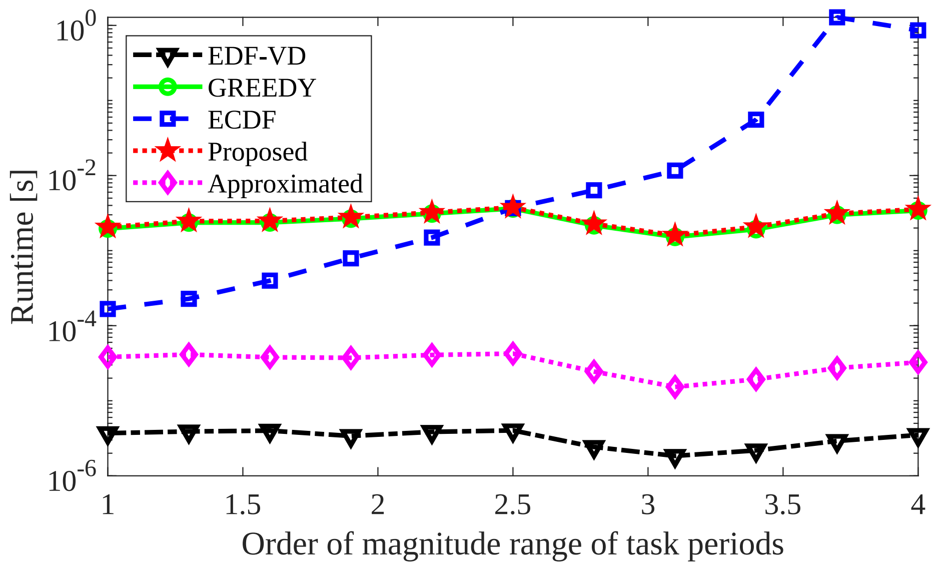


Figure B.8.: Runtime vs. range of task periods for $|\tau| = 20$, 30% HI tasks and 50% increase of HI execution demand — uniform distribution of task periods

Summary of Findings

1. **Utilization caps:** We introduced utilization caps to the original EDF-VD (Earliest Deadline First with Virtual Deadlines) algorithm, which is typically used for scheduling mixed-criticality (MC) tasks. To this end, an MC task set is partitioned into disjoint subsets, each of which is assigned a portion of the total processor utilization. EDF-VD is then applied to each such subset or partition independently. As a result, low-criticality (LO) tasks within one partition are not affected by high-criticality (HI) tasks from other partitions in case that the latter switch to HI mode, which normally cause the abortion of LO tasks (as per EDF-VD) to accommodate increases in execution demand by HI tasks. 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. This already allows for some LO tasks to be protected from switching to HI mode enabling for more design flexibility. This resulted in the proposed Utilization Caps algorithm of Chapter 4.

2. **Better exact schedulability test:** The idea is to better bound execution demand under mixed-criticality EDF. To this end, we derive a separate demand bound function for transitions from LO to HI mode and prove its validity. This technique allows us to work around the computation of carry-over execution demand and, hence, to reduce the amount of pessimism in characterizing mixed-criticality EDF. We further proved that the proposed technique leads to a tighter bound on the execution demand under mixed criticality EDF. It is interesting to notice that the proposed technique reduces the problem of testing schedulability under mixed criticality EDF to testing schedulability of three almost unrelated task sets: the one in LO mode, the one in HI mode and the equivalent task set for transitions between LO and HI mode. This leads to a considerably simpler schedulability test and improves our understanding of this problem. This has led to the proposed a new Demand Bound Function of Chapter 5.

3. **Better approximated tests:** Since we showed that testing schedulability for mixed-criticality EDF boils down to testing three separate task sets under standard EDF scheduling, we are now able to extend and apply known approximation techniques from the literature (originally conceived for standard EDF). In particular, we extend the so-called Devi's test to be used in the context of MC systems. This extension of Devi's test is also sufficient but not necessary, however, it is more accurate than using utilization caps (our first approach) and is considerably faster than any exact test (including the one proposed in this thesis). This represents a good trade-off between accuracy in testing schedulability for MC systems and running time of the test (relevant in on-line settings where testing needs to be done while the system is running, e.g., admission control). Moreover, we presented a large set of experiments based on synthetic data illustrating the benefits of the proposed approach in term of weighted schedulability and runtime compared to the most prominent approaches from the literature. This resulted in approving new demand bound function and presenting an approximation technique of Chapter 6.

List of own publications

The following publications were published in the context of this work.

Journal publications

- M. Mahdiani and A. Masrur, *A Novel View on Bounding Execution Demand under Mixed-Criticality EDF*, In Springer Real-Time Systems (RTS)-The International Journal of Time-Critical Computing Systems, 2020

Peer-reviewed conferences (acceptance ratio < 30 %)

- M. Mahdiani and A. Masrur, *On Bounding Execution Demand under Mixed-Criticality EDF*, (Outstanding paper), In Proceedings of the 26th International Conference on Real-Time Networks and Systems (RTNS'18), 2018
- M. Mahdiani and A. Masrur, *Introducing Utilization Caps into Mixed-Criticality Scheduling*, In Proceedings of the 19th Euromicro Conference on Digital Systems Design (DSD), 2016

Mitra Mahdiani — Curriculum Vitae

Contact Information	Dept. of Computer Science TU Chemnitz Straße der Nationen 62 09111 Chemnitz, Germany	**Office:** **Mobile:** **Email:**	+49 371 531 34672 +49 1516 3967 503 mitra.mahdiani.83@gmail.com

Research Interest

My research interests cover synchronous languages and real-time scheduling for embedded systems, in particular, scheduling of mixed-criticality real-time systems. To this end, a mix of Low and High criticality tasks should be considered to schedule on one or more processors under the different scheduling algorithms and provide guarantee meeting timing constraints for this kind of systems to operate correctly. This includes new approaches to improve scheduling techniques.

Education

2015 - 2019 PhD, Computer Science - Technical University Chemnitz, Germany

Field: Embedded Systems and Real-Time Scheduling
Thesis (PhD): *Advanced Scheduling Techniques for Mixed Criticality Systems*

2010 - 2013 M.Sc., Information Science and Technology - National University of Malaysia (UKM), Malaysia

Field: Information Technology - Industrial Computing (total grade 1.4)
Thesis (M.S.): *Abnormal Control Chart Pattern Classification Optimisation Using Multilayered Perceptron and Bees Algorithm*

2003 - 2007 B.Sc., Sciences - Azad University of Gorgan, Iran

Field: Applied Mathematics (total grade 2.0)

2001 - 2002 Post-Secondary Diploma - Basirat Pre–University School, Kordkuy, Iran

Major: Mathematics and Physics (grade 1.3)

1998 - 2001 High School Diploma - Shahid Ali Miandare High School, Kordkuy, Iran

Major: Theoretical and Techno–vocational branch of Mathematics and Physics Discipline (grade 1.7)

Teaching

- Software Platforms for Automotive Systems (En), Winter Term 2015/16
- Seminar AUTOSAR Based Software Design (En), Winter Term 2016/17 - Summer Term 2019

Employment History

2014 - 2015 Faculty of Information Science and Technology, The National University of Malaysia (UKM) *(43600 UKM Bangi, Selangor, Malaysia)*

Research Assistant

Working for one year as a research assistant in department of visual informatic to bridge the time between studying and PhD.

Scope: Cooperation in publishing papers, research collaboration with visual informatic group and doing research on different thema such as stock prediction, CAD converter for product design innovation in manufacturing.

2007 - 2008 Resana Afzar Sharif Ltd. (No.10, Sanaii St. Karimkhan Zand Ave., Tehran, Iran)
Project Managment Assistant

Working for 3 months as a trainee and 1 year full position.

Scope: Directing, scheduling and optimizing the mid-size IT and Telecommunication projects.

2008 - 2009 Behbood Gostaran Sanat Tabarestan Co. (No.1, Ansar Alley, Gharan St., Sari, Iran)
IT Assistant

Working for 3 month as a trainee and 9 months full position.

Scope: Managing of IT projects in research and industrial projects, assist in planning, coordination and monitoring IT projects, implementing, support and developing of systems (Soft-/Hardware).

Software Engineering Skills

- **Programming:** Java (intermediate), Matlab (good), Javascript (good), Scade (basic), Esterel (basic), C/C++ (basic)
- **Frameworks/IDEs:** ReactJS (intermediate), ES6 (intermediate), HTML5 (good), CSS (good)

Languages

- Persian (native)
- English (fluent)
- German (intermediate)
- Malay (basic)

Interests

- Dancing
- Traveling
- Cycling, Jogging and Gym
- Volleyball, Badminton and Zumba
- Reading Books, Watching Movie and Listening to Music

Erklärungen

[✓] Ich versichere, dass die vorgelegte Arbeit weder im Inland noch im Ausland in gleicher oder in ähnlicher Form einer anderen Prüfungsbehörde zum Zwecke einer Promotion oder eines anderen Prüfungsverfahren vorgelegt wurde und auch noch nicht veröffentlicht wurde.

☐ Es fand ein früheres Promotionsverfahren statt. ☐ Ja [✓ Nein]
(wenn ja)

Thema:			
Bescheid:		Zeit:	
Hochschule:			

[✓] Ich versichere, dass die vorliegende Arbeit ohne unzulässige Hilfe und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt wurde und die aus fremden Quellen direkt oder indirekt übernommenen Gedanken in der Arbeit als solche kenntlich gemacht sind.

[✓] Ich versichere, dass weitere Personen bei der geistigen Herstellung der vorliegenden Arbeit nicht beteiligt waren, insbesondere auch nicht die Hilfe eines Promotionsberaters in Anspruch genommen wurde, und dass Dritte vom Bewerber weder unmittelbar noch mittelbar geldwerte Leistungen für Arbeiten erhalten haben, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.

[✓] Ich bin mit der elektronischen Überprüfung meiner Dissertation auf etwaige Plagiate hin einverstanden.

_______________________ _______________________
Datum *Unterschrift*