Realistic Scheduling Models and Analyses for Advanced Real-Time Embedded Systems

Dissertation

zur Erlangung des Grades eines

Doktors der Ingenieurwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik

von

Georg von der Brüggen

Dortmund

2019

Tag der mündlichen Prüfung:	14. November 2019
Dekan / Dekanin:	Prof. Dr. Gernot Fink
Gutachter / Gutachterinnen:	Prof. Dr. Jian-Jia Chen (TU Dortmund University)
	Dr. Robert I. Davis (Reader at the University of York)

ABSTRACT

In real-time embedded systems, for each task the compliance to timing constraints has to be guaranteed in addition to the functional correctness. From a researcher's perspective, the examination of real-time scheduling focuses on three components that build on one another: 1) the system and task model, 2) the scheduling algorithm with related schedulability test, and 3) a theoretical and/or empirical performance evaluation of the scheduling algorithm. These three components are examined in this dissertation considering the following hypothesis:

> *Realistic scheduling models and analyses are essential for guaranteeing timing correctness in advanced real-time systems while ensuring that the system resources necessary to provide these guarantees are not over-provisioned.*

The dissertation is structured in three parts according to the considered task and system model. The first part primarily examines the classic periodic and sporadic task model and focuses on general theoretical methods for comparison. It is shown how utilization bounds can be parametrized to provide significantly higher utilization bounds when analyzing non-preemptive Rate Monotonic scheduling as well as task sets inspired by automotive applications. Afterwards, new speedup factors for non-preemptive Deadline Monotonic scheduling compared to non-preemptive Earliest Deadline First are provided and it is shown that tight speedup factors cannot only be achieved for schedulability tests with an exponential time complexity but also for linear time tests which in practice result in a significantly worse acceptance ratio. The findings for utilization bounds and speedup factors lead to a general discussion about these long standing standard techniques and guidance for their meaning and interpretation is provided. Furthermore, parametric augmentation functions are proposed as a possible solution.

The second part of the dissertation considers uncertain execution environments where tasks have multiple execution modes that differ regarding their worst-case execution time and where modes with large execution times are assumed to be rare. Such a setting is common in mixed criticality systems or when software-based fault tolerance mechanisms are exploited. However, this connection has never been examined in the literature. First, criticism on previous work in the area of mixed-criticality is detailed, i.e., that such systems are assumed to never return to the low-criticality mode, that low-criticality tasks are treated without any service guarantees, and that most mixed-criticality scheduling approaches assume online adaption. Hence, a new system model, namely Systems with Dynamic Real-Time Guarantees, is provided that allows a better applicability to realistic scenarios by providing guarantees offline without online adaptation under static-priority scheduling. Nevertheless, the approach is shown to be comparable to the state-of-the-art for mixed-criticality systems. The system model is afterwards extended to a multiprocessor scenario, considering both partitioned and semi-partitioned scheduling, and introducing task migration techniques to provide guarantees under intermittent faults. In addition, a new way to determine the worst-case

deadline failure probability for such systems is provided that drastically reduces the runtime of such calculations compared to the state-of-the-art.

The third part of the dissertation focuses on tasks with self-suspension behaviour. A new algorithm called SEIFDA that increases the schedulability under a fixed-relative-deadline strategy when considering segmented self-suspension tasks with one suspension interval is provided that outperforms the state-of-the-art. Considering multiprocessor resource sharing, SEIFDA is afterwards utilized in a resource-oriented partitioned scheduling with release enforcement, again outperforming the state-of-the-art. Furthermore, the gap between the dynamic and the segmented self-suspension model is examined. The dynamic self-suspension model can be utilized when only limited information about the suspension behavior is known and hence has a high flexibility, but results in more pessimistic analyses and designs of scheduling policies if the suspending pattern can be defined precisely. The segmented self-suspension has a lower flexibility, but the self-suspending structure can be exploited by the scheduling algorithms. However, a static execution pattern for each execution is a rather strong assumption. This gap is bridged by introducing multiple hybrid self-suspension models, which assume the self-suspending tasks to be specified by a set of possible execution patterns that are known offline.

The results presented in this dissertation show the importance of realistic models and analyses in real-time systems research and, therefore, support the dissertation hypothesis.

PUBLICATIONS

The majority of the ideas and findings presented in this dissertation have been published in the following peer-reviewed articles that appeared in international journals and proceedings of international conferences:

[BCH15] Georg von der Brüggen, Jian-Jia Chen, and Wen-Hung Huang. "Schedulability and Optimization Analysis for Non-preemptive Static Priority Scheduling Based on Task Utilization and Blocking Factors." In: *27th Euromicro Conference on Real-Time Systems, ECRTS 2015, Lund, Sweden, July 8-10, 2015*. 2015, pp. 90–101.

[BCH+16] Georg von der Brüggen, Kuan-Hsun Chen, Wen-Hung Huang, and Jian-Jia Chen. "Systems with Dynamic Real-Time Guarantees in Uncertain and Faulty Execution Environments." In: *2016 IEEE Real-Time Systems Symposium, RTSS 2016, Porto, Portugal, November 29 - December 2, 2016*. 2016, pp. 303–314.

[BHC+16] Georg von der Brüggen, Wen-Hung Huang, Jian-Jia Chen, and Cong Liu. "Uniprocessor Scheduling Strategies for Self-Suspending Task Systems." In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems, RTNS 2016, Brest, France, October 19-21, 2016*. 2016, pp. 119–128.

[BCD+17] Georg von der Brüggen, Jian-Jia Chen, Robert I. Davis, and Wen-Hung Huang. "Exact speedup factors for linear-time schedulability tests for fixed-priority preemptive and non-preemptive scheduling." In: *Information Processing Letters* 117 (2017).

[BHC17] Georg von der Brüggen, Wen-Hung Huang, and Jian-Jia Chen. "Hybrid self-suspension models in real-time embedded systems." In: *23rd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2017, Hsinchu, Taiwan, August 16-18, 2017*. 2017, pp. 1–9.

[BUC+17] Georg von der Brüggen, Niklas Ueter, Jian-Jia Chen, and Matthias Freier. "Parametric utilization bounds for implicit-deadline periodic tasks in automotive systems." In: *Proceedings of the 25th International Conference on Real-Time Networks and Systems, RTNS 2017, Grenoble, France, October 04 - 06, 2017*. 2017, pp. 108–117.

[BCH+17] Georg von der Brüggen, Jian-Jia Chen, Wen-Hung Huang, and Maolin Yang. "Release enforcement in resource-oriented partitioned scheduling for multiprocessor systems." In: *Proceedings of the 25th International Conference on Real-Time Networks and Systems, RTNS 2017, Grenoble, France, October 04 - 06, 2017*. 2017, pp. 287–296.

[BPC+18] Georg von der Brüggen, Nico Piatkowski, Kuan-Hsun Chen, Jian-Jia Chen, and Katharina Morik. "Efficiently Approximating the Probability of Deadline Misses in Real-Time Systems." In: *30th Euromicro Conference on Real-Time Systems, ECRTS 2018, July 3-6, 2018, Barcelona, Spain*. 2018, 6:1–6:22.

[BSC18] Georg von der Brüggen, Lea Schönberger, and Jian-Jia Chen. "Do Nothing, But Carefully: Fault Tolerance with Timing Guarantees for Multiprocessor Systems Devoid of Online Adaptation." In: *23rd IEEE Pacific Rim International Symposium on Dependable Computing, PRDC 2018, Taipei, Taiwan, December 4-7, 2018*. 2018, pp. 1–10.

[CBH+17a] Jian-Jia Chen, Georg von der Brüggen, Wen-Hung Huang, and Robert I. Davis. "On the Pitfalls of Resource Augmentation Factors and Utilization Bounds in Real-Time Scheduling." In: *29th Euromicro Conference on Real-Time Systems, ECRTS 2017, June 27-30, 2017, Dubrovnik, Croatia*. 2017, 9:1–9:25.

As part of my research, I also contributed to the following peer-reviewed articles that appeared in international journals and proceedings of international conferences but are ot part of this dissertation:

[CBC+18] Jian-Jia Chen, Nikhil Bansal, Samarjit Chakraborty, and Georg von der Brüggen. "Packing Sporadic Real-Time Tasks on Identical Multiprocessor Systems." In: *29th International Symposium on Algorithms and Computation, ISAAC 2018, December 16-19, 2018, Jiaoxi, Yilan, Taiwan*. Vol. 123. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018, 71:1–71:14.

[CBH+17b] Jian-Jia Chen, Georg von der Brüggen, Wen-Hung Huang, and Cong Liu. "State of the art for scheduling and analyzing self-suspending sporadic real-time tasks." In: *23rd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2017, Hsinchu, Taiwan, August 16-18, 2017*. IEEE Computer Society, 2017, pp. 1–10.

[CBS+18] Jian-Jia Chen, Georg von der Brüggen, Junjie Shi, and Niklas Ueter. "Dependency Graph Approach for Multiprocessor Real-Time Synchronization." In: *2018 IEEE Real-Time Systems Symposium, RTSS 2018, Nashville, TN, USA, December 11-14, 2018*. IEEE Computer Society, 2018, pp. 434–446.

[CBU18] Jian-Jia Chen, Georg von der Brüggen, and Niklas Ueter. "Push Forward: Global Fixed-Priority Scheduling of Arbitrary-Deadline Sporadic Task Systems." In: *30th Euromicro Conference on Real-Time Systems, ECRTS 2018, July 3-6, 2018, Barcelona, Spain*. Vol. 106. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018, 8:1–8:24.

[CHH+19] Jian-Jia Chen, Tobias Hahn, Ruben Hoeksma, Nicole Megow, and Georg von der Brüggen. "Scheduling Self-Suspending Tasks: New and Old Results." In: *31st Euromicro Conference on Real-Time Systems, ECRTS 2019, July 9-12, 2019, Stuttgart, Germany.* Vol. 133. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019, 16:1–16:23.

[CNH+19] Jian-Jia Chen, Geoffrey Nelissen, Wen-Hung Huang, Maolin Yang, Björn B. Brandenburg, Konstantinos Bletsas, Cong Liu, Pascal Richard, Frédéric Ridouard, Neil C. Audsley, Raj Rajkumar, Dionisio de Niz, and Georg von der Brüggen. "Many suspensions, many problems: a review of self-suspending tasks in real-time systems." In: *Real-Time Systems* 55.1 (2019), pp. 144–207.

[CBC18a] Kuan-Hsun Chen, Georg von der Brüggen, and Jian-Jia Chen. "Analysis of Deadline Miss Rates for Uniprocessor Fixed-Priority Scheduling." In: *24th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2018, Hakodate, Japan, August 28-31, 2018*. IEEE Computer Society, 2018, pp. 168–178.

[CBC16] Kuan-Hsun Chen, Georg von der Brüggen, and Jian-Jia Chen. "Overrun Handling for Mixed-Criticality Support in RTEMS." In: *WMC 2016*. Proceedings of WMC 2016. Porto, Portugal, 2016.

[CBC18b] Kuan-Hsun Chen, Georg von der Brüggen, and Jian-Jia Chen. "Reliability Optimization on Multi-Core Systems with Multi-Tasking and Redundant Multi-Threading." In: *IEEE Trans. Computers* 67.4 (2018), pp. 484–497.

[CUB+19] Kuan-Hsun Chen, Niklas Ueter, Georg von der Bruggen, and Jian-Jia Chen. "Efficient Computation of Deadline-Miss Probability and Potential Pitfalls." In: *Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy, March 25-29, 2019*. IEEE, 2019, pp. 896–901.

[DLB+18] Zheng Dong, Cong Liu, Soroush Bateni, Kuan-Hsun Chen, Jian-Jia Chen, Georg von der Brüggen, and Junjie Shi. "Shared-Resource-Centric Limited Preemptive Scheduling: A Comprehensive Study of Suspension-Based Partitioning Approaches." In: *IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2018, 11-13 April 2018, Porto, Portugal*. IEEE Computer Society, 2018, pp. 164–176.

[HFB+18] Tim Harde, Matthias Freier, Georg von der Brüggen, and Jian-Jia Chen. "Configurations and Optimizations of TDMA Schedules for Periodic Packet Communication on Networks on Chip." In: *Proceedings of the 26th International Conference on Real-Time Networks and Systems, RTNS 2018, Chasseneuil-du-Poitou, France, October 10-12, 2018*. ACM, 2018, pp. 202–212.

[HCB+18] Nils Hölscher, Kuan-Hsun Chen, Georg von der Brüggen, and Jian-Jia Chen. "Examining and Supporting Multi-Tasking in EV3OSEK." In: *OSPERT 2018* (2018), p. 25.

[SBS+19] Lea Schönberger, Georg von der Bruggen, Horst Schirmeier, and Jian-Jia Chen. "Design Optimization for Hardware-Based Message Filters in Broadcast Buses." In: *Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy, March 25-29, 2019*. IEEE, 2019, pp. 606–609.

[SHB+18] Lea Schönberger, Wen-Hung Huang, Georg von der Brüggen, Kuan-Hsun Chen, and Jian-Jia Chen. "Schedulability Analysis and Priority Assignment for Segmented Self-Suspending Tasks." In: *24th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2018, Hakodate, Japan, August 28-31, 2018*. IEEE Computer Society, 2018, pp. 157–167.

[UBC+18] Niklas Ueter, Georg von der Brüggen, Jian-Jia Chen, Jing Li, and Kunal Agrawal. "Reservation-Based Federated Scheduling for Parallel Real-Time Tasks." In: *2018 IEEE Real-Time Systems Symposium, RTSS 2018, Nashville, TN, USA, December 11-14, 2018*. IEEE Computer Society, 2018, pp. 482–494.

ACKNOWLEDGMENTS

I want to take this chance to express my gratitude to the people who helped and supported me during my years as a PhD student. Without them, I would not be where I am today.

First, I would like to thank my PhD adviser Jian-Jia Chen for his guidance, advice, support, trust, and understanding. I could not have asked for a better adviser. I also would like to thank my second reviewer Rob Davis for his guidance, support, and advice. I surely cannot thank him enough for all his time and effort. Furthermore, I want to thank Heinrich Müller and Jens Teubner for being part of my committee, and Peter Marwedel for being my PhD mentor.

I would like to express my thanks and appreciation to all my co-authors, namely Jian-Jia Chen, Wen-Hung Huang, Kuan-Hsun Chen, Niklas Ueter, Junjie Shi, Lea Schönberger, Marco Dürr, Mikail Yayla, and Christian Hakert from the real-time systems group, Horst Schirmeier, Sebastian Buschjäger, Niko Piatkowski, and Katharina Morik from other groups at TU Dortmund, as well as my external co-authors Cong Liu, Rob Davis, Nicole Megow, Ruben Hoeksma, Maolin Yang, Jing Li, Kunal Agrawal, Nikhil Bansal, Samarjit Chakraborty, Zheng Dong, Tobias Hahn, Tim Harde, Matthias Freier, Geoffrey Nelissen, Björn Brandenburg, Konstantinos Bletsas, Pascal Richard, Frédéric Ridouard, Neil Audsley, Raj Rajkumar, Dionisio de Niz, Soroush Bateni, Paul Genssler, Lars Bauer, Hussam Amrouch, Jörg Henkel, Tulika Mitra, and Vanchinathan Venkataramani. I am very thankful for the opportunity to work with all of them.

I want to thank the members and visitors of the real-time systems group, in addition to the ones mentioned above especially Anas Toma who I unfortunately did not have the chance to collaborate with for a paper, as well as the other members of the embedded systems chair for making my time at TU Dortmund both interesting and enjoyable, our secretary Claudia Graute for taking care of so many things over the years, and Kevin, Horst, Olaf, Jan, Boguslaw, Markus, Alex, Hendrik, and Timon for all their help when I started and over the years.

I would like to thank my parents Willi and Maria as well as my brother Richard and my sister Andrea for their support, both during my PhD and before. I especially want to thank Andrea for all her feedback while writing my thesis.

Last but certainly not least I want to thank my friends for bearing with me through all these years and for their unwavering support, understanding, and encouragement. I want to thank Marc, Charlotte, Miri, and Anne for the music, TK, Carsten, and Nicole for letting me win, Hajoh for the drinks, Dennis for the food, Markus for gravity, Lena for maybe coffee, Becky for her optimistic attitude towards life, JC for not working at night, and Kuan-Hsun for suffering with me.

My work has been supported by the Deutsche Forschungsgemeinschaft (DFG) as part of the Collaborative Research Center SFB 876 in the project A1. Multiple of my co-authors have been (partially) funded by SFB 876 - A1 and from other projects, namely SFB 876 - A3 and SFB 876 - B2, as well as from DFG SPP 1500.

CONTENTS

1 INTRODUCTION

Nowadays, society strongly relies on computing systems when controlling a large variety of complex physical plants, e.g., for the automation of industrial processes, automotive and avionic systems, and traffic control. These computing systems are often not only *embedded* into the plant controlled by them but "*built from and depend upon the synergy of computational and physical components*" [Nat13], i.e., they are not only part of the plant but directly interact with it and its environment, and are, therefore, referred to as *cyber-physical systems*. A detailed introduction into embedded and cyber-physical systems can be found in the textbook by Marwedel [Mar11].

embedded system

cyber-physical system

While for general-purpose computation systems an accurate system behaviour only depends on the functional correctness, i.e., that for a given input the expected output is computed, for many cyber-physical systems timing correctness must be achieved as well, i.e., the expected output must be computed within certain timing constraints. Computing systems that require both functional and timing correctness are called *real-time systems* [SR88]. On the one hand, a system is called a *soft real-time system* if the computation result is of lower value if the timing constraints are not met, e.g., in multimedia systems. Hence, in soft real-time system, not meeting the timing constrains leads to a reduces quality of service. On the other hand, the system is called a *hard real-time system* if not fulfilling the timing constraints leads to useless results, e.g., avionic systems, automotive systems, and traffic control. Therefore, in hard real-time system, not meeting the timing constrains represents a system failure.[1] In this work, we focus on such hard real-time systems.

real-time system

soft real-time system

hard real-time system

For hard real-time systems a precise analysis of the timing behaviour is of the utmost importance. However, practical systems are too complex to decide the compliance to these constraints directly. Hence, the characteristics of the system should first be observed and then modelled as precisely as possible. Afterwards, the timing behaviour can be analyzed more efficiently based on this model. Due to the complex nature of these systems, the modelling usually focuses on some parts of the system behaviour while abstracting others.

The goal of this dissertation is to explore how more realistic models and analyses for real-time scheduling can be achieved. This chapter first explains some general concepts regarding real-time systems, followed by three sections that introduce the topics that are analyzed in depth later in this dissertation, i.e., in Chapter 4 through Chapter 6. The considered task and system models are detailed

1 Note that different ways of categorization for real-time systems can be found in the literature. For example, the textbook by Buttazzo [But11] distinguishes three categories based on the effect of a deadline miss: 1) *hard real-time*, if a deadline miss may have catastrophic consequences for the system, 2) *firm real-time*, if a result after a deadline miss is useless but does not cause any damage, and 3) *soft real-time*, if a result after the deadline leads to performance degradation.

in Chapter 2 while Chapter 3 provides insight into the related work. Chapter 4 discusses how scheduling algorithms and schedulability tests are compared using theoretical methods like speedup factors and utilization bounds, which problems may arise when these methods are used for comparison, what kind of conclusions should be avoided, and how utilization bounds can be parametrized to improve their meaningfulness. Chapter 5 focuses on uncertain execution behaviour of tasks, explores the link between mixed-criticality and fault tolerance, and answers to criticism that mixed-criticality approaches have received, by introducing *Systems with Dynamic Real-Time Guarantees* for uniprocessor scenarios, which are extended to multiprocessor systems afterwards. The chapter concludes with a method that allows to calculate the worst-case deadline failure probability for tasks in uncertain execution environments under static-priority scheduling for task sets that have a reasonably large number of tasks, i.e., up to 100 tasks while previous approaches only allow up to 10 tasks. In Chapter 6, self-suspending task systems are considered and a new scheduling algorithm is presented for tasks that can be described by the segmented self-suspension model with one suspension interval. This algorithm is shown to outperform the current state-of-the-art and is later exploited in a resource-oriented partitioning algorithm for multiprocessor resource sharing, where the state-of-the-art is again outperformed. The chapter is concluded by introducing hybrid self-suspension task models that bridge the gap between the overly restrictive segmented self-suspension model and the overly pessimistic dynamic self-suspension model by carefully including all given information into the modelling. The conclusion of this dissertation is presented in Chapter 7.

1.1 GENERAL CONCEPTS IN REAL-TIME SCHEDULING

real-time system

When investigating a *real-time system*, a schedulability analysis determines if a system is feasible under a scheduling algorithm, i.e., if for a given set of tasks all task instances will meet their deadlines under a specific scheduling algorithm. This process typically consists of modelling the tasks and the system in a suitable way, and determining the schedulability under a scheduling algorithm by applying a related schedulability test. Here, we take a look at basic modelling principles, describe how scheduling algorithms can be classified, and provide a high-level view on schedulability tests and the related problems. However, we focus on the topics relevant to this dissertation. For a detailed introduction into real-time systems the reader is referred to the textbook by Buttazzo [But11].

1.1.1 MODELLING OF REAL-TIME SYSTEMS

In many real-time systems, tasks are executed recurrently, e.g., a sensor value is read repeatedly with a certain delay between two readings, the determined value is processed, and the result may lead to an action that changes the current system behaviour. In a multitasking scenario, a set of tasks is executed on a given execution platform. Such a set of n tasks $\mathbf{T} = \{\tau_1, ..., \tau_n\}$ is usually described

by the *periodic* or the *sporadic task model* where each task $\tau_i = (C_i, D_i, T_i)$ is characterized by its *minimum inter-arrival time* (or period) T_i, its *relative deadline* D_i, and its *worst-case execution time* (WCET) C_i. Each task releases an infinite number of *task instances*, called *jobs*, according to its minimum inter-arrival constraint. A task is called (strictly) *periodic* [LL73] if two consequent job releases are always separated exactly by the task's minimum inter-arrival, and *sporadic* [Mok83] if two consequent job releases are separated at least by the task's minimum inter-arrival time. If a task τ_i releases jobs periodically, the *phase* of the task, i.e., the release time of its first instance, is denoted with ϕ_i. If ϕ_i is omitted in the task specification, it is assumed to be 0. A job released at time t must be able to be executed for up to C_i time units before its absolute deadline at time $t + D_i$ to fulfill its timing requirements. The *utilization* of a task is defined as $U_i = \frac{C_i}{T_i}$ and the total utilization of a task set is $U_{sum} = \sum_{\tau_i \in \mathbf{T}} U_i$.

period
relative deadline
worst-case execution time
job
periodic task
sporadic task
phase
utilization

Tasks and therefore the resulting task systems are often distinguished based on the relation between the inter-arrival times and relative deadlines of the tasks. A task τ_i is called an *implicit-deadline* task if its relative deadline is equal to its period, and a *constrained-deadline* task if its relative deadline is not larger than its period. Accordingly, a task set $\mathbf{T} = \{\tau_1, ..., \tau_n\}$ is an *implicit-deadline* task set if all tasks have implicit deadlines, a *constrained-deadline* task set if all tasks have constrained deadlines, and an *arbitrary-deadline* task set if tasks are allowed to have a relative deadline larger than their period. Note that the implicit-deadline task sets are a subset of the constrained-deadline task sets which are, in turn, a subset of the arbitrary-deadline task sets.

implicit-deadline
constrained-deadline
arbitrary-deadline

The periodic and the sporadic task model have been extended to cover more complex task systems, e.g., mixed-criticality task systems [Ves07], self-suspending task systems [CBH+17b], or multiframe task systems [MC96]. If not stated otherwise, we assume that all task parameters considered in the underlying model are predetermined, i.e., they are known both at design time and at runtime, and that tasks will always respect their parameters. Especially, a task will not release jobs more frequent than allowed by the inter-arrival time restriction, and no job of a task will execute for more than the task's *worst-case execution time*. However, we note that even for tasks that can be exactly described by the periodic or the sporadic task model and that are executed in a uniprocessor environment, a precise *WCET analysis* is still a very challenging problem. A survey of WCET analysis was provided by Wilhelm et al. [WEE+08].

worst-case execution time
WCET analysis

1.1.2 CLASSIFICATION OF SCHEDULING ALGORITHMS

In most modern real-time embedded systems resources are not exclusively assigned to one computation task but multiple jobs of a set of tasks have to be executed on a given execution platform. Therefore, if multiple jobs are active at the same time, these jobs have to compete for shared resources like execution devices, cache, communication channels, etc. We mainly consider the competition for the processor(s), i.e., the (main) execution device(s) of the platform. For convenience, uniprocessor systems are assumed unless stated otherwise. To enable *multi-tasking*, i.e., multiple tasks executing jobs on the same processor, a *scheduling*

multi-tasking

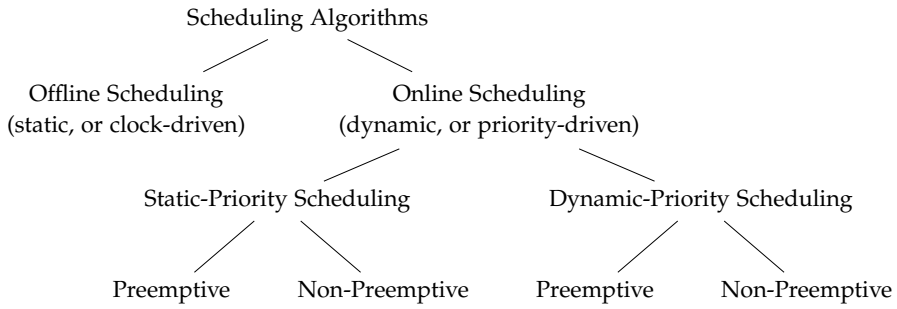


Figure 1.1: Classification of scheduling algorithms.

scheduling policy

policy is needed, i.e., a set of predefined criteria chosen by the designers and/or enforced by the system at hand. At any given point in time, the jobs that are currently available for execution are called *ready*. The actual scheduling decision, i.e., which *ready* job is executed at a specific point in time, is made based on a

scheduling algorithm

scheduling algorithm that implements the policy by providing a set of rules to determine the order in which jobs are executed at any given time. According to

dispatcher

the scheduling algorithm, the *dispatcher* ensures that during runtime the job with the highest priority among the ready jobs currently in the system is dispatched to and executed on the processor. Whether all jobs of all tasks are able to always

schedulability test

meet their absolute deadlines can be determined by a *schedulability test* that is related to the task and system model as well as to the scheduling algorithm.

classification of scheduling policies

Scheduling policies, and therefore the related scheduling algorithms, can be classified based on certain properties. An overview is provided in Figure 1.1. One major distinction is whether a scheduling algorithm is executed offline or online,

offline scheduling

sometimes also referred to as static or dynamic scheduling.[2] An *offline schedule* is statically created at design time, i.e., before the task set is actually executed, the generated schedule is stored in a suitable data structure, e.g., a table, and during runtime the dispatcher assigns the jobs to the processor based on this static schedule and the current system time. Offline scheduling approaches are therefore also called clock-driven or time-triggered. For periodic task sets, an offline schedule is usually constructed to cover a certain time frame, normally the

hyperperiod

hyperperiod, which is the least common multiple of all periods among tasks in the system, and repeated afterwards. While for such a schedule the timing correctness can easily be verified, constructing a suitable schedule that fulfills all timing constraints is a challenging problem. Furthermore, storing the schedule may lead to a large overhead if the hyperperiod is large. In addition, the scheduling of sporadic tasks or tasks with release jitter is problematic, since the actual release times of jobs are not known at design time. Further details are omitted since this work examines online scheduling algorithms.

online scheduling

priority-based scheduling

In *online scheduling* algorithms the currently executing job is determined at runtime based on a priority system. Hence, such scheduling algorithms are also called *priority-driven* or *priority-based scheduling* algorithms. They are further

2 The terms offline algorithm and online algorithm are used here to avoid confusion since the terms static and dynamic are also used for the subclasses of online scheduling policies, also called priority-based scheduling policies.

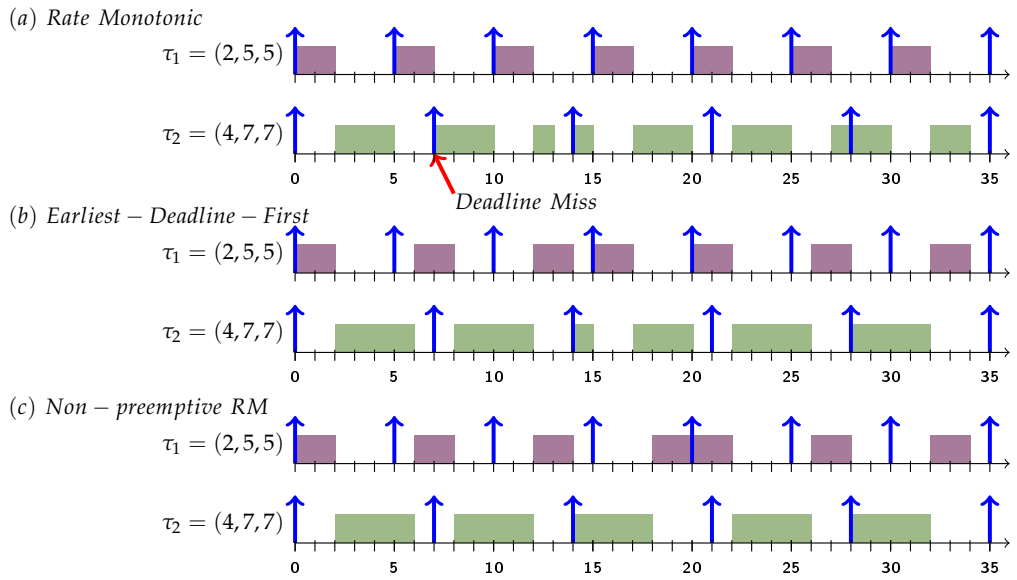


Figure 1.2: An implicit-deadline periodic task set not schedulable under preemptive *Rate Monotonic* scheduling (RM-P), but schedulable under both preemptive *Earliest Deadline First* scheduling (EDF-P) as well as *non-preemptive Rate Monotonic* scheduling (RM-NP).

divided into static-priority and dynamic-priority scheduling algorithms. In a *static-priority scheduling* algorithm, all jobs of a task have the same static (or fixed) priority. This static priority order is determined before the actual system execution. Assume that τ_i has a higher priority than task τ_j. This means, at any time t, if a job of τ_i and a job of τ_j are in the system, the job of task τ_i has the higher priority. For instance, in the *Rate Monotonic* (RM) scheduling algorithm [LL73], the task priorities are ordered according to the periods of the tasks where the task with the shortest period has the highest priority. On the other hand, in a *dynamic-priority scheduling* algorithm the priority of a job is determined at runtime based on parameters which may change over time. Therefore, it is possible that at a time point t_1 a job of task τ_i has a higher priority than a job of task τ_j while for another time point t_2 a job of task τ_j has a higher priority than a job of task τ_i. One example for dynamic-priority scheduling is the *Earliest Deadline First* (EDF) scheduling algorithm [LL73] where the job with the earliest absolute deadline has the highest priority. The difference between static-priority and dynamic-priority scheduling is exemplified in Figure 1.2. It considers two tasks τ_1 and τ_2, both described by a tuple $\tau_i = (C_i, D_i, T_i)$, where the blue arrows signal a release of the job and, since deadline and period are identical, also the absolute deadline of the previous job. In Figure 1.2(a) these two tasks are scheduled according to preemptive RM scheduling, i.e., all jobs of τ_1 have a higher priority than all jobs of τ_2 and the processor will always be assigned to a task of τ_1 the moment it enters the system. Thus, when the second job of τ_1 is released at time 5, the first job of τ_2 is preempted since the job of τ_1 has higher priority. This results in a deadline miss of the first job of τ_2 at time 7.[3] In contrast, for the dynamic EDF schedule in Figure 1.2(b), the first job of τ_2 at time 5 has a higher priority than

static-priority scheduling

Rate Monotonic

dynamic-priority scheduling

Earliest Deadline First

3 We assume that the first job of τ_2 is dropped after the deadline miss.

the second job of τ_1 since it has a deadline at 7 while the new arriving job of τ_1 has its deadline at time 10, and the EDF schedule meets all deadlines.

preemptive scheduling

Both online and offline scheduling algorithms can be further classified based on whether preemption is allowed during a job execution or not. For *preemptive scheduling* the execution of a job can be interrupted at any time, i.e., either due to a predefined schedule or because a job with higher priority arrived in the system, and the processor is assigned to another job according to the scheduling algorithm. Contrarily, in a *non-preemptive scheduling* policy a job is always executed until completion once it is granted the processor. To distinguish between preemptive and non-preemptive scheduling, we will use -P and -NP to denote the preemptive and non-preemptive version of a scheduling algorithm, e.g., RM-P and RM-NP for preemptive and non-preemptive Rate Monotonic scheduling. We note that preemption introduces context switch overhead to the system, e.g., for suspending the task, inserting it into the ready queue, flushing the processor pipeline, and dispatching the new incoming task. It is often assumed that this overhead is either neglectable or included into the tasks' worst-case execution time. Figure 1.2(c) details the non-preemptive RM schedule compared to the preemptive RM schedule in in Figure 1.2(a). *Limited-preemptive scheduling* techniques try to combine the advantages of preemptive and non-preemptive scheduling by allowing preemption only at predefined points in the task's code.

non-preemptive scheduling

limited-preemptive scheduling

Furthermore, scheduling algorithms can be classified according to their performance compared to other algorithms in the same class, as well as compared to algorithms from a different class, which is discussed in Section 1.1.4.

In practice, the class of the scheduling algorithm is chosen not only depending on considerations during the design process but also based on system properties that are enforced. For instance, many *real-time operating systems* only support static-priority scheduling [Bra11], and messages scheduled on a Controller Area Network (CAN) bus cannot be preempted [AT09].

real-time operating system

When examining multiprocessor platforms, homogeneous multiprocessor systems with m processors are assumed, i.e., all processors and therefore the task parameters on all processors are identical. Multiprocessor scheduling approaches usually follow one of three paradigms. Under *partitioned scheduling*, each task is statically allocated to a specific processor where all related task instances are executed under a uniprocessor scheduling algorithm, while under *global scheduling* tasks are allowed to migrate freely between processors and are scheduled based on one global scheduling algorithm. A *semi-partitioned scheduling* allocates the tasks to particular processors but allows a certain degree of migration, e.g., in predefined time slots or depending on specified constraints.

partitioned scheduling

global scheduling

semi-partitioned scheduling

1.1.3 SCHEDULABILITY ANALYSIS

performance metric

tardiness

The quality of different algorithms can be evaluated based on a *performance metric* or cost function. Such a metric is usually defined over the task set under analysis. We utilize the metric of *tardiness* $E_{\mathbf{T}}$ of the task set $\mathbf{T}$, i.e., the maximum amount of time any job is active after its absolute deadline, since it directly relates to the schedulability of the task set. If $E_{\mathbf{T}} = 0$ under a scheduling algorithm, then all

jobs meet their deadlines, i.e., the task set is schedulable. Since for *hard real-time systems* $E_{\mathbf{T}} = 0$ is a constraint for the correct functionality of the system, it is possible to optimize according to another metric, e.g., minimizing the average response time, as long as $E_{\mathbf{T}} = 0$ holds. However, the only criteria we consider in this work is whether $E_{\mathbf{T}} = 0$, i.e., if a task set is schedulable or not.

hard real-time system

Schedulability tests are usually related to a specific scheduling algorithm or a class thereof. They give indication whether a task set is schedulable or not and are classified as follows:

- *Sufficient tests* only deem task sets schedulable that are in fact *schedulable*. *sufficient test*
- *Necessary tests* only deem task sets not schedulable that are in fact *not schedulable*. *necessary test*
- *Exact tests* are both sufficient and necessary. *exact test*

Before a scheduling algorithm is chosen for a real-time system, it must first be determined if the considered task set is schedulable using either a sufficient or an exact test. Obviously, exact tests are preferable since they are more precise. However, it has been shown by Ekberg and Yi that, even for uniprocessor systems, an exact schedulability test for dynamic priority scheduling of constrained-deadline task sets is strongly coNP-complete [EY15] and that an exact schedulability test for sporadic tasks under static-priority scheduling is NP-hard [EY17]. Therefore, exact schedulability tests are not always a viable option and a lot of research focuses on finding tight[4] sufficient schedulability tests for different task models and scheduling algorithms.

One specific way to determine the schedulability of a task set is a (worst-case) *response time analysis*. If the *worst-case response time* R_i of task τ_i is less than or equal to the task's relative deadline D_i, the task will always fulfill its timing requirements. If for all tasks in the set the worst-case response time under the given scheduling algorithm is not larger than the relative deadline, the task set is schedulable under this algorithm. For such an analysis, a specific task τ_i must suffer the maximum interference from other tasks over a specific interval, also called the *interval of interest*. The considered interval is often between the job's release and its deadline. The situation where a task suffers the maximum possible interference is called a *critical instant* of τ_i.

response time analysis
worst-case response time
interval of interest
critical instant

The *Critical Instant Theorem* by Liu and Layland [LL73] states that for *periodic tasks* with *constrained* or *implicit deadlines* under *preemptive static-priority scheduling*, a critical instant for τ_i occurs if a job of τ_i is released together with a job of all higher-priority tasks. The same holds true for *sporadic tasks* if all subsequent jobs of higher priority tasks are released as early as possible, i.e., exactly periodically. Since no other job of τ_i can suffer from a higher interference, it is sufficient to determine the response time for the first job of τ_i under the critical instant. An example of the critical instant for task τ_4 under Rate Monotonic scheduling can be found in Figure 1.3. Note that finding a critical instant for task sets with other characteristics is (in general) not trivial. For instance, there have been several flaws in the literature regarding self-suspension that resulted from wrong assumptions for a critical instant, and no critical instant theorem for self-suspending tasks

Critical Instant Theorem
preemptive scheduling
static-priority scheduling
constrained-deadline
implicit-deadline

4 A schedulability test is called tight when it is nearly as precise as an exact test.

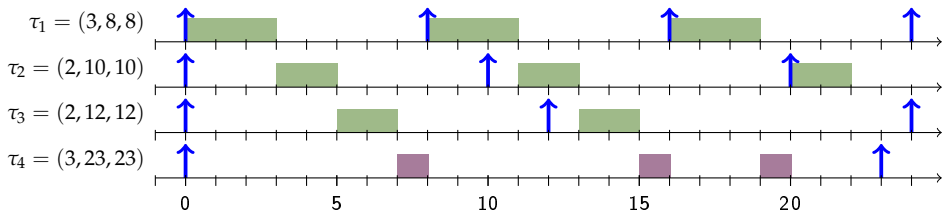


Figure 1.3: A schedule according to preemptive Rate Monotonic scheduling under the critical instant of τ_4.

has been established so far [CNH+19]. However, it is also possible to perform a response time analysis if no critical instant is known as long as the interference a task may suffer over any given interval can be safely upper bounded.

1.1.4 PERFORMANCE OF SCHEDULING ALGORITHMS

While schedulability tests analyze individual task sets, they do not directly allow a general comparison of the performance of scheduling algorithms. Hence, a way to compare different sufficient schedulability tests and/or scheduling algorithms is needed. The methods to evaluate and compare scheduling algorithms and their related schedulability tests can be divided into the following two categories:

theoretical evaluation method

- *Theoretical methods* focus on the worst-case behaviour of a scheduling algorithm or schedulability test and include *dominance relations*, *utilization bounds*, and *resource augmentation factors*. The resource augmentation factors typically compare the worst-case behaviour of a schedulability test to a specific competitor, i.e., an alternative schedulability test either for the same or for a different scheduling algorithm, and include *speedup factors*, *capacity augmentation bounds*, or *approximation ratios*.

empirical evaluation method

- *Empirical methods* evaluate the performance of a schedulability test considering typically average-case comparisons against a set of other scheduling algorithms based on a large number of considered task sets. They include the *simulation of the scheduling algorithm*, *evaluation on synthetic task sets*, *case studies*, and *experiments on real hardware*.

A review on the evaluation of scheduling algorithms was provided by Davis as part of the keynote in WATERS 2016 [Dav16].

empirical evaluation method

Among the *empirical methods*, we primarily consider the acceptance ratio of synthetic task sets. For each considered utilization value, e.g., from 1% to 100% with steps of 1%, a number of task sets, often 100 or 1000, is synthesized based on a specific setting. The task sets are evaluated under different scheduling algorithms or schedulability tests. Afterwards, for each test and algorithm, the *acceptance ratio*, i.e., the percentage of accepted task sets, for each utilization value is plotted against the utilization value. Figure 1.4 shows an example of the acceptance ratio for the *Hyperbolic Bound* (HB) [BBB01] and the *Time Demand Analysis* (TDA) [LSD89] for implicit-deadline task sets under Rate Monotonic scheduling. The utilization where the acceptance rate starts to drop drastically is called the *breakdown utilization* of the related schedulability test, i.e., 74% for HB

acceptance ratio

Hyperbolic Bound

Time Demand Analysis

breakdown utilization

and 93% for TDA in Figure 1.4. That the curve for TDA drops later than the one for HB shows that TDA is superior to HB which was expected since TDA is an exact schedulability test while HB is a sufficient test. The gap between the curves of TDA and HB depends on the actual setting used to create the synthetic tasks.

dominance relation

A *dominance relation* is a theoretical method to further classify scheduling algorithms and schedulability tests. A scheduling algorithm $\mathcal{A}$ dominates a scheduling algorithm $\mathcal{B}$ if every task set that is schedulable by algorithm $\mathcal{B}$ is also schedulable by algorithm $\mathcal{A}$. For instance, for a given scheduling algorithm, an exact test will always perform at least as good as a sufficient test, independent from the exact setting, e.g., TDA dominates HB as indicated in Figure 1.4. Based on a dominance relation between different scheduling algorithms, the optimality among a class of scheduling algorithms can be determined. An *optimal algorithm* is one that always optimizes the given metric, i.e., it dominates all other algorithms, while a *heuristic algorithm* tries to optimize for the metric but is not guaranteed to succeed. Hence, an optimal algorithm for minimizing the maximum lateness will ensure that all jobs of all tasks in a task set will meet their deadlines if such a schedule exists. We consider optimality among a certain class of scheduling algorithms according to the classification described in Section 1.1.2 and based on task set properties. Note that optimality with respect to one of the general classes does not mean that another algorithm from a different class may not perform better. For instance, RM is an optimal static-priority algorithm for preemptively scheduled task systems with implicit deadlines [LL73], and EDF is an optimal preemptive dynamic-priority scheduling algorithm for implicit deadlines [LL73]. Nevertheless, as detailed in Figure 1.2, there are task sets that are schedulable by EDF but not by RM. However, since EDF can schedule every implicit-deadline task set with a utilization of less than or equal to 100% [LL73], there is no implicit-deadline task set schedulable by RM that is not schedulable by EDF.

optimal algorithm

heuristic algorithm

utilization bound

A sufficient schedulability test (usually for implicit-deadline task sets) that determines the schedulability of a task set based on the task set utilization is called a *utilization bound*, e.g., the utilization bound of $U_{sum} \leq 100\%$ for EDF [LL73]. Since these bounds consider the worst-case task parameters among all possible task sets, they describe theoretical properties of the considered algorithm. In Figure 1.4 the well-known Liu and Layland Bound [LL73], also called total utilization bound, of $U_{sum} \leq ln(2) \approx 69.3\%$ for RM scheduling is shown. Hence, when considering the schedulability for RM, Figure 1.4 shows the complete spectrum from 1) the direct, linear time utilization bound, i.e., $U_{sum} \leq 69.3\%$, which is independent from the actual task set parameters, over 2) the more precise HB which is still a linear time test but considers information of the actual task set, to 3) the exact TDA which has a pseudo-polynomial runtime. Furthermore, a simple necessary condition is shown, i.e., that the task set can only be schedulable if $U_{sum} \leq 100\%$.

speedup factor

Another way to theoretically describe the worst-case performance of a scheduling algorithm $\mathcal{A}$ is the *speedup factor* [PST+02; KP00] compared to a scheduling algorithm $\mathcal{B}$. A speedup factor ρ describes how much the overall system speed must be increased to ensure that any task set schedulable by algorithm $\mathcal{B}$ is also schedulable by $\mathcal{A}$, where a speed increase by ρ means that for each task τ_i the WCET C_i will be reduced by the factor ρ, i.e., the WCET in the sped up platform is $\frac{C_i}{\rho}$. For instance, for implicit-deadline task sets, the speedup factor of RM-P

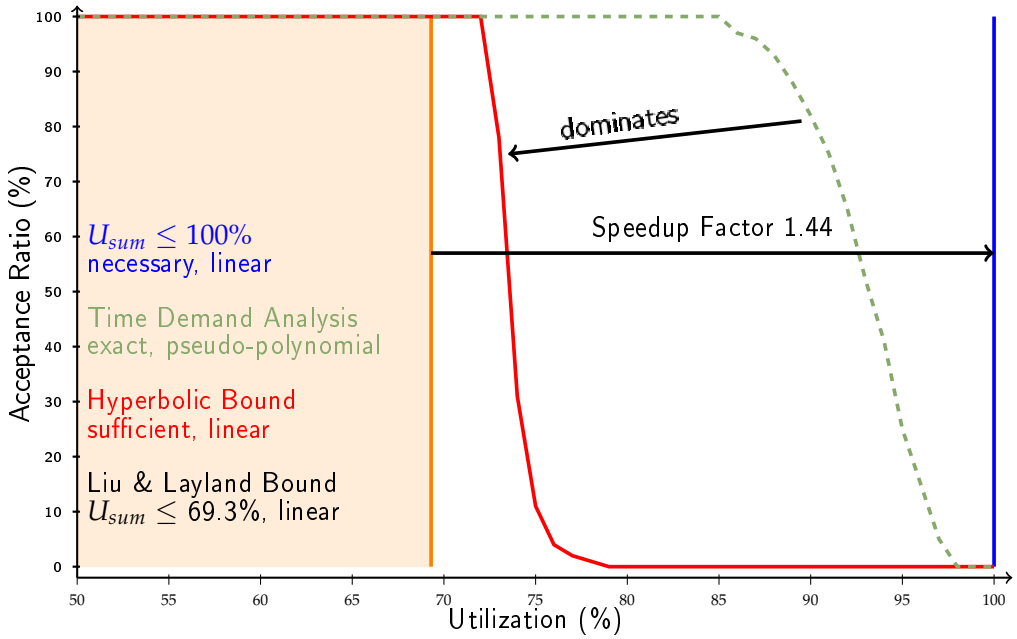


Figure 1.4: Example for theoretical and empirical methods to evaluate the performance of scheduling algorithms and schedulability tests.

compared to EDF-P is $\rho = \frac{1}{ln(2)} \approx 1.44$ since EDF-P can schedule no task set with $U_{sum} > 1$ and if a task set with $U_{sum} \leq 1$ is speed up by $\frac{1}{ln(2)}$ the total utilization of the resulting task set is smaller than $ln(2)$ which is the total utilization bound for RM-P. The speedup factor for comparing two scheduling algorithms $\mathcal{A}$ and $\mathcal{B}$ effectively compares their exact schedulability tests. Analogously, the speedup factor of a sufficient schedulability test $\mathcal{X}$ compared to another sufficient schedulability test $\mathcal{Y}$ can be defined as well, where $\mathcal{Y}$ can either relate to the same scheduling algorithm as $\mathcal{X}$ or to another scheduling algorithm. In some situations it is also meaningful to compare a sufficient test $\mathcal{S}$ to a necessary condition $\mathcal{N}$ to show that the test $\mathcal{S}$ has a bounded maximum loss.

1.2 CONTEXT AND CHALLENGES

From a researcher's perspective, the examination of real-time scheduling focuses on three components that build on one another: 1) the system and task model, 2) the scheduling algorithm with related schedulability test, and 3) a theoretical and/or empirical performance evaluation of the scheduling algorithm. The relation between these three components is detailed at the top of Figure 1.5.

SYSTEM AND TASK MODEL

Models abstract system characteristics to enable the analysis of timing constraints, since modern real-time systems are usually too complex to be analyzed directly. In most cases, if it is not sufficient to describe the system directly, the *periodic* or the *sporadic task model* is extended by introducing additional parameters and/or

constraints. One practically relevant example for such additional constraints are *automotive task systems*, where the task periods are not arbitrary but chosen from a specific set of possible values. In automotive systems, *angle-synchronous tasks*, where the period of the task depends on the rotation of the crankshaft, are also common. Other well known examples for additional task parameters are blocking time that results from mutually exclusive access to shared resources or release jitter. When considering uncertain execution environments, *mixed-criticality systems* [Ves07] have received attention in the real-time systems research community due to their practical relevance. Nevertheless, the model has been criticised [ENN+15; EN16], as it does not match the expectations of systems engineers. However, the modelling should result in an abstraction that is accurate and therefore able to precisely describe a system. Otherwise, the model either is not relevant for practical scenarios, or may lead to an over-pessimistic or even wrong analysis. On the other hand, a model should not be too restrictive but flexible enough to describe a variety of systems with similar behaviour. Otherwise, a large overhead for modelling and analysis must be expected when analysing a slightly different system.

automotive task set

angle-synchronous tasks

mixed-criticality systems

The self-suspension models in the literature show that finding a good tradeoff between flexibility and restrictiveness is not easy. The *dynamic self-suspension* model only introduces a bound on the maximum total suspension time as an additional parameter and assumes that suspension and computation can alternate arbitrarily, making the model *overly flexible* and *imprecise*. On the other hand, the *segmented self-suspension* model assumes a precisely known execution suspension pattern and is therefore *very precise* but *overly restrictive*.

dynamic self-suspension

segmented self-suspension

SCHEDULING ALGORITHMS AND SCHEDULABILITY TESTS

The scheduling algorithm should be chosen based on the system and task model. Under the classical periodic and sporadic task models, optimal scheduling algorithms are known for many classes of scheduling algorithms, e.g., EDF-P and RM-P for preemptive implicit-deadline task sets [LL73]. However, this optimality usually does not hold for extended task models. For instance, EDF and RM are both not optimal for non-preemptive scheduling, mixed-criticality systems, and self-suspending tasks. If optimal algorithms cannot be established, good heuristics are a suitable alternative. In addition to a good performance regarding schedulability, a scheduling algorithm must also be applicable to practical systems with reasonable overhead. Examples of problematic properties of algorithms are a large number of context switches, and online adaptations like disabling and restarting tasks at runtime or reordering of task/job priorities.

Furthermore, even for optimal algorithms, an exact or at least a sufficient schedulability test is essential to verify whether a specific task set is schedulable under the scheduling algorithm. Such a test should not only be as precise as possible but must also be computationally affordable. To achieve high precision, the test should consider the information available from both the model as well as from the scheduling algorithm. The schedulability test may also link back to the scheduling algorithm.

PERFORMANCE EVALUATION

While a schedulability test determines whether a specific task set is schedulable under a specific algorithm, the overall performance of scheduling algorithms and schedulability tests can be evaluated using theoretical and empirical methods.

empirical evaluation method

Empirical evaluation methods typically evaluate the average case performance. In many cases, randomly generated task sets are considered in the evaluation. Therefore, it is important that such task sets cover a large range of interesting scenarios. Otherwise, a scheduling algorithm or schedulability test may be classified as good even though it only performs well in certain, practically irrelevant scenario.

theoretical evaluation method

Theoretical evaluation methods provide worst-case guarantees for scheduling algorithms or schedulability tests. Hence, they focus on finding the worst-case setup among all possibilities. This leads to the question whether such worst-case scenarios are good representatives to evaluate the overall quality of an scheduling algorithm or schedulability test.

1.3 CONTRIBUTION OF THIS DISSERTATION

Resulting from the challenges determined in Section 1.2, this dissertation examines modelling as well as analysis in real-time systems and provides possible solutions. This investigation is performed considering the following hypothesis:

> *Realistic scheduling models and analyses are essential for guaranteeing timing correctness in advanced real-time systems while ensuring that the system resources necessary to provide these guarantees are not over-provisioned.*

The main part of Figure 1.5 displays the contributions of this dissertation, their interconnection, which of the three components the individual contributions relate to, and the related section in this dissertation. In this section, a detailed description based on the individual chapters, which are organized according to the considered task model, is provided, including a brief description how these contributions are related to the dissertation hypothesis. The main contributions of this dissertation were published in peer-reviewed international conferences and journals. For the ease of the reader, the references to publications related to this dissertation are *emphasized*, e.g., published in *[BCH15]*.

1.3.1 SPEEDUP FACTORS AND PARAMETRIC UTILIZATION BOUNDS

theoretical evaluation method
utilization bound
speedup factor

Theoretical methods like *utilization bounds* and *speedup factors*, which compare the worst-case behaviour of scheduling algorithms or schedulability tests, are the focus of Chapter 4. *Speedup factors* compare an algorithm or schedulability test with an optimal algorithm or test, or to a necessary schedulability condition, hence giving testimony to the maximum loss when the considered schedulability test or scheduling algorithm is used. Utilization bounds, on the other hand, always consider the worst-case setup, i.e., the unschedulable task set with the lowest

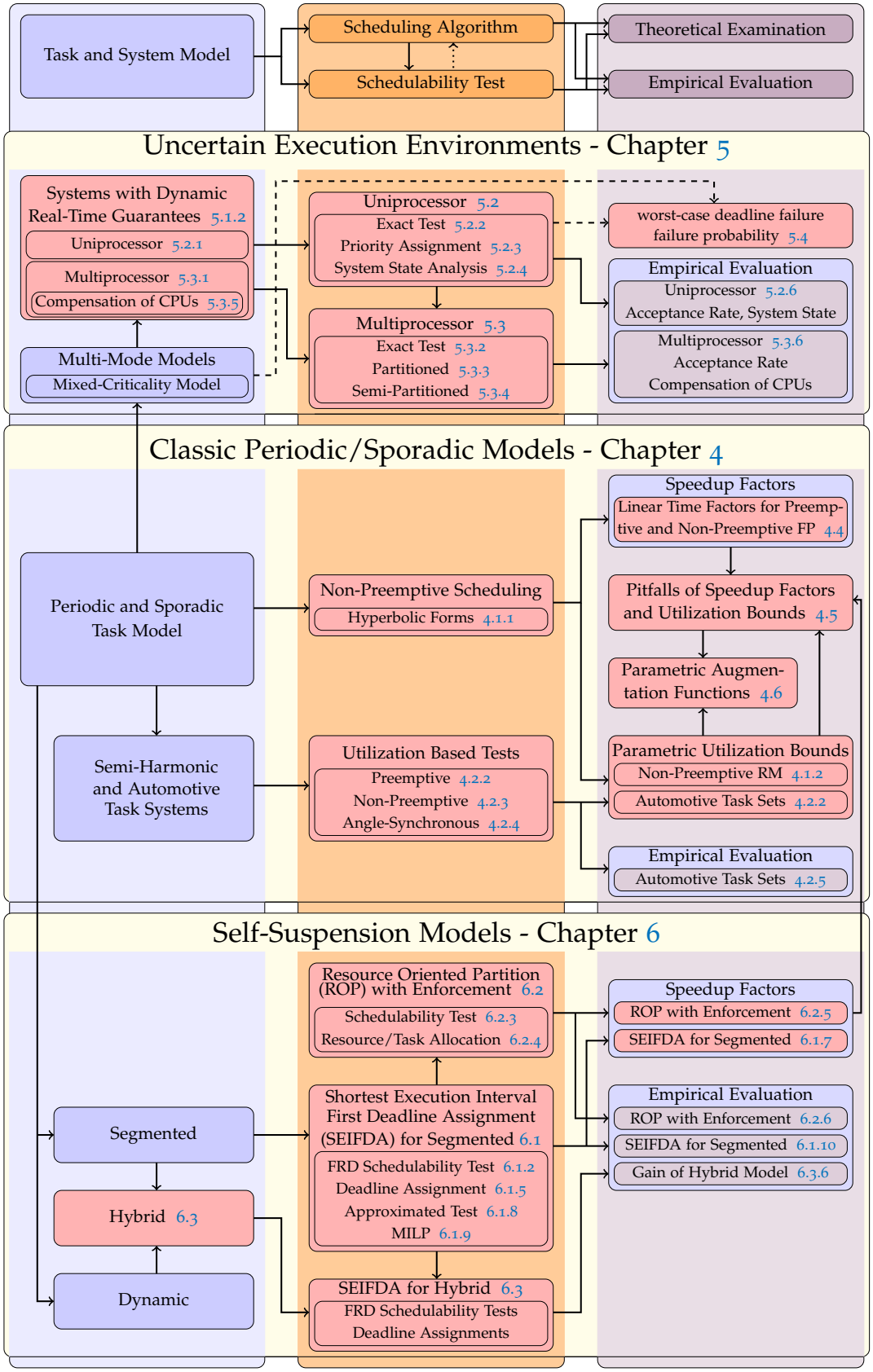


Figure 1.5: Contributions of this dissertation (marked red and with related section), their interconnection, and what part of real-time scheduling analysis they are related to. Blue boxes indicate pre-existing models and evaluation techniques while gray boxes are evaluations of proposed algorithms and tests.

possible utilization. Therefore, they are often seen as a way to describe the quality of an algorithm or test, i.e., it is assumed that a better utilization bound or speedup factor means that the related algorithm/test is better. Nevertheless, the question remains if a worst-case comparison is a good way to draw such conclusions and allows a realistic evaluation of scheduling algorithms and schedulability tests, since only the worst-case setting is considered and the average case behaviour may in fact be very different while the worst-case setting may be practically irrelevant.

utilization bound
Liu and Layland Bound

For instance, when considering the *utilization bound* of implicit-deadline tasks under RM-P, the *Liu and Layland Bound* [LL73] of $U_{sum} \leq ln(2) \approx 69.3\%$ is tight, i.e., all task sets with a utilization $\leq ln(2)$ are schedulable and a task set with a utilization of $ln(2) + \varepsilon$ that is not schedulable by RM-P exists for each $\varepsilon > 0$. For these task sets the periods of the tasks are distributed in a way that the largest period is at most two times the smallest period [LL73], e.g., all periods are in $[1;2]$. However, in commercial real-time systems, tasks periods typically range over multiple orders of magnitude [BB06]. For instance, in automotive systems periods usually range from 1 ms to 1000 ms [HDK+17; KZH15; TEH+16; SSD+13]. Furthermore, a larger range of periods leads to a higher breakdown utilization, as shown by Emberson et al. [ESD10]. Therefore, it seems reasonable to use additional information that is known about the considered task set, like the period range, to parametrize the utilization bound. Hence, in Section 4.1 and

parametric utilization bound

Section 4.2 respectively, we show how *parametric utilization bounds* can drastically increase the utilization bounds for RM-NP when considering the blocking factor of the task set as a parameter *[BCH15]* (appeared in ECRTS 2015), and for RM-P in automotive systems where the task periods are chosen from $\{1, 2, 5, 10, 20, 50, 100, 200, 1000\}$ ms *[BUC+17]* (appeared in RTNS 2017).

speedup factor

Regarding *speedup factors*, the speedup factor of $\rho = \frac{1}{ln(2)} \approx 1.44$ for RM-P compared to EDF-P that results from the Liu and Layland Bound [LL73] is tight, i.e., the upper bound follows directly from the *Liu and Layland Bound* and there exists a task set with utilization $U_{sum} = ln(2) + \varepsilon$ for each $\varepsilon > 0$ that is not schedulable under RM-P. As a result, the speedup factor for RM-P compared

Time Demand Analysis

to EDF-P is the same when considering *Time Demand Analysis* (TDA) [LSD89], which is an exact schedulability test with pseudo-polynomial runtime, as it is for the linear time Liu and Layland Bound. However, TDA strictly dominates the Liu and Layland Bound and, as exemplified in Figure 1.4, the acceptance ratio of TDA is usually significantly better. This leads to the observation that speedup factors are often not able to discriminate between different scheduling algorithms or schedulability test regarding performance. In Section 4.4, we first provide a speedup factor for non-preemptive Deadline Monotonic scheduling (DM-NP) compared to EDF-NP *[BCH15]* (appeared in ECRTS 2015). Hereinafter, we show that linear-time sufficient schedulability tests for DM-P compared to EDF-P, as well as for DM-NP compared to EDF-NP, result in exact speedup factors for implicit-, constrained-, and arbitrary-deadline task sets *[BCD+17]* (appeared in IPL in 2017). These factors hold regardless of the sub-optimality of the considered test and, in the arbitrary-deadline case, the sub-optimality of DM. Due to this rather surprising result, we take a closer look at speedup factors in the remainder of Chapter 4, observing additional possible problems, and provide guidance

for their meaning and interpretation. Furthermore, *parametric augmentation functions* are proposed an a possible solution and their use is exemplified in one scenario *[CBH+17a]* (appeared in ECRTS 2017).

parametric augmentation function

With respect to dissertation hypothesis, the provided parametric utilization bounds show that considering a more restricted scenario, which therefore is a more realistic model of the system at hand, can drastically increase the precision of the analysis and hence avoid over-provisioning of system resources. Furthermore, examining the meaning and practical relevance of speedup factors allows to determine the value of this metric in realistic scenarios.

1.3.2 UNCERTAIN EXECUTION BEHAVIOUR

While Chapter 4 critically scrutinizes theoretical methods to evaluate scheduling algorithms, it only briefly considers situations where the periodic or the sporadic task model [LL73] is not able to fully describe the system at hand. Chapter 5 focuses on such a situation, to be precise on *uncertain execution behaviour*. Specifically, it considers uncertainty regarding the *worst-case execution time* (WCET) of a task which is represented by a set of distinct execution modes with related WCETs. For each of these modes, however, it is assumed that the WCET can be precisely determined, i.e., the uncertainty is not related to the worst-case execution time analysis. Reasons for such a behaviour are, for instance, a reduced CPU frequency to prevent overheating, dynamic voltage frequency scaling to save energy, *mixed-criticality systems*, and software-based *fault tolerance* mechanisms. Furthermore, it is assumed that the WCET differs largely among the modes and that modes with a large execution time have a low probability to be executed, i.e., a small WCET is the *normal* and a large WCET the *abnormal* case. In this situation, one cannot simply consider the situation that the task will always run with the largest WCET in a schedulability analysis without largely overestimating the necessary system resources, which would result in an intolerable increase in hardware costs. On the other hand, it must be ensured that an execution mode with a large WCET does not destroy the timing guarantees that must be provided. Hence, to achieve practical applicability, a reasonable tradeoff between hardware costs and timing guarantees must be found.

uncertain execution behaviour

worst-case execution time

mixed-criticality systems

fault tolerance

In many real-time systems, rare deadline misses are acceptable at least for a subset of the tasks, facilitating the achievement of such a compromise. For instance, the work in [KT12; QHE12; QNE13; HQE14; XHK+15] assumes that rare deadline misses are acceptable and tries to quantify them. One main reason for such an assumption is that, even if all tasks in the systems have real-time constraints, some tasks are *more important* for the system stability while others are *not so important*. Hence, rare deadline misses can be tolerated for the latter since they will not lead to catastrophic consequences. As an example consider an unmanned aerial vehicle (UAV) where the tasks in the *more important* flight control system should always meet their deadline while for the *not so important* surveillance system occasional deadline misses are tolerable. In theory and in practical systems, to ensure a reasonable tradeoff between worst-case timeliness of the *more important* tasks and the system utilization, *not so important* tasks

mixed-criticality systems

are often aborted to guarantee the response time of the *more important* tasks if abnormal execution behaviour occurs. An important example for systems with such properties are *mixed-criticality systems* [Veso7] where, for dual-criticality systems, the tasks are partitioned into the more important high-criticality tasks and the less important low-criticality tasks. In the beginning of the system life time, the tasks execute in the low-criticality mode, i.e., with a smaller WCET, and at some point in time the system switches to high-criticality mode where tasks are executed with a larger WCET. It is assumed that the system stays in high-criticality mode for the remainder of its life time. Hence, for mixed-criticality systems the mode switch is usually handled by online adaptation, for example by dropping tasks, increasing inter-arrival times, or changing task priorities. Such approaches have been criticized lately, for instance by Esper et al. [ENN+15] who asked "How realistic is the mixed-criticality real-time system model?" and by Ernst and Di Natale [EN16] in "Mixed Criticality Systems - A History of Misconceptions?". Specifically, it is pointed out that the main stream of mixed-criticality research does not fit the expectations of system engineers since 1) low-criticality tasks should not be abandoned, and 2) systems should return to low-criticality mode after a sufficient amount of time.

Systems with Dynamic Real-Time Guarantees

timing tolerable tasks

timing strict tasks

normal mode

abnormal mode

full timing guarantees

limited timing guarantees

To answer this criticism and to provide a more suitable model, *Systems with Dynamic Real-Time Guarantees* are introduced in Section 5.1. Such systems assume a given task set partition into *not so important* tasks where rare deadline misses are acceptable, called *timing tolerable tasks*, and *more important* tasks that must always meet their deadlines, called *timing strict tasks*. On the one hand, they ensure that, if the systems runs *normally*, called the *normal mode*, the deadlines of all tasks should be satisfied. On the other hand, in case of a rare event, called the *abnormal mode*, the *timing strict tasks* are still guaranteed to meet their deadlines while the *timing tolerable tasks* may miss deadlines but are guaranteed bounded tardiness. All these guarantees are given in advance using static-priority scheduling and *no online adaptation* is performed. During runtime, a *System with Dynamic Real-Time Guarantees* provides either *full timing guarantees* if all jobs meet their deadline or *limited timing guarantees* if only the jobs of the *timing strict tasks* are guaranteed to meet their deadline while *timing tolerable tasks* have bounded tardiness. Furthermore, the time of a mode switch is not assumed to be given but detected by an online monitor. This model is not only suitable for mixed-criticality systems but can also be applied when the uncertain execution behaviour results from other sources like fault-tolerance mechanisms, overheating, or dynamic voltage frequency scaling.

In Section 5.2, *Systems with Dynamic Real-Time Guarantees* for uniprocessor environments are introduced, providing a precise definition of the system itself as well as for the related terminology like *full timing guarantees* and *limited timing guarantees*, followed by an exact schedulability test and several important properties. Furthermore, over-approximations for the maximum interval length with *limited timing guarantees* and an online monitoring is provided *[BCH+16]* (appeared in RTSS 2016). Afterwards, the system definition is extended to *Multiprocessor Systems with Dynamic Real-Time Guarantees* in Section 5.3, detailing the related definitions as well as schedulability tests and partitioning algorithms for both partitioned and semi-partitioned scheduling. Furthermore, it is shown how *Multi-*

Multiprocessor Systems with Dynamic Real-Time Guarantees

processor Systems with Dynamic Real-Time Guarantees can further increase the system reliability using *task migration* techniques *[BSC18]* (appeared in PRDC 2018). *task migration*

When uncertain execution behaviour is considered, for instance in *Systems with Dynamic Real-Time Guarantees*, one important problem is to determine the *worst-case deadline failure probability* of a task. However, previous approaches are either fast but imprecise analytical bounds or job-level convolution-based techniques that do not scale for task sets with a reasonable size. Therefore, a novel approach is introduced that convolves representatives of the individual tasks resulting from a multinomial distribution instead in Section 5.4. Furthermore, several optimization techniques are provided which ensure that the proposed approach is scalable to large task sets. The results in Section 5.4 appeared in ECRTS 2018 *[BPC+18]*. *worst-case deadline failure probability*

When uncertain execution behaviour is modeled as a mixed-criticality system, the result is in many situations an imprecise and therefore unrealistic description of the actual system. This may lead to an incorrect analysis which can be both optimistic or too pessimistic, which supports the dissertation hypothesis. One possible solution is a realistic system model that allows an analysis of the actual system like *Systems with Dynamic Real-Time Guarantees*.

As current techniques to approximate the *worst-case deadline failure probability* are either imprecise or not scalable to systems with a realistic system size, these techniques are not able to correctly quantify the system behaviour, which may result in over-provisioning of system resources. Therefore, new approaches are needed that are both precise and scalable like the proposed task-level convolution.

1.3.3 SELF-SUSPENSION

Self-suspension behaviour, which is examined in Chapter 6, may result from multiple sources like offloading, access to external devices, and multiprocessor resource synchronization, and may negatively impact real-time schedulability, especially when the suspension delays are long. Two self-suspension models are primarily studied in the literature. The *dynamic self-suspension* model that allows a job of task τ_i to suspend itself at any moment before it finishes as long as the worst-case self-suspension time S_i is not violated, and the *segmented self-suspension* model that assumes a specified execution/suspension pattern for each task. *dynamic self-suspension* *segmented self-suspension*

For the one-*segmented self-suspension* model, a *fixed-relative-deadline* (FRD) scheduling algorithm called *Shortest Execution Interval First Deadline Assignment* (SEIFDA) is introduced in Section 6.1. An FRD scheduling algorithm assigns individual relative deadlines to each computation segment and schedules them individually, using EDF. Therefore, setting the deadlines for the subjobs is the most challenging problem when developing an FRD approach. While previous assignment strategies suffer from the fact that they assign the deadlines for a task agnostic from the deadlines of other tasks, SEIFDA takes previously assigned deadlines into account and assigns the deadlines one by one, starting from the task with the shortest execution interval, i.e, the relative deadline minus the suspension interval. A general FRD schedulability test as well as an approximated test, three deadline assignment strategies, and an MILP formulation for FRD are provided. *fixed-relative-deadline* *SEIFDA*

In empirical evaluation based on synthesized task sets, SEIFDA shows a substantial performance gain compared to previous FRD assignment strategies from the literature while achieving the same *speedup factor*. The results in Section 6.1 were presented in RTNS 2016 *[BHC+16]*.

speedup factor

resource-oriented partitioned scheduling

Afterwards, SEIFDA is utilized in the design of a *resource-oriented partitioned* scheduling (ROP) for multiprocessor resource sharing in Section 6.2. The general concept of a ROP is to reduce the multiprocessor resource sharing problem to a set of uniprocessor resource sharing problems by specifying a set of synchronization processors where all resource access is performed. To be precise, each resource is assigned to a processor and all critical sections accessing that resource are moved to the related processor. The non-critical sections of the tasks are executed on the remaining processors. An algorithm is proposed that combines SEIFDA for the analysis of the non-critical sections with ROP under release enforcement for the critical and non-critical sections. A sufficient schedulability as well as a resource and task allocation strategy are proposed. The provided algorithm is shown to have a speedup-factor of 6, which is the best known *speedup factor* for multiprocessor resource sharing, and to outperform other multiprocessor resource sharing protocols in the evaluation. The results presented in Section 6.2 appeared in RTNS 2017 *[BCH+17]*.

speedup factor

Furthermore, the gap between the dynamic and the segmented self-suspension model is examined in Section 6.3. On the one hand, the *dynamic self-suspension* model can be utilized when only limited information about the suspension behavior is known. Therefore, it has a higher flexibility, but results in more pessimistic analyses and designs of scheduling policies if the suspending pattern can be defined precisely. On the other hand, the *segmented self-suspension* has a lower flexibility, but the self-suspending structure can be exploited by the scheduling algorithms to make better decisions. However, that the execution pattern is static for each execution and that the structure of the program is well designed are assumptions that often do not hold in practical situations. This gap is bridged by introducing multiple *hybrid self-suspension* models that assume the self-suspending task to be specified by a set of possible execution patterns that are known offline. These models are either applicable when no information as to which pattern is executed for a specific job is known online, so-called *pattern-oblivious models*, or when the information as to which pattern is executed can be determined when a job arrives, so-called *pattern-clairvoyant models*. The provided models have different tradeoffs between flexibility and precision that can be achieved based on the information that is known for the considered task set. The evaluation in shows that the information that can be accessed in addition to the information assumed by the dynamic self-suspension mode can be utilized to achieve a significantly better performance regarding schedulability. The results in Section 6.3 were presented in RTCSA 2017 *[BHC17]*.

hybrid self-suspension

pattern-oblivious

pattern-clairvoyant

Regarding the dissertation hypothesis, both SEIFDA as well as the provided resource-oriented partitioned scheduling show the possible performance gain when an algorithm can exploit properties of a restricted and therefore more precise system model instead of considering general models. Furthermore, the introduced hybrid self-suspension models exemplify how additional information can be included into the model to achieve a more realistic description of the

analysed system and therefore a more precise analysis. In all three cases, over-provisioning of resources can be reduced.

1.4 AUTHOR'S CONTRIBUTION TO THIS DISSERTATION

According to §10(2) of the "Promotionsordung der Fakultät für Informatik der Technischen Universität Dortmund vom 29. August 2011", a dissertation must include a list that highlights the author's contribution to research results that were obtained in cooperation with other researchers. The following overview lists the contribution on the results presented in the individual chapters:

- Chapter 4 focuses on the theoretical comparison of scheduling algorithms using *speedup factors* and *utilization bounds*. For the work regarding non-preemptive scheduling published at ECRTS 2015 *[BCH15]*, I was the principal author contributing the concepts and theorems. I was the principal author, contributing concepts, theorems, and evaluation, for the work published at RTNS 2017 *[BUC+17]* that considers *automotive systems*. Niklas Ueter provided the task generator used in the evaluation while Matthias Freier granted insight into automotive task systems from a designers perspective. The work on *speedup factors* published in the Information Processing Letters, Volume 177 *[BCD+17]*, stems from discussion over the work in *[BCH15]* at ECRTS 2015. The observations in *[BCD+17]*, together with other observations regarding work in the literature, then resulted in discussions that lead to the work on *speedup factors* and *utilization bounds* in ECRTS 2017 *[CBH+17a]*. Due to this discussion based process, the results in *[BCD+17]*, where I was the principal author, and *[CBH+17a]*, where I was a co-author, were developed by Robert I. Davis, Jian-Jia Chen, Wen-Hung Huang, and myself in close cooperation.
- Chapter 5 examines uncertain execution environments, introducing the model of *Systems with Dynamic Real-Time Guarantees* in uniprocessor and multiprocessor environments, and proposing a novel method to over-approximate the worst-case deadline failure probability in such an environment. For the work introducing *Systems with Dynamic Real-Time Guarantees [BCH+16]*, published in RTSS 2016, I was the principal author, contributing the concept, algorithms, the majority of the theorems, the schedulability evaluation, and the analysis of the evaluation results. Based on my system model, Wen-Hung Huang provided the system mode analysis. Kuan-Hsun Chen provided the implementation for the system state analysis. This implementation was released in RTEMS (now inherited in version 5) and details on the implementation were published at WMC 2016 [CBC16] with Kuan-Hsun Chen as principal author. The multiprocessor extension, where I am the principle author, was published at PRDC 2018 *[BSC18]*. It was written in cooperation with Lea Schönberger and was a result of her Master's Thesis under my supervision. For the work on calculating the worst-case deadline failure probability, published in ECRTS 2018 *[BPC+18]*, I was the principal author, contributing concept, analysis, theorems, and evaluation for the task-level convolution-based approach, i.e., the part that is presented in this

dissertation. Nico Piatkowski provided insight on probability theory and the related formalisms. The work in *[BPC+18]* also includes bounds based on *Hoeffding's inequality* and *Bernstein's inequality*, which are not part of this dissertation and were contributed by Kuan-Hsun Chen, who also provided the related implementation for the evaluation.

- Chapter 6 considers *self-suspension* and *multiprocessor resource sharing*. I was the principal author of the work published in RTNS 2016 *[BHC+16]*, contributing concepts, algorithms, theorems, and the analysis of the evaluation results. Wen-Hung Huang provided the implementation and the evaluation. For the work published in RTNS 2017 *[BCH+17]*, I was the principal author, contributing concepts, algorithms, theorems, and the analysis of the evaluation results. Wen-Hung Huang provided the evaluation for the methods that were proposed in *[BCH+17]*, and Maolin Yang provided the analysis results for methods from the literature. For the work published in RTCSA 2017 *[BHC17]*, I was the the principal author providing concepts, algorithms, theorems, and the evaluation results. The implementation was provided by Wen-Hung Huang.

2

TASK MODEL, SYSTEM MODEL, NOTATION, AND FUNDAMENTALS

This section formally introduces the task model, the system model, and the notations used in this work. We start by introducing periodic [LL73] and sporadic [Mok83] task models which are later extended to cover extensions for uncertain execution environments, self-suspending tasks, and resource sharing. Furthermore, some fundamental preliminary findings from the literature are introduced here, especially those used multiple times in the following chapters.

2.1 TASK MODEL

We assume recurrently executed real-time tasks, modeled as a set $\mathbf{T} = \{\tau_1, \ldots, \tau_n\}$ of n given tasks according to the periodic [LL73] or the sporadic [Mok83] task model. Each task is specified by a 3-tuple of parameters (C_i, T_i, D_i) where C_i is the *worst-case execution time* (WCET) of τ_i, T_i denotes the minimum inter-arrival time or *period* of τ_i, and D_i is the *relative deadline* of task τ_i. The *utilization* of τ_i is defined as $U_i = \frac{C_i}{T_i}$ and the total or *system utilization* as $U_{sum} = \sum_{\tau_i \in \mathbf{T}} U_i$. Every task in the system releases an infinite number of *task instances*, called *jobs*, where the j^{th} job of task τ_i is denoted $\tau_{i,j}$. Each job $\tau_{i,j}$ has a *release time* $r_{i,j}$, a *finishing time* $f_{i,j}$, and an *absolute deadline* $d_{i,j} = r_{i,j} + D_i$, i.e., a job released at time $r_{i,j}$ must be able to execute up to C_i time units before its absolute deadline at $d_{i,j}$. The response time $R_{i,j} = f_{i,j} - r_{i,j}$ of the j^{th} job $\tau_{i,j}$ of task $\tau_i \in \mathbf{T}$ (under a specific scheduling algorithm) is the interval between arrival and finishing time for job $\tau_{i,j}$ and we denote the *worst-case response time* (WCRT) of τ_i with R_i.

worst-case execution time
period
relative deadline
utilization
job
release time
finishing time
absolute deadline
worst-case response time

The *period* T_i is the minimum interarrival time between any two consecutive job releases of τ_i. A task is called (strictly) periodic [LL73] if subsequent jobs are always released exactly according to the period, i.e., if a job is released at time t, the next job is released *exactly* at time $t + T_i$. In the periodic case, the task is further described by a *phase* parameter ϕ_i which indicates the time the first instance of the job is released. Note that ϕ_i is omitted in the task description if $\phi_i = 0$. For a periodic task set, the *hyperperiod* H is the least common multiple of all T_i of tasks in $\mathbf{T}$. In contrast, a task is called sporadic [Mok83], if after a job is released at time t the next job is released *not before* $t + T_i$.

period
periodic task
phase
hyperperiod
sporadic task

If $D_i = T_i$, then τ_i is an *implicit-deadline* task, and if $D_i \leq T_i$, then τ_i is a *constrained-deadline* task. Accordingly, a task set $\mathbf{T}$ is called an implicit-deadline set if $D_i = T_i$ for each $\tau_i \in \mathbf{T}$, a constrained-deadline set if $D_i \leq T_i$ for each $\tau_i \in \mathbf{T}$, and an *arbitrary-deadline* set if $D_i > T_i$ is allowed. In this work, we focus on and therefore implicitly assume constrained- and implicit-deadline task sets. Note that each implicit-deadline task set is also a constrained-deadline task set, and each constrained-deadline task set is also an arbitrary-deadline task set. Hence,

implicit-deadline
constrained-deadline
arbitrary-deadline

an analysis for arbitrary deadlines always holds for constrained deadlines, and an analysis for constrained deadlines always holds for implicit deadlines. However, the additional deadline restriction usually leads to a more precise analysis.

harmonic task set

A task set $\mathbf{T}$ is called a *harmonic task set* (or a task set with harmonic periods), if all periods are integer multiples of each other, i.e., T_i is an integer multiple of T_j if $T_i \geq T_j$ for any two tasks τ_i and τ_j in $\mathbf{T}$. Furthermore, we call a task set *semi-harmonic*, if $T_i \cdot n_i = H \; \forall \tau_i \in \mathbf{T}$ where n_i is a small integer value for each period.[1] Automotive applications often consider the specific set of semi-harmonic periods $T_i \in \{1, 2, 5, 10, 20, 50, 100, 200, 1000\}$ *milliseconds* (ms), as mentioned in, for instance, [KZH15; HDK+17; SSD+13; TEH+16]. Therefore, we call a task set that only allows these periods an *automotive task set*. Such task systems usually assume that the scheduling entity is a Runnable [KZH15] and that multiple Runnables with the same period are combined into one task. However, since the distinction between Runnables and tasks has no impact on the analysis presented in this work, we use the terms equivalently here. Furthermore, we always assume implicit deadlines and that periods, deadlines, and WCETs are given in *milliseconds* (ms) for automotive task systems. In addition, we define $\mathbf{T}_x$ to be the subset of the tasks in $\mathbf{T}$ with period x, i.e., $\mathbf{T}_x = \{\tau_i \mid \tau_i \in \mathbf{T} \text{ and } T_i = x\}$.

semi-harmonic task set

automotive task set

2.2 SCHEDULABILITY

We assume a given task set $\mathbf{T}$ that is scheduled according to a given online scheduling algorithm, where the job that is executed at a given time is determined online based on the job's priority. This can either be a dynamic-priority or a static-priority scheduling algorithm. We use -P to denote the preemptive version of a given algorithm and -NP to denote the non-preemptive version of an algorithm, e.g., RM-P for preemptive rate-monotonic scheduling. We assume a *work-conserving* scheduler which means that the processor never idles if at least one job is ready to be executed. Unless stated differently, this work examines uniprocessor systems. Otherwise, for multiprocessor systems, we regard homogeneous multiprocessors, i.e., m identical processors with similar assumptions.

work-conserving

lateness

tardiness

For a job $\tau_{i,j}$ under the considered scheduling algorithm its *lateness* is defined as $L_{i,j} = f_{i,j} - d_{i,j}$ while its *tardiness* is $E_{i,j} = \max\{0, L_{i,j}\}$. Hence, the tardiness E_i of task τ_i is defined as $E_i = \max_j \{E_{i,j}\}$, i.e., the maximum over all jobs of τ_i, and the tardiness $E_{\mathbf{T}}$ of the task set $\mathbf{T}$ as the maximum over all tasks in the set, i.e., $E_{\mathbf{T}} = \max_{\tau_i \in \mathbf{T}} \{E_i\}$. A task is considered *schedulable* under a given algorithm if it always fulfils its timing constraints, i.e., its tardiness is 0, and a task set is schedulable under a given algorithm if all tasks in the set are schedulable.

schedulability

Whether a task set is schedulable can be determined based on a schedulability test, usually related to a specific scheduling algorithm.

sufficient test

- *Sufficient tests* allow false negatives but no false positives, i.e., deem task sets schedulable that are in fact schedulable.

1 This definition is informal as *small* is not precisely defined. Since in many practical cases the periods in a task set differ by two or three orders of magnitude [BB06], the largest n_i should not be much larger than the largest ratio between two periods in the set. An alternative definition is that for the largest period T_{max} in the set $n \cdot T_{max} = H$ holds for a very small integer n, e.g., $n \in [1, 10]$.

- *Necessary tests* allow false positives but no false negatives, i.e., deem task sets not schedulable that are in fact not schedulable. *necessary test*
- *Exact tests* are both sufficient and necessary, i.e., they allow neither false positives nor false negatives. Therefore, they deem exactly those task sets schedulable that are in fact schedulable by the related algorithms. *exact test*

One specific type of sufficient test are *utilization bounds*. They determine the schedulability of a task set based on whether $U_{sum} \leq x$, where x depends on the task set characteristics and the considered scheduling algorithm. *utilization bound*

A (worst-case) *response time analysis* [JP86; LSD89] is another type of schedulability tests. The idea is to determine the maximum interference a task τ_i can suffer from other tasks. If the response time under this interference, denoted as the *worst-case response time* R_i of task τ_i, is less than or equal to the task's relative deadline D_i, then the task will always fulfill its timing requirements. Hence, if $R_i \leq D_i$ for all tasks $\tau_i \in \mathbf{T}$ under the given scheduling algorithm, the task set is schedulable under this algorithm. When the maximum interference that a task can suffer is not precisely known but can be upper bounded, it is also possible to determine an upper bound of R_i, resulting in a *sufficient but not exact* test. *response time analysis* *worst-case response time*

When performing an empirical comparison of scheduling algorithms or schedulability tests, we usually look at the *acceptance ratio* over a range of utilization values, i.e., the percentage of generated task sets that are schedulable at each utilization level according to the algorithm or test. *acceptance ratio*

2.3 THEORETICAL COMPARISON OF SCHEDULING ALGORITHMS

Theoretical methods compare the worst-case performance of scheduling algorithms and schedulability tests. One possibility is to compare them based on the related utilization bounds, i.e., favoring the algorithm with the higher utilization bound. An additional theoretical method is a *dominance relation* between two scheduling algorithms or between two schedulability test. A scheduling algorithm $\mathcal{A}$ dominates a scheduling algorithm $\mathcal{B}$ if every task that is schedulable by algorithm $\mathcal{B}$ is also schedulable by algorithm $\mathcal{A}$. Furthermore, $\mathcal{A}$ strictly dominates $\mathcal{B}$, if $\mathcal{A}$ dominates $\mathcal{B}$ and there is at least one task set that is schedulable by $\mathcal{A}$ but not by $\mathcal{B}$. An algorithm $\mathcal{A}$ is termed *optimal* among a certain class of scheduling algorithms, if it dominates all other algorithms of the same class. Regarding schedulability test, a schedulability test $\mathcal{X}$ dominates a schedulability test $\mathcal{Y}$ if every task set that is deemed schedulable by schedulability test $\mathcal{Y}$ is also deemed schedulable by schedulability test $\mathcal{X}$. *dominance relation* *optimal algorithm*

A *speedup factor* ρ [KP00] details the sub-optimality of a scheduling algorithm $\mathcal{A}$ (or schedulability test) compared to another scheduling algorithm $\mathcal{B}$ (or schedulability test). We assume that speeding up the platform by ρ will lead to a WCET of $\frac{C_i}{\rho}$. The (maximum) speedup factor $\rho^{\mathcal{A}\to\mathcal{B}}$ between two scheduling algorithms $\mathcal{A}$ and $\mathcal{B}$ is the minimum increase in speed necessary to ensure that algorithm $\mathcal{A}$ can schedule every task set that is schedulable with algorithm $\mathcal{B}$. Let $\rho^{\mathcal{B}}(\mathbf{T})$ be the processor speed algorithm $\mathcal{B}$ needs to schedule $\mathbf{T}$, and let $\rho^{\mathcal{A}}(\mathbf{T})$ be the speed *speedup factor*

necessary for scheduling $\mathbf{T}$ under algorithm $\mathcal{A}$. We define the maximum speedup factor $\rho^{\mathcal{A}\to\mathcal{B}}$ similar to the definition provided by Davis et al. [DRB+09a] as:

$$\rho^{\mathcal{A}\to\mathcal{B}} = \max_{\forall\mathbf{T}} \left\{ \frac{\rho^{\mathcal{A}}(\mathbf{T})}{\rho^{\mathcal{B}}(\mathbf{T})} \right\} \tag{2.1}$$

We normally refer to the speedup factor as ρ if the algorithms (or tests) considered in the speedup factor are clear. Furthermore, $\mathcal{B}$ is often a (potentially unknown) optimal algorithm or a necessary schedulability condition, so we implicitly assume such a comparison if not stated differently. For convenience, it is assumed that the algorithm or condition we compare to is able to schedule the considered task set on a platform with speed 1, i.e., we normalize the speed, resulting in the following speedup factor definition:

$$\rho = \max_{\forall\mathbf{T}} \left\{ \rho^{\mathcal{A}}(\mathbf{T}) \right\} \tag{2.2}$$

Note that the terms *dominance* and *optimality* in this work are used regarding schedulability. It is also possible to define these relations regarding another metric and we denote this accordingly, e.g., for optimality regarding speedup factors we use the term *speedup-optimal*. A scheduling algorithm is *speedup-optimal* (for a certain class of scheduling algorithms) if the required speedup factor in comparison to an optimal algorithm is not larger than the speedup factor required by any other scheduling algorithm (of the same class) when all tests are performed using exact schedulability tests.

speedup-optimal

2.4 UNIPROCESSOR SCHEDULING

We assume a given task set to be scheduled based on an online, sometimes also called priority-driven, scheduling algorithm, which can be divided into static-priority and dynamic-priority scheduling algorithms. For a *static-priority scheduling* algorithm, each task is assigned a static priority, which means that if τ_i has a higher priority than τ_j, all jobs of τ_i have a higher priority than all jobs of τ_j. In contrast, for *dynamic-priority scheduling*, the priority of a job may change during its execution. Specifically, it is possible that at a time point t_1 a job of task τ_i has a higher priority than a job of task τ_j while for another time point t_2 a job of task τ_j has a higher priority than a job of task τ_i. Furthermore, we assume a *work-conserving* scheduling strategy, which means that the processor never idles if a job is ready to be executed.

static-priority scheduling

dynamic-priority scheduling

work-conserving

Whenever a scheduling decision has to be taken, the scheduler assigns the processor to the job in the system with the highest priority. If no job is in the system, the processor idles. For non-preemptive scheduling, a scheduling decision has to be made at any point in time when a job finishes executing or when a job arrives while the processor is idle. For preemptive scheduling, in addition a scheduling decision has to be made at every point in time where a new job arrives to the system while another job is executing. In this case, the currently executed job is preempted if the new arriving job has a higher priority.

Note that the abbreviation FP (for fixed-priority) is common in the literature when addressing static-priority scheduling algorithms in general. Hence, we use the abbreviation FP as well.

2.4.1 STATIC-PRIORITY SCHEDULING

static-priority scheduling

For *static-priority scheduling*, the tasks in $\mathbf{T}$ are ordered and indexed according to their priority, i.e., τ_1 has the highest and τ_n the lowest priority. The priority assignment is denoted as P and the priority of a task as $P(\tau_i)$. For a given task $\tau_k \in \mathbf{T}$ we define $hp(\tau_k)$ as the tasks in $\mathbf{T}$ with higher priority than τ_k and $lp(\tau_k)$ as the tasks in $\mathbf{T}$ with lower priority than τ_k under the considered scheduling algorithm. Furthermore, $hep(\tau_k) = hp(\tau_k) \cup \tau_k$.

Critical Instant Theorem

For implicit-deadline periodic tasks, the worst-case interference to a task can be determined using the *Critical Instant Theorem* by Liu and Layland [LL73]:

Theorem 2.1: Critical Instant Theorem (Liu and Layland [LL73]). *A critical instant for any task occurs whenever the task is requested simultaneously with requests for all higher priority tasks.*

It can be directly extended to constrained-deadline task sets. Furthermore, it can be extended to sporadic tasks if all subsequent jobs of the higher-priority tasks are released as early as possible. Note that for a given task set and a specific task, a critical instant may also occur for other setups, e.g., by releasing the first job of one of the higher-priority tasks later. However, for those other setups, the interference is the same as resulting from the Critical Instant Theorem.

PREEMPTIVE SCHEDULING (FP-P)

Time Demand Analysis
worst-case response time
exact test

For constrained- or implicit-deadline task sets under preemptive static-priority scheduling, *Time Demand Analysis* (TDA) [LSD89] can be used to decide the schedulability of a task τ_k by determining the *worst-case response time* (WCRT) R_k of task τ_k. This means that TDA is an *exact test* for preemptive static-priority scheduling. According to TDA, a task τ_k is schedulable if the following equation holds [LSD89]:

$$\exists t \text{ with } 0 < t \leq D_k \text{ and } C_k + \sum_{\tau_i \in hp(\tau_k)} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t \tag{2.3}$$

If such a t exists, the smallest value t where Eq. (2.3) holds is the WCRT of task τ_k. To determine the schedulability of a task set $\mathbf{T}$ using TDA, all tasks in $\mathbf{T}$ are tested in order of decreasing priority, and the task set is deemed schedulable if Eq. (2.3) holds for each $\tau_k \in \mathbf{T}$. Note that Time Demand Analysis cannot be used to determine the WCRT R_k of τ_k if $R_k > T_k$.

Rate Monotonic
optimal algorithm
Deadline Monotonic

Among *preemptive static-priority scheduling* algorithms *Rate Monotonic* (RM-P) scheduling, i.e., tasks with a smaller period have a higher priority, is an *optimal algorithm* for scheduling implicit-deadline task sets [LL73]. For constrained-deadline task sets [LW82], *Deadline Monotonic* scheduling (DM-P) is optimal, where tasks with a shorter relative deadline have a higher priority. For any priority order that is determined on task parameters, e.g., RM-P and DM-P, we assume that if two (or more) tasks have the same parameter value, the order among these tasks is determined arbitrary but fixed, i.e., if τ_i is chosen to have a

higher priority than τ_j it will always have a higher priority than τ_j. An optimal priority order for arbitrary-deadline task sets can be determined using Audsley's Algorithm, which is discussed at the end of this subsection.

utilization bound
Liu and Layland Bound

The well-known *Liu and Layland Bound* [LL73], also called total utilization bound, for RM-P is

$$U_{sum}(n) \leq n(2^{\frac{1}{n}} - 1) \tag{2.4}$$

where n is the number of tasks. When n converges to infinity, the total utilization bound is $U_{sum} \leq ln(2) \approx 0.693$.

Hyperbolic Bound

The *Hyperbolic Bound* (HB) for RM-P by Bini et al. [BBB01] is defined as

$$\prod_{i=1}^{n}(U_i + 1) \leq 2 \tag{2.5}$$

and dominates the Liu and Layland Bound while having the same runtime complexity and the same total utilization bound of 0.693.

Quadratic Bound

The *Quadratic Bound* (QB) by Davis and Burns [DB08] and Bini et al. [BNR+09] states that task τ_k is schedulable by RM scheduling if

$$\sum_{i=1}^{k} U_i + \frac{\sum_{i=1}^{k-1} C_i - \sum_{i=1}^{k-1} U_i C_i}{T_k} \leq 1 \tag{2.6}$$

This test has a utilization bound of $2 - \sqrt{2} \approx 0.58578$, as shown in [ASL04; HC15b] for more general task models.

NON-PREEMPTIVE SCHEDULING (FP-NP)

blocking time
work-conserving

In addition to the higher-priority interference, the *blocking time* by lower-priority tasks has to be taken into account when analyzing the schedulability under non-preemptive scheduling. For a *work-conserving* schedule, since the highest priority job in the ready queue is always chosen for execution, a job of τ_k can be blocked by a lower-priority task at most once, i.e., if at its arrival time a job $\tau_i \in lp(\tau_k)$ is executing. Hence, we can define the maximum blocking time B_k^* of a task τ_k under non-preemptive scheduling as

$$B_k^* = \max_{\tau_i \in lp(\tau_k)} \{C_i - \Delta\} \tag{2.7}$$

where $\Delta > 0$ but infinitesimally small.

One way to verify the schedulability of a task τ_k under FP-NP is to extended TDA to the non-preemptive case by including the maximum blocking time. Therefore, as shown in [Bur94; DGC10], a sufficient schedulability test of task τ_k under FP-NP is to verify whether

$$\exists t \text{ with } 0 < t \leq D_k \text{ and } B_k^* + C_k + \sum_{\tau_i \in hp(\tau_k)} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t \tag{2.8}$$

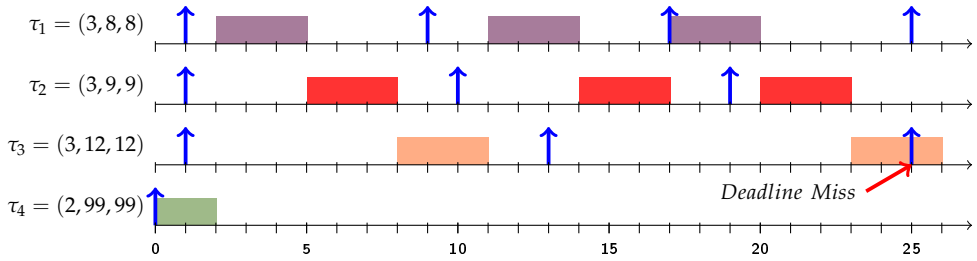


Figure 2.1: Self-pushing phenomenon for τ_3. Adapted from [But11].
Note that it is sufficient if τ_4 starts executing an infinitesimal amount of time before τ_1, τ_2, and τ_3 are released.

The analysis in Eq. (2.8) is pessimistic since it implies that task τ_k can be preempted by a higher-priority task. The pessimism can be removed by checking whether a job of task τ_k that arrives at time r can start no later than $r + T_k - C_k$. As shown by Tindell and Burns [TB94], this is equivalent to the validation of

$$\exists t \text{ with } 0 < t \leq D_k - C_k \text{ and } B_k^* + \sum_{\tau_i \in hp(\tau_k)} \left(\left\lfloor \frac{t}{T_i} \right\rfloor + 1 \right) C_i \leq t \tag{2.9}$$

However, solely testing Eq. (2.9) is not sufficient due to the well-known *self-pushing phenomenon* presented by Bril et al. [BLV09], which is exemplified in Figure 2.1. It shows that deadline misses are possible even if the condition in Eq. (2.9) is satisfied, as a deadline miss not necessarily happens for the first job of a task under FP-NP. An exact schedulability test for FP-NP was presented by Davis et al. [DBB+07], exploiting the *busy interval* concept. It requires checking all the jobs of task τ_k released in the busy interval of task τ_k, i.e., the longest interval starting with a job J_b blocking τ_k where only J_b, jobs of tasks in $hp(\tau_k)$, or jobs of τ_k itself are executed. Since B_1^* may be larger than T_1, the total utilization bound drops to 0 for RM-NP [NBF+14].

self-pushing phenomenon

busy interval

In some cases, another possibility to reduce the pessimism of the test in Eq. (2.8) is to adopt the following sufficient schedulability from Yao, Buttazzo, and Bertogna [YBB10].

Lemma 2.2 (Yao, Buttazzo, and Bertogna [YBB10])**.** *The worst-case response time of a non-preemptive task occurs in the first job if the task is activated at its critical instant and the following two conditions are both satisfied:*

1. *the task set is feasible under preemptive scheduling;*
2. *the relative deadlines are less than or equal to periods.*

Therefore, a sufficient schedulability test for task τ_k under FP-NP is to validate whether Eq. (2.3) and Eq. (2.9) both hold.

We use a *strict upper bound* B_k of the maximum blocking time B_k^* as

$$B_k = \max_{\tau_i \in lp(\tau_k)} \{C_i\} > \max_{\tau_i \in lp(\tau_k)} \{C_i - \Delta\} \tag{2.10}$$

Removing Δ introduces some slight pessimism into the analysis, since it now assumes the lower-priority task blocking τ_k started at the release time of the job of

τ_k and not an infinitesimal amount of time before. However, since Δ is assumed to be a very small positive number, this pessimism is neglectable. On the other hand, moving from B_k^* to B_k will sometimes simplify the analysis as shown later.

OPTIMAL PRIORITY ASSIGNMENT (OPA)

optimal priority assignment

If a priority assignment is not given in advance and no optimal a-priori priority assignment strategy is known, Audsley's Algorithm [Aud91] can sometimes determine an *optimal priority assignment* (OPA).

Definition 2.1 (Optimal Priority Assignment (from [DCB+16]))**.** *A priority order P is said to be optimal with respect to a configuration (task model $\mathbf{T}$, fixed priority scheduling policy G, and schedulability test S), if and only if every set of tasks that is compliant with the task model and is deemed schedulable under scheduling policy G by schedulability test S with some priority order is also deemed schedulable under scheduling policy G by schedulability test S using priority order P.*

In other words, P is optimal if it is as least as good as any other priority order.

The idea of Audsley's Algorithm [Aud91] is to first find a task τ_i that can take lowest priority, i.e., that is schedulable according to S if all other tasks in $\mathbf{T}$ have higher priority than τ_i. Afterwards, τ_i is removed from the set and another task is determined that can take lowest priority among the remaining tasks $\mathbf{T} \setminus \{\tau_i\}$, etc.

OPA compatible

While OPA was originally designed for periodic tasks with arbitrary start times (phases), it was later shown by Davis and Burns [DB09] to be applicable to a wider range of problems if the schedulability test S is *OPA compatible*, i.e., the following three conditions all hold [DB09]:

1. The schedulability of a task τ_k, according to S, may be dependent on the set of $hp(\tau_k)$, but not on the relative priority ordering of tasks in $hp(\tau_k)$.
2. The schedulability of a task τ_k, according to S, may be dependent on the set of $lp(\tau_k)$, but not on the relative priority ordering of tasks in $lp(\tau_k)$.
3. When the priorities of any two tasks of adjacent priority levels are swapped, the task being assigned the higher priority after the swap cannot become unschedulable according to S, if it was previously schedulable at the lower priority.

Audsley's Algorithm [Aud91] can be used to find an optimal priority assignment for preemptive arbitrary-deadline task sets and for non-preemptive scheduling of implicit-, constrained-, and arbitrary-deadline task sets.

2.4.2 DYNAMIC-PRIORITY SCHEDULING

dynamic-priority scheduling
Earliest Deadline First

For *dynamic-priority scheduling*, we focus on *Earliest Deadline First* (EDF) in both the preemptive and the non-preemptive case.

PREEMPTIVE EDF (EDF-P)

For preemptive scheduling, EDF-P is optimal regarding schedulability [Der74]. The *total utilization bound* of EDF-P [LL73] is: *utilization bound*

$$U_{sum} \leq 1 \tag{2.11}$$

for implicit-deadline task sets, hence, Eq. (2.11) is an exact test.

To handle task sets with arbitrary deadlines, Baruah et al. [BMR90] introduced the *demand bound function* (dbf_i) of a task τ_i as *demand bound function*

$$dbf_i(t) \;=\; max\left\{0, \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right\} C_i \tag{2.12}$$

to calculate the demand generated by task τ_i in an interval of length t. As shown by Baruah et al. [BMR90] a task set $\mathbf{T}$ with arbitrary deadlines is schedulable under EDF-P *if and only if*

1. the total utilization is no greater than 1 (i.e. $\sum_{i=1}^{n} U_i \leq 1$), and
2. for any time interval of length t, the total processor demand $dbf(t)$ requested by the task set is not greater than the length of the interval, i.e.,

$$dbf(t) = \sum_{i=1}^{n} dbf_i(t) \leq t \qquad \forall\, t > 0 \tag{2.13}$$

NON-PREEMPTIVE EDF (EDF-NP)

George et al. [GRS96] extended this demand bound test to the non-preemptive case by introducing a blocking factor $B(t)$. They showed that an arbitrary-deadline task set is schedulable under EDF-NP *if and only if*

1. the total utilization is no greater than 1, i.e. $\sum_{i=1}^{n} U_i \leq 1$, and
2. for any interval of length $t \geq D_1$ (where D_1 is the smallest task deadline)

$$dbf(t) + B(t) \leq t \tag{2.14}$$

where

$$B(t) = \max_{\forall i, D_i > t} (C_i - \Delta) \tag{2.15}$$

with $\Delta > 0$ but infinitesimally small [DBB+15].

It is sufficient to test every time point t where the demand bound function changes, i.e., every $t \in S = \left\{\cup_{i=1}^{n} \{kT_i + D_i\}, k \in \mathbb{N}\right\} \cap (0,\ L]$ where L is the longest *busy interval*. S is the union of the deadlines of all tasks in $(0,\ L]$ [GRS96]. *busy interval*

Under non-preemptive scheduling, no *work-conserving* scheduling policy is optimal, since it is possible that for an optimal schedule under a non-preemptive policy the processor must sometimes idle even if jobs are ready to be executed. However, EDF-NP is optimal among all work-conserving scheduling algorithms [GMR95]. Similar to FP-NP, for EDF-NP the total *utilization bound* is 0. *work-conserving* *utilization bound*

2.5 MULTIPROCESSOR SCHEDULING

We assume homogeneous multiprocessor systems with m processors, i.e., all processors and therefore the task parameters on all processors are identical. In multiprocessor systems, three scheduling paradigms are commonly followed:

partitioned scheduling

- **Partitioned scheduling**: Each task is statically allocated to a specific processor, i.e., all its instances are executed on the allocated processor. On each processor, the actual schedule is determined by means of a uniprocessor scheduling policy. Hence, for each processor an individual ready queue is maintained and on each processor the highest-priority job in the related ready queue is scheduled. Well known examples are partitioned EDF and partitioned RM.

global scheduling

- **Global scheduling**: Tasks are allowed to migrate freely between processors, such that the m highest-priority jobs among all ready jobs in the system are executed at any point in time. Hence, the jobs are scheduled based on one global ready queue. Well known examples are global EDF and global RM.

semi-partitioned scheduling

- **Semi-partitioned scheduling**: The tasks are allocated to particular processors but a certain degree of migration is allowed, e.g., in predefined time slots or depending on specified constraints.

A comprehensive survey on multiprocessor scheduling was provided by Davis and Burns in [DB11a].

TASK PARTITIONING STRATEGIES

Determining an optimal task partition is NP-hard in the strong sense, owing to the underlying bin-packing problem. Hence, several heuristics that usually consist of two phases, based on the initial work by Baruah and Fisher [BF05], have been exploited:

1. A preprocessing where the tasks are pre-sorted (usually) based on task parameters, e.g., according to DM order or decreasing regarding task utilization.
2. The actual partitioning where the tasks are assigned to the processors one by one according to the pre-sorting (also called pre-order). For each task the processors are considered according to an *assignment strategy*. All tasks are allocated to the first processor (according to the strategy) that fulfils a specific condition.

assignment strategy

The following assignment strategies are commonly used for the task partitioning:

- First-Fit (FF): If a task can be allocated is always tested in increasing order based on the processor ID.
- Best-Fit (BF): For each task, the processors are considered in decreasing order with respect to their utilization.
- Worst-Fit (WF): For each task, the processors are tested in increasing order with respect to their utilization.

- Arbitrary-Fit (AF): For each task, the processors are tested in random order.

The specific condition used to decide whether a task can be allocated is often a sufficient or exact schedulability test. For instance, if RM is assumed on each processor, the Liu and Layland Bound [LL73] or TDA [LSD89] can be exploited. used. However, some approaches use a necessary condition when assigning the tasks, e.g., that the individual processor utilization is $\leq 100\%$, and determine the schedulability on each individual processor after all tasks are partitioned.

2.6 UNCERTAIN EXECUTION BEHAVIOUR

When considering an uncertain execution behaviour, each task τ_i is described by a tuple $((C_{i,1}, ..., C_{i,h}), D_i, T_i)$, i.e., it has a set of h distinct execution modes $\mathcal{M}$ and each mode j with $j \in \{1, ..., h\}$ is associated with a WCET $C_{i,j}$. We assume those execution modes to be ordered increasingly according to their WCETs, i.e., $C_{i,m} \leq C_{i,m+1}\ \forall m \in \{1, ..., h-1\}$. In addition, $\mathbb{P}_i(j)$ denotes the probability that a job of task τ_i is executed in mode j and we assume that each job is executed in exactly one of these distinct execution modes, i.e., $\sum_{j=1}^{h} \mathbb{P}_i(j) = 1$. Furthermore, these probabilities are assumed to be independent from each other according to the following definition:

Definition 2.2 (Independent Random Variables)**.** *Two random variables are (probabilistically) independent if the realization of one does not have any impact on the probability of the other.*

Especially, the probability that a newly arriving job of τ_i has a certain execution mode is independent of the execution mode of all jobs currently in the system and of the execution mode of all previous jobs. For a task τ_j in $hp(\tau_i)$, $\rho_{j,t}$ is the maximum number of jobs that are released in an interval $[0, t)$, also called the interval of interest, and therefore interfere with task τ_i, i.e., the number of jobs released in the interval $[0, t)$ under the critical instance of τ_k. Furthermore, $\rho_{i,t}$ is the number of jobs of task τ_i in the analysis window. This notation implicitly assumes that the time window analyzed for τ_i starts at 0 for notational brevity.

normal mode
abnormal mode

In addition, if tasks have two distinct execution modes, we assume a more common execution mode, called *normal mode*, that has a smaller WCET C_i^N while the rare *abnormal mode* has a larger WCET C_i^A. The related probabilities are $\mathbb{P}_i(N)$ and $\mathbb{P}_i(A)$, and we assume that $\mathbb{P}_i(N) \gg \mathbb{P}_i(A)$ and $\mathbb{P}_i(N) + \mathbb{P}_i(A) = 1$. We denote the *normal utilization* of task τ_i as $U_i^N = C_i^N / T_i$ and the *abnormal utilization* of task τ_i as $U_i^A = C_i^A / T_i$. The total or system utilization in the normal mode is $U_{sum}^N = \sum_{\{\tau_i \in \mathbf{T}\}} U_i^N$ and the total utilization in the abnormal mode is referred to by $U_{sum}^A = \sum_{\{\tau_i \in \mathbf{T}\}} U_i^A$. Let E_i^N and E_i^A be the tardiness of tasks τ_i if all jobs are executed normally and abnormally.

2.7 SELF-SUSPENSION

dynamic self-suspension

Two self-suspension models are studied in the literature. For the *dynamic self-suspension* task model a task τ_i is specified like an ordinary sporadic task that has the worst-case self-suspension time S_i as an additional parameter. Task τ_i may suspend itself at any moment before it finishes as long as the total worst-case self-suspension time S_i is not violated.

segmented self-suspension

For the *segmented self-suspension* task model a task is specified by an array $(C_{i,1}, S_{i,1}, C_{i,2}, S_{i,2}, ..., S_{i,m_i}, C_{i,m_i+1})$, composed of $m_i + 1$ computation segments separated by m_i suspension intervals, where $C_{i,j}$ is the worst-case execution time of a computation segment and $S_{i,j}$ is the worst-case length of a self-suspension interval. For the segmented self-suspension model the total WCET $C_i = \sum_{j=1}^{m_i+1} C_{i,j}$ and the total self-suspension time $S_i = \sum_{j=1}^{m_i} S_{i,j}$.

suspension interval

We focus on the *segmented self-suspension* model and typically consider *one-segmented self-suspending* tasks where the execution of each job of τ_i is composed of two *computation* segments that are separated by one *suspension interval*. After the first computation segment is finished the job suspends itself, which means that it is removed from the ready queue for the length of the suspension interval and the job in the ready queue with the highest priority is executed. The second computation segment is eligible to be executed only after the completion of the suspension interval. Hence, after the suspension interval of a job ends, the job will be reentered into the ready queue. A one-segment self-suspending task τ_i is characterized by a tuple

$$\tau_i = ((C_{i,1}, S_i, C_{i,2}), T_i, D_i) \tag{2.16}$$

execution pattern

Unlike in a sporadic task the WCET is replaced by the *execution pattern* $(C_{i,1}, S_i, C_{i,2})$, where $C_{i,1}$ and $C_{i,2}$ are the WCETs of the first and second computation segment and S_i is an upper bound on the suspension time. For a self-suspending task we denote $C_i = C_{i,1} + C_{i,2}$ and assume that $C_i + S_i \leq D_i$ for any task $\tau_i \in \mathbf{T}$. Furthermore, we denote $C_i^{max} = \max\{C_{i,1},\ C_{i,2}\}$ and $C_i^{min} = \min\{C_{i,1},\ C_{i,2}\}$. In addition to the task utilization $U_i = C_i/T_i$, we denote $U_{i,1} = C_{i,1}/T_i$ and $U_{i,2} = C_{i,2}/T_i$ for notational brevity.

2.8 RESOURCE SHARING

shared resource

non-critical section

critical section

race conditions

When examining resource sharing, we assume a system with r mutually exclusive *shared resources* $\mathcal{R} = \{\mathcal{R}_1, \mathcal{R}_2, ..., \mathcal{R}_r\}$. Each sporadic task is characterized as $\tau_i = (C_i, A_i, T_i, D_i)$, and the WCET of τ_i is $C_i + A_i$ where C_i is an upper bound on the amount of execution time without resource access, called *non-critical section* WCET, and A_i an upper bound on the amount of execution time during resource access, called *critical section* WCET. Shared resources can be in-memory data, e.g., a set of variables, or external objects, like files, database connections, and network connections. To prevent *race conditions*, shared resources are accessed mutually exclusively, which means that for each shared resource $\mathcal{R}_j$ at any point in time no two jobs are both in a critical section that accesses the same resource $\mathcal{R}_j$. We focus

on *logical* shared resources, i.e., a piece of code executed on processors. Therefore, the considered shared resources are not processor-specific. The jobs of any task may request exclusive access to any of the shared resources $\mathcal{R}_1, \mathcal{R}_2, ..., \mathcal{R}_r$. Furthermore, the access to a shared resource is assumed to be non-preemptive from other accesses to the same resource, i.e., once a task τ_i accesses a shared resource $\mathcal{R}_j$ no other task is allowed to access $\mathcal{R}_j$ until τ_i finishes the execution on $\mathcal{R}_j$. We restrict ourselves to the case where each job of each task accesses only one shared resource at most once, thus, the critical sections are not nested by definition. As a result, each task is decomposed in three subtasks, representing the execution before, during, and after the critical section. We assume that these subtasks are described by a given *execution pattern*. To be precise, for each τ_i, we assume to know the share of C_i before the critical section, denoted as $C_{i,1}$, the WCET inside the critical section, denoted as A_i, and the share of C_i after the critical section, denoted as $C_{i,2}$. We assume this pattern is precisely known, i.e., $C_{i,1} + C_{i,2} = C_i$.

execution pattern

The utilization of task τ_i for non-critical sections is defined as $U_i^C = C_i/T_i$ and the total critical section utilization of task τ_i is denoted by $U_i^A = A_i/T_i$. Hence, the utilization of task τ_i is $U_i = (C_i + A_i)/T_i$. The utilization of τ_i on resource $\mathcal{R}_q$ is $U_i^{\mathcal{R}_q}$, i.e., $U_i^{\mathcal{R}_q} = U_i^A$ if τ_i accesses $\mathcal{R}_q$ and 0 otherwise. The total utilization of resource $\mathcal{R}_q$ is $U^{\mathcal{R}_q} = \sum_{\tau_i \in \tau} U_i^{\mathcal{R}_q}$, the total utilization of non-critical-sections is $U^C = \sum_{\tau_i \in \tau} U_i^C$, and $U^{\mathcal{R}} = \sum_{\mathcal{R}_q \in \mathcal{R}} U^{\mathcal{R}_q}$ is the total utilization of shared resources. Furthermore, $R_i(C_{i,1})$, $R_i(C_{i,2})$, and $R_i(A_i)$ denote the worst-case response time of the related subtask (under the considered scheduling algorithm).

UNIPROCESSOR RESOURCE SHARING

As shared resources must be serially executed to achieve mutual exclusion, the execution of critical sections inevitably causes some delay due to *priority inversion* when a task τ_k is prevented from executing by another task with a lower priority that holds the requested shared resource, also called *pi-blocking*. The resulting *blocking time* B_k must be included in the analysis. We consider two well known approaches under static-priority scheduling:

priority inversion

pi-blocking

blocking time

Non-Preemptive Protocol (NPP): A critical section that has started to be executed cannot be preempted by any other job, i.e., it runs non-preemptively until the critical section is finished. Under a static-priority NPP the maximum blocking time B_k for a task τ_k is

Non-Preemptive Protocol

$$B_k = \max_{\tau_i \in lp(\tau_k)} \{A_i\} \tag{2.17}$$

NPP can also directly be applied for FIFO queues or dynamic-priority scheduling. However, B_k must be calculated in a different way in this case.

Priority Inheritance Protocol (PIP) and Priority Ceiling Protocol (PCP): Both protocols were introduced by Sha et al. [SRL90] to avoid unnecessary blocking of high-priority tasks from unrelated shared resources. In the PIP, a lower-priority task temporarily inherits the priority of a higher-priority task that it blocks. Regardless, under PIP a job may still be blocked by at most $\min\{n, r\}$ tasks. Furthermore, PIP does not prevent deadlocks, even if resources are properly

Priority Ceiling Protocol

nested. To tackle these problems, in the PCP each resource $\mathcal{R}_q$ is assigned a priority ceiling $\mathcal{C}(\mathcal{R}_q)$ that is equivalent to the base priority of task τ_j of the highest-priority task that accesses $\mathcal{R}_q$. Under PCP, a job can only allocate a resource, if its priority is higher than the highest priority ceiling among the currently allocated resources. As a result, deadlocks are avoided and each task can be blocked by at most one lower-priority task. Let $\mathbf{L}_k$ be the subset of $lp(\tau_k)$ in which the resource ceiling of the shared resource requested by a task τ_i in $\mathbf{L}_k$ is higher than or equal to the priority of τ_k. As shown by Sha et al. in [SRL90], the blocking time under PCP is

$$B_k = \max_{\tau_i \in \mathbf{L}_k} \{A_i\} \tag{2.18}$$

3

RELATED WORK

As this dissertation covers a large spectrum of real-time systems, the goal of this chapter is not to provide a comprehensive survey of the individual areas, but to highlight fundamental results, to summarize the state-of-the-art, and to point out interesting recent work.

While the focus of this dissertation is recurrent real-time systems with a periodic or sporadic arrival pattern, first some details regarding the scheduling of aperiodic tasks are provided in Section 3.1, since some general concepts as well as complexity results yield from this setting. Section 3.2 is related to modelling of recurrent real-time task systems. Uniprocessor scheduling of periodic and sporadic tasks considering preemptive and non-preemptive scheduling is addressed in Section 3.3. Section 3.4 considers multiprocessor scheduling, assuming homogeneous processors, no inter-task parallelization, and preemptive scheduling. Section 3.5 focuses on automotive task systems and rate-dependent tasks. An overview of mixed-criticality systems is provided in Section 3.6 while probabilistic response time analysis is covered in Section 3.7. Self-suspending tasks systems are the focus of Section 3.8. Afterwards, related work for multiprocessor resource sharing is summarized in Section 3.9. The chapter is concluded in Section 3.10 by a short description of which parts of the related work the following 3 chapters relate to.

3.1 APERIODIC TASKS

aperiodic task sets

In this scenario, for a set of independent tasks $\mathbf{T}$ each task τ_i is assumed to release a single job. Each task $\tau_i \in \mathbf{T}$ is specified by its worst-case execution time (WCET) C_i, its release time a_i, and its absolute deadline d_i. Historically, the very first objective of operation research was to minimize the maximum lateness among the aperiodic tasks. If the maximum lateness is zero, all tasks meet their deadlines. The first algorithm considering the special case that all tasks are released synchronously, called *Earliest Due Date*, was provided by Jackson in 1955 [Jac55], stating that "any algorithm that executes the tasks in order of non-decreasing deadlines is optimal with respect to minimizing the maximum lateness", also called *Jackson's rule*. Due to the synchronous release, the resulting schedule is by construction non-preemptive. Assuming preemptive scheduling, this algorithm was extended to arbitrary arrival times by Horn in 1974 [Hor74] who proposed the *Earliest Deadline First* (EDF) algorithm which was formally shown to be *optimal* by Dertouzos [Der74] in the same year.

Earliest Deadline First
optimal algorithm

When considering tasks with arbitrary arrival times under *non-preemptive scheduling*, Lenstra et al. in 1977 [LRKB77] proved that the problem of minimizing the maximum lateness is $\mathcal{NP}$-hard in the strong sense. EDF is no longer optimal, since in the non-preemptive setting, an optimal schedule may contain idle time,

non-preemptive scheduling

work-conserving

even if tasks are ready to be executed, while EDF is *work-conserving*. If the arrival times are known a priori, branch-and-bound algorithms, like the one provided by Bratley et al. in 1971 [BFR71], may be able to find a feasible solution offline but can result in an exhaustive search with a time complexity of $O(|J|!)$, where $|J|$ is the number of task instances in the system. Multiple approximation algorithms are known, among them the *extended Jackson's rule* [HS92], *Potts Algorithm* [Pot80], and its extensions by Hall and Shmoys [HS92]. If the arrival times are not known beforehand, no online algorithm is able to determine whether the processor should be idle or not as shown by Howell and Venkatrao in 1995 [HV95]. Resulting from this, under work-conserving non-preemptive scheduling a task set may become unschedulable when the execution time of a task is reduced, a so-called *timing anomaly*. However, it has been shown by Jeffay et al. in 1991 [JSM91] that EDF is optimal among work-conserving non-preemptive scheduling algorithms.

timing anomaly

3.2 MODELLING OF REAL-TIME SYSTEMS

periodic task model
implicit-deadline
constrained-deadline
sporadic task model

The analysis of recurrent real-time tasks can be traced back to the seminal work by Liu and Layland [LL73] who introduced the *periodic task model* in 1973. They assume *implicit-deadline* tasks with a fixed runtime that release an infinite number of *jobs* exactly periodically. Leung and Whitehead [LW82] considered periodic *constrained-deadline* task systems in 1982. The periodic model was extended to *sporadic releases* by Mok in his Dissertation[1] in 1983 [Mok83].

The periodic and sporadic task models have been extended to cover more complex task systems. Mok and Chen [MC96] proposed the multiframe task model to cover the case where the execution time of a task varies greatly but follows a known pattern. This approach was extended to generalized multiframe (GMF) task systems by Baruah et al. [BCG+99], where tasks are represented by a 3-tuple of vectors representing the worst-case execution times, relative deadlines, and interarrival times of the frames. While both the multiframe and the generalized multiframe model consider a cyclic activation pattern, Moyo et al. [MNL+10] proposed non-cyclic generalized multiframe tasks, where an instance of any frame can be activated after the minimum inter-arrival time specified by the previous frame has passed. Stigge et at. [SEG+11] proposed the digraph real-time task model where each task is described by a directed acyclic graph. The nodes in this graph represent the possible types of jobs with related relative deadline and worst-case execution time, while the edges show the possible transitions between the job types and the related minimum inter-arrival time.

Other important extensions of real-time models are mixed-criticality task systems [Ves07], which are covered in Section 3.6, and self-suspending task systems [CBH+17b], which are covered in Section 3.8. In automotive applications, rate-dependent tasks [FBD+18], which are discussed in Section 3.5, are common where the activation of jobs depends on the rotation of the crankshaft.

1 Note that the notation in Mok's Dissertation differs from the notation here both regarding the order of parameters as well as in the way they are denoted.

3.3 UNIPROCESSOR SCHEDULING

This section considers preemptive and non-preemptive scheduling of periodic and sporadic task sets in a uniprocessor setting.

3.3.1 PREEMPTIVE SCHEDULING

preemptive scheduling

In *preemptive uniprocessor systems*, the scheduler design problem has been solved for independent periodic and sporadic tasks under both static-priority and dynamic-priority scheduling. For *dynamic-priority scheduling*, preemptive Earliest-Deadline-First (EDF-P) scheduling has been shown to be an *optimal scheduling policy* for implicit-deadline task sets by Liu and Layland [LL73].[2] This optimality also holds for constrained- and arbitrary-deadline task sets [Der74]. For implicit-deadline task sets, an exact schedulability test only needs to verify whether the processor utilization is less than or equal to 100% [LL73]. Baruah et al. [BMR90; BRH90] provided exact schedulability tests for constrained- and arbitrary-deadline task sets under EDF-P *using demand bound functions*.

dynamic-priority scheduling

optimal algorithm

static-priority scheduling

optimal algorithm

For preemptive *static-priority scheduling*, Liu and Layland [LL73] showed that *Rate Monotonic* (RM-P) scheduling is *optimal* for *implicit-deadline* task sets. For *constrained-deadline* task sets, *Deadline Monotonic* (DM-P) scheduling is optimal as shown by Leung and Whitehead [LW82]. Both results hold for synchronous periodic releases as well as for sporadic task sets, but not for periodic task sets with arbitrary phases [LW82; Aud91], i.e., the first jobs are not synchronously released.[3] Audsley [Aud91] provided an *optimal static-priority assignment* (OPA) strategy for this situation. An exact schedulability test for constrained- and implicit-deadline task sets called *Time Demand Analysis* was introduced by Lehoczky et al. [LSD89]. TDA exploits the concept of (worst-case) *response time analysis* that was introduced by Joseph and Pandya [JP86]. For *arbitrary-deadline* task sets, neither RM-P nor DM-P are optimal [Leh90]. Audsley's Algorithm is also applicable to find an optimal static-priority assignment for *arbitrary-deadline* task sets, using the exact schedulability test for arbitrary-deadline task sets under static-priority scheduling that was provided by Lehoczky [Leh90]. Davis and Burns [DB09] proved that three conditions must be met to ensure that Audsley's Algorithm can be applied for a schedulability test, i.e., the schedulability test is *OPA compatible*.

optimal priority assignment

response time analysis

OPA compatible

theoretical evaluation method

utilization bound

breakdown utilization

That EDF-P has a *utilization bound* of $\sum_{\tau_i} U_i \leq 100\%$, was shown by Liu and Layland [LL73] who also provided the utilization bound of $\sum_{\tau_i} U_i \leq 69.3\%$ for RM-P [LL73]. The *Hyperbolic Bound* (HB) for RM-P by Bini et al. [BBB01] dominates this result but leads to the same total utilization bound. Kuo and Mok proved that the utilization bound for RM-P is 100% for *harmonic task sets* [KM91]. While the utilization bound of 69.3% for RM-P is tight, Lehoczky et al. [LSD89] showed that the average *breakdown utilization* of RM-P is 88% in a stochastic analysis. For uniformly distributed utilization values, Bini and Buttazzo [BB05] determined an average breakdown utilization of over 90%.

2 EDF is called *Deadline Driven Scheduling Algorithm* in their work.

3 Examples for this behaviour can be found, for instance, in the work by Audsley [Aud91].

speedup factor

The *speedup factors* of RM-P compared to the optimal EDF-P for implicit-deadline task sets follows directly from the utilization bounds by Liu and Layland [LL73]. Davis et al. [DRB+09a] showed that for constrained-deadline task sets the speedup factor of DM-P compared to EDF-P is ≈ 1.76322. Since RM-P and DM-P are scheduling optimal for static-priority scheduling of implicit- and constrained-deadline task sets, respectively, they are also *speedup-optimal*. While not being scheduling optimal, DM-P is also speedup-optimal for arbitrary deadline task sets as shown by Davis et al. [DBB+15]. Therefore, the speedup factor of 2 that Davis et al. [DRB+09b; DBB+15] provided for DM-P compared to EDF-P when scheduling arbitrary-deadline task sets is the same as for the the optimal priority assignment resulting from Audsley's Algorithm.

speedup-optimal

3.3.2 NON-PREEMPTIVE SCHEDULING

dynamic-priority scheduling

Some results for non-preemptive scheduling directly stem from the aperiodic case. Scheduling of periodic non-preemptive tasks is $\mathcal{NP}$-hard in the strong sense [LRKB77], since periodic releases are a special case of aperiodic releases with given arrival times. For periodic non-preemptive task sets branch-and-bound algorithms, e.g., by Bratley et al. [BFR71], as well as approximation algorithms for aperiodic task sets like the *extended Jackson's rule* [HS92], Potts Algorithm [Pot80], and its extensions by Hall and Shmoys [HS92] can be used if the complexity for unrolling all job releases in the hyperperiod is affordable.[4] Nasri and Fohler [NF16] investigated non-work-conserving EDF-NP scheduling considering critical time windows. For sporadic releases, no algorithm can determine whether the processor should be idle or not [HV95], and EDF-NP is optimal among work-conserving scheduling algorithms [JSM91; GMR95]. George et al. [GRS96] provided an exact schedulability test for EDF-NP for sporadic tasks with arbitrary deadlines.

static-priority scheduling

George et al. [GRS96] proved that DM-NP is not optimal for constrained-deadline task sets, provided an exact schedulability test for *static-priority non-preemptive scheduling*, and showed that Audsley's Algorithm [Aud91] finds an optimal priority assignment. Nasri and Kargahi [NK14] proposed an online algorithm called Precautious-RM and showed that it can increase the schedulability under RM-NP for periodic systems by introducing idle times.

theoretical evaluation method
utilization bound

Nasri et al. [NBF+14] proved that the *utilization bound* of non-preemptive scheduling is 0, both for static- and dynamic-priority scheduling, which results from a situation where the worst-case execution time of one task is larger than the period of another. However, to the best of our knowledge, there is no analysis that examines the utilization bounds for non-preemptive scheduling in situations where the blocking time can be bounded with respect to the period or the worst-case execution time of tasks.

speedup factor

Davis et at. [DGC10] showed that the *speedup factor* for FP-NP compared to EDF-NP is lower bounded by ≈ 1.76322 and upper bounded by 2 for implicit-, constrained-, and arbitrary-deadline task sets. A lower bound of 2 in the arbitrary-deadline case was later provided by Davis et al. [DBB+15].

4 Note that the number of jobs in the hyperperiod can be exponential in the number of tasks.

3.3.3 LIMITED-PREEMPTIVE SCHEDULING

limited-preemptive scheduling

The idea of *limited-preemptive scheduling* techniques is to combine the advantages of preemptive and non-preemptive scheduling, i.e., the improved schedulability of preemptive scheduling where high-priority jobs can be allocated to the processor nearly immediately with the reduced worst-case execution time that results from preventing preemption at worst-case points in the code and from reducing the number of possible preemptions in general. The *preemption thresholds* by Wang and Saksena [WS99] allow to set a specific priority level for each task, i.e., the preemption threshold, and the task can only be preempted by tasks with a higher priority than this preemption threshold. Wang and Saksena [WS99] also introduced an algorithm that assigns these thresholds in a way that ensures feasibility if a feasible schedule exists. The *co-operative scheduling* by Burns [Bur94] allows preemption only at predefined preemption points in the code, therefore splitting the tasks into non-preemptive subtasks. An algorithm that selects optimal preemption points, in the sense that it achieves feasibility while minimizing the preemption costs, was presented by Bertogna et al. [BXM+11]. Under *deferred preemption* [BLV07], the maximum length of a non-preemptive interval is defined for each task, and the actual preemption is deferred accordingly, either based on a timer that is triggered by the arrival of an higher-priority job or by inserting specific primitives that disable and enable preemption into the code, so called floating non-preemptive regions [Bar05]. The problem of finding the longest non-preemptive region while still ensuring schedulability was solved by Baruah [Bar05] for EDF and by Yao et al. [YBB09] for static-priority scheduling. For a given task set, the scheduling analysis under limited-preemptive scheduling is similar to the analysis under non-preemptive scheduling but with a reduced blocking time due to lower-priority tasks. A survey on limited-preemptive scheduling in real-time systems has been provided by Buttazzo et al. [BBY13].

preemption threshold

co-operative scheduling

deferred preemption

3.4 MULTIPROCESSOR SCHEDULING

timing anomaly

Multiple *timing anomalies* can affect the schedulability of real-time tasks in multiprocessor systems as detailed by Graham in 1969 [Gra69], who pointed out that a task set that is schedulable under a specific priority assignment may become unschedulable[5] when the number of processors is increased, the execution times of tasks are reduced, or precedence constraints are removed.

optimal algorithm

When arrival times, deadlines, and execution times for the set J of all jobs in the system are precisely known, Horn [Hor74] provided an *optimal algorithm* in 1974. While his algorithm can be applied for periodic task sets, it is usually not applicable in practice, since it has a runtime of $O(|J|^3)$ and the number of jobs $|J|$ in one hyperperiod is known to be exponential in the number of tasks. Baruah et al. [BCP+96] provided an *optimal algorithm* for implicit-deadline periodic task sets called Proportionate Fair or Pfair, which is often not applicable due to the very high runtime overhead. Fisher [Fis07] showed that no optimal online algorithm for sporadic tasks with arbitrary or constrained deadlines exists.

5 The original argument is based on increasing the makespan which translates to missing a deadline.

Multiprocessor scheduling approaches are categorized based on the underlying scheduling paradigm into 1) partitioned approaches, where tasks are statically allocated to processors, 2) global approaches, where tasks can freely migrate, and 3) hybrid approaches that combine properties of partitioned and global scheduling. Such hybrid approaches can be further categorized into two classes. In semi-partitioned approaches, some tasks are split into multiple subtasks and the subtasks are allocated to different processors. In clustering approaches, the processors are partitioned into clusters, each task is statically allocated to one cluster, and tasks can (usually) freely migrate among the processors of the cluster. Clustering approaches are not further discussed here since they are out of scope of this work. A comprehensive survey on preemptive multiprocessor scheduling on homogeneous processors has been provided by Davis and Burns in [DB11a].

3.4.1 GLOBAL SCHEDULING

global scheduling
Dhall's effect
utilization bound

Global multiprocessor scheduling suffers from *Dhall's effect*, which was found by Dhall and Liu in 1978 [DL78]. For implicit-deadline tasks, *Dhall's effect* leads to a *utilization bound* of 1 for global EDF and global RM, independent from the number of processors m. Andersson et al. [ABJ01] proved that the maximum utilization bound for periodic implicit-deadline tasks under any global dynamic scheduling algorithm with fixed job level priorities is $(m+1)/2$. However, Philipps et al. [PST+97] proved that global EDF has a speedup factor of $2 - \frac{1}{m}$. A conclusion of their work is that *Dhall's effect* needs at least one task with very high utilization, which lead to algorithms that exploited this property by giving high utilization tasks a higher priority, e.g., EDF-US by Srinvasan and Baruah [SB02]. A utilization bound depending on the maximum task utilization U_{max} was given by Goossens et al. [GFB03] as $m - (m-1)U_{max}$. Considering the density instead of the utilization, this bound has been extended to constrained-deadline task sets by Bertogna et al. [BCL05a] and arbitrary-deadline task sets by Baker and Baruah [BB07].

worst-case response time

It was shown by Lauzac et al. [LMM98b] that under global static-priority scheduling the *worst-case response time* (WCRT) of a task cannot necessarily be obtained by releasing a job simultaneously with all higher priority tasks. Instead, the WCRT may, for example, happen for a later job or when subsequent higher priority jobs are not released as early as possible. The problem that no general worst-case arrival pattern can be determined also exists for dynamic-priority global schedulers like global EDF as pointed out by Baruah [Bar07].

time complexity

Geeraerts et al. [GGL13] proved that deciding whether a sporadic task set is schedulable on m processors, e.g., by global EDF or global FP, is *PSPACE-complete*. Therefore, the known exact schedulability tests for global FP [SL16; SL14] and global EDF [BC07a; GGL13; BM12] are not scalable [BBT15].

response time analysis
work-conserving

Most sufficient schedulability analysis for sporadic task sets under global scheduling extends the *response time analysis* by Baker [Bak03]. The key concepts of this approach are to find a necessary condition for a task not be schedulable using the *work-conserving* property of the scheduling protocols, and to upper bound the interference a task may suffer in a specific time interval (the so-called

problem window) based on the the number of releases in the problem window and the carry-in interference.

For global EDF, Baruah [Baro7] provided a sufficient schedulability test with pseudo-polynomial runtime for constrained-deadline task sets by extending the problem window to the last point in time any of the processors idled, thus limiting the carry-in interference to $m - 1$ tasks. Baruah and Baker further improved on the result in [Baro7] and showed a speedup factor of 2.62 in [BBo8b]. They also extend the approach in [Baro7] to arbitrary deadlines [BBo8a].

For global RM, a *utilization bound* of $\frac{m}{2}(1 - U_{max}) + u_{max}$ that also holds for global DM when considering the density instead of the utilization was provided by Bertogna et al. [BCL05b]. Based on the work by Baker [Bako3], sufficient schedulability tests for constrained-deadline task sets under global FP have been proposed by, for instance, Baker [Bako6], Baruah, alone and with multiple co-authors, [Baro7; BFo8; BBM+10], Chen et al. [CHL16a], Bertogna et al. [BCL05b; BCo7b], Fisher and Baruah [FBo6], and Guan et al. [GSY+09]. Here, the approach by Guan et al. [GSY+09] can be seen as the state-of-the-art. *utilization bound*

For arbitrary-deadline task sets under global FP the analysis by Baker [Bako6] is based on his work in [Bako3]. Baruah and Fisher [BFo7a; BFo8] extended the analysis window and derived corresponding exponential-time schedulability tests using annotations. Guan et al. [GSY+09] provided a response-time analysis for arbitrary-deadline task systems utilizing the insight proposed by Baruah [Baro7] to limit the number of carry-in jobs, and applying the workload function proposed by Bertogna et al. [BCL05b] to quantify the demand of higher-priority tasks. However, Sun et al. [SLG+14] showed that the analysis in [GSY+09] is optimistic. Huang and Chen [HC15a] quantified the number of carry-in jobs of a task more precisely than in [Bako6; BFo8]. Recently, Chen et al. [CBU18] provided a series of schedulability tests that analytically dominate the tests by Baruah and Fisher [BFo8] and has an asymptotically tight speedup factor for global DM.

3.4.2 PARTITIONED SCHEDULING

The *multiprocessor partitioned scheduling problem* is $\mathcal{NP}$-hard as shown by Garey and Johnson [GJ79]. It has been shown by Andersson et al. [ABJo1] that no partitioning algorithm for implicit-deadline tasks can have a general *utilization bound* that is larger than $\frac{m+1}{2}$. Lopez et al. [LGD+oo] provided upper and lower bounds for partitioned EDF based on the maximum task utilization. Baruah and Fisher [BFo5] provided the deadline monotonic task partitioning strategy that can be combined with multiple assignment strategies like best-fit, worst-fit, first-fit, and arbitrary-fit. They extended their work in a series of papers for both EDF [BFo6; BFo7b] as well as to static-priority scheduling [FBBo6]. They also provided the *speedup factors* of their partitioning algorithms which are $2 - \frac{1}{m}$ for task sets with implicit-deadlines, $3 - \frac{1}{m}$ for constrained-deadlines, and $4 - \frac{2}{m}$ for arbitrary-deadlines. *partitioned scheduling* *utilization bound* *speedup factor*

3.4.3 SEMI-PARTITIONED SCHEDULING

semi-partitioned scheduling

In 2006, Andersson and Tovar [AT06] proposed a partitioned scheduling approach for implicit-deadline periodic tasks that splits some tasks into two subtasks that are executed on different processors without time overlap. Andersson and Bletsas [AB08] considered sporadic implicit-deadline tasks and proposed to split tasks in a way that each processor executes at most two split tasks, i.e., processor p shares one task with processor $p-1$ and one task with processor $p+1$. In their approach, the subtasks are scheduled in fixed time slots. Kato and Yamasaki provided a *semi-partitioned scheduling* approaches under EDF [KY07; KY08].

Considering static-priority scheduling, Kato and Yamasaki [KY09] proposed Deadline Monotonic scheduling with Priority Migration for sporadic task sets, which dominates partitioned static-priority approaches. In their approach, tasks that are migrated are always executed at highest priority and are migrated as soon as the subtask finished its predefined execution budged on a processor. Lakshmanan et al [LRL09] developed a semi-partitioned scheduling for implicit- or constrained-deadline task sets where always the highest priority task is split.

3.4.4 COMPARISON OF SCHEDULING PARADIGMS

Global scheduling has a higher runtime overhead than partitioned scheduling for managing one global ready queue, task migration costs have to be considered in the analysis, and additional cache misses may occur after migration. On the other hand, global scheduling should (on average) be feasible for larger system utilization values since the underlying bin packing problem results in unused capacity on the individual processors for partitioned scheduling and the utilization bound of partitioned scheduling is 50%.

However, the state-of-the-art analysis for global static-priority scheduling and global EDF stems from the seminal work by Baker [Bak03]. For such approaches, the interference upper bounds resulting from this general approach are multiplied with $1/m$ in the resulting sufficient schedulability tests. Recently, Sun and Di Natale [SN18] proved that for global static-priority scheduling the pessimism of such a response time analysis is so large that the analysis is dominated by partitioned static-priority scheduling. A similar result has been shown by Biondi and Sun [BS18] for global EDF and global FIFO scheduling. Hence, a fundamentally different analysis technique is needed to exploit the potentially higher utilization of global scheduling compared to partitioned scheduling.

Furthermore, a study by Brandenburg and Gul [BG16] showed that semi-partitioned approaches are able to schedule task sets with up to nearly 100% average processor utilization, demonstrating that the potential improvement of system utilization under global scheduling is very limited.

3.5 AUTOMOTIVE SYSTEMS AND RATE-DEPENDENT TASKS

Automotive systems are a field with high practical relevance where real-time constraints are extremely important. However, realistic automotive benchmarks are

usually not available to researchers, mainly due to intellectual property concerns. In 2015, Kramer et al. [KZH15] from Bosch Corporate Research released a paper that allows the generation of realistic benchmarks by providing the structure of typical automotive real-time systems, e.g., regarding period distribution, worst-case execution times, and average-case execution times. One specific characteristic of such task systems is that for the majority of tasks the release pattern is periodic and that the period is chosen from $\{1, 2, 5, 10, 20, 50, 100, 200, 1000\}$ ms, i.e., the periods are *semi-harmonic*. Such automotive task sets are usually scheduled using static-priority scheduling [KZH15]. While general scheduling analysis for period task sets can be applied for such systems, to the best of our knowledge, there is no analysis that provides a specific schedulability analysis and utilization bounds for this restricted period setting.

semi-harmonic task set

Furthermore, parts of the engine behaviour is often controlled by tasks that are triggered by the rotation of the crankshaft, i.e., the inter-arrival time depends on the revolutions per minute (rpm). Such rate-dependent tasks are denoted by different names in the literature, e.g., rate-dependent tasks [FBD+18], *angle-synchronous tasks* [KZH15], or tasks with variable rate-dependent behaviour [DFP+14]. For these tasks not only the inter-arrival time but also the relative deadline depends on the crankshaft rotation. Furthermore, in many cases the worst-case execution time of these tasks also changes, i.e., the WCET can be described by a set of values that are related to a certain interval of rotation speeds. Hence, modelling these tasks as sporadic tasks based on the shortest possible interarrival time and on the shortest deadline as well as on the largest WCET value can be extremely pessimistic. Such tasks can also be modelled as non-cyclic generalized multiframe tasks [MNL+10] or, even more general, as digraph real-time tasks [SEG+11], and the related analysis techniques can be applied. However, these approaches are not able to cover the fact that the rotation of the crankshaft does not change arbitrary but only with a maximum acceleration. Furthermore, an exact analysis for the digraph real-time task model and the generalized multiframe model is intractable[6] under static-priority scheduling [SY12]. Therefore, a specific modelling and a specialized analysis is needed. The importance of these tasks and their challenging analysis was highlighted by Buttle [But12] in the keynote talk at ECRTS 2012. The first specifically designed analyses were provided by Kim et al. [KLR12], assuming a single rate-dependent task at highest priority with an interarrival time that is always smaller than all other periods, and by Pollex et al. [PFS+13], assuming a constant angular speed.

angle-synchronous tasks

Sufficient schedulability tests for rate-dependent tasks with constrained deadlines under static-priority scheduling have been proposed by Davis et al. [DFP+14]. They also provided linear-time approximations to bound the interference of such tasks. Their evaluations show a huge performance gain compared to the scenario where rate-dependent tasks are modeled as simple sporadic tasks. Biondi et al. [BNB16] provided an exact schedulability test building on their previous result in [BNB15]. Huang and Chen [HC15b] considered static-priority mode-level scheduling, where each execution mode of a task is assigned to an individual static-priority, and showed a utilization bound of approximately 0.5857. However,

6 The complexity of an exact analysis of the non-cyclic generalized multiframe model is unknown [DFP+14].

to the best of our knowledge, there is no work that addresses how the priorities of angle-synchronous tasks should be assigned in relation to periodic tasks.

Considering Earliest Deadline First scheduling, Buttazzo et al. [BBB14] and Guo and Baruah [GB15] provided utilization based test. Biondi et al. [BBS15] introduced an exact schedulability test under the assumption that the acceleration is constant between two jobs.

A survey on schedulability analysis for rate-dependent tasks was provided by Feld et al. [FBD+18].

3.6 MIXED-CRITICALITY SYSTEMS

mixed-criticality systems

The *mixed-criticality* model has been introduced into the real-time systems research in the seminal work by Vestal [Veso7] in 2007, assuming constrained-deadline tasks with 4 criticality levels,[7] and a set of WCET related to the execution mode of the system. The WCET of each task is assumed to be increasing from the first (least critical) mode to the last (most critical) mode. The system also has a number of criticality modes (same as the number of criticality levels and usually with the same names) and starts executing in the lowest one. When a mode switch to the next mode, according to the criticality order, happens, all tasks are assumed to execute up to the WCET of the related level and it is only important that all tasks with the same or a higher criticality level always meet their deadline while no guarantees are given for the tasks with a lower criticality level. The model has been extended to level-dependent inter-arrival times and deadlines by Baruah and Burns [BB11a]. In most mixed-criticality research, systems with two criticality modes are assumed. In such systems, the terms *high-criticality* and *low-criticality* are commonly used both for the modes as well as for the criticality levels.

Vestal [Veso7] utilized *static-priority scheduling* and *Audsley's Algorithm*. Baruah and Vestal [BVo8] considered sporadic tasks under static-priority and dynamic-priority scheduling. They showed the existence of feasible mixed-criticality systems that are not schedulable by EDF and that EDF does not dominate static-priority scheduling. For *static-priority scheduling*, Baruah et al. [BBD11; BB11a] as well as Burns and Baruah [BB11b] provided a series of papers for systems with two criticality levels that abandon all low-criticality tasks at the moment of a mode change, i.e., when some task in the system executes for more than its WCET in the low-criticality mode. Their result in [BBD11] outperformed all other know static-priority approaches.

static-priority scheduling

dynamic-priority scheduling

Extending the work in [BVo8] regarding EDF, Guan et al. [GES+11] as well as Ekberg and Yi [EY12] assigned shorter deadlines to high-criticality tasks in low-criticality mode to ensure that they are executed before low-criticality tasks. When a system mode change happens, all low-criticality tasks are dropped and the high-criticality tasks are scheduled by their original deadlines. Baruah et al. [BBD+11; BBD+15] provided an algorithm called EDF-VD, where VD stands for virtual deadline, that uses a similar strategy. In EDF-VD all deadlines are reduced by

7 Which are are called design assurance levels the work by Vestal [Veso7].

the same factor. EDF-VD is the current reference algorithm for scheduling of dual-criticality systems under dynamic-priority scheduling.

Considering that mixed-criticality systems have been introduced in 2007, an astonishing number of research results has been provided in the area, not only for uniprocessor scheduling but also for multiprocessor scheduling, complexity results, and theoretical evaluation methods. Mixed-criticality is also linked to multiple other research topics. A comprehensive survey of the state-of-the-art for mixed-criticality was provided by Burns and Davis [BD18].

However, the research for mixed-criticality systems has been criticised since it does not match the expectations of system engineers. Specifically, it has been pointed out by Esper et al. [ENN+15] in "How realistic is the mixed-criticality real-time system model?" that there is a clear mismatch between industrial standards and academic work regarding the interpretation of key concepts like system criticality and importance of tasks. In "Mixed Criticality Systems - A History of Misconceptions?", Ernst and Di Natale [EN16] detail 5 assumptions of the model introduced by Vestal [Ves07] which they see as unrealistic and which in most cases are similar to concerns raised by Esper et al. [ENN+15]:

1. The criticality level is applied to a task rather than to a system level function, which they deem acceptable but not precise since tasks can be part of multiple functions.
2. They question that tasks of higher criticality have multiple WCET estimates, since it not clear why a certification authority would accept multiple values that are achieved by different measuring processes. Instead, mechanisms like timing isolation are needed to avoid failure propagation.
3. Mixed-criticality scheduling approaches rely on a significant difference between the WCET that can be utilized after a mode switch.
4. A violation of timing assumptions like the WCET is not necessarily a criticality mode change.
5. Low-criticality tasks can be dropped when high-criticality tasks use more than their low-criticality WCET, since only non-critical tasks can safely be dropped while low-criticality tasks are still critical, which means that dropping them is not graceful degradation.

Ernst and Di Natale [EN16] also state that techniques that try to provide some timing guarantees by reducing the priorities of low-criticality tasks [BB11a; HGS+14], reducing their computation time [BB13], or reducing their periods or deadlines [SZ13; JZP03] are "hardly acceptable in practice".

The reader is also referred to Section 6 in the survey by Burns and Davis [BD18] where system issues like isolation are addressed while more realistic mixed-criticality models are considered in Section 5 in [BD18].

3.7 PROBABILISTIC RESPONSE TIME ANALYSIS AND SCHEDULABILITY TESTS

Probabilistic schedulability analysis examines the situation where one or multiple task parameters are not given by a specific value but described by random

variables. Examples are a probabilistic inter-arrival time, resulting in sporadic behaviour, or a probabilistic WCET, which may stem from, for instance, software-based fault recovery or mixed-criticality systems. The probabilistic parameters may either be described by a set of possible values with related probability or by a continuous distribution. The focus here is on analysis assuming a set of possible WCET values. In the following, independent probabilistic random variables are considered. A survey of probabilistic schedulability analysis has recently been provided by Davis and Cucu-Grosjean [DC19a], covering also a large variety of topics in addition to probabilistic response time analysis. Davis and Cucu-Grosjean [DC19b] also provided a survey on probabilistic timing analysis.

worst-case deadline failure probability
deadline miss rate

Two important metrics are examined in the literature to quantify the probabilistic behaviour of real-time systems with respect to timeliness: the *worst-case deadline failure probability* of an individual job of a task and the *deadline miss rate* of a task, were the former is the maximum probability that a job of task misses its deadline and the latter is the average probability for all jobs of the task. Note that different publications use different terms here. Most notably, the survey by Davis and Cucu-Grosjean [DC19a] used the term *deadline miss probability* instead of *deadline miss rate*, defining it as the average probability that a job of a periodic task misses its deadline when examining over one hyperperiod. However, the term *deadline miss probability*, and the related abbreviation DMP, is also used to describe the *worst-case deadline failure probability* in multiple publications, e.g., by Maxim and Cucu-Grosjean [MC13] and by Chen et al. [CC17; CUB+19]. Hence, the terms *worst-case deadline failure probability* and *deadline miss rate* are used to avoid possible ambiguity.

For periodic real-time task systems, Diaz et al. [DGK+02] provided a framework to calculate the *deadline miss rate* based on convolution. Tanasa et al. [TBE+15] allowed to approximate any arbitrary execution time distribution based on the Weierstrass Approximation. They applied a customized decomposition procedure to search all the possible combinations, in which the decomposition results in a list with $O(4^{|J|})$ elements where $|J|$ is the number of jobs in the interval of interest. Both results have an exponential-time complexity with respect to the number of jobs in the interval of interest, and thus both suffer from a limited scalability, i.e., the experimental results in [DGK+02] and [TBE+15] considered 7 and 25 jobs in the hyper-period, respectively.

Considering sporadic real-time task systems under non-preemptive static-priority scheduling, Axer et al. [AE13] proposed to evaluate the response-time distribution and iterated over the activations of job releases. Maxim and Cucu-Grosjean [MC13] assumed probabilistic minimum inter-arrival as well as probabilistic worst-case execution times and provided a precise probabilistic response time analysis for static-priority scheduling policy. Their approach was extended by Ben-Amor et al. [BAMCG16] to tasks with precedence constraints. All these approaches consider the jobs in the interval of interest in increasing order of the arrival time and convolve the related probability distributions. Therefore, they are also heavily dependent on the number of jobs in the interval of interest. Approximation techniques can be utilized to provide an upper bound on the probability. For example, re-sampling [MC13; RH10] and dynamic-programming based on user-defined granularity can be applied to reduce the time complexity.

As an alternative to job-level convolution, analytical bounds have been proposed recently. Chen and Chen [CC17] provided a scalable approximation based on the Chernoff bounds, which is applicable for 20 tasks and more than thousand jobs in the hyper-period. Chen et al. [CUB+19] also proposed an optimization technique which results in tighter results due to an optimized runtime. Hoeffding's and Bernstein's inequalities were utilized by von der Brüggen et al. *[BPC+18]*.[8] Note that the analytical bounds have a better runtime but do not provide tight results but over-approximations.

deadline miss rate

Chen et al. [CBC18a] proved that the *deadline miss rate* may be significantly larger than the *worst-case deadline failure probability* and derived an analytic upper bound for the deadline miss rate, using the calculation of the *worst-case deadline failure probability* as a subroutine in the analysis.

3.8 SELF-SUSPENSION

The examination of self-suspending tasks in real-time systems can be traced back to a work by Rajkumar in 1991 [Raj91]. Research on self-suspension usually considers either the segmented self-suspension model or the dynamic self-suspension model and this section is arranged accordingly. An exception is the DAG self-suspension model proposed by Bletsas' in his dissertation [Ble07] that, to the best of our knowledge, has never been applied in the literature. Unfortunately, a large number of research results have recently been reported flawed by Chen et al. [CNH+19]. The authors list 6 categories of flaws and over 20 affected publications. This Section is restricted to publications that are not listed as flawed in [CNH+19]. A survey on self-suspension has been provided by Chen et al. [CBH+17b].

speedup factor

Chen et al [CHH+19] provided *speedup factor* analysis for the segmented, dynamic, and hybrid self-suspension model for frame-based task systems, i.e., all tasks have the same release time and deadline, considering both uniprocessor and homogeneous multiprocessor systems.

3.8.1 SEGMENTED SELF-SUSPENSION MODEL

segmented self-suspension

In the *segmented self-suspension* task model a task is described by an interleaving execution/suspension pattern. As shown by Ridouard et al [RRC04], the scheduler design problem is $\mathcal{NP}$-hard in the strong sense, even for one self-suspension segment. Chen [Che16a] and Mohaqeqi et al. [MEY16] showed that verifying the schedulability under static-priority scheduling is co$\mathcal{NP}$-hard in the strong sense. This result also holds when the task under analysis is the only task with suspension behaviour while all higher-priority tasks are sporadic.

Recently, Chen et al [CHH+19] pointed out the link of segmented self-suspension to the *master-slave* problem which is examined in the operations research community. They surveyed the related complexity results, most notably that Yu

8 While most results from this work are presented in Section 5.4, the results for Hoeffding's and Bernstein's inequalities are not presented here but are part of the dissertation of Kuan-Hsun Chen.

et al. [YHL04] showed that the scheduler design problem for segmented self-suspension tasks is $\mathcal{NP}$-hard in the strong sense for a very simple setting, i.e., one-segmented self-suspension tasks where all tasks are released at the same time, have the same absolute deadline, and all segments of all tasks have the same computation time. Hence, the computational complexity of the scheduler design problem is not due to the recurrence of real-time jobs or due to the execution time of the segments but a direct result of the suspension behaviour.

A *period enforcer* algorithm to handle the impact of self-suspensions has been proposed by Rajkumar [Raj91]. It can be applied to segmented self-suspension tasks with multiple computation segments but Chen and Brandenburg [CB17] showed that it can be a cause of deadline misses for otherwise schedulable self-suspending tasks sets. They also pointed out that a schedulability test for the period enforcer is unknown.

For static-priority scheduling, Nelissen et al. [NFR+15] provided a sufficient schedulability test that requires exponential-time complexity, even when the tasks in the system only have one self-suspension interval, by transforming higher-priority tasks into periodic tasks with jitter. Note that the original work in [NFR+15] was flawed but later corrected by the authors in [NFR+17]. Schönberger et al. [SHB+18] used Audsley's Algorithm [Aud91], combining suspension as computation and restarting inference in the schedulability test.

Chen and Liu [CL14] proposed a release time enforcement called fixed-relative-deadline (FRD) under EDF and the equal deadline assignment strategy that has a *speedup factor* of 3 for one-segmented self-suspension. The fixed-relative-deadline approach was applied to static-priority scheduling by Huang and Chen [HC16].

speedup factor

3.8.2 DYNAMIC SELF-SUSPENSION MODEL

dynamic self-suspension

In the *dynamic self-suspension* model a task is described by the worst-case execution time and the maximum suspension time. It assumes no restriction on the number of suspension intervals, i.e., a task can suspend at any time as long as the maximum suspension time condition is not violated. For the dynamic self-suspension model the complexity of the scheduler design problem remains an open problem [CBH+17b]. If the suspension time cannot be reduced by speeding up, Chen [Che16a] showed that the speedup factor for the dynamic self-suspension model is unbounded for EDF, least-laxity-first, and earliest-deadline-zero-laxity.

For static-priority scheduling, Chen et al [CNH16] provided a unifying response time analysis framework, considering multiple approaches to model the self-suspension time under a dynamic self-suspension behaviour, i.e., as computation, carry-in, blocking, or jitter. Huang et al. [HCZ+15] provided PASS-OPA using Audsley's Algorithm [Aud91] for a static-priority assignment.

3.9 MULTIPROCESSOR RESOURCE SHARING

If not treated carefully, priority inversion resulting from resource access may lead to an unnecessary long blocking time for high-priority tasks or a system deadlock.

In uniprocessor systems, mutual exclusion and resource synchronization is usually provided based on longstanding *priority inheritance* techniques. Under static-priority scheduling, such techniques were provided by Sha et. al [SRL90] in the Priority Inheritance Protocol (PIP) and the *Priority Ceiling Protocol* (PCP). The latter ensures the minimum possible number of *pi-blocking*, i.e., 1, and prevents deadlocks. Baker [Bak91] introduced the *Stack Resource Policy* (SRP) that provides similar guarantees and is applicable for dynamic-priority scheduling. SRP is typically preferred to PCP, since it requires a smaller number of context switches and permits execution on a single stack.

Priority Ceiling Protocol
pi-blocking
Stack Resource Policy

Multiprocessor real-time locking protocols can be classified into suspension-based protocols [RSL88; Raj90; BA10; Bra14b] and spin-based protocols [GLN01; BW13; WB13b]. The Flexible Multiprocessor Locking Protocol (FMLP) [BLB+07] considers both, depending on the length of the critical section.

Brandenburg and Anderson [BA10] proved that $\Omega(m)$ *pi-blocking* is unavoidable under suspension-oblivious schedulability analysis. FMLP [BLB+07], $O(m)$ multiprocessor locking protocol (OMLP) [BA13], the Generalized FIFO Multiprocessor Locking Protocol (FMLP$^+$) [Bra14b], and the Distributed FIFO Locking Protocol (DFLP) [Bra14a] are asymptotically optimal for minimizing the pi-blocking when using FIFO-waiting queues. Yang et al. [YWB15] have summarized and compared the protocols that can be utilized under global scheduling. Their empirical results show that asymptotically optimal protocols do not necessarily perform well. In their evaluation, the FMLP and the Priority Inheritance Protocol (PIP) perform best among the existing protocols under global rate-monotonic scheduling when the linear-programming (LP) based schedulability tests in [Bra13] are used.

Several real-time locking protocols have been proposed for partitioned and semi-partitioned scheduling, among them the Distributed PCP (DPCP) [RSL88], the Multiprocessor PCP (MPCP) [Raj90], the Multiprocessor Stack Resource Policy (MSRP) [GLN01], the Flexible Multiprocessor Locking Protocol (FMLP) [BLB+07], and the Multiprocessor resource sharing Protocol (MrsP) [BW13]. The performance of these protocols highly depends on the task partition. Lakshmanan et al. [LNR09] proposed a synchronization-aware partitioned heuristic for MPCP, organizing the tasks that share resources into groups and attempting to assign each group of tasks to the same processor. Nemati et al. [NNB10] presented a blocking-aware partitioning method to split a task group that could not be assigned to one processor, an approach that was later extended by Hsiu et al. [HLK11] such that each resource request can be blocked by at most one lower-priority request. For the MSRP protocol, a Greedy Slacker (GS) algorithm using *integer linear programming* was proposed by Wieder and Brandenburg [WB13a] for the task partition. In their approach called *resource-oriented partitioned scheduling*, Huang et al. [HYC16] proposed to change the view angle and to partition the shared resources first to so-called synchronization processors where all resource access is handled. Afterwards, the non-critical sections are partitioned.

resource-oriented partitioned scheduling

Recently, Chen et al. [CBS+18] introduced the Dependency Graph Approach. The idea is to preconstruct the order in which the critical sessions are accessed as a directed acyclic graph, and afterwards schedule the tasks using any DAG scheduling policy. While the approach in [CBS+18] was restricted to frame-based tasks, the approach was extended to periodic tasks by Shi et al. [SUB+19].

speedup factor

Andersson and Easwaran [AE10] provided *gEDF-vpr*, the first multiprocessor resource sharing protocol with a bounded *speedup factor* of $12(1 + 3r/4m)$. This bound was improved in the *LP-EE-vpr* algorithm by Andersson and Raravi [AR14] that has a speedup factor of $4 \cdot (1 + max_r \cdot \lceil \frac{r \cdot max_r}{m} \rceil) \geq 8$ where max_r is the maximum number of resources a task requests. The resource-oriented partitioned scheduling introduced by Huang et al. [HYC16] has a speedup factor of $11 - \frac{6}{m+1}$ when PCP is utilized on the synchronization processors. These speedup factors (besides for *LP-EE-vpr*) are only valid when tasks have at most one critical section.

Chen et al. [CBS+18] proved that for multiprocessor resource sharing the decision whether all tasks meet one common deadline is $\mathcal{NP}$-hard in the strong sense, independent from the number of processors. They also concluded that allowing migration or preemption does not reduce the computational complexity.

The performance of multiprocessor resource-sharing protocols highly depends on the considered system. Hence, contrary to the uniprocessor case, no single *best* protocol is known and it is unlikely that one can be established. However, it seems important to develop criteria to determine which protocol should be applied, depending on the system architecture as well as the structure of the task set. However, there is only limited work in this direction, even though resource sharing is often a bottleneck for the system performance. Some explorations in which situations suspension-based protocols and in which situations spin-based protocols are preferable were presented. Gai et al. [GNL+03] compared MPCP [Raj90] and MSRP [GLN01] and determined that MSRP performs better when critical sections are short and access to local resources dominates. Brandenburg et al. [BCB+08] evaluated FMLP [BLB+07], showing that the suspension-based approach only very rarely was superior to the spin-based approach. However, to the best of our knowledge, a general evaluation of suspension vs. spinning has not been provided. Furthermore, how tasks should be partitioned and prioritized to achieve a good performance is an open questions for many resource-sharing protocols.

3.10 CONNECTION TO SUBSEQUENT CHAPTERS

Since the related work covers a large area of real-time system research, a brief summary of how the related work connects to the following chapters is provided.

The examination of speedup factors and utilization bounds in Chapter 4 is primarily related to preemptive, non-preemptive, and limited-preemptive uniprocessor scheduling. Section 4.2 considers automotive task systems and rate-dependent tasks. Section 4.5 also shows examples based on self-suspension and multiprocessor resource synchronization.

For uncertain execution environments a novel system model that is related to mixed-criticality systems is provided in Chapter 5. After Section 5.2 considers preemptive uniprocessor scheduling, preemptive partitioned and semi-partitioned multiprocessor scheduling is examined in Section 5.3. Probabilistic response-time analysis is the focus of Section 5.4.

The main scope of Chapter 6 is self-suspending task systems. While Section 6.1 and Section 6.3 consider uniprocessor scheduling, Section 6.2 examines

partitioned multiprocessor scheduling and multiprocessor resource sharing. Furthermore, Section 6.3 has a focus on modelling of real-time systems.

4

SPEEDUP FACTORS AND PARAMETRIC UTILIZATION BOUNDS

In this chapter, we take a careful look at theoretical methods to compare scheduling algorithms and schedulability tests. We focus on *utilization bounds*, *speedup factors*, and *capacity augmentation bounds* to compare these algorithms, since they are widely adopted and accepted as the *de facto* standard theoretical tools for assessing scheduling algorithms and schedulability tests in the real-time research community. Nevertheless, it is not always clear how researchers and designers should view or use such theoretical results. We point out a number of surprising results that show how these metrics can be misinterpreted or misunderstood. Therefore, we aim to provide a perspective on the use of these metrics, guide researchers on their meaning and interpretation, and help avoid common pitfalls.

utilization bound
speedup factor
capacity augmentation bound

We first consider *parametric utilization bounds* for *non-preemptive Rate Monotonic* scheduling (RM-NP) as well as for automotive task sets. In Section 4.1, we examine the utilization bound of RM-NP, showing that depending on the blocking factor γ, which is the relation between the blocking time of a task and its execution time, the utilization bound can still be up to 69.3%, the same as the *Liu and Layland Bound* for preemptive RM (RM-P) [LL73], if γ is sufficiently small. In Section 4.2 we focus on *automotive task sets* where the periods of tasks are limited to $\{1, 2, 5, 10, 20, 50, 100, 200, 1000\}$ ms, i.e., the periods are *semi-harmonic*, providing tailor-made utilization based schedulability tests. We show that the utilization bound of RM-P is $90\% + x$, where x depends on the total utilization of tasks in the set with certain periods. Empirically, in realistic settings the acceptance ratio is still 100% for task sets with total utilization very close to 100%. We recap our findings in Section 4.3, observing that for both analyzed cases parametric bounds, which consider further parameters in addition to the total utilization, can be substantially superior to classical utilization bounds, which do not consider additional parameters.

parametric utilization bound
non-preemptive scheduling
Rate Monotonic
Liu and Layland Bound
automotive task set
semi-harmonic task set

We move our focus to *speedup factors* in Section 4.4, where we show that linear time tests for *Deadline Monotonic* (DM) scheduling are *speedup-optimal* for both preemptive and non-preemptive scheduling of implicit-, constrained-, and arbitrary-deadline task sets. Since DM is not an optimal scheduling algorithm for arbitrary-deadline task sets and the linear-time tests result in the same speedup factor as the exact tests, this raises the questions whether and in which situation speedup factors are a reasonable tool to compare the performance of scheduling algorithms and schedulability tests. Hence, Section 4.5 provides perspective on how to understand and utilize speedup factors, pointing out several misinterpretations or misunderstandings from the real-time systems literature. Resulting from these findings, in Section 4.6 we propose using *parametric augmentation functions*, which means describing theoretical comparisons not with a single value but based on a vector of values that detail the augmentation function based on

speedup factor
Deadline Monotonic
speedup-optimal
parametric augmentation function

the vector. An example on how an examination based on additional parameters is performed and results in a parametric augmentation function is provided in Section 4.7. We summarize these findings and draw conclusions in Section 4.8.

4.1 PARAMETRIC UTILIZATION BOUNDS FOR NON-PREEMPTIVE SCHEDULING

preemptive scheduling

Allowing *preemptions* is often considered an important feature to increase the schedulability, as it enables the scheduler to allocate the processor to high priority tasks almost immediately. This ensures that these tasks are able to meet their deadlines, which may be impossible if they experience long *blocking time* from the execution of lower priority tasks in an non-preemptive manner.

blocking time

non-preemptive scheduling

WCET analysis

However, *non-preemptive scheduling* also has advantages compared to preemptive scheduling. First, calculating a *good upper bound on the WCET* is more difficult if preemption is allowed. The reason is that preemption introduces additional overhead, e.g., for suspending the task, inserting it into the ready queue, flushing the processor pipeline, and dispatching the newly incoming task. This overhead has to be taken into account when analyzing the schedulability, but it is not easy to estimate these costs without getting very pessimistic, e.g., assuming a large number of preemptions at worst-case points. Second, during the execution of a task, preemption may even be impossible in some situations, for instance, due to I/O operations or access to shared resources. Third, ensuring mutual exclusion of shared resources introduces additional overhead to preemptive systems, e.g., to prevent deadlocks and chained blocking, while mutual exclusion is trivial in non-preemptive uniprocessor systems. Context switches also destroy the program locality, making it hard to analyse the effectiveness of caches, as context switches may cause additional cache misses. The number of additional cache misses depends on the number of preemptions a task experiences and the specific point those context switches occur [AG08], making it even harder to calculate those cache related costs, resulting in even more pessimistic bounds for the WCET. Note that cache-related preemption delays are a possibility to tackle this problem. A related survey is provided by Altmeyer and Maiza [AM11]. Nevertheless, the WCET under preemptive scheduling is (sometimes substantially) larger than under non-preemptive scheduling since the preemption overheads cannot be neglected and are, therefore, usually included into the WCETs of the tasks under preemptive scheduling.

limited-preemptive scheduling

preemption threshold

co-operative scheduling

deferred preemption

To combine the advantages of preemptive and non-preemptive scheduling, *limited-preemptive scheduling* techniques have been introduced, e.g., *preemption thresholds* by Wang and Saksena [WS99], *co-operative scheduling* by Burns [Bur94], and *deferred preemption* [BLV07]. For a given task, the analysis under limited-preemptive scheduling is similar to the analysis under non-preemptive scheduling but with a reduced blocking time due to lower-priority tasks.

non-preemptive scheduling

While the utilization bound of RM-NP is 0 [NBF+14], the advantages of *non-preemptive scheduling* over preemptive scheduling and the possibility to reduce the disadvantages by using limited-preemptive techniques motivate the investigation

of non-preemptive scheduling. We introduce the first schedulability test for non-preemptive scheduling in *hyperbolic form*, which is improved afterwards. Moreover, *parametric utilization bounds* for RM-NP are provided that take the blocking factor γ into account, where γ is the relation between the execution time of lower-priority tasks and the task itself. Depending on the value of γ, this significantly improves the bound from 0 to up to 69.3%, the same as for RM-P, if the blocking time is sufficiently small. The results presented in this section appeared in *Schedulability and Optimization Analysis for Non-Preemptive Static Priority Scheduling Based on Task Utilization and Blocking Factors* in ECRTS 2015 *[BCH15]*.

hyperbolic form

parametric utilization bound

4.1.1 HYPERBOLIC SCHEDULABILITY TEST

The *Time Demand Analysis* (TDA) [LSD89] in Eq. (2.3), extended to the non-preemptive case by including the maximum blocking time in Eq. (2.8), is a sufficient schedulability test for a constrained-deadline or implicit-deadline task τ_k [Bur94; DGC10]. We restate the equation, using the upper bound B_k on the *maximum blocking time* instead of B_k^*:

Time Demand Analysis

maximum blocking time

$$\exists t \text{ with } 0 < t \leq D_k \quad \text{and} \quad B_k + C_k + \sum_{\tau_i \in hp(\tau_k)} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t \tag{4.1}$$

To simplify Eq. (4.1), we partition $hp(\tau_k)$ into two disjunct subsets:

- $hp_1(\tau_k)$ consists of all tasks $\tau_i \in hp(\tau_k)$ with $T_i < D_k$
- $hp_2(\tau_k)$ consists of all tasks $\tau_i \in hp(\tau_k)$ with $T_i \geq D_k$

Since in Eq. (4.1) the tasks in $hp_2(\tau_k)$, task τ_k itself, and the blocking time B_k contribute to the workload in the interval $[0, D_k]$ exactly once, we directly include them into the tasks execution time as

$$\widehat{C_k} = B_k + C_k + \sum_{\tau_i \in hp_2(\tau_k)} C_i \tag{4.2}$$

Now, Eq. (4.1) can be rewritten as

$$\exists t \text{ with } 0 < t \leq D_k \quad \text{and} \quad \widehat{C_k} + \sum_{\tau_i \in hp_1(\tau_k)} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t \tag{4.3}$$

For brevity of notation, we *abuse* k by resetting it to $k = |hp_1(\tau_k)| + 1$, where $|hp_1(\tau_k)|$ is the cardinality of $hp_1(\tau_k)$. Introducing another variable here, say k^*, would be precise. However, since all tasks in $\tau_i \in hp_2(\tau_k)$ are summed up in $\widehat{C_k}$, we only have to consider the $\tau_i \in hp_1(\tau_k)$, and such a notation has no value for the analysis but makes the following argumentation more difficult to read.

The sufficient schedulability test in Eq. (4.3) has a pseudo-polynomial runtime, since it has to test all time points where a job of a higher priority task arrives. If only a subset of these points is tested, the test becomes more pessimistic. However, to get to a hyperbolic form, we only test k time points $\{t_1, \ldots, t_{k-1}, t_k\}$, namely the last arrival points of higher priority tasks and the absolute deadline of τ_k:

$$t_i = \left\lfloor \frac{D_k}{T_i} \right\rfloor T_i \; \forall \; \tau_i \in hp_1(\tau_k) \quad \text{and} \quad t_k = D_k \tag{4.4}$$

Assuming that the schedulability of $\{\tau_1, \ldots, \tau_{k-1}\}$ has been ensured beforehand, this results in the following sufficient test for the schedulability of τ_k:

$$\exists t_j \in \{t_1, \ldots, t_k\} \quad \text{with} \quad \widehat{C_k} + \sum_{i=1}^{k-1} \left\lceil \frac{t_j}{T_i} \right\rceil C_i \leq t_j \tag{4.5}$$

Since the workload of the higher-priority tasks is independent from the actual priority order, we can reorder the tasks in $hp_1(\tau_k)$ according to their last release times, i.e., $t_1 \leq t_2 \leq \ldots \leq t_{k-1} \leq t_k$. When evaluating time t_i, we can remove the ceiling function for τ_i since $\left\lceil \frac{t_i}{T_i} \right\rceil$ is always an integer. Furthermore, as t_j is the last release time for a job of τ_j before D_k and $t_j < D_k$ for all tasks $\tau_j \in \{\tau_1, \ldots, \tau_k\}$ we know that

$$\frac{t_i}{T_i} + 1 \geq \left\lceil \frac{t_j}{T_i} \right\rceil \quad \forall \tau_i, \tau_j \in \{\tau_1, \ldots, \tau_k\} \tag{4.6}$$

Considering time $t_j \in \{t_1, \ldots, t_k\}$ after reordering according to the time of the last release, the last release of $\tau_i \in hp(\tau_k)$ only happens before t_j if $i < j$. If $t_j > t_i$, then $\left\lceil \frac{t_j}{T_i} \right\rceil = \left\lceil \frac{t_i}{T_i} \right\rceil + 1 = \frac{t_i}{T_i} + 1$ since t_i is the last release of τ_i before D_k. If $t_j \leq t_i$, we get $\left\lceil \frac{t_j}{T_i} \right\rceil \leq \left\lceil \frac{t_i}{T_i} \right\rceil = \frac{t_i}{T_i}$.

Hence, for each time t_j the summation in Eq. (4.5) can be split into two parts as

$$\widehat{C_k} + \sum_{i=1}^{k-1} \left\lceil \frac{t_j}{T_i} \right\rceil C_i \leq \widehat{C_k} + \sum_{i=1}^{k-1} \frac{t_i}{T_i} C_i + \sum_{i=1}^{j-1} C_i \tag{4.7}$$

where the first summation represents all jobs of higher priority tasks but the last one, and the second summation represents the last job for the tasks where this last job is already released at t_j.

We unify these considerations in a safe sufficient schedulability test for a task τ_k stated in the the following lemma:

Lemma 4.1. *If the schedulability of all higher priority tasks is ensured already, task τ_k is schedulable by a non-preemptive static-priority scheduling policy if*

$$\exists t_j \in \{t_1, \ldots, t_k\} \quad \textit{such that} \quad \widehat{C_k} + \sum_{i=1}^{k-1} \frac{t_i}{T_i} C_i + \sum_{i=1}^{j-1} C_i \leq t_j \tag{4.8}$$

Proof. That Eq. (4.8) is a sufficient schedulability test follows directly from the argumentation above. □

To achieve a polynomial-time schedulability test in a hyperbolic form, we need a test based on the utilization of the higher-priority tasks and the execution and blocking time of the task that is currently tested. In Eq. (4.8) in Lemma 4.1 the left summation can directly be converted to be utilization-based as $U_i = \frac{C_i}{T_i}$. The right summation in Eq. (4.8) can be easily transformed to be utilization-based:

$$\sum_{i=1}^{j-1} C_i = \sum_{i=1}^{j-1} \frac{T_i}{T_i} C_i = \sum_{i=1}^{j-1} T_i U_i \leq \sum_{i=1}^{j-1} t_i U_i \tag{4.9}$$

where the inequality holds since $t_i = f_i \cdot T_i$ for some $f_i \in \mathbb{Z}^+$, resulting in the following utilization-based schedulability test

$$\exists t_j \in \{t_1, \ldots, t_k\} \quad \text{and} \quad \widehat{C_k} + \sum_{i=1}^{k-1} t_i U_i + \sum_{i=1}^{j-1} t_i U_i \leq t_j \tag{4.10}$$

that is more pessimistic than Lemma 4.1. For non-preemptive scheduling, we must verify the schedulability for each task individually. The reason is that, contrary to the utilization, the blocking time is a monotonically decreasing function with respect to the priority. Eq. (4.10) allows a schedulability test in a *hyperbolic form* for static-priority non-preemptive scheduling that is stated in the following theorem:

hyperbolic form

Theorem 4.2. *A task τ_k in a non-preemptive sporadic task system with constrained deadlines can be feasibly scheduled by a static-priority scheduling algorithm, if the schedulability for all higher priority tasks has already been ensured and the following condition holds:*

$$\left(\frac{\widehat{C_k}}{D_k} + 1\right) \prod_{\tau_j \in hp_1(\tau_k)} (U_j + 1) \leq 2 \tag{4.11}$$

Proof. We prove the theorem by showing that if the condition in Eq. (4.11) is satisfied, the condition in Eq. (4.10) is satisfied as well. We use contrapositive, thus showing that if Eq. (4.10) is not satisfied, Eq. (4.11) is not satisfied as well. The proof uses the same strategy as the proof of Lemma 1 in [CHL15b]. For completeness, we list the corresponding linear programming and the optimal extreme point solution.

If a task τ_k is not schedulable, by Eq. (4.3) we know that

$$\forall t \text{ with } 0 < t \leq D_k : \widehat{C_k} + \sum_{\tau_i \in hp_1(\tau_k)} \left\lceil \frac{t}{T_i} \right\rceil C_i > t$$

This must hold true $\forall\ t \in (0, D_k]$. Hence, it must hold true for times of the last releases of higher priority tasks. All transformations made starting from Eq. (4.3) until we reached Eq. (4.10) only increased the left side of the equations. Therefore, an unschedulable task τ_k will fail Eq. (4.10) as well. Hence, if τ_k is not schedulable

$$\forall\ j \in \{1, \ldots, k-1, k\} \text{ holds } \widehat{C_k} + \sum_{i=1}^{k-1} t_i U_i + \sum_{i=1}^{j-1} t_i U_i > t_j \tag{4.12}$$

Since Eq. (4.10) is only a sufficient scheduling condition, a task set might still be schedulable if Eq. (4.12) holds. However, if a task is not schedulable Eq. (4.12) holds and we prove, that Eq. (4.11) never holds if Eq. (4.12) holds. Thus Eq. (4.11) does not hold for any unschedulable task τ_k. We get the following optimization problem represented by a *linear programming*:

linear programming

$$\inf \quad C_k^* \tag{4.13a}$$

$$\text{s.t} \quad C_k^* + \sum_{i=1}^{k-1} t_i^* U_i + \sum_{i=1}^{j-1} t_i^* U_i > t_j^* \qquad \forall\ 1 \leq j \leq k \tag{4.13b}$$

$$t_j^* \geq 0 \qquad \forall\ 1 \leq j \leq k \tag{4.13c}$$

where $t_1^*, \ldots, t_{k-1}^*$ and C_k^* are variables and t_k^* is defined as t_k for notational brevity. We replace $>$ with $\geq$ in (4.13b) as infimum and minimum are the same if $\geq$ is used. Thus Eq. (4.12) holds $\forall\ C_k > C_k^*$.

For $j = k$, Eq. (4.13b) leads to $C_k^* \geq t_k^* - \left(\sum_{i=1}^{k-1} t_i^* U_i + \sum_{i=1}^{k-1} t_i^* U_i\right)$. Based on this inequality, C_k^* is replaced in Eq. (4.13a) and Eq. (4.13b). As for t_k^* the two summations are the same, we get $t_k^* - 2\sum_{i=1}^{k-1} t_i^* U_i$ in Eq. (4.13a). Since t_k^* is a constant, we have to maximize $\sum_{i=1}^{k-1} t_i^* U_i$ to find a minimum value for C_k^*. Replacing C_k^* in Eq. (4.13b) results in

$$\begin{aligned} & t_k^* - 2\sum_{i=1}^{k-1} t_i^* U_i + \sum_{i=1}^{k-1} t_i^* U_i + \sum_{i=1}^{j-1} t_i^* U_i \\ & \quad = t_k^* - \sum_{i=j}^{k-1} t_i^* U_i \geq t_j^* \qquad \forall\ 1 \leq j \leq k-1 \end{aligned} \tag{4.14}$$

These reformulations result in the following linear programming:

$$\max \quad \sum_{i=1}^{k-1} t_i^* U_i \tag{4.15a}$$

$$\text{s.t} \quad t_k^* - \sum_{i=j}^{k-1} t_i^* U_i \geq t_j^* \qquad \forall\ 1 \leq j \leq k-1 \tag{4.15b}$$

$$t_j^* \geq 0 \qquad \forall\ 1 \leq j \leq k-1 \tag{4.15c}$$

According to the extreme point theorem [LY15], these conditions form a polyhedron of possible solutions. Furthermore, if the polyhedron is not empty (in which case there is no feasible solution), one of the extreme points of the polyhedron is an optimal solution for the optimization problem. Since $0 \leq t_i^* \leq t_k^* < \infty$ for all $1 \leq j \leq k-1$, the objective function in Eq. (4.15a) is bounded, and the $2(k-1)$ constraints in Eq. (4.15b) and Eq. (4.15c) form a polyhedron of feasible solutions. As there are $k-1$ variables, at least $k-1$ of the constraints in Eq. (4.15b) and Eq. (4.15c) have to be active, which means that $\geq$ holds with $=$ in the solution.

One extreme point solution with $t_j^* > 0, \forall\ 1 \leq j \leq k-1$ can be found by setting $t_j^* = t_k^* - \sum_{i=j}^{k-1} t_i^* U_i$ with

$$t_{i+1}^* - t_i^* = t_i^* U_i \qquad \forall\ 1 \leq j \leq k-1 \tag{4.16}$$

Thus we know

$$\frac{t_{i+1}^*}{t_i^*} = U_i + 1 \qquad \forall\ 1 \leq j \leq k-1 \tag{4.17}$$

and

$$\frac{t_i^*}{t_k^*} = \prod_{j=1}^{k-1} \frac{t_j^*}{t_{j+1}^*} = \frac{1}{\prod_{j=i}^{k-1}(U_j + 1)} \tag{4.18}$$

Based on Eq. (4.13b) with $j = k$ the minimum value of C_k^* is:

$$\begin{aligned} C_k^* &= t_k^* - 2\sum_{i=1}^{k-1} t_i^* U_i \overset{(4.16)}{=} t_k^* - 2(t_k^* - t_1^*) \\ &\overset{(4.18)}{=} t_k^* - 2\left(t_k^* - \frac{t_k^*}{\prod_{j=1}^{k-1}(U_j + 1)}\right) \\ \Rightarrow\ C_k^* &= t_k^*\left(\frac{2}{\prod_{j=1}^{k-1}(U_j + 1)} - 1\right) \end{aligned} \tag{4.19}$$

We now show that no feasible solution results in a smaller value for C_k^* than the one in Eq. (4.19). An extreme point solution for $k-1$ variables must have at least $k-1$ active constraints, i.e., $k-1$ constraints out of the Eqs. (4.15b) and (4.15c) hold with equality.

By assuming that $0 = t_p^* = t_k^* - \sum_{i=p}^{k-1} t_i^* U_i$ for a specific task $\tau_p \in \{\tau_1, \ldots, \tau_{k-1}\}$, we show that in an extreme point solution for each $\tau_j \in \{\tau_1, \ldots, \tau_{k-1}\}$ either Eq. (4.15b) or Eq. (4.15c) is active but not both. Let $\tau_q \in \{\tau_{p+1}, \ldots, \tau_{k-1}\}$ be the next task with $t_q^* > 0$ in the extreme point solution, thus $t_p^* = t_{p+1}^* = \ldots = t_{q-1}^* = 0$. If no such task τ_q exists, we set $q = k^*$ and $t_q^* = t_k^*$. With these two conditions, we get the contradiction that $0 = t_p^* = t_k^* - \sum_{i=p}^{k-1} t_i^* U_i = \sum_{i=q}^{k-1} t_i^* U_i \geq t_q^* > 0$ for $q \leq k-1$ and $0 = t_p^* = t_k^* - \sum_{i=p}^{k-1} t_i^* U_i = t_k^* > 0$ if $q = k$. Hence, for a feasible solution of Eq. (4.15) we can partition $\{\tau_1, \ldots, \tau_{k-1}\}$ into two tasks sets $\mathbf{T}_1$ and $\mathbf{T}_2$, where $\tau_j \in \mathbf{T}_1$ if $t_j^* = 0$ (Eq. (4.15c) holds) and $\tau_j \in \mathbf{T}_2$ if $t_j^* = t_k^* - \sum_{i=j}^{k-1} t_i^* U_i > 0$ (Eq. (4.15b) holds). As we maximize $\sum_{i=1}^{k-1} t_i^* U_i$ we only use the tasks in $\mathbf{T}_2$ and drop all tasks in $\mathbf{T}_1$, leading to the objective function as in Eq. (4.19) where only $\tau_j \in \mathbf{T}_2$ are considered. As $\prod_{\tau_j \in \mathbf{T}_2}(U_j + 1) \leq \prod_{j=1}^{k-1}(U_j + 1)$ we maximize the objective in Eq. (4.15) if all higher priority tasks are in $\mathbf{T}_2$.

Thus we know that Eq. (4.12) always holds if $\widehat{C_k} > C_k^*$ and get

$$\left(\frac{C_k^*}{t_k^*} + 1\right) \prod_{j=1}^{k-1}(U_j + 1) > 2 \tag{4.20}$$

Thus we know τ_k is schedulable if

$$\left(\frac{C_k^*}{t_k^*} + 1\right) \prod_{j=1}^{k-1}(U_j + 1) \leq 2 \tag{4.21}$$

holds and all higher priority tasks are schedulable. We know that $t_k^* = D_k$ and replace C_k^* with $\widehat{C_k}$, as it is constructed as the minimum of the values Eq. (4.12) holds for, therefore reaching the conclusion of Theorem 4.2. □

Instead of minimizing C_k^* to ensure Eq. (4.12) holds, by minimizing $\widehat{C_k} + \sum_{i=1}^{k-1} t_i U_i$ we can also get another a sufficient schedulability test:

Theorem 4.3. *A task τ_k in a non-preemptive sporadic task system with constrained deadlines can be feasibly scheduled by a static-priority scheduling algorithm, if the*

schedulability for all higher priority tasks has already been ensured and the following condition holds:

$$\frac{\widehat{C_k} + \sum_{i=1}^{k-1} t_i U_i}{D_k} \leq \frac{1}{\prod_{\tau_j \in hp_1(\tau_k)} (U_i + 1)} \tag{4.22}$$

Since the proof of Theorem 4.3 is very similar to the proof of Theorem 4.2 it is not shown here but provided in the Appendix.

Observation 4.4. *The schedulability tests in Theorem 4.2 and Theorem 4.3 provide the same result.*

The reason is that the optimization problem for both approaches lead to the same linear programming and, therefore, provide the same solution, i.e, Eq. (4.13b) and Eq. (8.1b) are the same for $j = k$. Since there are no estimations in the following steps, both hold if $\widehat{C_k} \leq C_k^*$ and both will fail for $\widehat{C_k} > C_k^*$.

time complexity

linear time

If for **T** the task order according to the periods and according to the relative deadlines are both given, the schedulability test in Theorem 4.2 can be conducted in *linear time* under RM-NP and DM-NP. The blocking time for all tasks in **T** can be computed in $O(n)$ when the task set is traversed in reversed priority order, i.e., by starting from the lowest priority task. Afterwards, the actual test starts from the highest priority task. We analyze the changes in hp_1 and hp_2 for one step, i.e., from τ_k to τ_{k+1}. No task can ever move from hp_1 to hp_2 since the relative deadline is increasing with the task priority. Assume τ_k is placed in hp_2 for τ_{k+1}. Then all tasks $\tau_i \in hp_2$ with $D_k \leq T_i < D_{k+1}$ are moved to hp_1 which can be determined for each task in $O(1)$. If the tasks in hp_2 are tested in increasing order of their period, we can stop for this step once $T_i \geq D_{k+1}$. Due to the monotonicity of the deadlines in DM, each task is only moved from hp_2 to hp_1 at most once, and for each move $\prod_{\tau_j \in hp_1} (U_j + 1)$ can be computed in $O(1)$. Therefore, the test in Theorem 4.2 has an amortized cost $O(1)$ for each task, i.e., it is considered once in the product of utilizations and moved from hp_2 to hp_1 at most once, resulting in $O(n)$ for testing **T**. If the two orders are not given, they can be determined in $O(n \log n)$. Note that for RM-NP hp_2 is always empty. For the general case this argumentation does not hold, as tasks may be moved from hp_1 to hp_2 as well. Therefore, in the general case the time complexity can be $O(n)$ for each step, resulting in a total complexity of $O(n^2)$. The utilization based tests in hyperbolic form we present subsequently have the same runtime complexity.

The sufficient schedulability test in Lemma 4.1 and, therefore, the resulting hyperbolic tests are pessimistic, as the concept of Lemma 4.1 includes the blocking time due to lower-priority tasks but allows task τ_k to be preempted after it starts. However, for non-preemptive scheduling, we merely have to verify whether a job of task τ_k, arriving at time t, can be started before $t + D_k - C_k$ and higher-priority jobs arriving in $[t + D_k - C_k,\ t + D_k]$ do not have to be considered. Hence, we

can determine the schedulability of a job of τ_k released under the critical instant (at time 0) by testing whether[1]

$$\exists t \in (0,\ D_k - C_k] \quad \text{with} \quad B_k + \sum_{i=1}^{k-1} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t \tag{4.23}$$

thus ensuring enough time for the job of τ_k to start executing.

While testing Eq. (4.23) alone is not safe enough due to the self-pushing phenomenon [BLV09], we adopt Lemma 2.2 from Yao, Buttazzo, and Bertogna [YBB10], which directly results in:

Lemma 4.5. *A task τ_k is schedulable by a static-priority non-preemptive scheduling (FP-NP) algorithm A^{NP}, if all higher-priority tasks are schedulable, and the following two conditions hold:*

1. *the first job of τ_k will be executed before its deadline:*
   $\exists t \in (0,\ D_k - C_k] \quad with \quad B_k + \sum_{\tau_i \in hp(\tau_k)} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t$
2. *the task set is schedulable by A^P, i.e., the preemptive version of A:*
   $\exists t \in (0,\ D_k] \quad with \quad C_k + \sum_{\tau_i \in hp(\tau_k)} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t$

From Lemma 4.5, we to construct a tighter sufficient schedulability test based on two hyperbolic equations. Since this construction is similar to the one in Theorem 4.2, we only state the differences here.

As we examine different time intervals for the preemptive and the non-preemptive test, i.e., $(0,\ D_k]$ and $(0,\ D_k - C_k]$, the sets $hp_1(\tau_k)$ and $hp_2(\tau_k)$ are not necessarily identical for both tests, i.e., a task $\tau_i \in hp(\tau_k)$ with $D_k - C_k \leq T_i < D_k$ is in $hp_1(\tau_k)$ for the preemptive case and in $hp_2(\tau_k)$ for the non-preemptive case. We denote these sets $hp_1^P(\tau_k)$ and $hp_2^P(\tau_k)$ for the preemptive case, and $hp_1^{NP}(\tau_k)$ and $hp_2^{NP}(\tau_k)$ for the non-preemptive case. As the examined deadline differs, the order of the jobs in $hp_1^P(\tau_k)$ and $hp_1^{NP}(\tau_k)$ may differ as well if the last release time t_i of τ_i is in $(D_k - C_k,\ D_k]$. As a result, the permutation of the $\tau_i \in hp_1^P(\tau_k)$ may differ from the permutation of the $\tau_i \in hp_1^{NP}(\tau_k)$ as well since they are permutated according to the last release of τ_i. We denote those last release times t_i^P and t_i^{NP} and the resulting permutations as π_k^P and π_k^{NP}.

Theorem 4.6. *A task τ_k is schedulable by a static-priority non-preemptive scheduling algorithm A^{NP} if all higher priority tasks are schedulable and the following two conditions hold:* *hyperbolic form*

$$\left(\frac{B_k + \sum_{\tau_i \in hp_2^{NP}(\tau_k)} C_i}{D_k - C_k} + 1 \right) \prod_{\tau_j \in hp_1^{NP}(\tau_k)} (U_j + 1) \leq 2 \tag{4.24}$$

$$\left(\frac{C_k + \sum_{\tau_i \in hp_2^{P}(\tau_k)} C_i}{D_k} + 1 \right) \prod_{\tau_j \in hp_1^{P}(\tau_k)} (U_j + 1) \leq 2 \tag{4.25}$$

1 When considering a tight blocking time $B_k = \max_{\tau_i \in lp(\tau_k)}\{C_i - \Delta\}$ with $\Delta > 0$ (instead of a strict upper bound), e.g., [YBB10; BLV09], $\left\lfloor \frac{t}{T_i} \right\rfloor + 1$ (instead of $\left\lceil \frac{t}{T_i} \right\rceil$) must be used in Eq. (4.23). The simplification of setting B_k to $\max_{\tau_i \in lp(\tau_k)}\{C_i\}$ allows us to put $\leq$ instead of $<$ in the condition.

Proof. We need to show that if both conditions hold, Lemma 4.5 holds as well. The proof for Eq. (4.25) is similar to the one of Theorem 4.2 but looking for the smallest C_k instead of the smallest $\widehat{C_k}$.

When considering Eq. (4.24), the sets $hp_1^{NP}(\tau_k)$ and $hp_2^{NP}(\tau_k)$ may differ from $hp_1^{P}(\tau_k)$ and $hp_2^{P}(\tau_k)$, i.e., a tasks τ_i with $D_k - C_k \leq T_i < D_k$ is moved from $hp_1^{P}(\tau_k)$ to $hp_2^{NP}(\tau_k)$, and thus $|hp_1^{NP}(\tau_k)| \leq |hp_1^{P}(\tau_k)|$. The tasks in $hp_1^{NP}(\tau_k)$ will be ordered according to

$$t_i^{NP} = \left\lfloor \frac{D_k - C_k}{T_i} \right\rfloor T_i \; \forall \tau_i \in \; hp_1^{NP}(\tau_k) \;\text{ and }\; t_k = D_k \tag{4.26}$$

We are again looking for the smallest $B_k^* > B_k' = B_k + \sum_{\tau_i \in hp_1^{NP}(\tau_k)} C_i$ to ensure that τ_k cannot start by considering an optimization problem. If we replace $>$ with $\geq$ to look for the minimum instead of the infimum, the conditions are the same as the conditions in Eq. (4.13b) with B_k^* instead of C_k^*. With $B_k^* \geq t_k^* - \sum_{i=1}^{k-1} t_i^* U_i - \sum_{i=1}^{k-1} t_i^* U_i$ we get a maximization problem of the $\sum_{i=1}^{k-1} t_i^* U_i$ with the same constraints as in Eq. (4.13), and therefore getting the same solution for B_k^*. □

We now show an interesting property of the blocking time, namely that under certain conditions and if the blocking time is not too long, the blocking time has no impact on the schedulability test in Lemma 4.5.

Theorem 4.7. *The schedulability of task τ_k under a non-preemptive static-priority scheduling A^{NP} solely depends on the schedulability of τ_k under its preemptive version A^P if the following two conditions hold:*

$$T_i \;\leq\; D_k \;-\; C_k \qquad \forall \tau_i \in hp(\tau_k) \tag{4.27a}$$

$$B_k \;\leq\; \left(1 - \frac{C_k}{D_k}\right) C_k \tag{4.27b}$$

Proof. We need to show that under the given assumptions, if Eq. (4.25) holds, Eq. (4.24) holds as well. As $T_i \leq D_k - C_k$ for all $\tau_i \in hp(\tau_k)$, we know, that $hp_2^P(\tau_k)$ and $hp_2^{NP}(\tau_k)$ are both empty, and therefore $hp_1^P(\tau_k) = hp_1^{NP}(\tau_k) = hp(\tau_k)$. As a result, we know $\prod_{\tau_j \in hp_1^N(\tau_k)} (U_j + 1)$ and $\prod_{\tau_j \in hp_1^{NP}(\tau_k)} (U_j + 1)$ are the same. We know $\frac{B_k}{D_k - C_k} \leq \frac{C_k}{D_k}$ by Eq. (4.27b). Therefore, the success of Eq. (4.25) implies the success of Eq. (4.24). □

limited-preemptive scheduling

The proposed schedulability tests can easily be extended to *limited-preemptive scheduling*, as long as the pseudo-polynomial time schedulability tests have a similar form as the ones presented here. For instance, for the model in [YBB10], a strict upper bound of the blocking time for τ_k can be computed as

$$B_k \;=\; \max_{\tau_i \,\in\, lp(\tau_k)} \Big\{ \max_{j \,\in\, np(\tau_i)} \{C_{i,j}\} \Big\} \tag{4.28}$$

where $np(\tau_i)$ is the number of non-preemptive regions in τ_i and $C_{i,j}$ is the WCET of the j-th non-preemptive region of τ_i. With this upper-bounded blocking time,

the TDA-based schedulability test for task τ_k in [Bur94] can be revised by using the definition of B_k in Eq. (4.28) to replace B_k^* in Eq. (2.7) or B_k in Eq. (2.10). Hence, the tests in Theorem 4.2 and Theorem 4.3 can be applied for limited preemption.

A tighter schedulability test for tasks that end with a final non-preemptive interval has been proposed by Yao et al. [YBB10]. Let $C_{k,f}$ be the length of this final non-preemptive section of τ_k and let $C_{k,s} = C_k - C_{k,f}$ be the WCET of τ_k without the final section. We need to ensure that the upper-bounded blocking time, all higher priority tasks $\tau_i \in hp_2^{NP}(\tau_k)$, and the part of τ_k represented by $C_{k,s}$ can be executed before the last non-preemptive section of τ_k. Hence, similar to Theorem 2 in [YBB10], the first condition in Lemma 4.5 is changed to verify whether there exists $t \in (0, D_k - C_{k,f}]$ with $B_k + C_{k,s} + \sum_{\tau_i \in hp(\tau_k)} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t$. Thus we can reformulate Eq. (4.24) in Theorem 4.6 as

$$\left(\frac{B_k + C_{k,s} + \sum\limits_{\tau_i \in hp_2^{NP}(\tau_k)} C_i}{D_k - C_{k,f}} + 1 \right) \prod_{\tau_j \in hp_1^{NP}(\tau_k)} (U_j + 1) \leq 2 \tag{4.29}$$

if $hp_2^{NP}(\tau_k)$ and $hp_1^{NP}(\tau_k)$ are constructed accordingly.

4.1.2 PARAMETRIC UTILIZATION BOUND

parametric utilization bound

maximum blocking factor

We now provide the first general *parametric utilization bounds* for implicit-deadline task sets under non-preemptive Rate Monotonic scheduling (RM-NP), where, in addition to the utilization, the *maximum blocking factor* γ of the task set $\mathbf{T}$ is considered. We start with a parametric bound for a specific task τ_k and then extend it to $\mathbf{T}$, both based on Theorem 4.2. Afterwards, we provide improved bounds based on the improved scheduling condition in Theorem 4.6.

Theorem 4.8. *Suppose that the tasks are indexed such that $T_i \leq T_{i+1}$ and that $\gamma_k = \max_{\tau_i \in lp(\tau_k)} \left\{ \frac{C_i}{C_k} \right\} = \frac{B_k}{C_k}$. Task τ_k is schedulable by RM-NP if*

$$U_{sum} \leq \begin{cases} \left(\left(\frac{2}{1+\gamma_k} \right)^{\frac{1}{k}} - \frac{1}{1+\gamma_k} \right) + (k-1) \left(\left(\frac{2}{1+\gamma_k} \right)^{\frac{1}{k}} - 1 \right) & \text{if } \gamma_k \leq 1 \\ \frac{1}{1+\gamma_k} & \text{if } \gamma_k > 1 \end{cases} \tag{4.30}$$

Proof. This has a similar proof as the Liu and Layland bound using the Lagrange Multiplier Method. Details are provided in the Appendix. □

Theorem 4.9. *Suppose that $\gamma = max_{\tau_k \in \mathbf{T}} \{\gamma_k\}$. A task set can be feasibly scheduled by RM-NP if*

$$U_{sum} \leq \begin{cases} \frac{\gamma}{1+\gamma} + ln \left(\frac{2}{1+\gamma} \right) & \text{if } \gamma \leq 1 \\ \frac{1}{1+\gamma} & \text{if } \gamma > 1 \end{cases} \tag{4.31}$$

Proof. This follows directly from Theorem 4.8 by calculating the utilization bound when $k \to \infty$, i.e.,

$$\lim_{k\to\infty}\left(\left(\frac{2}{1+\gamma}\right)^{\frac{1}{k}} - \frac{1}{1+\gamma}\right) + (k-1)\left(\left(\frac{2}{1+\gamma}\right)^{\frac{1}{k}} - 1\right)$$
$$= k\left(\left(\frac{2}{1+\gamma}\right)^{\frac{1}{k}} - 1\right) + \left(1 - \frac{1}{1+\gamma}\right) = ln\left(\frac{2}{1+\gamma}\right) + \frac{\gamma}{\gamma+1}$$

for the cases when $\gamma \leq 1$. For $\gamma > 1$ the result is identical to Theorem 4.8 regardless of k. □

As Theorem 4.6 improves Theorem 4.2, it can also be applied to achieve a tighter parametric utilization bound.

Theorem 4.10. *Suppose that the tasks are indexed in rate monotonic order, i.e., such that* $T_i \leq T_{i+1}$. *If* $\gamma = \max_{\tau_i \in lp(\tau_k)}\left\{\frac{C_i}{C_k}\right\} = \frac{B_k}{C_k} > 0$, *then* τ_k *is schedulable by RM-NP if*

$$\sum_{i=1}^{k} U_i \leq \min\left\{k(2^{\frac{1}{k}} - 1), \frac{1}{1+\gamma}\right\} \tag{4.32}$$

Proof. The utilization bound of Eq. (4.25) for RM-NP is the well-known Liu and Layland bound $k(2^{\frac{1}{k}} - 1)$, as shown in [LL73; BBB01]. We only have to focus on the utilization bound of Eq. (4.24). Since we consider implicit-deadline task sets, we can substitute D_k with T_k. Due to RM-NP, we know $T_i \leq T_k$ for any higher-priority task τ_i, which means $\frac{C_i}{T_k} \leq U_i$. Therefore, a more pessimistic test than Eq. (4.24) is to test whether

$$\left(\frac{\gamma U_k + \sum\limits_{\tau_i \in hp_2^{NP}(\tau_k)} U_i}{1 - U_k} + 1\right) \prod_{\tau_j \in hp_1^{NP}(\tau_k)} (U_j + 1) \leq 2 \tag{4.33}$$

The utilization bound can be proven by finding the infimum $\sum\limits_{i=1}^{k} U_i$ such that Eq. (4.33) does not hold. We first show that the condition in Eq. (4.33) can be simplified. Suppose that $U_k + \sum_{\tau_i \in hp_2^{NP}(\tau_k)} U_i$ is specified, denoted as f. Since $\frac{\gamma U_k + f - U_k}{1 - U_k}$ is maximized when U_k is either 0 or f we only consider these two cases.

We search for the infimum of $\sum\limits_{i=1}^{k} U_i$ such that $\left(\frac{\gamma U_k}{1-U_k} + 1\right) \prod\limits_{j=1}^{k-1}(U_j + 1) > 2$. The detailed proof showing that the infimum happens for one of the boundary values of $U_1 \in [0;\ 2^{\frac{1}{k-1}} - 1]$ is in the Appendix. The utilization bound for $U_1 = 0$ is $\frac{1}{1+\gamma}$. If $U_1 = (2^{\frac{1}{k-1}} - 1)$ the utilization bound is $(k-1) \cdot (2^{\frac{1}{k-1}} - 1) > k(2^{\frac{1}{k}} - 1)$. Therefore, the conclude the Theorem. □

parametric utilization bound

Based on the property shown in Theorem 4.10, we provide an improved *parametric utilization bound* of RM-NP with respect to γ for sporadic real-time tasks with implicit deadlines.

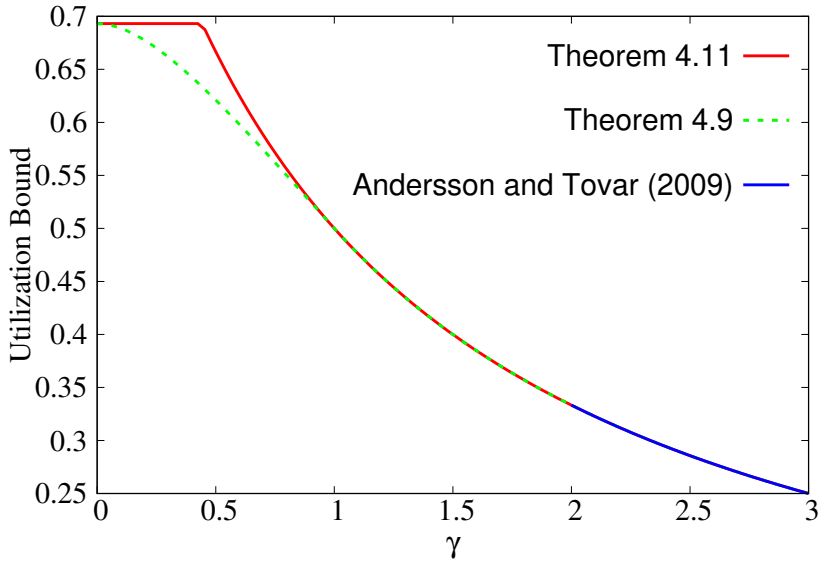


Figure 4.1: Comparison of the parametric utilization bound of RM-NP with respect to $\gamma = \max_{\tau_k \in \mathbf{T}} \left\{ \max_{\tau_i \in lp(\tau_k)} \left\{ \frac{C_i}{C_k} \right\} \right\}$ by Theorem 4.9 and Theorem 4.11 with the result by Andersson and Tovar [AT09] for RM-NP on CAN buses [Bos91]. *Adapted from [BCH15].*

Theorem 4.11. *For* $\gamma = \max_{\tau_k \in \mathbf{T}} \left\{ \max_{\tau_i \in lp(\tau_k)} \left\{ \frac{C_i}{C_k} \right\} \right\}$*, a task set can be feasibly scheduled by RM-NP if*

$$U_{sum} \leq \begin{cases} ln(2) \approx 0.693 & \text{if } \gamma \leq \frac{1-ln(2)}{ln(2)} \\ \frac{1}{1+\gamma} & \text{if } \gamma > \frac{1-ln(2)}{ln(2)} \end{cases} \tag{4.34}$$

Proof. This follows directly from Theorem 4.10 by calculating the utilization bound when $k \to \infty$. □

The result in Theorem 4.11 further improves the result in Theorem 4.9. With the analysis in Theorem 4.11, we conclude that the utilization bound of RM-NP with respect to γ can still be up to 69.3%, if $\gamma \leq \frac{1-ln(2)}{ln(2)} \approx 0.44269$. We illustrate the results of Theorems 4.9 and Theorem 4.11 in Figure 4.1. We also show the result by Andersson and Tovar [AT09] who provided a utilization bound for RM-NP on a CAN bus [Bos91] which covers the case when $\gamma \geq 2$.

4.2 PARAMETRIC UTILIZATION BOUNDS FOR AUTOMOTIVE TASK SYSTEMS

parametric utilization bound

In the previous section, we provided a *parametric utilization bound* for non-preemptive scheduling where the considered parameter γ represented the relation between blocking time and execution time. However, while the WCETs of all tasks were assumed to be known in addition to their utilization, there were no restrictions regarding the relation among task parameters. In this section, we explore how to exploit the relation of task parameters among tasks in the task set. Motivated by the following two reasons, we examine restrictions for the

harmonic task set

relation of periods in the system. First, it is known that for *harmonic task sets* the utilization bound of RM-P is 100% [KM91]. Therefore, it seems reasonable that a better result than the Liu and Layland bound of 69.3% [LL73] can be achieved if the relation between periods is not arbitrary but some tasks in the set have harmonic relations of their periods, e.g, for *semi-harmonic task sets*. Second, such harmonic relations among periods are common for real-world systems, since periodic releases will be triggered recurrently based on a timer and the period of the timer will in many cases be a round value of milliseconds or microseconds. One prime example are *automotive task sets* where the period is chosen from $\{1, 2, 5, 10, 20, 50, 100, 200, 1000\}$ ms for the majority of tasks [KZH15; HDK+17; SSD+13; TEH+16]. We prove that the utilization bound for automotive task sets is always at least 90%. In addition, we provide efficient utilization based schedulability tests and parametric utilization bounds for such automotive task sets, increasing this bound even further. We also show that representative automotive task sets are usually schedulable for up to 100%, by considering randomized task sets created according to the "Real world automotive benchmark for free" provided by Bosch [KZH15]. The results presented in this section appeared in *Parametric Utilization Bounds for Implicit-Deadline Periodic Tasks in Automotive Systems* in RTNS 2017 *[BUC+17]*. Note that the provided results can directly include additional harmonic periods, i.e., 0.1 ms, 0.5 ms, and 2000 ms, and can be extended to additional nearly harmonic periods, i.e, including both 0.2 ms and 0.5 ms, or 500 ms. Furthermore, other semi-harmonic settings than the automotive case can be handled taking similar steps.

semi-harmonic task set

automotive task set

4.2.1 PRELIMINARY RESULTS

utilization bound
automotive task set

While tailor-made analysis and *utilization bounds* for *automotive task sets* have not been considered in the literature, some of the known general results for preemptive scheduling can be applied directly or with some slight modification. We start by introducing these results.

harmonic task set
utilization bound
Rate Monotonic

For *harmonic periods*, the *utilization bound* under RM-P is 100% [KM91]. Hence, a task with a period of $1, 2, 10, 20$, 100, 200, or 1000 ms can miss its deadline if and only if the task set has more than 100% utilization.

Lemma 4.12 (Harmonic Subset). *In an automotive implicit-deadline task set, a task τ_k with $T_k \in \{1, 2, 10, 20, 100, 200, 1000\}$ is schedulable under RM-P scheduling if and only if*

$$U_k + \sum_{\tau_i \in hp(\tau_k)} U_i \leq 1 \tag{4.35}$$

Proof. The only-if part is obvious. We sketch the if-part, that has been proved by Nasri et al. [NMF16], for completeness. Using the TDA in Eq. (2.3), we only test at time $t = T_k$. By definition, T_k has a period in $\{1, 2, 10, 20, 100, 200, 1000\}$ and for

a higher-priority task τ_i the period T_i is in $\{1, 2, 5, 10, 20, 50, 100, 200, 1000\}$ with $T_i \leq T_k$. Therefore, T_k is an integer multiple of T_i for any task τ_i in $hp(\tau_k)$, and

$$\begin{aligned} & C_k + \sum_{\tau_i \in hp(\tau_k)} \left\lceil \frac{T_k}{T_i} \right\rceil C_i = C_k + \sum_{\tau_i \in hp(\tau_k)} \frac{T_k}{T_i} C_i \\ = \; & T_k (U_k + \sum_{\tau_i \in hp(\tau_k)} U_i) \leq T_k \end{aligned}$$

where the inequality is due to the if-condition. □

Hence, if a task misses its deadline under RM-P in an automotive implicit-deadline periodic task set with $U_{sum} \leq 100\%$, the period of this task must be either 5 ms or 50 ms. The following lemma combines the results in [LSP04; BB04].

Lemma 4.13 (Utilization-Bound Non-Harmonic Subset). *In an automotive implicit-deadline task set, a task τ_k with $T_k \in \{5, 50\}$ is schedulable under RM-P scheduling if*

$$U_k + \sum_{\tau_i \in hp(\tau_k)} U_i \leq 0.9 \tag{4.36}$$

Proof. We only sketch the proof. Let Y_ℓ be the total utilization of the tasks with periods equal to ℓ, i.e., $Y_\ell = \sum_{\tau_i \in \mathbf{T}_\ell} U_i$ for $\ell = 1, 2, 5, 10, 20, 50$. We prove this lemma only for $T_k = 5$. The objective is equivalent to finding the infimum $Y_1 + Y_2 + Y_5$ such that task τ_k misses its deadline. If we only test the schedulability condition in Eq. (2.3) when $t = 4$ and $t = 5$, we can equivalently formulate this problem as a *linear programming*: *linear programming*

$$\begin{aligned} \text{minimize} \quad & Y_1 + Y_2 + Y_5 \\ \text{such that} \quad & 4Y_1 + 4Y_2 + 5Y_5 \geq 4 \\ & 5Y_1 + 6Y_2 + 5Y_5 \geq 5 \\ & Y_1, Y_2, Y_5 \geq 0 \end{aligned}$$

The utilization bound is 0.9 since the optimal solution of the linear programming is $Y_1 = 0, Y_2 = 0.5, Y_5 = 0.4$. The proof for $T_k = 50$ is almost identical by testing only at time $t = 40$ and $t = 50$. □

Combining Lemmas 4.12 and 4.13 directly shows that the utilization bound of automotive implicit-deadline task sets under RM-P is 90%. Therefore, our focus is to push this bound further upwards. Hence, in the following section, we explain how to derive a parametric utilization bound that is superior to 90%.

4.2.2 ANALYSIS FOR RM-P

We first present a parametric utilization bound and tight schedulability analyses for RM-P. Moreover, we provide an exact schedulability test that only needs to validate 3 inequalities. *Rate Monotonic*

The following two theorems present *parametric utilization bounds* and a concrete example for the utilization lower bounds. *parametric utilization bound*

Theorem 4.14 (Parametric-Bound Non-harmonic). *In an automotive implicit-deadline periodic task set, task τ_k is schedulable under RM-P scheduling if T_k is 5 and*

$$U_k + \sum_{\tau_i \in hp(\tau_k)} U_i \leq 0.9 + \sum_{\tau_i \in \boldsymbol{T}_1} \frac{U_i}{10} \tag{4.37}$$

When T_k is 50, task τ_k is schedulable under RM-P scheduling if

$$U_k + \sum_{\tau_i \in hp(\tau_k)} U_i \leq 0.9 + \sum_{\tau_i \in \widehat{\boldsymbol{T}}} \frac{U_i}{10} \tag{4.38}$$

where $\boldsymbol{T}_x = \{\tau_i \mid \tau_i \in \boldsymbol{T} \text{ and } T_i = x\}$ and $\widehat{\boldsymbol{T}}$ is $\boldsymbol{T}_1 \cup \boldsymbol{T}_2 \cup \boldsymbol{T}_5 \cup \boldsymbol{T}_{10}$ for notational brevity. The above utilization bounds are lower bounded by 0.9.

Proof. We classify the higher-priority tasks in $hp(\tau_k)$ into two subsets $hp^<(\tau_k)$ and $hp^=(\tau_k)$, in which $\tau_i \in hp(\tau_k)$ is in $hp^<(\tau_k)$ if $T_i < T_k$ and is in $hp^=(\tau_k)$ if $T_i = T_k$. Let C'_k be $C_k + \sum_{\tau_i \in hp^=(\tau_k)} C_i$ for notational brevity. For a specific t with $0 < t \leq T_k$, the left-hand side in the schedulability test in Eq. (2.3) is equivalent to

$$C'_k + \sum_{\tau_i \in hp^<(\tau_k)} \left\lceil \frac{t}{T_i} \right\rceil C_i$$

First, assume T_k is 5. By the definition of RM-P scheduling, we know that $hp^<(\tau_k)$ is $\mathbf{T}_1$ and $\mathbf{T}_2$. Suppose that task τ_k cannot pass the test in Eq. (2.3) when we test only $t = 4$ and $t = 5$. For $t = 4$, we get

$$\begin{aligned} & C'_k + \sum_{\tau_i \in \mathbf{T}_1} U_i \times 4 + \sum_{\tau_i \in \mathbf{T}_2} U_i \times 4 > 4 \\ \Rightarrow\ & \frac{C'_k}{5} + \sum_{\tau_i \in \mathbf{T}_1} U_i \times \frac{4}{5} + \sum_{\tau_i \in \mathbf{T}_2} U_i \times \frac{4}{5} > 0.8 \\ \Rightarrow\ & U_k + \sum_{\tau_i \in hp(\tau_k)} U_i > 0.8 + \sum_{\tau_i \in \mathbf{T}_1} \frac{U_i}{5} + \sum_{\tau_i \in \mathbf{T}_2} \frac{U_i}{5} \end{aligned} \tag{4.39}$$

Likewise, for $t = 5$, this results in

$$\begin{aligned} & C'_k + \sum_{\tau_i \in \mathbf{T}_1} U_i \times 5 + \sum_{\tau_i \in \mathbf{T}_2} U_i \times 6 > 5 \\ \Rightarrow\ & \frac{C'_k}{5} + \sum_{\tau_i \in \mathbf{T}_1} U_i + \sum_{\tau_i \in \mathbf{T}_2} 1.2 U_i > 1 \\ \Rightarrow\ & U_k + \sum_{\tau_i \in hp(\tau_k)} U_i > 1 - \sum_{\tau_i \in \mathbf{T}_2} \frac{U_i}{5} \end{aligned} \tag{4.40}$$

By the inequalities in Eq. (4.39) and Eq. (4.40), for a task τ_k with $T_k = 5$ the test in Eq. (2.3) can only fail at $t = 4$ and $t = 5$ if

$$U_k + \sum_{\tau_i \in hp(\tau_k)} U_i > \max \left\{ 1 - \sum_{\tau_i \in \mathbf{T}_2} \frac{U_i}{5}, 0.8 + \sum_{\tau_i \in \mathbf{T}_1} \frac{U_i}{5} + \sum_{\tau_i \in \mathbf{T}_2} \frac{U_i}{5} \right\} \tag{4.41}$$

$$\geq 0.9 + \sum_{\tau_i \in \mathbf{T}_1} \frac{U_i}{10} \tag{4.42}$$

where the $\geq$ is due to the intersection of the two upper bounds in Eq. (4.39) and Eq. (4.40). With similar arguments to the case with $T_k = 5$, we know that the test in Eq. (2.3) fails at $t = 40$ and $t = 50$ for a task τ_k with $T_k = 50$ if

$$U_k + \sum_{\tau_i \in hp(\tau_k)} U_i > \max \left\{ 1 - \sum_{\tau_i \in \mathbf{T}_{20}} \frac{U_i}{5}, 0.8 + \sum_{\tau_i \in \widehat{\mathbf{T}}} \frac{U_i}{5} + \sum_{\tau_i \in \mathbf{T}_{20}} \frac{U_i}{5} \right\} \tag{4.43}$$

$$\geq 0.9 + \sum_{\tau_i \in \widehat{\mathbf{T}}} \frac{U_i}{10} \tag{4.44}$$

where the $\geq$ again comes from the intersection of the two upper bounds. Therefore, by using contrapositive based on Eqs. (4.42) and (4.44), we reach the conclusion. □

Note that Eq. (4.35), Eq. (4.42), and Eq. (4.44) determine parametric utilization bounds of $90\% + z_5$ and $90\% + z_{50}$, where

$$z_5 = \sum_{\tau_i \in \mathbf{T}_1} \frac{U_i}{10} \quad \text{and} \quad z_{50} = \sum_{\tau_i \in \mathbf{T}_1 \cup \mathbf{T}_2 \cup \mathbf{T}_5 \cup \mathbf{T}_{10}} \frac{U_i}{10}$$

We rewrite the conditions in Eqs. (4.35), (4.42), and (4.44) as $\sum_{\tau_i \in \mathbf{T}} U_i \leq 100\%$, $\sum_{\tau_i \in \mathbf{T}_1 \cup \mathbf{T}_2 \cup \mathbf{T}_5} U_i \leq 90\% + z_5$, and $\sum_{\tau_i \in \mathbf{T}_1 \cup \mathbf{T}_2 \cup \mathbf{T}_5 \cup \mathbf{T}_{10} \cup \mathbf{T}_{20} \cup \mathbf{T}_{50}} U_i \leq 90\% + z_{50}$.

We now show that the bounds in Theorem 4.14 are tight.

Theorem 4.15 (Tight-Bound-Non-harmonic). *There exists an automotive implicit-deadline periodic task set with $U_k + \sum_{\tau_i \in hp(\tau_k)} U_i > 0.9$ in which task τ_k is not schedulable by RM-P for a task τ_k in $\boldsymbol{T}_x$ with $x \in \{5, 50\}$.*

Proof. We prove this theorem by providing two concrete examples. Suppose that $T_k = 5$ and let $\mathbf{T}$ consist of τ_1 with $T_1 = 2, C_1 = 1$, and τ_2 with $T_2 = 5, C_2 = 2 + \varepsilon$ with $\varepsilon > 0$ but arbitrarily small.

The utilization of the task set is $0.9 + \varepsilon/5$ and task τ_2 misses its deadline using the exact test in Eq. (2.3). For $T_k = 50$, we multiple T_1, C_1, T_2, and C_2 with 10, leading to a task set with utilization $0.9 + \varepsilon/5$ again that is not schedulable according to Eq. (2.3). □

This leads to the following corollaries:

Corollary 4.16. *The utilization bound of an automotive implicit-deadline task set is 90% which is analytically tight.*

Proof. This corollary follows directly by combining Lemma 4.12, Theorem 4.14, and Theorem 4.15. □

Corollary 4.17. *An automotive implicit-deadline task set is schedulable by RM-P, if $\sum_{\tau_i \in \boldsymbol{T}} U_i \leq 100\%$ and*

$$\sum_{\tau_i \in \boldsymbol{T}} U_i \leq 0.9 + \sum_{\tau_i \in \boldsymbol{T}_1} \frac{U_i}{10} + \Big(\sum_{\tau_i \in \boldsymbol{T}_{100} \cup \boldsymbol{T}_{200} \cup \boldsymbol{T}_{1000}} U_i \Big) \tag{4.45}$$

Proof. Let $\sum_{\tau_i \in \mathbf{T}} U_i \leq 100\%$. According to Lemma 4.12, task τ_k meets its deadline if $T_k = 1, 2, 10, 20, 100, 200, 1000$. Since the satisfaction of the condition in Eq. (4.45) also implies the satisfaction of the condition in Eq. (4.37), task τ_k with $T_k = 5$ always meets its deadline according to Theorem 4.14. Moreover, task τ_k with $T_k = 50$ always meets its deadline since the satisfaction of the condition in Eq. (4.45) also implies the satisfaction of the condition in Eq. (4.38). Thus, we reach the conclusion. □

In the proof of Theorem 4.14, we showed that testing $t = 4$ and $t = 5$ in Eq. (2.3) for $T_k = 5$ ($t = 40$ and $t = 50$ for $T_k = 50$, respectively) is sufficient to achieve the utilization bound of 90%. The following lemma shows that an exact test only needs to test these two specific t values in Eq. (2.3) as well.

Lemma 4.18. *A task τ_k in $\mathbf{T}_5$ is schedulable under RM-P scheduling **if and only if** the schedulability condition in Eq. (2.3) holds for $t = 4$ or $t = 5$. A task τ_k in $\mathbf{T}_{50}$ is schedulable under RM-P scheduling **if and only if** the schedulability condition in Eq. (2.3) holds for $t = 40$ or $t = 50$.*

Proof. We only prove the case $T_k = 50$ since the procedure is similar for $T_k = 5$. Let t^* be the minimum value with $0 < t^* \leq 50$ such that $C_k + \sum_{\tau_i \in hp(\tau_k)} \left\lceil \frac{t^*}{T_i} \right\rceil C_i = t^*$. We show that the existence of t^* implies either $C_k + \sum_{\tau_i \in hp(\tau_k)} \left\lceil \frac{40}{T_i} \right\rceil C_i \leq 40$ or $C_k + \sum_{\tau_i \in hp(\tau_k)} \left\lceil \frac{50}{T_i} \right\rceil C_i \leq 50$.

Recall the definition of C'_k, $hp^<(\tau_k)$, and $hp^=(\tau_k)$ in the proof of Theorem 4.14.

- Case 1 when $0 < t^* \leq 40$: This means that

$$t^* = C_k + \sum_{\tau_i \in hp(\tau_k)} \left\lceil \frac{t^*}{T_i} \right\rceil C_i \geq C'_k + \sum_{\tau_i \in hp^<(\tau_k)} t^* U_i$$

  Clearly, $\sum_{\tau_i \in hp^<(\tau_k)} U_i \leq 1$. As 40 is an integer multiple of $1, 2, 5, 10$, and 20,

$$\begin{aligned} &C_k + \sum_{\tau_i \in hp(\tau_k)} \left\lceil \frac{40}{T_i} \right\rceil C_i = C'_k + \sum_{\tau_i \in hp^<(\tau_k)} 40 U_i \\ &\leq t^* \left(1 - \sum_{\tau_i \in hp^<(\tau_k)} U_i\right) + \sum_{\tau_i \in hp^<(\tau_k)} 40 U_i \leq 40 \end{aligned}$$

  we reach the conclusion $C_k + \sum_{\tau_i \in hp(\tau_k)} \left\lceil \frac{40}{T_i} \right\rceil C_i \leq 40$.

- Case 2 when $40 < t^* \leq 50$: This means that

$$t^* = C_k + \sum_{\tau_i \in hp(\tau_k)} \left\lceil \frac{t^*}{T_i} \right\rceil C_i \geq C'_k + \sum_{\tau_i \in \widehat{\mathbf{T}}} t^* U_i + \sum_{\tau_i \in \mathbf{T}_{20}} 3 C_i$$

  where $\widehat{\mathbf{T}}$ is $\mathbf{T}_1 \cup \mathbf{T}_2 \cup \mathbf{T}_5 \cup \mathbf{T}_{10}$. Clearly, $\sum_{\tau_i \in \widehat{\mathbf{T}}} U_i \leq 1$. Since 50 is an integer multiple of $1, 2, 5$, and 10,

$$\begin{aligned} &C_k + \sum_{\tau_i \in hp(\tau_k)} \left\lceil \frac{50}{T_i} \right\rceil C_i = C'_k + \sum_{\tau_i \in \widehat{\mathbf{T}}} 50 U_i + \sum_{\tau_i \in \mathbf{T}_{20}} 3 C_i \\ &\leq t^* \left(1 - \sum_{\tau_i \in \widehat{\mathbf{T}}} U_i\right) + \sum_{\tau_i \in \widehat{\mathbf{T}}} 50 U_i \leq 50 \left(1 - \sum_{\tau_i \in \widehat{\mathbf{T}}} U_i\right) + \sum_{\tau_i \in \widehat{\mathbf{T}}} 50 U_i = 50 \end{aligned}$$

  we reach the conclusion $C_k + \sum_{\tau_i \in hp(\tau_k)} \left\lceil \frac{50}{T_i} \right\rceil C_i \leq 50$. □

This leads to the following *exact linear time schedulability test*: *exact test*

Theorem 4.19. *The given automotive implicit-deadline periodic task set is schedulable by RM-P if and only if all of the following conditions are satisfied:*

$$\sum_{\tau_i \in \boldsymbol{T}} U_i \leq 1 \tag{4.46}$$

$$\sum_{\tau_i \in \boldsymbol{T}_1 \cup \boldsymbol{T}_2 \cup \boldsymbol{T}_5} U_i \leq \max \left\{ 1 - \sum_{\tau_i \in \boldsymbol{T}_2} \frac{U_i}{5}, 0.8 + \sum_{\tau_i \in \boldsymbol{T}_1} \frac{U_i}{5} + \sum_{\tau_i \in \boldsymbol{T}_2} \frac{U_i}{5} \right\} \tag{4.47}$$

$$\sum_{\tau_i \in \widehat{\boldsymbol{T}} \cup \boldsymbol{T}_{20} \cup \boldsymbol{T}_{50}} U_i \leq \max \left\{ 1 - \sum_{\tau_i \in \boldsymbol{T}_{20}} \frac{U_i}{5}, 0.8 + \sum_{\tau_i \in \widehat{\boldsymbol{T}}} \frac{U_i}{5} + \sum_{\tau_i \in \boldsymbol{T}_{20}} \frac{U_i}{5} \right\} \tag{4.48}$$

where $\widehat{\boldsymbol{T}}$ *is* $\boldsymbol{T}_1 \cup \boldsymbol{T}_2 \cup \boldsymbol{T}_5 \cup \boldsymbol{T}_{10}$. *Therefore, testing Eq. (4.46), Eq. (4.47), and Eq. (4.48) is an exact schedulability test.*

Proof. This is based on Lemma 4.12 and Lemma 4.18. The last two conditions represent the tests at time $t = 4$ and $t = 5$ for a task τ_k with $T_k = 5$ (from Eq. (4.41)) and at time $t = 40$ and $t = 50$ for a task τ_k with $T_k = 50$ (from Eq. (4.43)), respectively. □

4.2.3 NON-PREEMPTIVE SCHEDULING

We now provide a sufficient schedulability test for automotive task sets under RM-NP, considering the utilization of the task τ_k itself, the utilization of the higher-priority tasks, and the blocking time for task τ_k. In this test, we assume that that $C_i < 1$ for each task τ_i in the set. This is not too restrictive, since if $C_i \geq 1$ for any task, the task set is unschedulable by default if at least one task τ_b in the system has a period of 1 due to the blocking time for τ_b. *Rate Monotonic non-preemptive scheduling*

Theorem 4.20. *Suppose that* $\boldsymbol{T}$ *is an automotive implicit-deadline periodic task set and suppose that* $C_i < 1$ *for every task* τ_i *in* $\boldsymbol{T}$. *When* T_k *is in* $\{1, 2, 10, 20, 100, 200, 1000\}$, *task* τ_k *is schedulable under RM-NP if*

$$\frac{\max_{\tau_i \in lp(\tau_k)} C_i}{T_k} + U_k + \sum_{\tau_i \in hp(\tau_k)} U_i \leq 1 \tag{4.49}$$

When T_k *is* 5, *task* τ_k *is schedulable under RM-NP if the condition in Eq. (4.37) holds and*

$$\begin{aligned} &\frac{\max_{\tau_i \in lp(\tau_k)} C_i}{T_k} + \sum_{\tau_i \in hp(\tau_k)} U_i \\ &\leq \max \left\{ 1 - \sum_{\tau_i \in \boldsymbol{T}_2} \frac{U_i}{5} - U_k, 0.8 + \sum_{\tau_i \in \boldsymbol{T}_1} \frac{U_i}{5} + \sum_{\tau_i \in \boldsymbol{T}_2} \frac{U_i}{5} \right\} \end{aligned} \tag{4.50}$$

When T_k is 50, task τ_k is schedulable under RM-NP if the condition in Eq. (4.38) holds and

$$\frac{\max_{\tau_i \in lp(\tau_k)} C_i}{T_k} + \sum_{\tau_i \in hp(\tau_k)} U_i \leq \max \left\{ 1 - \sum_{\tau_i \in \boldsymbol{T}_{20}} \frac{U_i}{5} - U_k, 0.8 + \sum_{\tau_i \in \widehat{\boldsymbol{T}}} \frac{U_i}{5} + \sum_{\tau_i \in \boldsymbol{T}_{20}} \frac{U_i}{5} \right\} \tag{4.51}$$

where $\widehat{\boldsymbol{T}}$ is $\boldsymbol{T}_1 \cup \boldsymbol{T}_2 \cup \boldsymbol{T}_5 \cup \boldsymbol{T}_{10}$ for notational brevity.

Proof. The condition from Eq. (4.49) comes from the same analysis as for the if-part in Lemma 4.12 when applying Eq. (2.8) instead of Eq. (2.3). We focus on the other cases, i.e., when $T_k = 5$ and $T_k = 50$, by using Lemma 2.2 that states that a sufficient schedulability test is to validate whether both conditions in Eq. (2.3) and Eq. (2.9) hold. The condition in Eq. (2.3) can be tested by using Theorem 4.19. We focus on a utilization-based test by simplifying Eq. (2.9).

The test in Eq. (2.9) uses $\left\lfloor \frac{t}{T_i} \right\rfloor + 1$ for the summation of the interference from the higher-priority tasks. This can be translated into using $\left\lceil \frac{t}{T_i} \right\rceil$ instead if we use $B_k = \max_{\tau_i \in lp(\tau_k)} C_i$ as the blocking time instead of $B_k^* = \max_{\tau_i \in lp(\tau_k)} C_i - \varepsilon$. Increasing the blocking time by ε makes the test a bit more pessimistic. However, if ε can be considered to be small compared to the WCETs of the tasks in $\mathbf{T}$ this pessimism is negligible. For the simplicity of presentation, instead of using Eq. (2.9), we therefore use the following test

$$\exists t | 0 < t \leq T_k - C_k, \qquad B_k + \sum_{\tau_i \in hp(\tau_k)} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t \tag{4.52}$$

Recall the definition of $hp^<(\tau_k)$ and $hp^=(\tau_k)$ in the proof of Theorem 4.14. Furthermore, let $C_k^\dagger = \sum_{\tau_i \in hp^=(\tau_k)} C_i$. We first consider the case when T_k is 5, i.e., $hp^<(\tau_k)$ is $\mathbf{T}_1 \cup \mathbf{T}_2$. Suppose that task τ_k cannot pass the test in Lemma 2.2 due to the non-preemptive case using condition Eq. (4.52). Due to the assumption $C_k < 1$, it is sufficient to test the condition in Eq. (4.52) at time $t = 4$ and at time $t = 5 - C_k$ with $C_k < 1$.

For $t = 4$, we get

$$\begin{aligned} & B_k + C_k^\dagger + \sum_{\tau_i \in \mathbf{T}_1} U_i \times 4 + \sum_{\tau_i \in \mathbf{T}_2} U_i \times 4 > 4 \\ \Rightarrow \; & \frac{B_k}{5} + \frac{C_k^\dagger}{5} + \sum_{\tau_i \in \mathbf{T}_1} U_i \times \frac{4}{5} + \sum_{\tau_i \in \mathbf{T}_2} U_i \times \frac{4}{5} > 0.8 \\ \Rightarrow \; & \frac{B_k}{T_k} + \sum_{\tau_i \in hp(\tau_k)} U_i > 0.8 + \sum_{\tau_i \in \mathbf{T}_1} \frac{U_i}{5} + \sum_{\tau_i \in \mathbf{T}_2} \frac{U_i}{5} \end{aligned} \tag{4.53}$$

For $t = 5 - C_k$, we get

$$\begin{aligned} & B_k + C_k^\dagger + \sum_{\tau_i \in \mathbf{T}_1} U_i \times 5 + \sum_{\tau_i \in \mathbf{T}_2} U_i \times 6 > 5 - C_k \\ \Rightarrow \; & \frac{B_k}{5} + \frac{C_k^\dagger}{5} + \sum_{\tau_i \in \mathbf{T}_1} U_i + \sum_{\tau_i \in \mathbf{T}_2} 1.2 U_i > 1 - U_k \\ \Rightarrow \; & \frac{B_k}{T_k} + \sum_{\tau_i \in hp(\tau_k)} U_i > 1 - \sum_{\tau_i \in \mathbf{T}_2} \frac{U_i}{5} - U_k \end{aligned} \tag{4.54}$$

By the inequalities in Eq. (4.53) and Eq. (4.54), the failure of the test in Eq. (4.52) at $t = 4$ and $t = 5 - C_k$ happens if

$$\frac{B_k}{T_k} + \sum_{\tau_i \in hp(\tau_k)} U_i > \max \left\{ 1 - \sum_{\tau_i \in \mathbf{T}_2} \frac{U_i}{5} - U_k, 0.8 + \sum_{\tau_i \in \mathbf{T}_1} \frac{U_i}{5} + \sum_{\tau_i \in \mathbf{T}_2} \frac{U_i}{5} \right\} \tag{4.55}$$

We reach the conclusion in Eq (4.50) using contrapositive.

Furthermore, we test $t = 40$ and $t = 50 - C_k$ when T_k is 50, which leads to the conclusion in Eq (4.51). □

While Theorem 4.19 is an exact test, the test in Theorem 4.20 is only sufficient since the blocking time is greedily included. In general, to verify the schedulability under FP-NP, the schedulability of each task has to be verified individually. This is due to the fact that the blocking time is a decreasing function with respect to the priority and, thus, a task τ_j with a lower priority than task τ_i may be schedulable while τ_i is not schedulable because the blocking time for τ_i is larger.

4.2.4 ANGLE-SYNCHRONOUS TASKS

In addition to periodic tasks, an automotive task system may involve *event-triggered* aperiodic/sporadic tasks [FBD+18]. One specific type are *angle-synchronous tasks* where the jobs are triggered by the rotation of the crankshaft. According to [KZH15], the inter-arrival time between two jobs of an angle-synchronous engine control task can be modeled as

angle-synchronous tasks

$$\frac{120}{rpm \times \#cyl} \times 1000 \text{ milliseconds} \tag{4.56}$$

where rpm is the revolutions per minute of the engine and #cyl is the number of cylinders. Even though the inter-arrival time of these jobs may change over time, they are scheduled based on static-priority scheduling. We consider two general approaches for the priority assignment of those angle-synchronous tasks:

1. Assigning them to the highest priority.
2. The priorities are assigned according to the inter-arrival time at the maximum rotation speed, i.e., the shortest possible inter-arrival time between two jobs. For example, for #cyl = 4 and 6000 rpm, the minimum inter-arrival time is $\frac{120 \times 1000}{6000 \times 4} = 5$ ms.

An angle-synchronous task τ_i with q execution modes can be modeled by a tuple $< C_i^1, T_i^1, C_i^2, T_i^2, \ldots, C_i^q, T_i^q >$ where C_i^j is the WCET for the j^th mode, and T_i^j is the minimum inter-arrival time after a job in the j^th mode of task τ_i is released. Such tasks are also called variable-rate-behaviour tasks [DFP+14] or multi-mode tasks [HC15b], and schedulability tests of under FP scheduling have been proposed in [HC15b; DFP+14] as well.

Two existing methods allow to calculate the interference due to an angle-synchronous task in an interval length Δ. On the one hand, the worst-case workload can be determined by investigating the worst-case release patterns using integer linear programming (ILP) [DFP+14] or dynamic programming [HC15b]. On the other hand, the interference due to an angle-synchronous task τ_i can be safely approximated by examining $U_i^{max} = \max_{j\in\{1,\ldots,q\}} \left\{ \frac{C_i^j}{T_i^j} \right\}$, $C_i^{max} = \max_{j\in\{1,\ldots,q\}} \left\{ C_i^j \right\}$, and $T_i^{min} = \max_{j\in\{1,\ldots,q\}} \left\{ T_i^j \right\}$ as shown in the following lemma:

Lemma 4.21. *The maximum interference $I_i(\Delta)$ incurred by an angle-synchronous task τ_i in an interval of length Δ is at most*

$$I_i(\Delta) = \begin{cases} U_i^{max} \times \Delta + C_i^{max} & \text{if } \Delta > T_i^{min} \\ C_i^{max} & \text{if } \Delta \leq T_i^{min} \end{cases} \tag{4.57}$$

Proof. This is based on Theorem 1 by Davis et al. [DFP+14] and Lemma 2 by Huang and Chen [HC15b]. Without loss of generality, let the interval start at time 0. According to Theorem 1 in [DFP+14], the maximum interference from an angle-synchronous task τ_i to a lower-priority job arriving at time 0 happens in the following worst-case pattern: a) release the first job at time 0, b) follow the minimum period needed in the particular execution mode, and c) release the last job with execution time C_i^{max} before Δ. We consider the two cases individually. For $\Delta > T_i^{min}$, let $t^* < \Delta$ be the arrival time of the last job in the above pattern. Lemma 2 in [HC15b] that the maximum interference from 0 to t^* is at most $U_i^{max} \times t^*$ if the last job is excluded. Therefore, by including the job released at or after t^*, the maximum interference incurred by τ_i is at most $U_i^{max} \times \Delta + C_i^{max}$. For $\Delta \leq T_i^{min}$, only one job of the angle-synchronous task is released and the interference is, obviously, at most C_i^{max}. □

We revise the schedulability test in Eq. (2.3) to consider angle-synchronous tasks. Let $hp(\tau_k)$ be the set of the *periodic* tasks with priorities higher than task τ_k and let $\mathbf{T}_{as}$ be the set of the *angle-synchronous* tasks with higher priority. An implicit-deadline periodic task τ_k is schedulable under FP-P scheduling if

$$\exists t | 0 < t \leq T_k, \;\; C_k + \sum_{\tau_i \in hp(\tau_k)} \left\lceil \frac{t}{T_i} \right\rceil C_i + \sum_{\tau_i \in \mathbf{T}_{as}} I_i(t) \leq t \tag{4.58}$$

The schedulability test in Eq. (4.58) does not significantly increase the difficulty for testing the schedulability of a periodic task τ_k under FP-P compared to the case without angle-synchronous tasks. All utilization-based schedulability tests in Section 4.2.2 and the test in Theorem 4.20 can be revised easily by including the interference from the angle-synchronous tasks based on Eq. (4.58). For example, we can revise Theorem 4.19 as:

Theorem 4.22. *Suppose that tasks in $\boldsymbol{T}_{as}$ are assigned to higher priorities than any periodic task, and that the priorities of the periodic tasks are assigned by the rate-monotonic priority assignment. For each value y in $\{1, 2, 5, 10, 20, 50, 100, 200, 1000\}$ let $\widehat{\boldsymbol{T}}_y$ be defined as the set of periodic tasks with period less than or equal to y for notational brevity. The given automotive implicit-deadline periodic task set is schedulable under static-priority preemptive scheduling if the angle-synchronous tasks are schedulable at the highest priority and all the following conditions are satisfied:*

$$\sum_{\tau_i \in \widehat{\boldsymbol{T}}_x} U_i + \sum_{\tau_i \in \boldsymbol{T}_{as}} \frac{I_i(x)}{x} \leq 1 \qquad \forall x \in \{1, 2, 10, 20, 100, 200, 1000\} \tag{4.59}$$

$$\sum_{\tau_i \in \widehat{\boldsymbol{T}}_5} U_i \leq \max \left\{ 1 - \sum_{\tau_i \in \boldsymbol{T}_2} \frac{U_i}{5} - \sum_{\tau_i \in \boldsymbol{T}_{as}} \frac{I_i(5)}{5}, 0.8 + \sum_{\tau_i \in \boldsymbol{T}_1 \cup \boldsymbol{T}_2} \frac{U_i}{5} - \sum_{\tau_i \in \boldsymbol{T}_{as}} \frac{I_i(4)}{5} \right\} \tag{4.60}$$

$$\sum_{\tau_i \in \widehat{\boldsymbol{T}}_{50}} U_i \leq \max \left\{ 1 - \sum_{\tau_i \in \boldsymbol{T}_{20}} \frac{U_i}{5} - \sum_{\tau_i \in \boldsymbol{T}_{as}} \frac{I_i(50)}{50}, 0.8 + \sum_{\tau_i \in \widehat{\boldsymbol{T}} \cup \boldsymbol{T}_{20}} \frac{U_i}{5} - \sum_{\tau_i \in \boldsymbol{T}_{as}} \frac{I_i(40)}{50} \right\} \tag{4.61}$$

Proof. This follows directly from considering the interference due to the angle-synchronous tasks in Eq. (4.58) and repeating the same procedures as in the proofs of Section 4.2.2. □

Note that the schedulability has to be tested individually for each period. The reason is that while the interference due to the utilization U_i^{max} of the angle synchronous tasks is constant for each period, the interference due to C_i^{max} decreases when the period is increased. Furthermore, Theorem 4.22 is only a sufficient test while Theorem 4.19 is an exact schedulability test. This is due to the fact that the terms we introduce to calculate the interference from angle-synchronous tasks are safe approximations but not tight.

Regarding schedulability, letting the angle-synchronous tasks have the highest priority introduces unnecessary pessimism to the system. The reason is that tasks with high priority are postponed while angle-synchronous tasks that arrived later and have a larger relative deadline are executed. Instead, from a scheduling point of view, the angle-synchronous tasks should be scheduled according to their minimal inter-arrival time if this value can determined safely. However, this would most likely not lead to harmonic periods which are key for a high schedulability. Therefore, we propose to determine the priority of the angle-synchronous tasks based on the minimal inter-arrival time T_{as} and to analyse them assuming they have a period that is the maximum $p \in \{1, 2, 5, 10, 20, 50, 100, 200, 1000\}$ with $p \leq T_{as}$. Hence, T_{as} would be removed from the analysis when only tasks in $\widehat{\mathbf{T}}_p$ are considered and the schedulability test for the angle-synchronous tasks can be done by using Eq. (4.59), Eq. (4.60), or Eq. (4.61), depending on p. For example, for a maximum of 6000 rpm and 6 cylinders, according to Eq. (4.56), the angle-synchronous tasks have a minimal inter-arrival time of ≈ 3.33. Thus their priority would be between the tasks in $\mathbf{T}_2$ and $\mathbf{T}_5$. Hence, they would be removed from the analysis for $x = 1$ and $x = 2$ and the schedulability of the angle-synchronous tasks would be determined using Eq. (4.59) for $x = 2$.

4.2.5 EVALUATION

To analyze the schedulability of implicit-deadline periodic tasks in automotive systems, it would be best to analyze task sets of real-world applications. Unfortunately, real-world automotive task sets are not available to the public. Hence, we use synthetic task sets that are similar to real-world systems, based on "Real world automotive benchmarks for free" by Kramer, Ziegenbein, and Hamann [KZH15]. Note that similar period distributions are used in other works related to automotive applications, e.g., in [TEH+16; SSD+13; HDK+17].

In automotive systems, the entity for scheduling is a Runnable, which is equivalent to a task in this paper, i.e, the information provided in [KZH15] about Runnables is used to create tasks. We analyzed the schedulability of the resulting task sets using the schedulability tests presented in this paper, both for scaled and for unscaled task sets as well as both for RM-P and RM-NP. We conducted evaluations with and without considering angle-synchronous tasks.

EVALUATION SETUP

The information to create the real-world automotive task sets was collected from Tables III, IV, and V in [KZH15] and is summarized in Table 4.1 in a compact form. Table III in [KZH15] provided the distribution of tasks over the periods in $\{1, 2, 5, 10, 20, 50, 100, 200, 1000\}$ ms and for angle-synchronous tasks. For each period, the minimum, average, and maximum value of the average-case execution time (ACET) of tasks is given in Table IV in [KZH15]. According to [KZH15], the value distribution can be approximated using a Weibull distribution with the probability density function

$$f(x) = \frac{k}{\lambda} \cdot \left(\frac{x}{\lambda}\right)^{k-1} \cdot e^{-\left(\frac{x}{\lambda}\right)^k} \tag{4.62}$$

for $x \geq 0$, if the values for the shape parameter k and the scale parameter λ are given. We numerically approximated k and λ for each period, since only C_{min}, $C_{average}$, and C_{max} of the distributions are provided. We used the maximum likelihood estimators for k and λ as the starting values, drew a sample of 10000 random numbers based on those values, compared the results with the given C_{min}, $C_{average}$, and C_{max}, and adjusted the values for k and λ based on the resulting distribution. We iterated until the resulting distribution matched the values for C_{min}, $C_{average}$, and C_{max} provided in [KZH15]. Since we only approximated the Weibull distribution up to a certain accuracy, drawn C_i values that were not in the related $[C_{min}, C_{max}]$ interval were discarded in the actual evaluation. Based on the average execution time values the WCETs can be calculated by scaling up the average execution time with a randomly distributed factor in the interval $[f_{min}, f_{max}]$ related to the period of the task. As only the scaling factors are provided in Table V in [KZH15] but no further information regarding the distribution was given, we used a uniform distribution over $[f_{min}, f_{max}]$.

We conducted evaluations in two general setups: 1) based on the ACETs reported in [KZH15], i.e., the values based on the Weibull distributions, referred

Period	Share	Average ET in μs			WCET factor	
		Min	Avg.	Max	f_{min}	f_{max}
1 ms	3%	0.34	5.00	30.11	1.30	29.11
2 ms	2%	0.32	4.20	40.69	1.54	19.04
5 ms	2%	0.36	11.04	83.38	1.13	18.44
10 ms	25%	0.21	10.09	309.87	1.06	30.03
20 ms	25%	0.25	8.74	291.42	1.06	15.61
50 ms	3%	0.29	17.56	92.98	1.13	7.76
100 ms	20%	0.21	10.53	420.43	1.02	8.88
200 ms	1%	0.22	2.56	21.95	1.03	4.90
1000 ms	4%	0.37	0.43	0.46	1.84	4.75
angle-sync.	15%	0.45	6.52	88.58	1.20	28.17

Table 4.1: The information used to generate the automotive task sets, combined from Table III, Table IV, and Table V in [KZH15]. *Adapted from [BUC+17].*

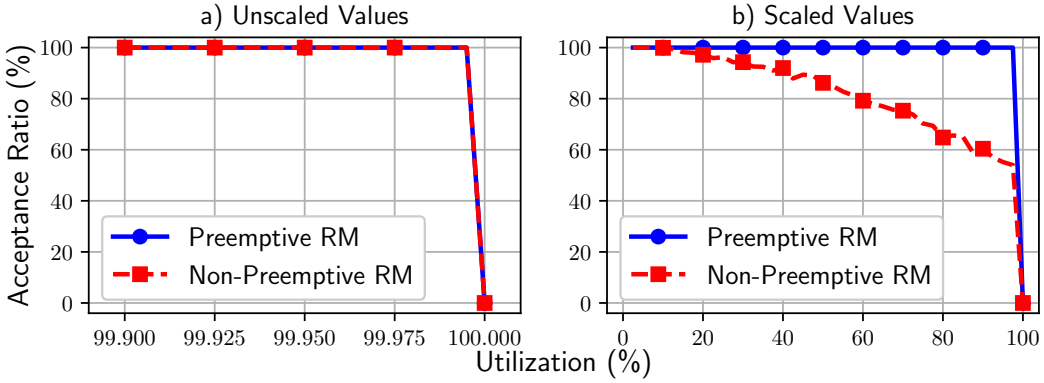


Figure 4.2: The acceptance ratio of unscaled and scaled task sets for both preemptive (RM-P) and non-preemptive Rate Monotonic scheduling (RM-NP). *Adapted from [BUC+17].*

to as *unscaled tasks*, and 2) based on the worst-case execution times after *scaling* the ACETs with the WCET scaling factors. For a given target utilization U_t we randomized task sets with a total utilization in the interval $[U_t, U_t + \gamma]$ for a small $\gamma > 0$. Details can be found in the Appendix. Independent from the setting, we always created 1000 task sets for each utilization value we analyzed.

GENERAL SCHEDULABILITY

We evaluated the schedulability under RM-P and RM-NP for task sets without angle-synchronous tasks, considering random task sets based on Table 4.1 under the schedulability tests in Theorem 4.19 and Theorem 4.20 for the preemptive and the non-preemptive case, respectively. The results with and without scaling are shown in Figure 4.2a and Figure 4.2b, respectively. The task sets with unscaled values are (nearly) always schedulable under RM-P. However, when considering 99.99% utilzation as an example, the setting in Table 4.1 did not lead to the case where Corollary 4.17 could be applied directly, i.e., the combined utilization of the tasks with periods 100, 200, and 1000 was always below 10%. Therefore,

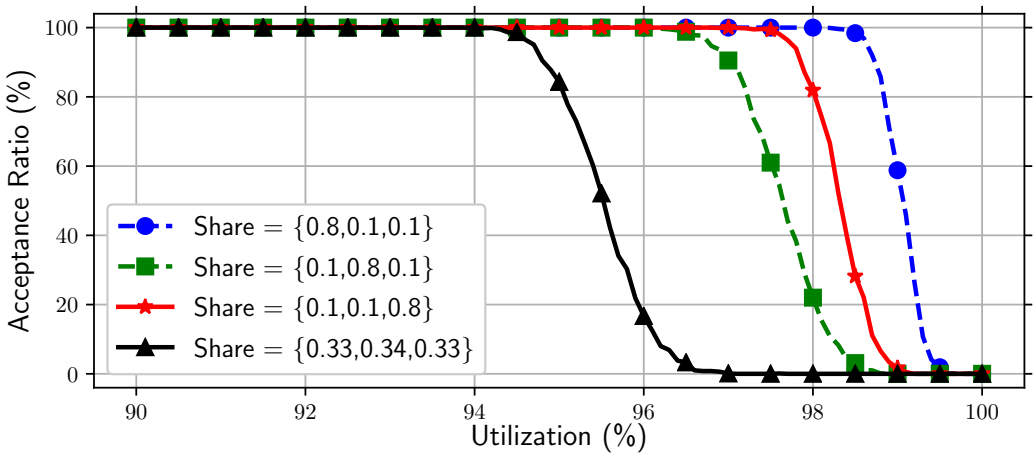


Figure 4.3: The effect of non-harmonic subsets in Theorem 4.19. The share values indicate the average percentage of tasks with period 1, 2, and 5 ms, respectively. *Adapted from [BUC+17].*

we analyzed the utilization values of the individual periods, i.e., looking at the non-harmonic periods 5 and 50. The combined utilization for periods $1, 2$, and 5 was always $\leq 47.68\%$. Hence, the tasks up to period 5 are always schedulable as the total utilization up to period 5 is below 90%. Furthermore, the total utilization of periods $1, 2, 5, 10, 20$, and 50 was at most 97.52% while the total utilization of periods $1, 2, 5, 10$, and 20 was at least 92.78%. Putting these values to Eq (4.48) results in a guaranteed schedulability since $80\% + 92.78\%/5 = 98.556\%$ which is larger than 97.52%. In addition, task sets with 100% utilization are never schedulable for RM-P due to their construction, as we created sets with a utilization in $[U_t, U_t + \gamma]$, i.e., the actual task set utilization in this case was strictly larger than 100%. Note that it is possible that task sets with a utilization U^* with $90\% < U^* < 100\%$ are created that are not schedulable in the preemptive case due to the random distribution of task periods. However, this is very unlikely and never happened in our evaluation.

To determine the impact of the distribution of tasks among non-harmonic periods, we analyzed task sets where all periods were in $\{1, 2, 5\}$ under RM-P, considering different distributions over the periods. The individual tasks were created according to the C_i distribution given in Table 4.1. The results are shown in Figure 4.3. The probability that a task has period x depends on the share value given for that period in the related label, i.e., it shows the probability that a task is in $\{T_1, T_2, T_5\}$. If the distribution of probabilities is $\{0.8, 0.1, 0.1\}$ (blue curve) the task sets were always schedulable up to 98.1%, since the utilization of $\mathbf{T}_1$ is very large and task sets are schedulable up to a utilization of $0.9 + \sum_{\tau_i \in \mathbf{T}_1} \frac{U_i}{10}$ according to Eq. (4.42). To explain the other cases, we look at Eq. (4.47) in Theorem 4.19, i.e., $\sum_{\tau_i \in \widehat{\mathbf{T}}_5} \leq \max \left\{ 1 - \sum_{\tau_i \in \mathbf{T}_2} \frac{U_i}{5}, 0.8 + \sum_{\tau_i \in \mathbf{T}_1} \frac{U_i}{5} + \sum_{\tau_i \in \mathbf{T}_2} \frac{U_i}{5} \right\}$. For the green and the red curve, the acceptance ratio drops a bit earlier than for the blue curve. The reason is that in Eq. (4.47) a large value on the right hand side happens for either a large or a small utilization of tasks with period 2. If the tasks are distributed equally over the periods $1, 2$, and 5 (black curve), we observe the earliest drop of the acceptance ratio. The reason is that none of the terms on the right hand side of Eq. (4.47) is as large as in the previous cases.

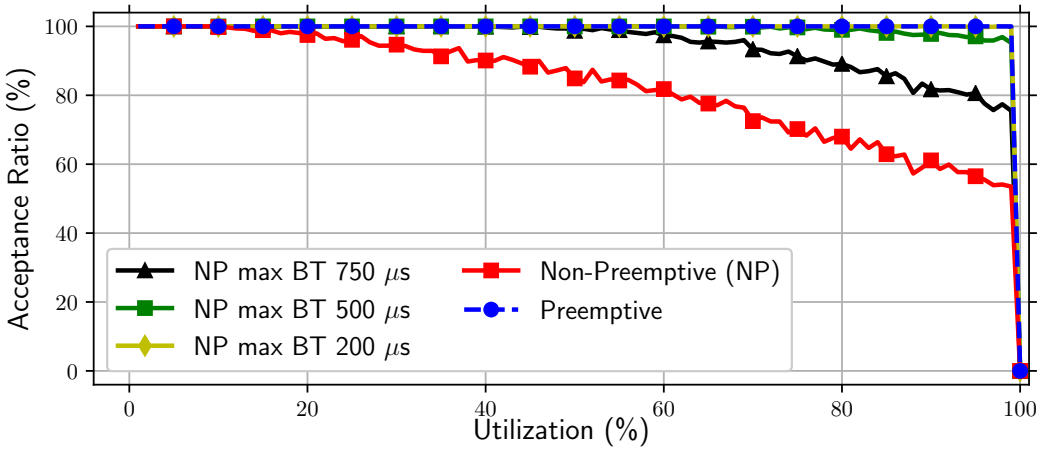


Figure 4.4: Impact of the maximum blocking time on the schedulability of automotive task sets under for RM-NP. *Adapted from [BUC+17].*

When the tasks are scaled (Figure 4.2b), all task sets with a utilization less than 100% are still schedulable under RM-P with similar reasons as in the unscaled case. On the other hand, for RM-NP the schedulability drops from 10% utilization onwards, since the execution time can be larger than 1 ms for some tasks after scaling them. Therefore, we analyzed the effect of bounded blocking times which is similar to *limited-preemptive scheduling* approaches, where tasks are separated into non-preemptive subtasks with a given maximum length. As shown in Figure 4.4, the acceptance ratio can still be very reasonable for such a setup. Even for a comparatively large maximum blocking time of 750 μs the improvement compared to the strictly non-preemptive case is significant. For a maximum blocking time of 500 μs the acceptance ratio is always above 95.6%. If the maximum blocking time is set to 200 μs or less, the acceptance ratio is the same as in the preemptive case, i.e., the task sets are always schedulable.

limited-preemptive scheduling

This result shows that moderate blocking times due to non-preemptive execution have no negative impact on the schedulability of automotive task sets compared to preemptive scheduling. Hence, tasks with a WCET of 200 μs or less can be executed non-preemptively. For tasks with a WCET that is larger than 200 μs, *limited-preemptive scheduling* approaches like *co-operative scheduling* [Bur94] can be utilized when the length of the non-preemptive sections is chosen based on the result of our evaluation, i.e., the non-preemptive sections are 200 μs or less. Note that non-preemptive execution of jobs or of certain job intervals also reduce the WCET of tasks. The reason is that overhead for preemptions and cache misses is usually included in the WCET to guarantee a safe schedulability analysis. While bounding such overheads is problematic under preemptive scheduling, the number of preemptions and the effect of cache misses can more easily be determined when the number of preemptions is zero or bounded by the number of non-preemptive intervals. Therefore, the provided results enable to combine the advantages of preemptive and non-preemptive scheduling in practical scenarios.

limited-preemptive scheduling

co-operative scheduling

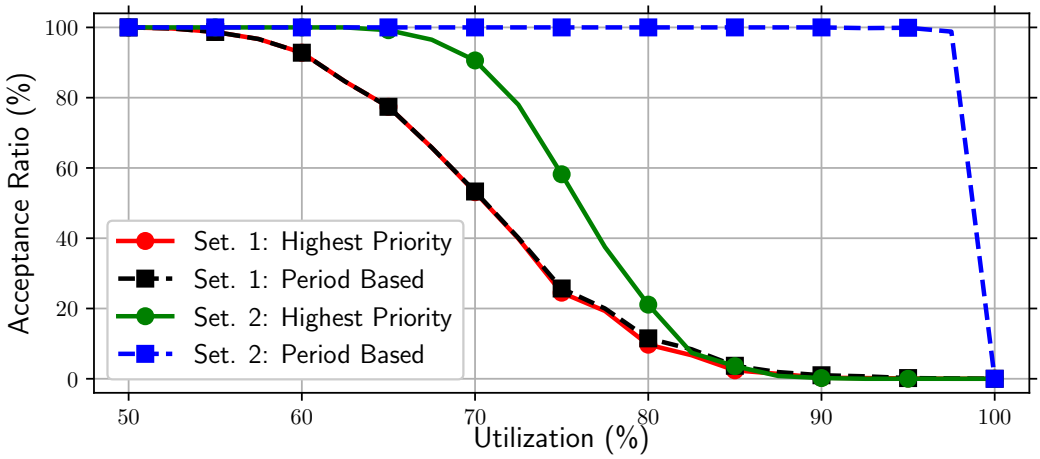


Figure 4.5: Acceptance ratio of different priority-assignment strategies for angle-synchronous tasks under different period distributions. *Adapted from [BUC+17].*

ANGLE-SYNCHRONOUS TASKS

Besides providing the schedulability tests for automotive task sets with angle-synchronous tasks, in Section 4.2.4 we also stated that, from a scheduling point of view, the priorities of the angle-synchronous tasks should be set according to their minimum inter-arrival times instead of to the highest-priority level, if possible. We compare those two approaches in Figure 4.5, considering angle synchronous tasks that have a minimum inter-arrival time of 5 ms, i.e., 6000 rpm and 4 cylinders in Eq. (4.56). We drew C_i randomly according to Table 4.1, and set $C_i^{max} = C_i$ and $U_i^{max} = C_i/6000$. We considered two settings:

- **Setting 1**: the periods are distributed according to Table 4.1, resulting in a nearly identical acceptance ratio for both approaches. However, assigning the priority of angle-synchronous tasks based on their minimum inter-arrival time (black, dashed curve) is never worse than assigning them to the highest priority (red, solid curve).
- **Setting 2**: the period distribution is based on Table 4.1 after exchanging the probabilities for tasks to have a period of 1 ms and a period of 10 ms, i.e., a period of 1 ms has a probability of 25% and a period of 10 ms one of 3%. The tasks are nearly always schedulable if the priority for the angle-synchronous tasks is assigned according to the minimal inter-arrival time (blue, dashed curve). However, if the angle-synchronous tasks have the highest priority (green, solid curve), the acceptance ratio starts dropping at 65% and is under 20% for a utilization of 80%. While a higher utilization for $\mathbf{T}_1$ usually increases the schedulability, if the angle-synchronous tasks are scheduled with highest priority, they keep the tasks in $\mathbf{T}_1$ from executing, potentially leading to deadline misses.

4.3 PARAMETRIC UTILIZATION BOUNDS - RECAPITULATION

In the two previous sections, we showed that utilization bounds are often of limited value when describing the actual system utilization possible in a realistic setting. The reason is that they, by design, consider the worst-case over all possible setups. Moreover, we showed that *parameterising the utilization bounds* helps to provide more realistic results since they allow to exclude these worst-case scenarios from the consideration if they do not occur in the examined setting.

parametric utilization bound

For example, consider the utilization bound for *non-preemptive scheduling*, which is 0 in the general case [NBF+14]. The reason is that, if for a task τ_x the WCET C_x is larger than the smallest period T_1 in the system, task τ_1 is never schedulable and, thus, C_1 can be arbitrary small and T_x arbitrarily large, resulting in a utilization bound of 0. However, this example only has a theoretical value, since non-preemptive scheduling should not be considered in such a situation. Instead, a *parametric utilization bound* is able to capture the situation after excluding such extreme cases. Technically, the only previous results in this area by Andersson and Tovar [AT09], who consider RM-NP for a *Controller Area Network (CAN)* [Bos91], can also be considered as a parametric utilization bound. For CAN, the length of the transmitted messages is lower- and upper-bounded by the CAN configuration, which results in utilization bound of RM-NP of 25.8% for CAN 2.0A due to $\gamma \leq \frac{135}{47}$ and of 29.5% for CAN 2.0B due to $\gamma \leq \frac{160}{67}$. Hence, the maximum and minimum length of the messages is a parameter in this utilization bound. However, the analysis in [AT09], due to the restriction to CAN, only covered the case of $\gamma \geq 2$. With the analysis in Section 4.1, we showed that the utilization bound of RM-NP for CAN bus utilization is 50% if all the messages have the same length, i.e., $\gamma = 1$. Furthermore, we covered the general case by considering arbitrary values of γ. That the utilization bound is still 69.3% if $\gamma \leq \frac{1-ln(2)}{ln(2)} \approx 0.44269$ can be utilized, for instance, in *limited-preemptive scheduling* since it provides a bound on the maximum blocking time and, therefore, the length of the longest non-preemptive intervals. Hence, the bound can be used at design time to combine the advantages of non-preemptive and preemptive scheduling.

non-preemptive scheduling

Controller Area Network (CAN)

limited-preemptive scheduling

Section 4.2 showed that harmonic relations among the task periods can be used to achieve a general bound of 90% for *automotive task sets* which can be further improved by considering the utilization of tasks with specific periods as additional parameters. The evaluation of our parametric utilization bounds and schedulability tests showed that nearly 100% utilization is achieved for *preemptive scheduling*. Furthermore, very high utilizations for *non-preemptive scheduling* are possible, depending on the maximum length of non-preemptive intervals.

automotive task set

preemptive scheduling

non-preemptive scheduling

Some parametric utilization bounds are already provided in the literature. The utilization bounds for global multiprocessor scheduling that consider the maximum task utilization as a parameter, e.g., by Goossens et al. [GFB03] and Bertogna et al. [BCL05b], are parameterized. The utilization bound of 100% for *harmonic task sets* by Kuo and Mok [KM91] is technically also a parametric bound where the parameter is whether the task set has harmonic periods. Otherwise, the

harmonic task set

Liu and Layland Bound

Liu and Layland Bound of 69.3% holds, which itself is parametric in the number of tasks in the set.

It is noted that the schedulability of a task set will eventually be decided by a sufficient or at best exact schedulability test, since they have a higher precision than utilization bounds. However, utilization bounds are helpful tools at design time, as a precise schedulability analysis cannot be carried out for every change in the implementation, especially if a large number of tasks are assigned to the same processor. Both examples show that parametric utilization bounds are able to provide the required schedulability information in a more precise way than the more general bounds. Nevertheless, it is important that obtaining the additional parameters must have a *time complexity* similar to the parametric utilization bound or utilization based test to be exploitable in practical scenarios. Specifically, the complexity must be polynomial and not pseudo-polynomial or exponential.

time complexity

4.4 LINEAR TIME SPEEDUP FACTORS

speedup factor
static-priority scheduling
preemptive scheduling
non-preemptive scheduling
Deadline Monotonic
time complexity

After focusing on parametric utilization bounds, we now move our attention to *speedup factors*. Specifically, we consider speedup factors for *static-priority preemptive* and *non-preemptive scheduling* compared to optimal work-conserving algorithms, i.e., FP-P vs. EDF-P and FP-NP vs. EDF-NP. While previous work has mainly focused on determining speedup factors assuming exact schedulability tests and optimal priority assignment policies, we explore how the acquired speedup factor changes, based on both the priority assignment policy and the considered schedulability tests. To be precise, we show that a *Deadline Monotonic* priority assignment combined with *linear-time schedulability tests* leads to optimal speedup factors of FP-P vs. EDF-P as well as of FP-NP vs. EDF-NP for implicit-, constrained-, and arbitrary-deadline task sets. This surprising result reveals that speedup factors are neither able to capture the sub-optimality of linear-time sufficient tests compared to exact exponential-time tests, nor able to capture the sub-optimality of the Deadline Monotonic priority assignment in the arbitrary-deadline case. The results presented in this section appeared in *Schedulability and Optimization Analysis for Non-Preemptive Static Priority Scheduling Based on Task Utilization and Blocking Factors* in ECRTS 2015 *[BCH15]* and in *Exact Speedup Factors for Linear-Time Schedulability Tests for Fixed-Priority Preemptive and Non-preemptive Scheduling* in the Information Processing Letters, Volume 177 *[BCD+17]*.

4.4.1 SPEEDUP-OPTIMAL PRIORITY ASSIGNMENT

optimal algorithm
speedup factor
speedup-optimal

Optimality cannot only be defined according to schedulability but according to any metric, in our case the *speedup factor*. Hence, a scheduling algorithm is *speedup-optimal* if the speedup factor that it requires when combined with an exact schedulability test is not larger than the speedup factor required by any other scheduling algorithm. Again, this can be applied to different classes of task set and different scheduling algorithms. We explore the optimality of FP-P vs. EDF-P and of FP-NP vs. EDF-NP, i.e., the speedup-optimality of a static priority assignment compared to EDF in both the preemptive and the non-preemptive case. On the

	Preemptive		
Constraints	lower bound	upper bound (DM, linear)	upper bound (DM, expo.)
implicit-deadline	$1/ln(2) \approx 1.44269$ [LL73]		
constrained-deadline	$1/\Omega \approx 1.76322$ [DRB+09a]	$1/\Omega \approx 1.76322$ [CHL15b]	$1/\Omega \approx 1.76322$ [DRB+09a]
arbitrary-deadline	2 [DBB+15]	2 *[BCD+17]*	2 [DRB+09b]
	Non-Preemptive		
Constraints	lower bound	upper bound (DM, linear)	upper bound (DM, expo.)
implicit-deadline	$1/\Omega \approx 1.76322$ [DGC10]	$1/\Omega \approx 1.76322$ *[BCH15]*	$1/\Omega \approx 1.76322$ *[BCH15]*
constrained-deadline	$1/\Omega \approx 1.76322$ [DGC10]	$1/\Omega \approx 1.76322$ *[BCH15]*	$1/\Omega \approx 1.76322$ *[BCH15]*
arbitrary-deadline	2 [DBB+15]	2 *[BCD+17]*	2 [DGC10]

Table 4.2: The speedup factor lower bounds, upper bounds for linear-time schedulability tests, and upper bounds for pseudo-polynomial or exponential-time schedulability tests for FP-P vs. EDF-P as well as for FP-NP vs. EDF-NP. Note that the factors provided in ECRTS 2015 *[BCH15]* and ILP 177 *[BCD+17]* are part of the work presented here. *Adapted from [BCD+17].*

one hand, optimality of a priority assignment with respect to schedulability implies that it is also speedup-optimal for the same class of task sets. On the other hand, non-optimality with respect to schedulability does not necessarily imply that the priority assignment is not speedup-optimal.

The speedup factors for FP-P vs. EDF-P and FP-NP vs. EDF-NP are summarized in Table 4.2, presenting the lower bounds, the upper bounds for linear-time schedulability tests, and the upper bounds for pseudo-polynomial or exponential-time schedulability tests. The constant $\Omega \approx 0.56714$ is defined by the transcendental equation $\Omega = ln\left(\frac{1}{\Omega}\right)$. The table refers to the work that appeared in ECRTS 2015 *[BCH15]* and ILP 177 *[BCD+17]* with the related citations. However, the provided speedup factors are detailed in this section.

Deadline Monotonic (DM) scheduling is a *schedulability-optimal preemptive static-priority scheduling algorithm* (FP-P) for constrained-deadline task sets [LW82] as well as for implicit-deadline task sets [LL73], since Rate Monotonic and DM are the same for implicit deadline task sets. Hence, DM is also speedup-optimal in those cases. However, DM is not schedulability-optimal for arbitrary-deadline task sets under FP-P [Leh90] or for non-preemptive static-priority scheduling (FP-NP) of implicit-, constrained-, and arbitrary-deadline task sets [GRS96]. Nevertheless, DM is speedup-optimal for arbitrary-deadline task sets under both FP-P scheduling (Theorem 1 in [DBB+15]) and FP-NP scheduling (Theorem 7 in [DBB+15]). These theorems prove that the exact speedup factors are unchanged under DM compared to Audsley's algorithm (OPA) [Aud91]. The lower bounds for non-preemptive scheduling of implicit- and constrained-deadline task sets are deter-

mined with Audsley's algorithm [DGC10]. In the following, we show that DM is also speedup-optimal in these cases by providing the related upper bounds with a linear-time test. Furthermore, we provide the linear-time tests that lead to the tight speedup factor for the preemptive and the non-preemptive arbitrary-deadline case and briefly recap the linear-time tests for implicit- and constrained-deadline preemptive scheduling. Based on these results, we conclude that the upper bounds on the speedup factors for static-priority scheduling using DM are the same as the lower bounds (proven for exact tests and schedulability optimal priority assignment policies) even when simple linear-time schedulability tests are used. Therefore, we showed that the simplification from schedulability-optimal priority assignment to DM priority assignment is penalty-free in terms of speedup factors. Furthermore, the simplification from exact pseudo-polynomial or exponential schedulability tests to linear-time sufficient tests is penalty-free with respect to the speedup factor as well.

4.4.2 SPEEDUP FACTOR OF DM-NP FOR CONSTRAINED DEADLINES

Based on the hyperbolic tests in Theorem 4.2, we can provide the speedup factor of DM-NP with respect to EDF-NP for constrained-deadline task sets.

Theorem 4.23. *The speedup factor of non-preemptive Deadline Monotonic scheduling for task sets with constrained deadline is* $\frac{1}{\Omega} \approx 1.76322$ *with respect to non-preemptive Earliest Deadline First scheduling.*

Proof. The lower bound of $\frac{1}{\Omega} \approx 1.76322$ has been provided by Davis et. al [DGC10], who constructed an example that shows $\frac{1}{\Omega} \approx 1.76322$ is nearly reached for some task sets. Hence, we only provide the upper bound of $\frac{1}{\Omega} \approx 1.76322$ by showing that all task sets that are accepted by the exact schedulability test for EDF-NP on a processor with speed 1 will be accepted for DM-NP on a processor with speed $\frac{1}{\Omega} \approx 1.76322$ as well. We use contrapositive to show that if a task τ_k is not accepted by our schedulability test in Theorem 4.2, it is also not accepted by the exact schedulability test for EDF-NP on a processor with speed Ω.

If $\prod_{i=1}^{k-1}(U_i + 1) \geq 2$ we know that $\sum_{i=1}^{k-1} U_i \geq \ln 2$, which directly results in the speed-up factor of $\frac{1}{\ln 2} < 1.76322$.

If $\prod_{i=1}^{k-1}(U_i + 1) < 2$ we know that $\sum_{i=1}^{k-1} U_i < 1$ and $C_k^* \geq 0$. From the proof of Theorem 4.2 we know that the minimum value C_k^* that ensures that the schedulability test fails for a task τ_k can be constructed by solving the linear programming in Eq. (4.13). From Eq. (4.11) we know that for the extreme case

$$\prod_{i=1}^{k-1}(U_i + 1) = \frac{2}{\left(\frac{C_k^*}{D_k} + 1\right)} \tag{4.63}$$

By the definition of $B(t)$ for EDF-NP in Eq. (2.15), $B(D_k) = \max_{\forall i, D_i > t}\{C_i\}$ which is identical to B_k under DM-NP. If a task is not accepted by the schedulability test in Theorem 4.2 we know, that

$$\begin{aligned}
&\frac{B_k(D_k) + dbf(D_k)}{D_k} = \frac{B_k + \sum_{i=1}^{k} max\left\{0, \left\lfloor \frac{t-D_i}{T_i} \right\rfloor + 1\right\} C_i}{D_k} \\
&= \frac{B_k + C_k + \sum\limits_{\tau_i \in hp_2(\tau_k)} C_i + \sum\limits_{\tau_i \in hp_1(\tau_k)} max\left\{0, \left\lfloor \frac{t-D_i}{T_i} \right\rfloor + 1\right\} C_i}{D_k} \\
&\geq \frac{\widehat{C_k} + \sum_{i=1}^{k-1} max\left\{0, \left\lfloor \frac{t-T_i}{T_i} \right\rfloor + 1\right\} C_i}{D_k} \geq \frac{\widehat{C_k} + \sum_{i=1}^{k-1} t_i U_i}{D_k} \\
&\overset{1}{>} \frac{C_k^* + \sum_{i=1}^{k-1} t_i U_i}{t_k} = \frac{1}{\prod_{i=1}^{k-1}(U_i + 1)} = \frac{1 + \frac{C_k^*}{D_k}}{2}
\end{aligned}$$

where $\overset{1}{>}$ follows by Theorem 4.3 and Observation 4.4. For notational brevity, let $\frac{C_k^*}{D_k} = x$. Therefore, $\frac{1+\frac{C_k^*}{D_k}}{2} = \frac{1+x}{2}$.

We have to find the infimum $\sum_{i=1}^{k-1} U_i$ to ensure $\prod_{i=1}^{k-1}(U_i + 1) > \frac{2}{x+1}$. Since the arithmetic mean is always larger than or equal to the geometric mean, it follows that $\left(\frac{\sum_{i=1}^{k-1} U_i}{k-1} + 1\right)^{k-1} \geq \prod_{i=1}^{k-1}(U_i + 1)$. Furthermore, $\left(\frac{\sum_{i=1}^{k-1} U_i}{k-1} + 1\right)^{k-1} \leq e^{\sum_{i=1}^{k-1} U_i}$, where e is the Euler number and the right-hand side represents the case when k goes to ∞. From $\prod_{i=1}^{k-1}(U_i + 1) > \frac{2}{x+1}$, we conclude

$$\sum_{i=1}^{k-1} U_i > ln\left(\frac{2}{1+x}\right) \tag{4.65}$$

While $\frac{1+x}{2}$ is an increasing function of x, $ln\left(\frac{2}{1+x}\right)$ is a decreasing function of x. As a result,

$$\inf_{0 \leq x < 1} \left\{\max\left\{\frac{x+1}{2}, ln\left(\frac{2}{1+x}\right)\right\}\right\} = \Omega \tag{4.66}$$

The infimum occurs for the intersection of these two functions, i.e., when $\frac{2}{x+1} = ln\left(\frac{2}{1+x}\right)$. Hence, the proof is concluded by

$$\begin{aligned}
&\max\left\{\frac{dbf(D_k) + B_k(D_k)}{D_k}, \sum_{i=1}^{k-1} U_i\right\} \\
&= \max\left\{\frac{\widehat{C_k} + \sum\limits_{i=1}^{k-1} dbf_i(D_k)}{D_k}, \sum_{i=1}^{k-1} U_i\right\} \\
&> \max\left\{\frac{2}{x+1}, ln\left(\frac{2}{1+x}\right)\right\} \geq \Omega
\end{aligned}$$

□

Theorem 4.23 directly provides the following corollary:

Corollary 4.24. *The speedup factor of non-preemptive Rate Monotonic scheduling for task sets with implicit deadline is* 1.76322 *with respect to non-preemptive Earliest Deadline First.*

4.4.3 LINEAR-TIME SCHEDULABILITY TESTS

When exploring the linear-time schedulability tests for DM, we implicitly assume that the tasks are indexed in order of non-decreasing relative deadlines, i.e., $D_1 \leq D_2 \leq D_3 \leq \cdots \leq D_n$. We focus on testing the schedulability of task τ_k in linear time under the assumption that the first $k-1$ tasks are already verified to be schedulable under DM. We note that all tests for *static-priority scheduling* in this section can be efficiently implemented by using appropriate data structures to amortize the overall time complexity to $O(n)$ for testing *all* n tasks with similar reasons as discussed in Section 4.1 regarding Theorem 4.2.

TECHNICAL PRELIMINARIES

We first introduce the techniques that are used to prove the speedup factors in the case of arbitrary-deadline task sets. It is important to realize that the the maximum blocking time $B(D_k)$ under EDF-NP is never smaller than the maximum blocking time B_k under DM-NP. Furthermore, $B(D_k) = B_k$ unless a lower priority task with the same deadline and a long execution time exists.

Regarding EDF, we assume the more general form of the demand bound test presented in Eq. (2.14) for both preemptive and non-preemptive scheduling, i.e., $\sum_{i=1}^{n} U_i \leq 1$ and

$$dbf(t) + B(t) \leq t \quad \forall t \geq D_1 \tag{4.68}$$

In the in the preemptive case we set $B(D_k) = B_k = 0$. To show that a static-priority algorithm has a speedup factor of ρ, it is sufficient to prove that any task set $\mathbf{T}'$ that is *unschedulable* according to some schedulability test for FP-P (FP-NP) scheduling is also unschedulable under EDF-P (EDF-NP) on a processor whose speed has been reduced by ρ, i.e., scaled by a factor of $1/\rho$. Hence, to prove an upper bound on the speedup factor ρ for a linear-time schedulability test for preemptive or non-preemptive DM scheduling, we need to show that failure of the schedulability test implies either

$$\sum_{i=1}^{n} U_i > \frac{1}{\rho} \tag{4.69}$$

or

$$\frac{dbf(D_k) + B(D_k)}{D_k} > \frac{1}{\rho} \tag{4.70}$$

which implies that the task set cannot be scheduled under EDF-P (EDF-NP) on a processor of speed $1/\rho$. We observe the following relationship for the DM priority order:

$$\frac{dbf(D_k) \; + \; B(D_k)}{D_k} \geq \frac{C_k + \sum_{i=1}^{k-1} C_i + B_k}{D_k} \tag{4.71}$$

Furthermore, any lower bound on a speedup factor for FP-P versus EDF-P (FP-NP versus EDF-NP, respectively) provided for an exact test for FP-P (FP-NP, respectively) is also valid for any sufficient test for the same scheduling algorithm. The reason is that an exact test dominates any sufficient test for the same algorithm, since there are no task sets which are deemed schedulable according to the sufficient test that are not also deemed schedulable by the exact test. We provide a set of speedup factor upper bounds for simple linear-time sufficient schedulability tests. In addition, we show that the speedup factors are also tight (exact) for the linear-time tests, since these upper bounds are the same as the lower bounds (and exact values) previously published for exact tests.

PREEMPTIVE DM SCHEDULING

IMPLICIT-DEADLINES The Liu and Layland bound [LL73] in Eq. (2.4) provides a schedulability test by verifying $\sum_{i=1}^{k} U_i \leq k(2^{1/k} - 1)$. As EDF-P can schedule all implicit-deadline tasks sets with $\sum_{i=1}^{k} U_i \leq 1$ [LL73], this directly provides a linear-time schedulability tests with an exact speedup factor of $1/\ln 2$.

CONSTRAINED-DEADLINES The linear-time test provided by Chen et al. in Section 5.1 of [CHL15b] can be exploited to determine the schedulability of τ_k by evaluating the following condition in hyperbolic form

$$\left(\frac{C_k + \sum_{\tau_i \in hp_2(\tau_k)} C_i}{D_k} + 1 \right) \prod_{\tau_i \in hp_1(\tau_k)} (1 + U_i) \leq 2 \tag{4.72}$$

In Eq. (4.72), $hp_1(\tau_k)$ are the tasks in $hp(\tau_k)$ with periods less than D_k, therefore empty in DM order, and $hp_2(\tau_k)$ are the tasks in $hp(\tau_k)$ with periods greater than or equal to D_k. The upper bound speedup factor for this test is $1/\Omega \approx 1.76322$ as shown in Theorem 2 in [CHL15b]. Since this is the same as the lower bound proven in [DRB+09a], the bound is also exact.

ARBITRARY-DEADLINE The linear-time test we utilize is a weaker form of the response time upper bounds provided by Davis and Burns in Eq. (26) in [DB08] and Bini et al. in Eq. (11) in [BNR+09]. The schedulability can be verified by verifying whether

$$D_k \geq \frac{C_k + \sum_{i=1}^{k-1} C_i}{1 - \sum_{i=1}^{k-1} U_i} \tag{4.73}$$

If task τ_k cannot pass the test in Eq. (4.73), it follows that

$$1 < \frac{C_k + \sum_{i=1}^{k-1} C_i}{D_k} + \sum_{i=1}^{k-1} U_i \tag{4.74}$$

Hence, either (i) $\frac{C_k+\sum_{i=1}^{k-1} C_i}{D_k} > 0.5$, (ii) $\sum_{i=1}^{k-1} U_i > 0.5$, or both (i) and (ii) hold. This directly implies that EDF-P cannot schedule the task set on a processor of speed 0.5, and therefore 2 is an upper bound on the speedup factor. Since the lower bound provided in [DBB+15] is also 2, the speedup factor is exact.

NON-PREEMPTIVE DM SCHEDULING

IMPLICIT- AND CONSTRAINED-DEADLINES The upper bound of $1/\Omega \approx 1.76322$ on the speedup factor for linear-time schedulability test when comparing DM-NP to EDF-NP was provided in Section 4.4.2. Since the lower bound shown in [DGC10] and [DBB+15] is the same, the speedup factor is exact.

ARBITRARY-DEADLINES We use is a weaker form of the linear-time response time upper bounds by Davis and Burns in Eq. (33) in [DB08] as well as by Bini et al. in Eq. (14) in [BNR+09]. The schedulability can be verified by evaluating whether

$$D_k \geq \frac{B_k + C_k + \sum_{i=1}^{k-1} C_i}{1 - \sum_{i=1}^{k-1} U_i} \tag{4.75}$$

Hence, if task τ_k cannot pass the above test, we conclude that

$$1 < \frac{B_k + C_k + \sum_{i=1}^{k-1} C_i}{D_k} + \sum_{i=1}^{k-1} U_i \tag{4.76}$$

Following the same logic as in the preemptive case, either (i) $\frac{B_k+C_k+\sum_{i=1}^{k-1} C_i}{D_k} > 0.5$ or (ii) $\sum_{i=1}^{k-1} U_i > 0.5$ or both (i) and (ii) hold, again implying that the task set is not schedulable under EDF-NP on a processor of speed 0.5, i.e., the upper bounded speedup factor is 2. Since the the lower bound provided in [DBB+15] is the same, this is also exact.

4.5 PITFALLS OF SPEEDUP FACTORS AND UTILIZATION BOUNDS

utilization bound
parametric utilization bound
speedup factor

The results we presented in this Chapter so far are rather surprising. On the one hand, we showed that *utilization bounds* can be very pessimistic since they cover corner cases that may be excluded by *parametric utilization bounds*, i.e., if a small set of easily determined parameters is used to enhance the utilization bound. On the other hand, we showed that the *speedup factors* of linear-time schedulability tests for DM are identical to the factors that can be obtained by the dominating exact tests. This holds for all three classes of task set (implicit-, constrained- and arbitrary-deadline) as well as for both preemptive and non-preemptive scheduling (FP-P and FP-NP). Furthermore, DM is not even an optimal priority assignment with respect to schedulability for arbitrary-deadline task sets. Therefore, speedup factors are neither able to capture the sub-optimality of the schedulability test

nor, when considering preemptive arbitrary-deadline task sets or non-preemptive task sets, of the priority assignment.

This raises some serious questions regarding speedup factors, utilization bounds, and capacity augmentation bounds that we want to cover in the remainder of this chapter. While these three metrics, referred to as *the resource augmentation factors and bounds*, have been widely adopted and accepted by the real-time scheduling research community and are the *de facto* standard theoretical tools to compare scheduling algorithms as well as schedulability tests, how researchers and designers should view or use these theoretical results is not always clear. In addition to the examples previously provided in this chapter, we found a number of surprising results and related ways how these metrics can be misinterpreted or misunderstood in the literature. Hence, we try to provide perspective on the use of these metrics and help avoid common pitfalls by guiding researchers on their meaning and interpretation. Specifically, we want to shed light on the following questions:

1. What is the actual meaning of resource augmentation factors and bounds and how should they be interpreted?
2. If Algorithm $\mathcal{A}$ has a better resource augmentation factor or bound than Algorithm $\mathcal{B}$, does this also mean that the performance of Algorithm $\mathcal{A}$ is *always* better than that of Algorithm $\mathcal{B}$?
3. Enforcement may be used in algorithm design to achieve good resource augmentation factors or bounds. Can these enforcements result in design pitfalls that reduce the performance?
4. Are resource augmentation factors meaningful when they refer to an algorithm that is not optimal?
5. Is it possible to enhance the information provided by resource augmentation factors and bounds to give a broader perspective on performance?

We answer these questions and present our key observations based on several research results for different models and scheduling problems. The results presented in the remainder of this Chapter appeared in *On the Pitfalls of Resource Augmentation Factors and Utilization Bounds* in ECRTS 2017 [CBH+17a].

4.5.1 THE MEANING AND INTERPRETATION OF AUGMENTATION FACTORS

While utilization bounds, speedup factors, and capacity augmentation bounds have been widely used in the literature to theoretically quantify the performance of scheduling algorithms and schedulability tests, they *focus entirely on the worst-case scenario* and quantify it via a single value. However, this way to define the quantification metrics can make the augmentation bounds poorly suited to distinguishing between the performance of different algorithms or tests. These algorithms or tests may have an identical worst-case performance, but a very different performance in the average-case or across a broad spectrum of other cases. One reason is that the worst-case scenario may be a specific corner case

that is far removed from practical interest. We first consider uniprocessor static-priority preemptive scheduling of implicit-deadline task sets as an example to illustrate this behaviour. Afterwards, we extend our view to constrained- and arbitrary-deadline task sets and to non-preemptive scheduling.

The seminal utilization bound of $\ln 2 \approx 69.3\%$ for RM scheduling of periodic tasks by Liu and Layland [LL73] in 1973 directly leads to a speedup factor of $1/ln(2) \approx 1.44269$ with respect to EDF-P. However, a stochastic analysis by Lehoczky et al. [LSD89] in 1989 showed that the average case behaviour is much better and that the average breakdown utilization is 88%. If the task utilization is uniformly distributed, the breakdown utilization is even higher, over 90%, as shown by Bini and Buttazzo [BB05] in 2005.

Furthermore, a number of schedulability tests for RM-P that analytically dominate and are more precise than the Liu and Layland Bound have been established. In 1995, Burchard et al. [BLO+95] presented a test that also considers the ratios of task periods. In 1997, Han and Tyan [HT97] proposed a task transformation technique that converts a set of periodic tasks into a corresponding harmonic task set, such that T_i is an integer multiple of T_j if $T_i \geq T_j$. The utilization bound provided by Han and Tyan [HT97] analytically dominates the Liu and Layland bound [LL73] and the bound by Burchard et al. [BLO+95]. To improve the utilization bound, the harmonic relationship of the task periods was further exploited by Kuo et al. [KCL+03]. Based on the ratio r of the minimum task period to the maximum task period, in 1998 Lauzac et al. [LMM98a] proposed a utilization bound of $\ln r + 2/r - 1$ if $1 \leq r \leq 2$. This bound is $\ln 2$, the same as the Liu and Layland bound, if r is 2. Furthermore, Bini and Buttazzo [BBB01] presented the hyperbolic bound $\prod_{\tau_i \in \tau}(1 + U_i) \leq 2$ in 2001. More general utilization based tests in hyperbolic form have been recently provided by Chen et al. [CHL15b]. While the improvements in the other utilization bounds are based on characteristics[2] of the task set, the Liu and Layland utilization bound of $\ln 2$ is independent of the task parameters. However, the following worst-case scenario for RM-P, provided by Liu and Layland [LL73], is valid for all of the tests mentioned above when n is sufficiently large:

- $T_1 = D_1 = 1$, $C_1 = (2^{\frac{1}{n}} - 1)$
- $T_i = T_{i-1} + C_{i-1}$, $C_i = (2^{\frac{1}{n}} - 1)T_i, \forall i = 2, 3, \ldots, n-1$.
- $T_n = T_{n-1} + C_{n-1}$, $C_n = (2^{\frac{1}{n}} - 1)T_n + \varepsilon T_n$ where $\varepsilon > 0$ is arbitrarily small.

This task set, denoted by $\mathbf{T}^{RM}$, is not schedulable by RM-P since task τ_n misses its deadline. U_{sum} of $\mathbf{T}^{RM}$ is $n(2^{\frac{1}{n}} - 1) + \varepsilon = \ln 2 + \varepsilon$ for $n \to \infty$. Hence, all the schedulability tests mentioned above, i.e, [LL73; LSD89; BLO+95; HT97; BBB01; KCL+03; LMM98a; CHL15b], conclude that $\mathbf{T}^{RM}$ is not schedulable under RM-P. Therefore, we know that the speedup factor with respect to EDF-P of any of the above schedulability tests is $\frac{1}{\ln 2}$, since all the schedulability tests mentioned above analytically dominate the Liu and Layland bound of $\ln 2$. Furthermore, the lower bound on the speedup factor of RM-P compared to EDF-P is also $\frac{1}{\ln 2}$ since τ^{RM} is schedulable under EDF-P at speed s as long as $\sum_{\tau_i \in \tau^{RM}} \frac{U_i}{s} \leq 1$.

2 Note that therefore the tests in [LL73; LSD89; BLO+95; HT97; BBB01; KCL+03; LMM98a; CHL15b] can all be seen as parametric utilization bounds.

Hence, for the comparison of RM-P with respect to the optimal scheduling algorithm EDF-P, each of the schedulability tests presented in [LL73; LSD89; BLO+95; HT97; BBB01; KCL+03; LMM98a; CHL15b] is a *speedup-optimal* schedulability test for implicit-deadline task sets. In other words, they all have the minimum possible speedup factor for the class of scheduling algorithms considered, i.e., static-priority preemptive scheduling. This is the case despite the fact that of the tests mentioned above only the test in [LSD89] is an exact test while the others are merely sufficient. Furthermore, it is well known that in terms of schedulability they perform very differently, which can be demonstrated via empirically evaluating the acceptance ratio of synthetic task sets.

speedup-optimal

Therefore, by analyzing the very restricted but important example of RM-P, we could show the lack of discrimination between schedulability tests when assessed using speedup factors. This observation is similar to the one we made in Section 4.4 when we considered DM scheduling in both preemptive and non-preemptive cases. Our results in Table 4.2 show the lack of discrimination between exact tests, with pseudo-polynomial or exponential time complexity, and some corresponding linear-time sufficient schedulability tests when speedup factor are considered. Furthermore, although DM is neither an optimal priority assignment policy for arbitrary-deadline task sets under FP-P nor for FP-NP scheduling considering any of the three classes of task sets, it is a *speedup-optimal* static-priority scheduling strategy in all these cases.

The problem that speedup factors are not able to discriminate between different schedulability tests and scheduling algorithms is not restricted to the sporadic task model or to uniprocessor scenarios. For uniprocessor mixed-criticality scheduling, a series of papers by Baruah showed that EDF-VD and its generalization have the same speedup factor for different task models [Bar16a; Bar16b; BBD+15]. For multiprocessor partitioned static-priority scheduling of constrained-deadline and arbitrary-deadline sporadic task sets, an analysis by Chen [Che16c] shows that the achieved speedup factors for exponential-time exact schedulability tests and for polynomial-time sufficient schedulability tests are the same.

Observation 1: *Speedup factors, utilization bounds, and capacity augmentation bounds often lack the power to discriminate between the performance of different scheduling algorithms and schedulability tests even though the performance of these algorithms and tests may be very different when viewed from the perspective of empirical evaluation.*

The reason is that utilization bounds, capacity augmentation bounds, and speedup factors *only* consider the worst-case corner cases. Therefore, a constant factor or bound, e.g., $\frac{1}{\ln 2}$ or $\ln 2$, does not have any implication for the performance of the algorithm or test in typical or average cases. While the structure of the corner cases may be easily captured by simple tests, e.g., the Hyperbolic Bound [BBB01] or the Liu and Layland bound [LL73], the more common cases do not contribute to the metric. This means that the more common case is simply ignored in the metric, even if the algorithm or the test has relatively poor performance in the broad space of possible task sets. As a result, a very simple algorithm or very imprecise sufficient schedulability tests may be classified as excellent or even optimal as long as they are able to handle these corner cases well. This explains the results listed in Table 4.2 where DM is always classified

as *speedup-optimal*, although its performance, in terms of schedulability across a wide range of task sets, is not necessarily good.

Observation 2: *Speedup factors, utilization bounds and capacity augmentation bounds should only be considered for their negative implications, since these metrics only provide information on performance in the worst case.*

Observation 3: *Proving that an algorithm or test has the best possible (or optimal) speedup factor or bound for that class of algorithms does not imply that the algorithm or test cannot be substantially improved upon.*

Hence, an algorithm that does not have a constant speedup factor, utilization bound, or capacity augmentation bound may still perform reasonably well. Nevertheless, it may also perform terribly in the worst case. Furthermore, a constant bound or factor only ensures that the algorithm performs at least at some minimum level in the worst case. Even showing that for the studied problem an algorithm or test has the best possible (or optimal) speedup factor or bound [BBD+12; BBM+09] does *not* imply that its performance will necessarily be good in other cases. Hence, as researchers, we should not be satisfied with just deriving algorithms or tests that have optimal speedup factors or bounds. These can rather be seen as a step towards developing algorithms and tests that provide improved performance in practice while retaining some guaranteed worst-case performance.

4.5.2 BETTER SPEEDUP FACTORS DO NOT IMPLY A DOMINANCE RELATION

We again use the simplest setting, two implicit-deadline task sets under uniprocessor RM-P, to examine the relationship between dominance results based on speedup factors and utilization bounds, and empirical schedulability based on acceptance ratios in evaluation. To be precise, we compare the *Hyperbolic Bound* (HB) by Bini et al. [BBB01] and the *Quadratic Bound* (QB) by Davis and Burns [DB08] as well as Bini et al. [BNR+09].

Hyperbolic Bound

Quadratic Bound

Only examining speedup factors and utilization bounds, one would conclude that the HB is better than the QB. However, this does not hold true when considering different settings of $\frac{T_1}{T_2}$. Let $U_1 = 0.4$. Utilizing the HB in Eq. (2.5), task τ_2 is schedulable under RM-P if $U_2 \leq 2/1.4 - 1 \approx 42.8\%$. For the QB in Eq. (2.6), task τ_2 is schedulable under RM-P if $0.4 + U_2 + \frac{0.4T_1 - 0.4^2 T_1}{T_2} = 0.4 + U_2 + 0.24\frac{T_1}{T_2} \leq 1$. Thus, the QB is better if $\frac{T_1}{T_2} > 0.715$ and the HB is better otherwise. This shows that the HB and the QB are incomparable, which means neither dominates the other. We demonstrate the impact of different distributions of T_1/T_2 by evaluations for four different configurations. We always set T_1 to 1 and T_2 was chosen randomly uniform from (a) $[1, 1.5]$, (b) $[1, 2]$, (c) $[1.5, 2]$, and (d)$[1, 10]$. In each configuration and for each utilization level, we generated 10,000 task sets:

- For a given U_{sum}, i.e., the target utilization level, U_1 was chosen randomly uniform from $[0, U_{sum}]$ and U_2 was set to $U_{sum} - U_1$.
- T_1 was always set to 1 and C_1 was set to $U_1 T_1$.

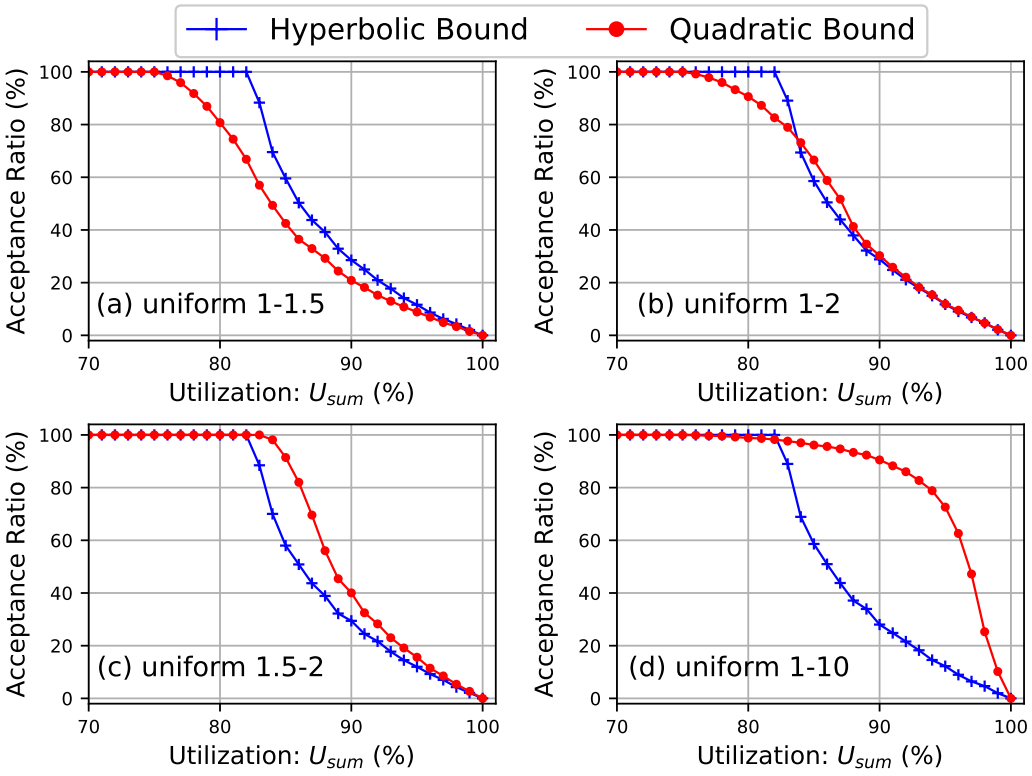


Figure 4.6: Comparison of the Hyperbolic Bound (HB) and the Quadratic Bound (QB) for RM uniprocessor scheduling for two tasks and different uniform distributions of T_2/T_1. *Adapted from [CBH+17a].*

- T_2 was chosen from a uniform distribution specified by the four configurations (a) to (d) and C_2 was set to U_2T_2.

We compared the performance of the HB and the QB based on the *acceptance ratio* with respect to a given task set utilization level U_{sum}.

The evaluation results are displayed in Figure 4.6. The acceptance ratio of the QB is highly dependent on the configuration for T_2/T_1, while the acceptance ratio of the HB is in general independent of these settings. Furthermore, the QB is worse than HB if T_2/T_1 is small and superior if T_2/T_1 is large. Hence, the results in Figure 4.6 show that the relative performance of these two schedulability tests is highly dependent on the task set parameters, although the HB has a superior speedup factor and utilization bound to the QB. Furthermore, a study by Burns and Baxter [BB06] shows that in many real-word settings the periods in a task set differ by two or three orders of magnitude. Hence, it seems preferable that a test perform well if the periods of the considered tasks are not too close.

Observation 4: *A scheduling algorithm or schedulability test with a worse speedup factor or utilization bound may perform (much) better in practice than another algorithm or test with a superior speedup factor or utilization bound, dependent on the task set configurations and parameters used. Conclusions on the relative merits of algorithms or tests drawn from speedup factors or utilization bounds can therefore be in direct contradiction with those drawn from empirical performance evaluation.*

The reason for this apparent contradiction between dominance in terms of speedup factors or utilization bounds and performance observed from evaluation is that speedup factors and utilization bounds depend solely on corner cases, which may have parameters that rarely occur or are far removed from practical settings, while an empirical evaluation usually covers a broader spectrum of values. Further examples can be found in the literature:

- When comparing global-RM scheduling to an optimal algorithm, the forced forward method by Baruah et al. [BBM+10] and the test by Bertogna and Cirinei [BC07b] both have a speedup factor of 3. On the other hand, the schedulability tests based on the k2U and k2Q frameworks by Chen et al. [CHL15b; CHL16b], that use a bounded carry-in [GSY+09], have worse speedup factors, i.e., 3.62143 for k2U and 3.73 for k2Q. Nevertheless, the evaluation results in [CHL16b; CHL15a] show that k2U and k2Q have a much better performance than the tests in [BBM+10; BC07b].
- On the one hand, the capacity augmentation bound for scheduling implicit-deadline DAG task sets on m homogeneous processors under federated scheduling was proved to be $2 - \frac{1}{m}$ by Li et al. [LCA+14]. On the other hand, Jiang et al. [JLG+16] developed a decomposition algorithm that assigns a relative deadline for each DAG subtask and has a capacity augmentation bound in the range of $[2 - \frac{1}{m},\ 4 - \frac{2}{m})$. Based on the capacity augmentation bounds, one may conclude that federated scheduling dominates the decomposition algorithm. However, according to the evaluation in [JLG+16], the decomposition algorithm outperforms other algorithms for the considered experimental settings.

Observation 5: *Identifying regions of dominance in terms of schedulability, between scheduling algorithms and schedulability tests provides valuable information in addition to theoretical analysis in terms of speedup factors or bounds, and empirical evaluations in terms of acceptance ratios.*

4.5.3 SPEEDUP FACTORS BASED ON ENFORCED ALGORITHMS

enforcement

Sometimes *enforcements* are used in the design of scheduling algorithms to simplify the structure of the scheduling problem. Such enforcements may make it easier to derive a speedup factor for the scheduling algorithm or schedulability test, especially if they are strong and/or are applied at an early stage of the algorithm. However, this may lead to poor performance in practical settings when compared to other algorithms or tests that have worse speedup factors or no speedup factor at all. We illustrate this effect with two examples, one for the scheduling of self-suspending tasks on a uniprocessor, and the other for tasks that share resources on multiprocessors.

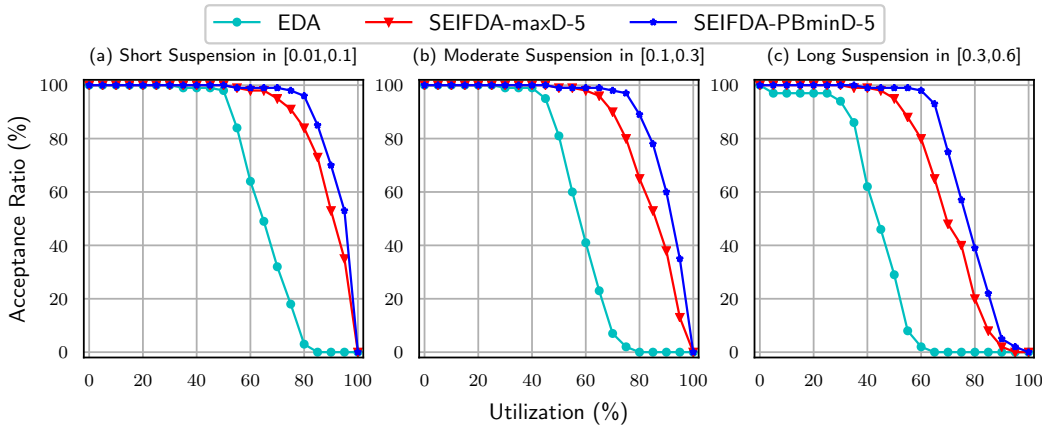


Figure 4.7: Comparison of the acceptance ratio for EDA [CL14] and SEIFDA *[BHC+16]* in Section 6.1, showing that enforcements in the algorithm design can lead to a huge performance loss. *Adapted from [CBH+17a]*.

ONE-SEGMENTED SELF-SUSPENSION

We consider the scheduling of implicit-deadline one-segmented self-suspending tasks,[3] each described as $\tau_i = ((C_{i,1}, S_i, C_{i,2}), T_i = D_i)$, where two execution segments $C_{i,1}$ and $C_{i,2}$ are separated by a suspension interval S_i. A *fixed-relative-deadline* (FRD) scheduling strategy, proposed by Chen and Liu [CL14], assigns two individual relative deadlines $D_{i,1}$ and $D_{i,2}$ to the first and second computation segment, respectively. Based on the resulting deadlines, the computation segments are scheduled using EDF. Due to the implicit-deadline assumption we know $D_{i,1} + D_{i,2} + S_i = T_i$. The main question for FRD is how to assign $D_{i,1}$ and $D_{i,2}$, i.e., how to partition the *execution interval* $T_i - S_i$. We compare the Equal Deadline Assignment (EDA) by Chen and Liu [CL14] and the *Shortest Execution Interval First Deadline Assignment* (SEIFDA), which appeared in RTNS 2016 *[BHC+16]* and is presented in Section 6.1.

fixed-relative-deadline

execution interval

SEIFDA

The most intuitive FRD strategy is the proportional deadline assignment provided by Liu et al. [LCT+14]: $D_{i,1} = \frac{C_{i,1}}{C_i} \cdot (T_i - S_i)$ and $D_{i,2} = \frac{C_{i,2}}{C_i} \cdot (T_i - S_i)$. While seeming very reasonable, the speedup factor is unbounded as shown by Chen and Liu [CL14]. They propose to instead use EDA: $D_{i,1} = D_{i,2} = (T_i - S_i)/2$, and derive a speedup factor of 3 compared to an optimal scheduling algorithm for one-segmented self-suspension task sets as well as a speedup factor of 2 compared to any other FRD strategy. The speedup factor compared to other FRD strategies easily follows from the strong enforcement for the relative deadlines, since both segments have half of the execution interval $T_i - S_i$ to execute the necessary workload and any other FRD scheduling strategy could assign at most twice the relative deadline that EDA assigns to any segment. However, EDA jeopardizes the schedulability by assigning the same relative deadline to both segments, even if $C_{i,1} = \varepsilon$ and $C_{i,2} = C_i - \varepsilon$ for a very small $\varepsilon > 0$.

3 Regardless of the fact that the argumentation here is mainly based on the results provided in Section 6.1, we provide a brief introduction into the topic here for the convenience of the reader, since this enables us to keep the flow of argumentation.

The main problem with the proportional assignment strategy is that the deadline assignment for one task is independent from the deadline assignment for other tasks. This problem is tackled by the SEIFDA algorithm that assigns the relative deadlines of the tasks in decreasing order of their execution intervals $T_i - S_i$ while taking into account the previously assigned deadlines. For the shorter execution segment, we assume it is $C_{i,1}$ for ease of explanation, the possible values of $D_{i,1} \in [C_{i,1}, T_i - S_i]$ are calculated and afterwards one of these values is chosen according to one of three strategies: (i) the minima value (minD), (ii) the maxima value (maxD), or (iii) the minima value larger than $\frac{C_{i,1}}{C_i} \cdot (T_i - S_i)$, called proportionally bounded minD (PBminD). While all three assignment strategies have the same speedup factor as EDA, they clearly outperform EDA in terms of acceptance ratios. This is shown in Figure 4.7[4] for SEIFDA-maxD and SEIFDA-PBminD when considering randomly generated task sets with 10 tasks under different settings for the length of the suspension interval. The reason for the significantly better performance is that SEIFDA does not enforce the deadlines but chooses them dependent on the other tasks.

MULTIPROCESSOR RESOURCE SHARING

Another example that shows how enforcement can compromise performance comes from scheduling algorithms for tasks that share r resources and execute on a platform with m homogeneous processors. We assume that $r \leq m$ and consider the simplified execution structure introduced by Andersson and Raravi [AR14], i.e., each task has only one critical section where it may access shared resources guarded by semaphores. Hence, each sporadic task τ_i has three execution segments with WCETs $C_{i,1}^N$, C_i^{Crit}, and $C_{i,2}^N$ representing the part before the critical section, the critical section itself, and the part after the critical section. Furthermore, $C_i = C_{i,1}^N + C_i^{Crit} + C_{i,2}^N$. We compare two algorithms for implicit-deadline task sets with known speedup factors.

- Andersson and Raravi [AR14] developed the *LP-EE-vpr* algorithm that has a speedup factor of $4 \cdot (1 + \lceil \frac{r}{m} \rceil) = 8$ in the given setting, since $r \leq m$. Note that we simplify the formulas from [AR14] to match the case we analyse while the model in [AR14] is more general. They create m virtual processors with speed $\frac{1}{2}$ to schedule the two non-critical sections of task τ_i with relative deadlines of $\frac{C_{i,1}^N}{C_i} \cdot \frac{T_i}{2}$ and $\frac{C_{i,2}^N}{C_i} \cdot \frac{T_i}{2}$ under partitioned EDF-P. Furthermore, m virtual processors with speed $\frac{1}{2}$ are created to schedule the critical section C_i^{Crit} with a relative deadline $\frac{T_i}{2}$. Critical sections guarded by one semaphore are executed exclusively on one virtual processor using EDF-NP.
- The resource-oriented partitioned PCP, called *ROP-PCP*, by Huang et al. [HYC16] has a speedup factor of $11 - \frac{6}{m+1}$. It uses two dedicated subsets of the m processors to execute the critical and the non-critical sections individually. Although *ROP-PCP* is applicable for tasks with multiple critical sections, we assume one critical section to compare with *LP-EE-vpr*.

4 The results shown in Figure 4.7 are a subset of the results presented in Section 6.1 and in *[BHC+16]*, i.e., the results where in the test for SEIFDA five periods are calculated exactly before an over-approximation takes place.

Comparing the speedup factors, one might assume that *LP-EE-vpr* outperforms *ROP-PCP*. However, this is not the case since *LP-EE-vpr* uses enforcements early in the algorithm to provide the speedup factor while *ROP-PCP* first provides the algorithm and then analyzes the resulting speedup factor. To be precise, the enforcements of *virtualization* at slower speeds in *LP-EE-vpr* and shortened relative deadlines substantially reduce the schedulability. The sufficient schedulability test used in the comparison can be found in [HYC16]. Since no schedulability test for *LP-EE-vpr* was provided by Andersson and Raravi [AR14], we used two necessary conditions for the schedulability of the non-critical execution and the critical section, respectively, based on demand bound functions. Details on how the necessary conditions were constructed are in the Appendix.

A comparison between the sufficient test for *ROP-PCP* and the necessary condition for *LP-EE-vpr* is shown in Figure 4.8. We considered a system with 8 processors, 80 tasks, and 1 resource per task that is accessed at most once. Three ratios $\alpha = 5$, $\alpha = 10$, and $\alpha = 20$ for the length of non-critical sections to the length of critical sections were evaluated, i.e., $C^{Crit} = \frac{1}{1+\alpha} \cdot C_i$ and $C^N_{i,1} + C^N_{i,2} = \frac{\alpha}{1+\alpha} \cdot C_i$. Detailed configurations can be found in [HYC16]. Furthermore, we compared to a necessary condition for *gEDF-vpr* by Anderson and Easwaran [AE10], which is the predecessor of *LP-EE-vpr* and the first algorithm with a proven speedup factor, i.e., $12(1 + 3r/4m) = 21$ when $r = m$, and also uses strong enforcement techniques. Figure 4.8 shows that the acceptance ratio for *LP-EE-vpr* drops dramatically from a utilization of roughly $m \times 25\%$ and is 0 by $m \times 28\%$ utilization in all cases while *gEDF-vpr* drops even sooner, before $m \times 20\%$ utilization. In contrast, the acceptance ratio for *ROP-PCP* decreases more slowly and is still over 50% when the utilization is at $m \times 76\%$ for $\alpha = 20$, $m \times 61\%$ for $\alpha = 10$, and $m \times 36\%$ for $\alpha = 5$. These results clearly show that the enforcements, i.e., assigning stringent relative deadlines to subtasks and enforcing slow virtual processors in *LP-EE-vpr*, have significant performance drawbacks even though the speedup factor obtained is better than that for *ROP-PCP*. Hence, both *gEDF-vpr* and *LP-EE-vpr* have barely any chance to schedule task sets with a total utilization above a certain small threshold value that depends on the exact configuration. Note that we adopt necessary conditions for *LP-EE-vpr* and *gEDF-vpr* to show that the performance loss for those methods is due to the early and restrictive enforcements and not due to a sufficient schedulability test that performs badly.

Observation 6: *Adding enforcements tailoring the design of a scheduling algorithm or test to facilitate the derivation of a bounded speedup factor can be counterproductive; it may severely compromise performance in practical settings.*

4.5.4 RELATIVE SPEEDUP FACTORS

The speedup factor for a scheduling algorithm or schedulability test is typically provided with respect to an optimal algorithm for the same class of problem. For example, the speedup factor of $\frac{1}{\ln 2}$ for RM-P is provided with respect to EDF-P, which is an optimal scheduling algorithm for implicit-deadline sporadic task sets on a uniprocessor. However, speedup factors may also be derived relative to another non-optimal scheduling algorithm or schedulability test. One example

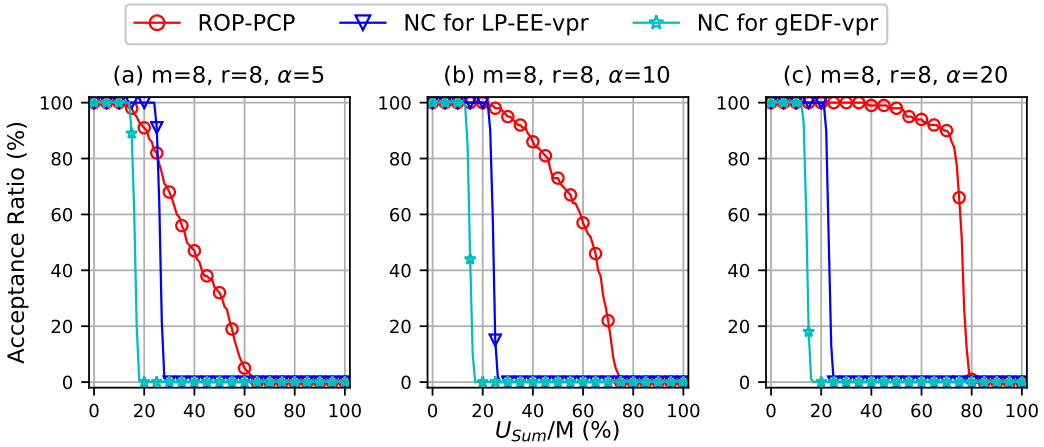


Figure 4.8: Comparison of the necessary condition for *LP-EE-vpr* [AR14], the necessary condition for gEDF-vpr [AE10], and the sufficient schedulability test for *ROP-PCP* [HYC16], showing that enforcements made to guarantee a speedup factor can lead to a huge performance loss. *Adapted from [CBH+17a].*

is the set of speedup factors for FP-NP with respect to EDF-NP, which is an optimal work-conserving algorithm but not in general optimal, that is provided in Section 4.4 and in [DGC10].

Furthermore, bounds on the values of speedup factors can be composed from existing results. If an algorithm $\mathcal{O}$ strictly dominates another algorithm $\mathcal{X}$ in terms of schedulability, we know that $\rho^{\mathcal{O}\to\mathcal{X}} = 1$ and $\rho^{\mathcal{X}\to\mathcal{O}} > 1$. This relation holds if $\mathcal{O}$ is an optimal algorithm for the class of problem. Further, if two algorithms $\mathcal{X}$ and $\mathcal{Y}$ are incomparable, then $\rho^{\mathcal{X}\to\mathcal{Y}} > 1$ and $\rho^{\mathcal{Y}\to\mathcal{X}} > 1$, i.e., there are non-trivial speedup factors in both directions. Such speedup factors may be combined into graphs or chains. Let $\mathcal{O}$ dominate $\mathcal{Z}$ which in turn dominates $\mathcal{Y}$. This results in the following relationships between the speedup factors:

$$max(\rho^{\mathcal{Y}\to\mathcal{Z}}, \rho^{\mathcal{Z}\to\mathcal{O}}) \leq S^{\mathcal{Y}\to\mathcal{O}} \leq S^{\mathcal{Y}\to\mathcal{Z}} \times S^{\mathcal{Z}\to\mathcal{O}}$$

Note that while the first inequality holds due to the dominance relationships, the second holds regardless. In addition, if $\mathcal{O}$ dominates both $\mathcal{Z}$ as well as $\mathcal{X}$, and $\mathcal{Z}$ and $\mathcal{X}$ are incomparable, the following relations hold:

$$S^{\mathcal{Z}\to\mathcal{X}} \leq S^{\mathcal{Z}\to\mathcal{O}} \text{ and } S^{\mathcal{X}\to\mathcal{Z}} \leq S^{\mathcal{X}\to\mathcal{O}}$$

When a speedup factor $S^{\mathcal{X}\to\mathcal{Z}}$ for some algorithm or test $\mathcal{X}$ is determined relative to some other algorithm $\mathcal{Z}$ that dominates it, the results must be interpreted carefully if $\mathcal{Z}$ is not optimal for the considered scheduling problem. In particular, if the speedup factor of $\mathcal{Z}$ is unbounded with respect to the optimal algorithm $\mathcal{O}$, i.e. $\rho^{\mathcal{Z}\to\mathcal{O}} = \infty$, the speedup factor of $\mathcal{X}$ relative to the optimal algorithm is also unbounded, i.e., $\rho^{\mathcal{X}\to\mathcal{O}} = \infty$ regardless of the value of $\rho^{\mathcal{X}\to\mathcal{Z}}$. Two examples based on recent studies are given below to illustrate the pitfalls in using relative speedup factors, rather than those immediately grounded by optimal algorithms. Note that in this respect utilization and capacity augmentation bounds are more robust, since their reference is the capacity of the processor.

FEDERATED SCHEDULING

federated scheduling

Li et al. [LCA+14] and Baruah [Bar15b; Bar15a; Bar15c] proposed to use *federated scheduling* for sporadic real-time tasks with intra-task parallelism, detailed as directed acyclic graphs (DAGs), on multiprocessor platforms. In federated scheduling, a task is either executed sequentially on a single processor while sharing this processor with other tasks or it is assigned to a processor exclusively. The existing speedup factor results for federated scheduling on m identical processors are:

- The capacity augmentation bound of the federated scheduling algorithm for implicit-deadline task sets provided in [LCA+14] is 2. Therefore, the speedup factor with respect to *an optimal scheduling algorithm* is also 2.
- The speedup factor of the federated scheduling algorithms for constrained-deadline task sets provided in [Bar15b; Bar15c] is $3 - 1/m$ with respect to *an optimal federated scheduling algorithm*.
- The speedup factor of the federated scheduling algorithms for arbitrary-deadline task sets provided in [Bar15a; Bar15c] is $4 - 2/m$ with respect to *an optimal federated scheduling algorithm*.

These speedup factors are effectively the same as for the EDF-FFID partitioning algorithm [BF05; BF06; BF07b] for sporadic task sets. Therefore, based on these results, Baruah concluded for the algorithms in [Bar15b; Bar15a; Bar15c]:

> Baruah [Bar15b; Bar15a; Bar15c]: *... in terms of the speedup metric, there is no loss in going from the three-parameter sporadic tasks model to the more general sporadic DAG tasks model.*

However, the resulting speedup factors and conclusions that are provided in [Bar15b; Bar15a; Bar15c] are only meaningful if the speedup factor of an optimal federated scheduling algorithm with respect to an optimal scheduling algorithm is bounded for this problem. Therefore, the constant speedup factors of 3 and 4 become less useful due to the following result:

> Chen [Che16b]: *... in terms of the speedup metric with respect to any optimal scheduling algorithm, federated scheduling strategies do not yield any constant speedup factors for constrained-deadline task systems with DAG structures.*

As a result, the relative speedup factors derived in [Bar15b; Bar15a; Bar15c] cannot be related back to an optimal scheduling algorithm and this relation is effectively unbounded.

UNIPROCESSOR SELF-SUSPENSION SYSTEMS

Huang et al. [HCZ+15] studied constrained-deadline dynamic self-suspending tasks under task-level static-priority scheduling. In Theorem 1 in [HCZ+15] they showed that several heuristic priority assignments have unbounded speedup factors, including Rate Monotonic (RM), Deadline Monotonic (DM), and Laxity Monotonic (LM). Huang et al. [HCZ+15] proposed to use OPA [Aud91] together

with an OPA-compatible schedulability test [DB11b] and showed that this approach has a speedup factor of 2 with respect to the *optimal static-priority schedule*. Unfortunately, it has been shown that the (commonly used) existing scheduling algorithms do not yield any constant speedup factors in relation to an optimal algorithm for this problem:

> Chen [Che16a]: *For dynamic self-suspending task systems, . . . the speedup factor for any FP preemptive scheduling, compared to the optimal schedules, is not bounded by a constant if the suspension time cannot be reduced by speeding up. Such a statement of unbounded speedup factors can also be proved for Earliest Deadline First (EDF), Least Laxity First (LLF), and Earliest Deadline Zero Laxity (EDZL) scheduling algorithms.*

REMARKS ON RELATIVE SPEEDUP FACTORS

These two concrete examples show that arguments based on relative speedup factors may be undermined and inconclusive if the reference scheduling strategies cannot be related back to optimal algorithms. In both examples, the proposed algorithms as well as the reference class of algorithms have unbounded speedup factors with respect to an optimal algorithm.

Observation 7: *Where relative speedup factors are used in relation to a non-optimal algorithm or class of algorithms, then great care needs to be taken in the interpretation of the results. If the reference algorithm has an unbounded speedup factor with respect to optimal solutions, then the speedup factors may not be that meaningful.*

Nevertheless, relative speedup factors can still be meaningful if (i) the reference scheduling strategies are well-accepted, defined, and constrained by the system properties, or (ii) the reference strategy facilitates comparison with an optimal algorithm. For example, in some cases work-conserving non-preemptive uniprocessor scheduling may be the only implementation option and EDF-NP is an optimal scheduling strategy for sporadic real-time tasks. Hence, the speedup factors between FP-NP and EDF-NP in [DGC10; DBB+15] as well as in Section 4.4.2 can be useful. Furthermore, although they are both not optimal algorithms, they can be related back to EDF-P via speedup factors [DTG+15].

4.6 PARAMETRIC AUGMENTATION FUNCTIONS

As illustrated in this chapter, using a single factor or bound to represent the theoretical quantification of the performance of scheduling algorithms or schedulability tests can prove inadequate. While it would be preferable, it is not always possible to show the analytical or theoretical dominance of an algorithm. Hence, we propose a more nuanced way to compare algorithms using a *parametric augmentation function* $\mathcal{A}(\vec{x})$, defined as follows:

parametric augmentation function

- $\vec{x}$ is a vector of user-defined parameters of interest, like $\max_{\tau_i \in \tau} U_i$ or $\max_{\tau_i \in \tau} \frac{Critical_i}{T_i}$, to classify different (troublesome) cases.

- The parametric augmentation function $\mathcal{A}(\vec{x})$ represents the augmentation factor (or the utilization bound) respecting all of the parameters described in the vector $\vec{x}$.

The concept of parametric augmentation functions can be traced back to Liu and Layland's seminal utilization bound for RM-P which is parametric in the number of tasks: $n(2^{\frac{1}{n}} - 1) \geq \ln 2 \approx 69.3\%$. Similarly, work on speedup factors for DM scheduling of constrained-deadline task sets by Davis et al. [DRB+09a] explored a speedup factor upper bound that is parametric in n, i.e., Theorem A.2 in [DRB+09a] uses the hyperbolic bound to derive a speedup factor that is a function of n.

parametric utilization bound

The *parametric utilization bounds* we provided in Chapter 4.1 and in Chapter 4.2 show how much parametric augmentation functions can improve the general augmentation functions. In some cases, they may be unbounded unless some parameter is controlled. For instance, Davis et al. [DTG+15] compare non-preemptive uniprocessor scheduling (FP-NP and EDF-NP) against an optimal algorithm (EDF-P) based on parametric speedup factors. The speedup factors were obtained as a function of C_{max}/D_{min}, where C_{max} is the largest WCET of any task and D_{min} is the smallest relative deadline. Without this parameter, the speedup factors of $1 + C_{max}/D_{min}$ for EDF-NP and of $2 + C_{max}/D_{min}$ for FP-NP are unbounded as C_{max}/D_{min} could be an arbitrarily large value. In some practical settings this value can be relatively small, highlighting the utility of such an approach. Similarly, in Chapter 4.1 we use the blocking factor γ which can be arbitrarily large as well. Liu et al. [LSG+16] implicitly used parametric augmentation functions when studying EDF-VD scheduling for mixed-criticality systems with degraded quality guarantees. They showed that the augmentation factor depends on two task set dependent constants, denoted by α and λ, with the worst-case speedup factor reducing to 4/3.

Observation 8: *Parametric augmentation functions can reveal more detailed and nuanced information about the actual performance of schedulability tests or scheduling algorithms across a wide range of parameter values, including practical settings. In some cases parameterized augmentation functions are essential to avoid singularities and hence unbounded results due to unrealistic combinations of parameter values.*

In the next section, we detail how theoretical comparisons of two priority assignments for uniprocessor FP-P scheduling, i.e., RM and Slack Monotonic, may be performed using parametric augmentation functions. The results also show that parametric augmentation functions can be helpful when designing empirical evaluations and workload generators aimed at providing a comprehensive comparison between different scheduling algorithms and schedulability tests.

4.7 PARAMETRIC AUGMENTATION FUNCTION FOR RATE MONOTONIC VS. SLACK MONOTONIC

To explore the advantages of parameterized augmentation functions, we examine the theoretical comparisons of two priority assignment schemes for uniprocessor FP-P scheduling. We consider:

- Rate Monotonic (RM): If $T_i < T_j$, then task τ_i has higher-priority than τ_j;
- Slack Monotonic (SM): If $T_i - C_i < T_j - C_j$, then task τ_i has higher-priority than task τ_j.

In both cases, ties are broken arbitrarily. We consider sporadic task sets in which $D_i \geq T_i$ for every task τ_i in the task set. Instead of solely using utilization bounds (or speedup factors) for comparing these two algorithms (or the schedulability tests of these two algorithms), we show why it is more meaningful to compare the algorithms and tests across a broader spectrum. We first recall polynomial-time schedulability tests for RM from the literature.

Theorem 4.25 (Chen, Huang, and Liu [CHL15b]). *Suppose that $D_k = fT_k$ where f is a positive integer. Task τ_k is schedulable under RM scheduling if*

$$\prod_{i=1}^{k}(1 + U_i/f) \leq (f+1)/f. \tag{4.77}$$

The utilization bound can be further expressed as

$$\sum_{i=1}^{k-1} U_i \leq f \ln\left(\frac{f+1}{f(1+U_k/f)}\right) \qquad \text{and} \qquad U_k \leq 1 \tag{4.78}$$

Theorem 4.26 (Lehoczky [Leh90]). *Suppose that $D_k = fT_k$ where f is a positive integer. Task τ_k is schedulable under RM scheduling if*

$$\sum_{i=1}^{k} U_i \leq \begin{cases} k\left(2^{\frac{1}{k}} - 1\right) & \text{if } f = 1 \\ f(k-1)\left(\left(\frac{f+1}{f}\right)^{\frac{1}{k-1}} - 1\right) & \text{if } f = 2, 3, \ldots \end{cases} \tag{4.79}$$

$$\text{When } k \to \infty, \text{ the utilization bound is } \sum_{i=1}^{k} U_i \leq f \ln((f+1)/f) \tag{4.80}$$

We next derive sufficient schedulability tests for SM.

Theorem 4.27. *Suppose that $D_k = fT_k$ with $f > 0$. Task τ_k is schedulable under SM scheduling if*

$$\sum_{i=1}^{k} U_i \leq 1 \qquad \text{and} \qquad U_k + (1 - U_k + f)\sum_{i=1}^{k-1} U_i \leq f \tag{4.81}$$

Proof. We implicitly assume that $\sum_{i=1}^{k} U_i \leq 1$. Using the response time upper bounds given by both Bini et al. [BPD15] and Chen et al. [CHL16b], a simple schedulability test for task τ_k validates whether

$$\frac{C_k + \sum_{i=1}^{k-1} C_i - \sum_{i=1}^{k-1} U_i C_i}{1 - \sum_{i=1}^{k-1} U_i} \leq D_k$$

By the definition of SM, we know that

$$T_i - C_i \leq T_k - C_k \Rightarrow T_i(1 - U_i) \leq T_k(1 - U_k) \Rightarrow (1 - U_i) \leq \frac{T_k}{T_i}(1 - U_k)$$

Hence, we get

$$C_i - C_i U_i = C_i(1 - U_i) \leq C_i \frac{T_k}{T_i}(1 - U_k) = T_k U_i (1 - U_k)$$

As a result, the following inequality holds:

$$\frac{C_k + \sum_{i=1}^{k-1} C_i - \sum_{i=1}^{k-1} U_i C_i}{1 - \sum_{i=1}^{k-1} U_i} \leq \frac{C_k + T_k(1 - U_k)\sum_{i=1}^{k-1} U_i}{1 - \sum_{i=1}^{k-1} U_i}$$

When $D_k = fT_k$, a sufficient schedulability condition for SM is:

$$\frac{C_k + T_k(1 - U_k)\sum_{i=1}^{k-1} U_i}{1 - \sum_{i=1}^{k-1} U_i} \leq fT_k.$$

Dividing both sides by T_k leads to

$$U_k + (1 - U_k)\sum_{i=1}^{k-1} U_i \leq f\left(1 - \sum_{i=1}^{k-1} U_i\right) \Rightarrow U_k + (1 - U_k + f)\sum_{i=1}^{k-1} U_i \leq f$$

□

Theorem 4.28. *Suppose that $D_k = fT_k$ with $f \geq 1$. Task τ_k is schedulable under SM scheduling if*

$$\sum_{i=1}^{k} U_i \leq \frac{f}{f+1} \tag{4.82}$$

Proof. This theorem is proved by finding the infimum $\sum_{i=1}^{k} U_i$ under the condition that $U_k + (1 - U_k + f)\sum_{i=1}^{k-1} U_i > f$ using the schedulability condition in Theorem 4.27. Note that the condition $\sum_{i=1}^{k} U_i \leq 1$ automatically holds. This is equivalent to the following problem:

$$\text{minimum } x + y \quad \text{s. t. } x + (1 - x + f)y = f \text{ and } 0 \leq x \leq 1,$$

where x is U_k and y is $\sum_{i=1}^{k-1} U_i$. By $x + (1 - x + f)y = f$, we can write y as $(f - x)/(1 - x + f)$. Therefore, $x + y$ is $x + (f - x)/(1 - x + f)$. The first order derivative of $x + y$ is

$$\frac{\partial}{\partial x}\left(x + \frac{f - x}{1 - x + f}\right) = 1 - \frac{1}{(1 - x + f)^2} \geq 0,$$

since $f \geq 1$ and $0 \leq x \leq 1$ in our assumption. Hence, $x + y$ is minimized for $x = 0$ and $y = f/(1 + f)$. □

COMPARING RM/SM BASED ON TRADITIONAL UTILIZATION BOUNDS

Let $f = \min_{\forall \tau_i}(D_i/T_i)$ for the rest of this section, with $f \geq 1$ due to the problem definition. Based on Eq. (4.80), we know that the utilization bound of RM is $\left(\lfloor f \rfloor \ln \frac{\lfloor f \rfloor + 1}{\lfloor f \rfloor}\right)$. Similarly, based on Eq. (4.82) the utilization bound of SM is $\frac{f}{f+1}$.

Lemma 4.29. $\frac{x+1}{x+2} - x\ln(1+\frac{1}{x}) \leq 0$ *for any positive integer* x.

Proof. Using Taylor series expansion, $\ln(1+z)$ can be over-approximated as $z - \frac{z^2}{2} + \frac{z^3}{3}$ when $-1 < z < 1$. Hence, for any $x \geq 2$, we get

$$\begin{aligned}&\frac{x+1}{x+2} - x\ln(1+\frac{1}{x}) \leq 1 - \frac{1}{x+2} - x(\frac{1}{x} - \frac{1}{2x^2} + \frac{1}{3x^3})\\ &= \frac{-1}{x+2} + \frac{1}{2x} - \frac{1}{3x^2} < 0\end{aligned}$$

where the last inequality is due to $\frac{-1}{x+2} + \frac{1}{2x} \leq 0$ for any $x \geq 2$. For $x = 1$, we get $\frac{2}{3} - 2\ln 1.5 \approx -0.144$, hence the lemma is proved. □

Theorem 4.30. *Suppose that* $\min_{\forall\tau_i}(D_i/T_i) = f$ *where* $f \geq 1$. *The utilization bound* $\left(\lfloor f \rfloor \ln \frac{\lfloor f \rfloor + 1}{\lfloor f \rfloor}\right)$ *in Eq. (4.80) for RM dominates the utilization bound* $\frac{f}{f+1}$ *in Eq. (4.82) for SM.*

Proof. Suppose that $\lfloor f \rfloor$ is x. Since the utilization bound $\frac{f}{f+1}$ in Eq. (4.82) is upper bounded by $\frac{x+1}{x+2}$, the theorem follows directly from Lemma 4.29. □

Since the utilization bound for RM is better than the utilization bound for SM for a given $f \geq 1$, this also holds for the speedup factor with respect to EDF-P.

COMPARING RM/SM BASED ON PARAMETRIC-UTILIZATION FACTORS

The dominance in Theorem 4.30 only results from the utilization bounds (or the augmentation factors) under a given $f = \min_{\forall\tau_i}(D_i/T_i)$. More precise schedulability tests may lead to different conclusions.

Theorem 4.31. *Suppose that* $f = \min_{\forall\tau_i}(D_i/T_i)$ *where* $f \geq 1$. *SM is a feasible scheduling algorithm for task* τ_k *if* $U_k \geq \frac{1+f-\sqrt{(1+f)^2-4}}{2}$ *and* $\sum_{i=1}^{k} U_i \leq 1$.

Proof. The test in Eq. (4.81) for SM evaluates whether the conditions $\sum_{i=1}^{k} U_i \leq 1$ and $U_k + (1 - U_k + f)\sum_{i=1}^{k-1} U_i \leq f$ hold. The first and second condition constrain $\sum_{i=1}^{k-1} U_i$ to be at most $1 - U_k$ and $\frac{f-U_k}{1+f-U_k}$, respectively. Since $\frac{\partial(1-U_k)}{\partial U_k} = -1$ and $\frac{\partial(\frac{f-U_k}{1+f-U_k})}{\partial U_k} = \frac{-1}{(f-U_k+1)^2} \geq -1$ for $f \geq 1$ and $0 \leq U_k \leq 1$, the intersection $1 - U_k = \frac{f-U_k}{1+f-U_k}$ defines which of the two conditions in Eq. (4.81) dominates the schedulability test. Let $U^*(f)$ be the intersection of U_k for a given f. By solving $1 - U_k = \frac{f-U_k}{1+f-U_k}$, we know that $U^*(f)$ is defined as $\frac{1+f-\sqrt{(1+f)^2-4}}{2}$. Hence, $U^*(1) = 1$, $U^*(2) \approx 0.382$, $U^*(3) \approx 0.268$, $U^*(4) \approx 0.209$, $U^*(5) \approx 0.172$, $U^*(6) \approx 0.146$, $\cdots$, $U^*(10) \approx 0.092$, etc.

Therefore, when $U_k \geq U^*(f)$ under Slack Monotonic scheduling, we know that $1 - U_k \leq \frac{f-U_k}{1+f-U_k}$ and the condition $\sum_{i=1}^{k} U_i \leq 1$ dominates the condition that $U_k + (1 - U_k + f)\sum_{i=1}^{k-1} U_i \leq f$. □

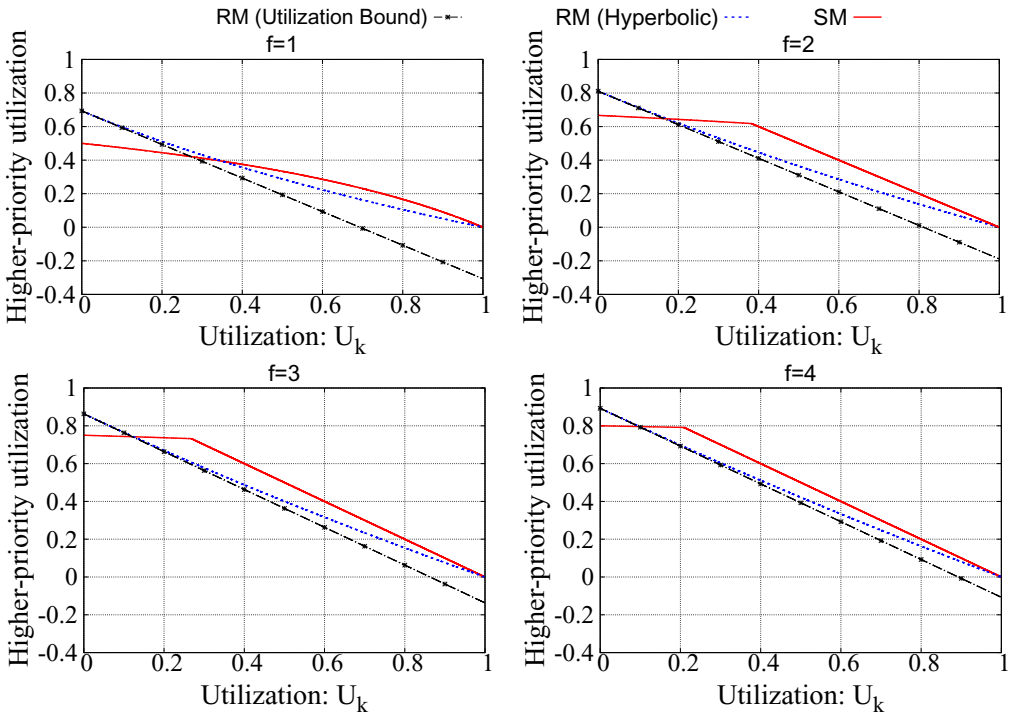


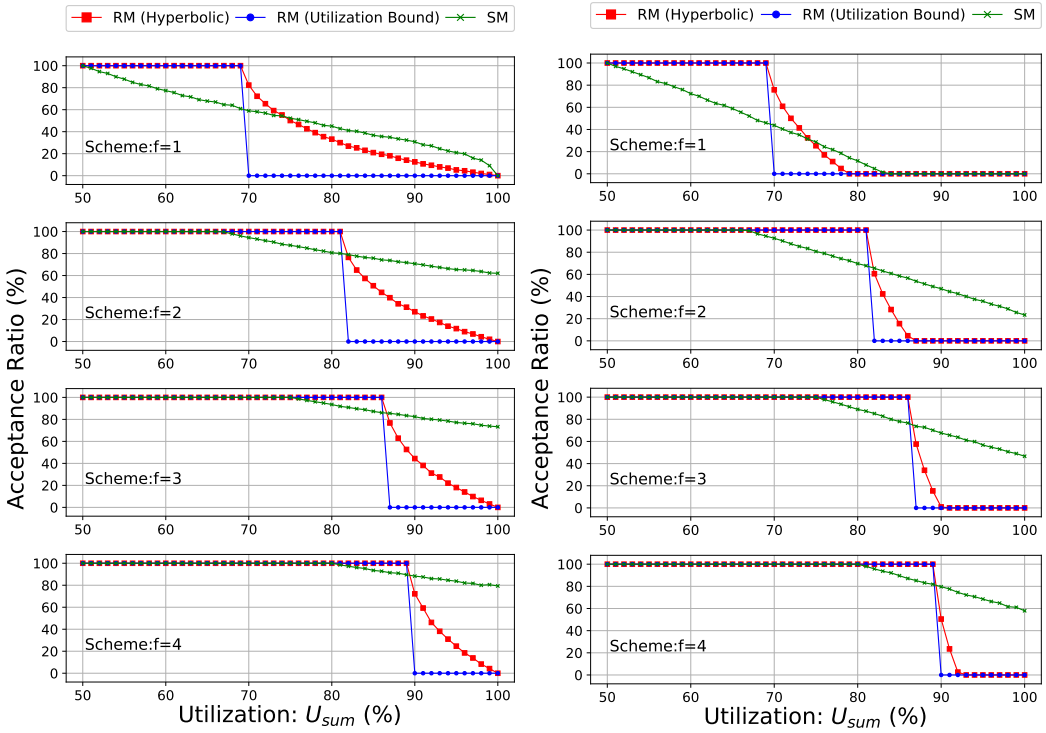
Figure 4.9: Theoretical comparison of SM and RM. Eq. (4.78) is denoted by RM (Hyperbolic), Eq. (4.80) is denoted by RM (Utilization Bound), and Eq. (4.81) is denoted by SM. *Adapted from [CBH+17a]*

This schedulability analysis shows that U_k has an important role. For the rest of this section, we assume that f is an integer for ease of comparison. Figure 4.9 provides the analytical results by comparing the conditions in Eq. (4.78) denoted by RM (Hyperbolic), Eq. (4.80) denoted by RM (Utilization Bound), and Eq. (4.81) denoted by SM. Figure 4.9 also shows that $U^*(2) \approx 0.382$, $U^*(3) \approx 0.268$, and $U^*(4) \approx 0.209$, since these are the values of U_k at the corner points on the line for SM. Furthermore, the utilization bound of SM is 100% when $U_k \geq U^*(f)$, as shown by the part of the line for SM with a 45 degree slope. As the y-axis measures total utilization for higher priority tasks, a line between (0,1) and (1,0) means 100% utilization.

As shown in Figure 4.9, the schedulability tests for RM in Eq. (4.78) and Eq. (4.80) are better than the test in Eq. (4.81) for SM when U_k is small. By contrast, the test for SM is better than the above tests for RM for larger values of U_k. Thus, conclusions on the analytical superiority of these tests for RM and SM should only be drawn when U_k is considered. This shows the importance of including U_k into the parametric utilization bound when testing the schedulability of task τ_k, i.e., $\vec{x}$ in the parametric augmentation function should include U_k.

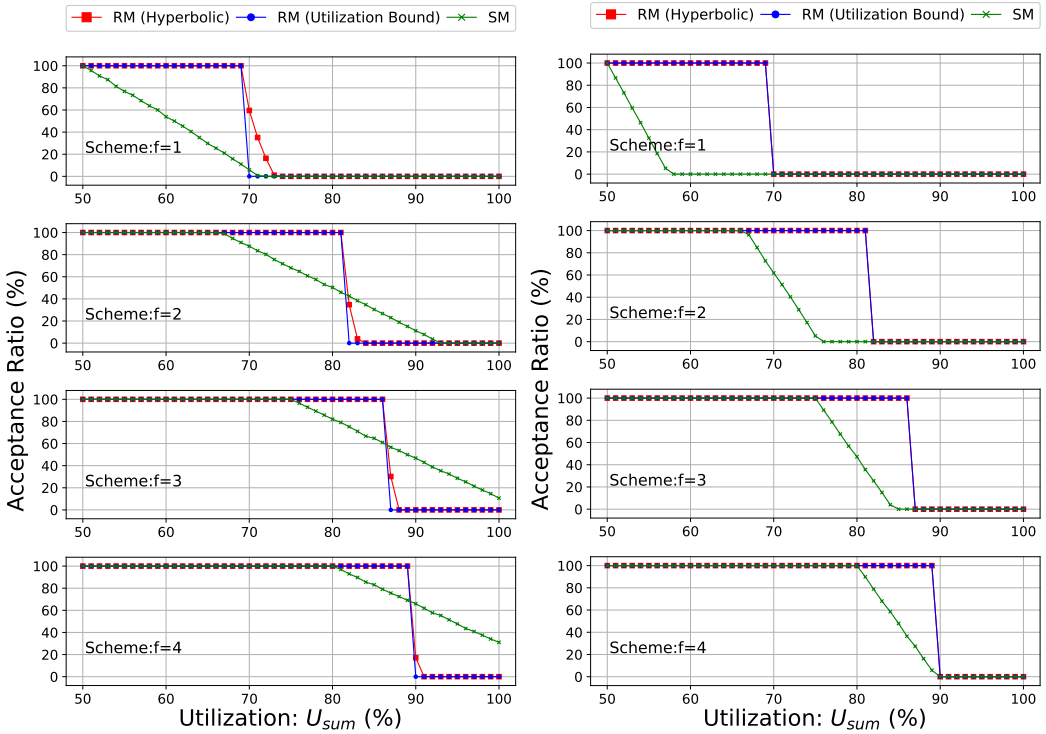
COMPARING SCHEDULABILITY TESTS BASED ON SYNTHETIC WORKLOAD

We conducted an evaluation for arbitrary-deadline sporadic task sets with k tasks, where we only considered the schedulability of task τ_k, to demonstrate the



(a) U_k: uniform distribution in (0, U_{sum})

(b) U_k: uniform distribution in (0, 0.5)



(c) U_k: uniform distribution in (0, 0.3)

(d) U_k: uniform distribution in (0, 0.1)

Figure 4.10: Experimental comparison of SM and RM scheduling. Eq. (4.78) is denoted as RM (Hyperbolic), Eq. (4.80) is denoted as RM (Utilization Bound), and Eq. (4.81) is denoted as SM. *Adapted from [CBH+17a]*

impact of different distributions of U_k. For $U_{sum} \geq 0.5$, we explored four different uniform distributions for U_k: (a) $[0, U_{sum}]$, (b) $[0, 0.5]$, (c) $[0, 0.3]$, (d) $[0, 0.1]$ where $\sum_{i=1}^{k-1} U_i = U_{sum} - U_k$ for each case. We tested $f = 1, f = 2, f = 3, f = 4$ for each of the configurations, generating 10000 task sets for each utilization level in each configuration. The performance was compared based on the *acceptance ratio* for the tests in Eq. (4.78), Eq. (4.80), and Eq. (4.81). The results are displayed in Figure 4.10. It shows that the acceptance ratios of the tests are highly dependent on both f and the distribution of U_k, i.e., the configuration used. Note that RM (Hyperbolic) and RM (Utilization Bound) have essentially the same performance when the range of values for U_k is small, i.e., $[0, 0.1]$, and thus the lines for RM (Hyperbolic) and RM (Utilization Bound) are identical in Figure 4.10(d).

The evaluation settings and configurations also consider the parameter U_k in the experimental setup since the parametric analysis showed that U_k plays a significant role. Hence, we get a much more comprehensive picture of the performance of the different algorithms and tests, showing how they vary with critical parameters.

4.8 SUMMARY AND CONCLUSIONS

speedup factor
utilization bound
capacity augmentation bound
parametric utilization bound

In this chapter, we studied the use of *speedup factors*, *utilization bounds*, and *capacity augmentation bounds*. First, we examined RM-NP and automotive task systems, providing *parametric utilization bounds* that significantly improved the general bounds. Afterwards, we took a closer look at the *speedup factors* of DM vs. EDF in both the preemptive and the non-preemptive case and showed that the speedup factors for linear-time and exponential-time schedulability tests are the same. These results motivated us to examine speedup factors, utilization bounds, and capacity augmentation bounds from a more general point of view. Through a series of examples, some of them by combining known results from the literature, we reached the following conclusions:

- These metrics often lack the power to discriminate between the performance of different scheduling algorithms and schedulability tests even though their performance may be very different in an empirical evaluation.
- These metrics should only be considered for their negative implications, since they may only provide information on performance in corner cases that may not be important in practical settings.
- Proving that an algorithm or test has an optimal speedup factor or bound for a class of algorithms or problems does not imply that the algorithm or test cannot be substantially improved. Furthermore, an algorithm or test with a worse speedup factor or bound may perform much better in practice. Therefore, conclusions solely based on speedup factors may directly contradict those drawn from empirical evaluation.
- Adding enforcements tailoring the design of an algorithm or test to facilitate the derivation of a bounded speedup factor can be counterproductive since it may severely compromise the performance in practical settings.

- Great care needs to be taken in interpreting the results when relative speedup factors are derived in relation to a non-optimal algorithm or class of algorithms. These results can be undermined if the reference algorithm has an unbounded speedup factor with respect to optimal solutions.
- Identifying regions of dominance between scheduling algorithms or schedulability tests in terms of schedulability provides valuable information. This information should be considered in addition to the information that can be provided by speedup factors or bounds as well as the information provided by empirical evaluations in terms of acceptance ratios.

parametric augmentation function
theoretical evaluation method

Resulting from our exploration, we recommend *parametric augmentation functions* as a *theoretical evaluation method* that is capable of revealing more detailed and nuanced information about the actual performance of schedulability tests or scheduling algorithms across a wide range of parameter values, including practical settings. We illustrated this technique by deriving such functions for two uniprocessor scheduling algorithms and schedulability tests, namely Rate Monotonic and Slack Monotonic. Furthermore, we showed that in some cases parameterized augmentation functions are essential to avoid singularities and hence unbounded results due to unrealistic combinations of parameter values.

speedup factor
utilization bound
capacity augmentation bound

Based on our studies of *speedup factors*, *utilization bounds*, and *capacity augmentation bounds*, our considered view is to *handle them with care*. While these theoretical metrics can provide useful information, there are also pitfalls that must be avoided. Problems may arise when algorithms are designed with speedup factors in mind, or when conclusions taking a positive perspective are drawn solely on the basis of these theoretical results. We welcome the additional information that theoretical metrics, particularly parametric augmentation functions, can provide. Nevertheless, we further recommend that any judgement on the practical utility or otherwise of scheduling algorithms or schedulability tests is backed up by a thorough performance evaluation that studies practical settings.

5

UNCERTAIN EXECUTION BEHAVIOUR

When examining *uncertain execution behaviour*, we focus on uncertainty regarding the *worst-case execution time* (WCET) of a task and not regarding other task parameters, e.g., an uncertain minimum inter-arrival time due to release jitter. An *uncertain execution behaviour* regarding WCET may stem from a variety of scenarios, among them software-based *fault tolerance* mechanisms, *mixed-criticality*, a reduced CPU frequency to prevent overheating, and dynamic voltage frequency scaling. We model such behaviour as a set of distinct execution modes with related WCETs, assuming that the WCET can be precisely determined for each mode, i.e., the uncertainty is not resulting from the worst-case execution time analysis. Furthermore, we assume that the WCET differs largely among the modes and that modes with a large execution time have a low probability to be executed, which means that a small WCET is the *normal* and a large WCET the *abnormal* case. If the situation is reversed, i.e., a large WCET is the normal case, standard schedulability analysis considering the largest WCET is sufficient. However, if a large WCET is less likely, simply assuming that tasks will always run abnormally would result in largely overestimating the necessary system resources and a not tolerable increase in hardware costs. Nevertheless, it must be ensured that execution modes with a large WCET do not destroy the necessary timing guarantees. Therefore, they must be analysed and handled properly.

uncertain execution behaviour
worst-case execution time
fault tolerance
mixed-criticality systems

One possible solution takes advantage of the fact that for many real-time systems, very rare deadline misses are acceptable. For instance, the safety standards in the industry, such as IEC-61508 [IEC10] and ISO-26262 [ISO00], require a low or very low probability of failure, e.g., due to deadline misses, but not a failure probability of 0. As a result, researchers have provided work that tries to quantify the deadline misses resulting from workload overloads due to rare events, e.g., in [KT12; QHE12; QNE13; HQE14; XHK+15]. Cause for such workload overloads may, for instance, be *intermittent faults* or (bursts of) *transient faults* and the additional execution time necessary when applying software-based *fault tolerance* mechanisms. The aforementioned work assumes that, in general, rare deadline misses are tolerable. Therefore, such work allows some tradeoff between the timeliness of the tasks and the necessary amount of system resources.

intermittent fault
transient fault
fault tolerance

Furthermore, *mixed-criticality systems* [Ves07] assume an uncertain execution behaviour with an ordered set of two or more distinct execution modes as well, but allow deadline misses only for a specific set of tasks that depends on the actual system mode. For convenience, we assume dual-criticality systems in the description, i.e., two disjunct sets of tasks and two execution modes. This concept of mixed-criticality is a result of the observation that, even if all tasks in the systems have real-time constraints, some tasks are *more important* for the system than others, e.g., they ensure the stability of the system. The system may switch from a *low-criticality* to a *high-criticality* mode, the time where this mode

mixed-criticality systems

switch happens is assumed to be given (or easily detectable), and the system is assumed to never return to the *low-criticality* mode. Moreover, to guarantee the timeliness of the *more important* tasks, the *not so important* tasks are often abandoned when a mode switch happens. The model of mixed-criticality as well as the related scheduling approaches and analysis have been criticized lately, for instance by Ernst and Di Natale [EN16] in "Mixed Criticality Systems - A History of Misconceptions?" as well as by Esper et al. [ENN+15] who asked "How realistic is the mixed-criticality real-time system model?". In particular, they pointed out that model and analysis do not fit the expectations of system engineers since 1) *not so important* tasks should not be abandoned, and 2) systems should return to the starting mode after a sufficient amount of time, e.g., after a rare event does not affect the system anymore.

Especially when considering such criticism, the connection and similarity between *mixed-criticality* and *fault tolerance* become apparent. Nevertheless, to the best of our knowledge, they have never been considered in the real-time systems research. This connection will be concretized in Section 5.1 before we introduce the model of *Systems with Dynamic Real-Time Guarantees*, which is suitable for systems with uncertain execution behaviour regarding the worst-case execution time in general and not limited to mixed-criticality. Similar to *mixed-criticality systems* with two modes, we assume a given task set partition into *not so important* tasks where rare deadline misses are acceptable, called *timing tolerable tasks*, and *more important* tasks that must always meet their deadline, called *timing strict tasks*. *Systems with Dynamic Real-Time Guarantees* ensure that the deadlines of all tasks are satisfied if the system runs normally. On the other hand, in case of a rare event, the *timing strict tasks* are still guaranteed to meet their deadlines while the *timing tolerable tasks* have at least bounded tardiness but may miss deadlines. We exploit static-priority scheduling and provide these guarantees in advance. During runtime, a *Systems with Dynamic Real-Time Guarantees* provides either

Systems with Dynamic Real-Time Guarantees

timing tolerable tasks

timing strict tasks

- *full timing guarantees* if all jobs meet their deadline, or
- *limited timing guarantees* if only the jobs of the *timing strict tasks* are guaranteed to meet their deadline while bounded tardiness is guaranteed for the *timing tolerable tasks*.

full timing guarantees

limited timing guarantees

Furthermore, in *Systems with Dynamic Real-Time Guarantees*, mode switches are detected by an online monitor and the time needed to return to *full timing guarantees* is approximated at runtime. A precise system definition, an exact schedulability test, and properties for uniprocessor systems are provided in Section 5.2. The definition is extended to *Multiprocessor Systems with Dynamic Real-Time Guarantees* in Section 5.3, also providing the schedulability test, both partitioned and semi-partitioned heuristics, and compensation techniques for intermittent faults by task migration.

Multiprocessor Systems with Dynamic Real-Time Guarantees

One important problem regarding uncertain execution behaviour is to determine the *worst-case deadline failure probability* of a task. Regardless, previous approaches are either fast but imprecise, i.e., analytical bounds, or not applicable for task sets with a reasonable size, i.e., job-level convolution-based techniques. Hence, we introduce an approach that utilizes task-level convolution and several optimization techniques to ensure the scalability to large task sets in Section 5.4.

worst-case deadline failure probability

5.1 DYNAMIC REAL-TIME GUARANTEES IN AN UNCERTAIN EXECUTION ENVIRONMENT

In this section, we first take a closer look at both uncertain execution behaviour induced by fault tolerance mechanisms and at the model of mixed-criticality before detailing their connection. As a result, we propose *Systems with Dynamic Real-Time Guarantees* as a general approach for systems with uncertain execution behaviour. Most parts of the argumentation presented in this section appeared in *Systems with Dynamic Real-Time Guarantees in Uncertain and Faulty Execution Environments* in RTSS 2016 *[BCH+16]* and in *Do Nothing, but Carefully: Fault Tolerance with Timing Guarantees for Multiprocessor Systems devoid of Online Adaptation* in PRDC 2018 *[BSC18]*.

Systems with Dynamic Real-Time Guarantees

5.1.1 MODELLING UNCERTAIN EXECUTION BEHAVIOUR

In many practical real-time systems, the physical environment and the system platform can impose *uncertain execution behaviour* and, therefore, reliability problems to the system. One cause is that continuous technology scaling has introduced multiple threats that reduce the reliability of computing hardware, not only considering the memory hierarchy but also in the logic components [Bau05; MWE+03; SKK+02]. Such reliability threats include transient faults, also called soft errors, aging, and process variations. To be able to neglect transient faults, which are results of electromagnetic interference and radiation [Bau05], recovery operations are necessary. Otherwise, a transient fault may corrupt the correct application execution state, and, therefore, lead to wrong execution results or even system failure. The related recovery mechanisms can be classified into hardware- and software-based techniques. Hardware-based techniques include spatial isolation, hardware redundancy with voting mechanisms, and remapping of logical components to a subset of hardware resources [ENN+15]. The alternative are software-based techniques like checkpointing, software redundancy with voting mechanisms, or re-execution of incorrectly executed jobs if an error was detected. These software-based techniques induce additional runtime for (at least partly) re-execution of an erroneous job [ENN+15], and the maximum execution time with and without performed fault recovery may differ largely, e.g., by a factor of ≈ 3 for two times re-execution compared to the fault-free execution. However, if software-based techniques are exploited, it is assumed that the fault rate is low and, hence, recovery only has to be performed rarely. The occasionally appearing faults can be modeled in different ways, depending on the considered systems and the cause of the faults. For instance, they may occur for each individual job with a certain probability or as a fault burst where a set of jobs is subjected to fault recovery, e.g., all jobs that are active in a certain time interval. Such faults can be modelled as overshoots and multiple publications examined the resulting worst-case response time (WCRT), the maximum settling time, or the number of deadline misses in such situations [KT12; QHE12; QNE13; HQE14; XHK+15].

uncertain execution behaviour

While the aforementioned work assumes rare deadline misses to be generally acceptable, in many practical situations deadline misses may be acceptable for

some tasks but not acceptable for others. For instance, in an unmanned aerial vehicle (UAV), the tasks in the *more important* flight control system should always meet their deadline while for the *not so important* surveillance system, occasional deadline misses are tolerable. The importance of tasks is considered in *mixed-criticality systems*, which have been introduced by Vestal in 2007 [Ves07], where the tasks are partitioned into *high-criticality* and *low-criticality task*.[1] The system is assumed to start its lifetime in *low-criticality mode* and to switch to *high-criticality mode* at some point in time, which is typically unknown but assumed to be easily detectable. While in *low-criticality mode*, timeliness for all tasks is guaranteed. After the mode switch, tasks are executed with a larger WCET and only timing guarantees for the *high-criticality tasks* are provided. To ensure these guarantees, scheduling approaches for mixed-criticality often perform online adaptation, e.g., changing task parameters or priorities, and neglect the low-criticality tasks in high-criticality mode, i.e., they are either abandoned or run as background workload with lowest priority. This is justified by the assumption that mixed-criticality systems stay in high-criticality mode indefinitely after the mode switch and sufficient utilization to provide guarantees for *low-criticality tasks* is not available. However, this assumption has been questioned lately, since it does not match the expectations of systems engineers. Such criticism has been detailed by Ernst and Di Natale [EN16] in "Mixed Criticality Systems - A History of Misconceptions?" as well as by Esper et al. [ENN+15] who ask "How realistic is the mixed-criticality real-time system model?", and is also mentioned in the survey by Burns and Davis [BD18, Section 6]. Specifically, it is argued that

mixed-criticality systems

- systems should be able to return from the high-criticality mode to the low-criticality mode after a sufficient amount of time, and
- that the low-criticality tasks are still critical and should therefore not be abandoned in high-criticality mode.

Note that if systems return to low-criticality mode, abandoning the low-criticality tasks is especially problematic since restarting them leads to an additional overhead. Moreover, if a system frequently switches between different execution modes, the overhead induced by online adaptation cannot be neglected, which is often assumed in the literature. In addition, such a mode switch may be completely unnecessary when intervals with abnormal execution behaviour are small. Besides, online adaptation, like changing task parameters or priorities at runtime, may not be possible in real-world systems. Furthermore, most research results for mixed-criticality systems assume the mode changes to be provided, i.e., the possibly complicated detection of the mode switch is not considered.

Resulting from the mentioned problems, some new scheduling approaches for mixed-criticality systems have emerged that give at least certain reduced timing guarantees for not so important tasks e.g., [BBG16; Pat17; Eri14; LSG+16; BB13]. Regardless, to the best of our knowledge, there is no research discussing whether such online adaptation is necessary beside the approach we presented in RTSS 2016 *[BCH+16]* and PRDC 2018 *[BSC18]* and detail in this chapter.

1 Note that we consider dual-criticality systems here since they are most commonly examined in mixed-criticality research. The general model considers multiple task sets and execution modes.

5.1.2 SYSTEMS WITH DYNAMIC REAL-TIME GUARANTEES

Especially when considering the aforementioned criticism in classical mixed-criticality research, as well as the resulting implications and concerns, the natural connection between *mixed-criticality* and *fault tolerance* becomes apparent, since there are more similarities than the fact that tasks have multiple execution modes. In both cases, the mode with larger execution time is induced by a rare event with resulting abnormal behaviour and after these events are handled properly, the system should return to the normal execution mode. Specifically, if assuming that the systems return to low-criticality mode, mixed-criticality behaviour is similar to a behaviour that occurs for both burst of transient faults as well as to so-called intermittent faults where a system steadily alternates between proper functionality and malfunction, e.g., due to a loose electrical contact [KK07].

mixed-criticality systems

fault tolerance

Therefore, we provide a general model, called *Systems with Dynamic Real-Time Guarantees* (SDRTGs), which is applicable in both cases as well as for other scenarios where uncertain execution behaviour occurs, for instance a CPU frequency reduction due to overheating. Similar to *mixed-criticality systems*, SDRTGs assume the tasks to be partitioned with respect to timeliness into *not so important* tasks, where rare deadline misses are acceptable, and *more important* tasks that must always meet their deadline. Hence, these tasks are called *timing tolerable tasks*, denoted $\mathbf{T}_{soft}$, and *timing strict tasks*, denoted $\mathbf{T}_{hard}$, respectively. Tasks may be executed either in a more likely *normal mode* with smaller WCET or in a rare *abnormal mode* with larger WCET. One prime feature of SDRTGs is that they provide service guarantees based on the current state of the system.

Systems with Dynamic Real-Time Guarantees

timing tolerable tasks

timing strict tasks

normal mode

abnormal mode

Assume that all tasks are executed normally over a long period of time. In this situation, the deadlines of all tasks should be satisfied, which is denoted as *full timing guarantees* in SDRTGs. In case of a rare event like fault-recovery, the *timing strict tasks* are still guaranteed to meet their deadlines, while the *timing tolerable tasks* may miss deadlines but are guaranteed a bounded tardiness. The situation where the guarantees are downgraded for the *timing tolerable tasks* is denoted *limited timing guarantees*. After the additional workload induced by the rare event and possible delayed jobs of *timing tolerable tasks* does not affect the system any more, it returns to *full timing guarantees*. All theses guarantees are given in advance using *static-priority scheduling* without any *online adaptation*. This avoids the online adaptation overhead that itself may lead to deadline misses. Moreover, such a system is implementable in current real-time operation systems, which often do not support online changes of task parameters and in addition often only support static-priority scheduling. Furthermore, under this approach a mode switch must not necessarily be detected or given, to ensure that the system functions correctly. However, during runtime, a *System with Dynamic Real-Time Guarantees* uses an online monitor to display the provided timing guarantees.

full timing guarantees

limited timing guarantees

static-priority scheduling

online adaptation

Based on our general description, we now provide a precise definition, an exact schedulability test, and some properties for *Systems with Dynamic Real-Time Guarantees* in a uniprocessor environment, and show that the resulting systems achieve a reasonable schedulability compared to the state-of-the-art for *mixed-criticality systems* in Section 5.2. Afterwards, the system model is extended to

multiprocessor platforms Section 5.3, considering both partitioned and semi-partitioned scheduling strategies. We also show how migration techniques can be utilized to compensate for processors that admit an abnormal execution behaviour over a longer interval of time, i.e., intermittent faults.

5.2 UNIPROCESSOR SYSTEMS WITH DYNAMIC REAL-TIME GUARANTEES

Systems with Dynamic Real-Time Guarantees

In this section, we provide the general model of *Systems with Dynamic Real-Time Guarantees* (SDRTGs) which can be adopted when uncertain execution behaviour of tasks is imposed by the environment and the system platform. We first define the system model as well as the related terms of *full timing guarantees* and *limited timing guarantees* in Section 5.2.1 and provide an exact schedulability test in Section 5.2.2. In Section 5.2.3, we continue by showing several important properties of an optimal priority order for SDRTGs, present an algorithm to determine such an order, and prove its optimality. How to calculate the maximum interval length until the system returns to *full timing guarantees* when only providing *limited timing guarantees* due to some abnormal execution behaviour in the past is detailed in Section 5.2.4, followed in Section 5.2.5 by the description of an online monitor for the system state that approximates the amount of time needed to return to *full timing guarantees*. The evaluation in Section 5.2.6 compares the acceptance ratio of the provided optimal static-priority assignment compared to other scheduling policies, to be precise to Rate Monotonic, Criticality Monotonic, and EDF-VD by Baruah et al. [BBD+15], a state-of-the-art dynamic-priority scheduling approach for mixed-criticality systems with online adaption.

The evaluation shows that regarding schedulability, our optimal priority assignment for *Systems with Dynamic Real-Time Guarantees* does not sacrifice much in comparison to EDF-VD in most settings. In some settings, especially when there is a larger difference between the WCETs in the abnormal and the normal mode, our approach can even achieve a higher acceptance ratio than EDF-VD for task sets with higher utilization values. These results show the good applicability of *Systems with Dynamic Real-Time Guarantees* and furthermore suggests that static-priority scheduling without any online adaptation is a sensible way to schedule mixed-criticality systems. This observation is supported by the fact that static-priority scheduling has less runtime overhead than dynamic-priority scheduling. In addition, our strategy does not abandon any tasks when only *limited timing guarantees* are provided and still guarantees bounded tardiness for the *timing tolerable tasks*, while most mixed-criticality approaches, including EDF-VD, allow to discard low-criticality tasks in the high-criticality mode. The results presented in this section appeared in *Systems with Dynamic Real-Time Guarantees in Uncertain and Faulty Execution Environments* in RTSS 2016 [BCH+16].

5.2.1 SYSTEM DEFINITION

implicit-deadline
constrained-deadline

We assume that the tasks in $\mathbf{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ have *implicit* or *constrained deadlines* and two execution modes with the properties detailed in Section 2.6,

referred to as the more common *normal mode* and the rare *abnormal mode* with related WCETs C_i^N and C_i^A. We further assume that we do not necessarily know the manner in which a job will be executed at the moment the job arrives or starts executing, but that this information may be available at some time point during or at the end of the execution process, e.g., due to a fault recovery routine that only detects the fault at the end of a job execution. Informally, we say that the system is abnormal in a given interval if some tasks are executed abnormally in a certain time interval. In addition, we assume that $\mathbf{T}$ is partitioned into $\mathbf{T}_{hard}$ and $\mathbf{T}_{soft}$ representing the *timing strict tasks* and *timing tolerable tasks*, respectively, i.e., $\mathbf{T}_{hard} \cap \mathbf{T}_{soft} = \emptyset$ and $\mathbf{T}_{hard} \cup \mathbf{T}_{soft} = \mathbf{T}$, and that for the *timing strict* tasks, missing a deadline will have catastrophic consequences.

normal mode

timing strict tasks

timing tolerable tasks

For all tasks, we assume the existence of two distinct execution modes, but that the other task parameters cannot be changed during runtime, i.e., releasing tasks at a lower rate, allowing some jobs to not be released, and enlarging a task's deadline are not valid reactions to abnormal execution. Hence, D_i and T_i are identical in the normal and the abnormal mode for all tasks. Furthermore, the tasks are scheduled with *static-priority scheduling* and have the same priorities regardless of the execution mode. One main reason for this decision regarding the scheduling algorithms is that many real-time operating systems only support one static-priority assignment that cannot be changed during runtime.

static-priority scheduling

In *Systems with Dynamic Real-Time Guarantees*, jobs are never aborted. One reason for this decision is that the results of these jobs may still be useful, even if they are a little late, as long as the timing behaviour of the *timing strict* tasks is not jeopardized. Therefore, we may reduce the guarantees for the *timing tolerable* tasks, i.e., we guarantee (at least) bounded tardiness instead of timeliness. However, the user should be informed that the results of *timing tolerable* tasks could be late. Furthermore, executing one or more jobs in the abnormal mode will not necessarily lead to missed deadlines. Lastly, when not assuming a given mode change, the trivial approach of abandoning the tasks in $\mathbf{T}_{soft}$ at the moment a fault occurs is not sufficient, but jeopardizes the deadline of a task in $\mathbf{T}_{hard}$, even if the system is proven safe in the normal mode:

Example 5.1. *Let $\tau_1 \in \mathbf{T}_{soft}$ with $C_1^N = 6$, $C_1^A = 6 + \varepsilon$, $T_1 = D_1 = 16$ and $\tau_2 \in \mathbf{T}_{hard}$ with $C_2^N = 11$, $C_2^A = 12 + \varepsilon$, $T_1 = D_1 = 24$, where $\varepsilon > 0$ but very small, be scheduled according to Rate Monotonic (RM). This task set is schedulable in the normal mode, but if we assume the critical instant of τ_2 and a fault happening at $t \in [22, 23]$, then τ_2 will miss its deadline as the first two jobs of τ_1 are already completed.*

Hence, trivially aborting jobs only works if all tasks $\tau_i \in \mathbf{T}_{soft}$ have lower priority than all tasks $\tau_i \in \mathbf{T}_{hard}$. However, in this situation there is anyway no effect on the schedulability of tasks in $\mathbf{T}_{hard}$.

Furthermore, after some abnormal execution occurred, jobs may still miss their deadline, even if all jobs run normally and no abnormal job remains in the system. The reason is that some remaining workload, which was postponed by the abnormal execution of higher priority tasks or an earlier job of the tasks itself, can *push back* the time intervals where the job runs. This behaviour is exemplified in Figure 5.1, where we consider the abnormal execution caused by faults, marked

self-pushing phenomenon

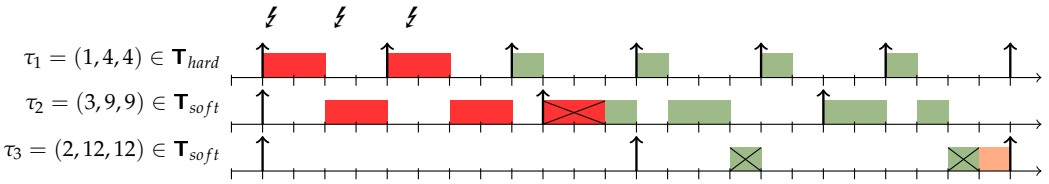


Figure 5.1: Tasks can miss a deadline due to self-pushing.
The first jobs of τ_2 and τ_3 miss their deadlines due to this additional workload. A black cross labels the late executions. The purple job of τ_3 misses its deadline due to the self-pushing by its earlier instance. *Adapted from [BCH+16].*

with $\lightning$, and $C_i^A = 2 \cdot C_i^N$. The first jobs of τ_2 and τ_3 miss their deadlines due to the additional workload created by the faults in the jobs of τ_1 and τ_2 (red) directly and the workload that is executed after the deadline is labeled by a black cross. After the first job of τ_2 finishes, all remaining jobs in the system execute normally, but the second job of τ_3 (orange) misses its deadline due to the additional workload induced by the late execution of the first job of τ_2 and τ_3. Note that the workload created by higher priority tasks and the second job of τ_3 in $[12, 24]$ is not sufficient for a deadline miss of the second job of τ_3 but that remaining workload from the first job of τ_3 in $[12, 24]$ leads to a *self-pushing phenomenon*.

The above observations have to be considered in the definition for *Systems with Dynamic Real-Time Guarantees*. We say a system provides *full timing guarantees*, if we can guarantee that all jobs that are currently ready to be executed will always meet their deadline should no further abnormal execution occur. Contrarily, a system provides *limited timing guarantees*, if we can only guarantee the timing behaviour for the *timing strict* tasks, while for the *timing tolerable* tasks bounded tardiness is guaranteed. We now formalize our argumentation and considerations.

Definition 5.1 (***System with Dynamic Real-Time Guarantees***)**.** *Consider a set $\mathbf{T}$ of tasks under a static-priority scheduling. A job of task $\tau_i \in \mathbf{T}$ cannot run until all the jobs of task τ_i that arrived earlier are completed. All jobs of all tasks always have to be run and can never be aborted. The tasks in $\mathbf{T}$ are partitioned into the timing strict tasks $\mathbf{T}_{hard}$ and the timing tolerable tasks $\mathbf{T}_{soft}$, hence $\mathbf{T}_{hard} \cap \mathbf{T}_{soft} = \emptyset$ and $\mathbf{T}_{hard} \cup \mathbf{T}_{soft} = \mathbf{T}$.*

*If the system provides **full timing guarantees**, hard real-time guarantees hold for each task:*

- *$\mathbf{T}$: Each task $\tau_i \in \mathbf{T}$ must meet the hard relative deadline.*

*If the system provides **limited timing guarantees**, the service level guarantees are downgraded from hard real time guarantees to bounded tardiness for some of the tasks:*

- *$\mathbf{T}_{hard} \subseteq \mathbf{T}$: Each task $\tau_i \in \mathbf{T}_{hard}$ is required to meet its deadline.*
- *$\mathbf{T}_{soft} \subseteq \mathbf{T}$: Each task $\tau_i \in \mathbf{T}_{soft}$ must have bounded tardiness, i.e., $0 \leq E_i < \gamma$ for a fixed value $\gamma > 0$.*

The partition into $\mathbf{T}_{hard}$ and $\mathbf{T}_{soft}$ is assumed to be given. We say a task set is *feasible* or *feasibly schedulable* as a *System with Dynamic Real-Time Guarantees*, if for the given partition and the given priority ordering the conditions for *full timing guarantees* and *limited timing guarantees* both hold. While we only consider

preemptive static-priority scheduling, this definition can be applied for both the preemptive and the non-preemptive case.

An important property of our approach is **not** to provide any runtime (online) adaptation. This has the nice consequence that the scheduling algorithm can be robust regardless of mode changes, assuming that the schedule is verified offline, to be always feasible to provide dynamic timing guarantees. Hence, the impact on the system behaviour due to the tardiness of the tasks in $\mathbf{T}_{soft}$, can be analysed in advance without disturbing the runtime system. The system may arbitrarily switch between normal and abnormal execution due to the impact of the physical environment, e.g., transient faults. If this impact is acceptable in the system behaviour, e.g., guaranteeing bounded tardiness instead of hard deadlines in $\mathbf{T}_{soft}$ is sufficient as long as this happens for less than 1% of systems runtime, there is no need for any runtime adaptation.

The remaining question is what information should be displayed to the user of a system at which time point. Usually, the user will not be interested in the information that an abnormal execution happened as long as the timing behaviour is not affected and all results can be trusted. We focus on the timing behaviour and assume that the calculated results can always be trusted, e.g., when considering fault recovery, the abnormal mode always leads to a correct result or to a partially incorrect but still acceptable result. However, a possible delay of currently provided results, the actual delay, and the expected time until the system returns to providing *full timing guarantees* - assuming no further faults occur - should be displayed.

5.2.2 EXACT SCHEDULABILITY TEST

Here, we assume the task set $\mathbf{T}$, a partition of $\mathbf{T}$ into two subsets $\mathbf{T}_{hard}$ and $\mathbf{T}_{soft}$, and a static-priority order P to be given. How such a priority order can be found is detailed in the next subsection. A sufficient schedulability test that determines whether $\mathbf{T}$ is a *System with Dynamic Real-Time Guarantees* (Definition 5.1) if scheduled according to P must test the following three conditions:

1. Each task $\tau_i \in \mathbf{T}$ meets its hard deadline if all tasks are executed in the normal mode.
2. Each task $\tau_i \in \mathbf{T}_{hard}$ meets its hard deadline if some (or all) tasks are executed in the abnormal mode.
3. Each task $\tau_i \in \mathbf{T}_{soft}$ has a bounded tardiness if some (or all) tasks are executed in the abnormal mode.

To test the schedulability of a preemptive task set with constrained deadlines under a static-priority assignment, we apply the Time Demand Analysis (TDA) [LSD89] as defined in Eq. (2.3) as an exact test with pseudo-polynomial runtime. It determines the schedulability of a task τ_k under the assumption that the priority order of the task set is given and the schedulability of all tasks in $hp(\tau_k)$ is already ensured. The task set is schedulable under preemptive static-priority scheduling, if Eq. (2.3) holds true for all $\tau_i \in \mathbf{T}$. This results in the following *exact* *exact test*

schedulability tests to determine the schedulability of a task set as a *System with Real-Time Service Level Guarantees*.

Theorem 5.1 (Exact Schedulability Test for Constrained Deadlines)**.** *For a given static-priority ordering P, a task set $\boldsymbol{T}$ is a System with Dynamic Real-Time Guarantees as defined in Definition 5.1, if the following three conditions hold:*

1. *Full timing guarantees hold, if $\boldsymbol{T}$ can be scheduled according to TDA [LSD89] when all tasks are executed in the normal mode, i.e., $C_i = C_i^N \; \forall \tau_i$.*
2. *When the system runs with limited timing guarantees, all $\tau_i \in \boldsymbol{T}_{hard}$ meet their hard deadlines, if they are schedulable according to TDA when all tasks are executed in the abnormal mode, i.e., $C_i = C_i^A \; \forall \tau_i$.*
3. *Each task $\tau_i \in \boldsymbol{T}_{soft}$ has bounded tardiness if $U_{sum}^A \leq 1$.*

Proof. Let $\Theta_{\text{soft}}^A := \left\{ \tau_i \in \mathbf{T}_{soft} \mid \tau_i \in hp(\tau_j), \tau_j \in \mathbf{T}_{hard} \right\}$ be the tasks in $\mathbf{T}_{soft}$ that have a higher priority than at lest one task in $\mathbf{T}_{hard}$.

1. Follows immediately, since TDA is an exact schedulability test for any preemptive static-priority scheduling algorithm if the maximum interference can be determined based on the critical instant theorem. If no abnormal execution occurs, all tasks are executed in normal mode and the situation is identical to the sporadic case where all tasks have only one execution mode.

2. We only must test the tasks in $\mathbf{T}_{hard}$ with TDA, since we only need to guarantee bounded tardiness for $\tau_i \in \mathbf{T}_{soft}$ which is evaluated in the third step. However, tasks in Θ_{soft}^A may contribute workload to the worst-case response time of tasks in $\mathbf{T}_{hard}$. The tasks in $\mathbf{T}_{soft}$ do not have hard real-time constraints if abnormal executions occur, but are executed with the same priority as in the normal mode. Therefore, they can be handled as hard real-time tasks in the analysis, since they contribute the same workload as tasks in $\mathbf{T}_{hard}$ would. The possibility that these tasks may miss their deadlines has no impact on the analysis, since TDA tests every task individually and we are only interested in the workload those tasks contribute if they are executed, but not in the concrete execution order of higher priority tasks or if the tasks meat or miss their deadline. According to the critical instant theorem, the worst case for $\tau_k \in \mathbf{T}_{hard}$ happens when it is released together with all higher priority tasks, all subsequent jobs of these tasks are released as early as possible, and all tasks are executed in abnormal mode. Considering C_i^N instead of C_i^A in the analysis would only decrease the workload generated by tasks in $hep(\tau_k)$.

3. For tasks in $\mathbf{T}_{soft}$, bounded tardiness has to be provided. When assuming exactly periodic releases, the total workload contributed by jobs that are released in one hyperperiod is always smaller than or equal to the hyperperiod length if $U_{sum}^A \leq 1$. If jobs are released sporadically, this workload may only decrease. Therefore, the latest possible time where a job can finish is one hyperperiod after its release if $U_{sum}^A \leq 1$. This leads to an upper bound on the worst-case response time for all tasks in $\mathbf{T}$ and thus to a bounded tardiness. If $U_{sum}^A > 1$, the maximum workload in one hyperperiod, due to jobs that arrive in the hyperperiod but are not fully executed in the hyperperiod, is larger than the length of the hyperperiod. Let this additional workload be $\gamma > 0$. As for each value β value $x \in \mathbb{N}$ with $\gamma \cdot x > \beta$ exists, the tardiness is not bounded.

This are the 3 conditions we have to match for a *System with Dynamic Real-Time Guarantees* (Definition 5.1). □

However, this definition of bounded tardiness seems to be too restrictive for some practical cases. For instance, when many executions are affected by faults, hardened hardware should be used instead of recovery mechanisms, and to successfully apply software-based recovery mechanisms, a low expected fault rate is usually a condition. Moreover, for *mixed-criticality systems* it is assumed that the system will usually run in low-criticality mode. Hence, without restricting ourselves to a specific kind of uncertain execution behaviour, we consider two general possibilities:

1. Abnormal execution happens with a very low probability over an interval with a small length and affects (nearly) all jobs.
2. For each individual job, abnormal execution happens with a low probability.

In both cases $U^A_{sum} > 1$ is tolerable if $U^N_{sum} < 1$ and if the intervals where no abnormal execution occurs are significantly longer than the intervals where abnormal execution occurs. Thus, in both scenarios, the setting itself will lead to a bounded tardiness for most practical cases. For instance, when it can be assumed that a burst of faults only affects a small number of jobs compared to the number of jobs between two bursts of faults. For mixed-criticality systems, a similar assumption is that the intervals in high-criticality mode are significantly shorter than the intervals in low-criticality mode. In both scenarios, the system has a sufficient amount of time to return to *full timing guarantees*. The length of the interval with *limited timing guarantees* for a given task set can be upper bounded if we suppose the maximum length Δ of such an interval with abnormal execution behavior to be known, as shown in Section 5.2.4. If we assume faults to happen with a given rate, this rate needs to be high to affect a sufficient number of tasks to lead to *limited timing guarantees* over a longer interval.

Due to this consideration, we drop the condition that $U^A_{sum} \leq 1$ in the evaluation regarding the acceptance rate of *Systems with Dynamic Real-Time Guarantees* in Section 5.2.6. However, we analyse the amount of time where only *limited timing guarantees* are provided for task sets with a high utilization in Section 5.2.6 as well, validating whether our decision to drop the condition has a high impact on the stability of the system based on the example of reasonable fault rates.

5.2.3 PROPERTIES OF PRIORITY ASSIGNMENTS

After providing the schedulability test for a *System with Dynamic Real-Time Guarantees* under a given priority ordering, we now explain how to construct such a priority ordering, if one exists, for a given task set. First, we show that existing or trivial priority orderings, namely Deadline Monotonic order and Criticality Monotonic order, are not optimal for *System with Dynamic Real-Time Guarantees*.

Lemma 5.2 (Deadline Monotonic Order is Not Optimal)**.** *For a System with Dynamic Real-Time Guarantees (Def. 5.1) a Deadline Monotonic priority order is not optimal for constrained-deadline task sets.*

Proof. Assume two tasks to be scheduled according to DM, where $\tau_1 \in \mathbf{T}_{soft}$ with $C_1^N = 1,\ C_1^A = 1+\varepsilon,\ T_1 = D_1 = 4$, where $\varepsilon > 0$ but very small, and $\tau_2 \in \mathbf{T}_{hard}$ with $C_2^N = 3,\ C_2^A = 4,\ T_1 = D_1 = 6$. In normal mode, both tasks meet their deadlines with WCRTs of $R_1^N = 1$ and $R_2^N = 4$, respectively. In the abnormal mode, $U_{sum}^A = \frac{1+\varepsilon}{4} + \frac{4}{6} = \frac{22+6\cdot\varepsilon}{24} < 1$ for small values of $\varepsilon > 0$, which leads to bounded tardiness for τ_1, and $R_2^A = 2\cdot(1+\varepsilon)+4 > 6$ and thus τ_2 will miss its deadline.

If the priorities of τ_1 and τ_2 are switched, τ_1 has bounded tardiness in abnormal mode since U_{sum}^A remains the same. Both tasks are schedulable in normal mode as $R_1^N = 4$ and $R_2^N = 3$. In abnormal mode the WCRT of τ_2 is $4 < 6$. □

As shown in Example 5.1, in general, aborting the execution of $\tau_i \in \mathbf{T}_{soft}$ is only able to keep up hard real time guarantees for $\mathbf{T}_{hard}$, if all tasks in $\mathbf{T}_{soft}$ have lower priorities than all tasks in $\mathbf{T}_{hard}$.

Definition 5.2 (Criticality Monotonic)**.** *We say a task set $\mathbf{T}$ with two subsets $\mathbf{T}_{hard}$ and $\mathbf{T}_{soft}$ has a Criticality Monotonic ordering, when all tasks $\tau_i \in \mathbf{T}_{hard}$ have higher priority than all tasks in $\mathbf{T}_{soft}$.*

For the following lemma, the internal order of $\mathbf{T}_{hard}$ and $\mathbf{T}_{soft}$ is not important. However, in general, we assume that $\mathbf{T}_{hard}$ and $\mathbf{T}_{soft}$ are internally ordered according to DM.

Lemma 5.3 (Criticality Monotonic Ordering is Not Optimal)**.** *For a System with Dynamic Real-Time Guarantees (Def. 5.1), Criticality Monotonic order is not optimal.*

Proof. Assume $\tau_1 \in \mathbf{T}_{soft}$ with $C_1^N = 1,\ C_1^A = 1+\varepsilon$, and $T_1 = D_1 = 3$, where $\varepsilon > 0$ but very small, and $\tau_2 \in \mathbf{T}_{hard}$ with $C_2^N = 3,\ C_2^A = 3+\varepsilon$, and $T_1 = D_1 = 6$. Let them be scheduled according to Criticality Monotonic, i.e., $P(\tau_1) > P(\tau_2)$. In normal mode we get $R_1^N = 4$ and $R_2^N = 3$ and thus τ_1 will not meet its deadline.

If the priorities of τ_1 and τ_2 are switched, τ_1 and τ_2 both meet their deadlines in normal mode, i.e., $R_1^N = 1$ and $R_2^N = 4$. In abnormal mode τ_2 meets its deadline, since $R_2^N = 2\cdot(1+\varepsilon)+3+\varepsilon = 5+3\cdot\varepsilon < 6$. Here τ_1 has bounded tardiness, as $U_{sum}^A = \frac{1+\varepsilon}{3} + \frac{3+\varepsilon}{6} = \frac{5+3\cdot\varepsilon}{6} < 1$. □

optimal priority assignment

OPA compatible

Since neither Deadline Monotonic nor Criticality Monotonic scheduling are optimal for *System with Dynamic Real-Time Guarantees*, we have to look at a more general approach. Audsley's Algorithm [Aud91], also called *optimal priority assignment* (OPA), can be applied to find a feasible static-priority assignment if the used schedulability test S is *OPA compatible* [DB09]. The related three conditions are detailed in Section 2.4.

Lemma 5.4. *The Schedulability Test in Theorem 5.1 is OPA compatible.*

Proof. We only sketch the proof.

1. **Schedulability is independent from order of $hp(\tau_k)$**: TDA sums up the workload of all jobs from tasks in $hp(\tau_k)$. Hence, the order of those tasks has no impact, as it only changes the order in which the workload is summed up. This holds true for both the normal and the abnormal case in Theorem 5.1. The task order has no impact on the condition $U_{sum}^A \leq 1$.

2. **Schedulability is independent from order of $lp(\tau_k)$**: Since the workload of tasks in $lp(\tau_k)$ is not considered in TDA at all, the order has no impact. The order of the tasks has no impact on the condition $U^A_{sum} \leq 1$ as well.
3. **Tasks do not get unschedulable at a higher priority**: If τ_k is assigned a higher priority by swapping with τ_{k-1}, the workload of the higher priority tasks in TDA will only be reduced and thus τ_k remains schedulable. The switch has no impact on the condition $U^A_{sum} \leq 1$ either.

Ensuring these three properties is sufficient to prove that a schedulability test is OPA compatible. □

We now show that the tasks in $\mathbf{T}_{hard}$ and $\mathbf{T}_{soft}$ can both be ordered according to DM if a feasible priority assignment exists.

Lemma 5.5 ($\mathbf{T}_{hard}$ in Deadline Monotonic Order)**.** *If a feasible priority assignment P for a System with Dynamic Real-Time Guarantees (Definition 5.1) exists for a given task set $\mathbf{T}$, the tasks in $\mathbf{T}_{hard}$ can be reordered according to Deadline Monotonic order and Dynamic Real-Time Guarantees are still provided.*

Proof. We only sketch the proof since it is very similar to the prove for the optimality of Deadline Monotonic scheduling [LW82]. Using the interchanging argument, we look at the first two consecutive tasks τ_j and τ_k in the internal priority order of $\mathbf{T}_{hard}$ that are not in DM order, i.e., $D_j > D_k$ and $P(\tau_j) < P(\tau_k)$. If τ_j and τ_k are direct successors in P, we can swap them directly due to the optimality of DM for constrained-deadline task sets.

The case that τ_j and τ_k are not direct successors in P remains, which means that $S := \{\tau_i \in \mathbf{T}_{soft} \mid P(\tau_j) < P(\tau_i) < P(\tau_k)\} \neq \emptyset$. We increase the priority of each $\tau_i \in S$ by 1 and set the priority of τ_j to $P(\tau_{k-1})$, thus τ_j and τ_k are now direct successors in P. All tasks in S remain schedulable as their priorities are increased, while τ_j remains schedulable since $D_j > D_k$ due to the precondition that τ_j and τ_k are not in DM order and, as τ_k is schedulable, the workload created by all tasks in $hp(\tau_j) \cup \tau_j \cup \tau_k$ up to D_k is smaller than D_k. Thus, we can now swap τ_j and τ_k since both tasks remain schedulable according to the first case and continue until all tasks in $\mathbf{T}_{hard}$ are in DM order. □

With a similar (omitted) proof, we achieve the same result for $\mathbf{T}_{soft}$.

Lemma 5.6 ($\mathbf{T}_{soft}$ in Deadline Monotonic Order)**.** *If a feasible priority assignment for a System with Dynamic Real-Time Guarantees (Definition 5.1) exists, the tasks in $\mathbf{T}_{soft}$ can be reordered according to Deadline Monotonic ordering while all Dynamic Real-Time Guarantees still hold.*

We now know that we can reorder both subsets $\mathbf{T}_{hard}$ and $\mathbf{T}_{soft}$ to be in Deadline Monotonic order, resulting in the following theorem.

Theorem 5.7 ($\mathbf{T}_{hard}$ and $\mathbf{T}_{soft}$ in DM order)**.** *If a feasible priority assignment for System with Dynamic Real-Time Guarantees (Definition 5.1) exists, a feasible priority assignment where the tasks in $\mathbf{T}_{hard}$ and $\mathbf{T}_{soft}$ are internally ordered according to the Deadline Monotonic order also exists.*

Proof. We know that $\mathbf{T}_{hard}$ and $\mathbf{T}_{soft}$ can be reordered individually according to DM order while obtaining the feasibility. Since reordering $\mathbf{T}_{hard}$ does not change the internal order of $\mathbf{T}_{soft}$ and vice versa, we first reorder the tasks in $\mathbf{T}_{hard}$ to be in DM order and then reorder the tasks in $\mathbf{T}_{soft}$ to be in DM order. Afterwards, we still obtain a feasible priority ordering for a *System with Dynamic Real-Time Guarantees*. □

Algorithm 1 Feasible Priority Assignment

Input: $\mathbf{T}_{hard}$, $\mathbf{T}_{soft}$
Output: Feasible Order P of $\mathbf{T}_{hard} \cup \mathbf{T}_{soft}$ or **NOT POSSIBLE**
1: Sort $\mathbf{T}_{hard}$ by D_i increasingly
2: Sort $\mathbf{T}_{soft}$ by D_i increasingly
3: Find Assignment($\mathbf{T}_{hard}$, $\mathbf{T}_{soft}$)

Procedure: Find Assignment($\mathbf{T}_{hard}$, $\mathbf{T}_{soft}$)
4: **for** ($n = |\mathbf{T}_{hard} \cup \mathbf{T}_{soft}|$; $n > 0$; $n := n - 1$) **do**
5: τ_t := last element of $\mathbf{T}_{hard}$
6: **if** (Try Priority(τ_t, $\left\{\mathbf{T}_{hard} \cup \mathbf{T}_{soft}\right\} \setminus \{\tau_t\}$,n,hard)) **then**
7: $P(\tau_t) := n$
8: $\mathbf{T}_{hard} := \mathbf{T}_{hard} \setminus \{\tau_t\}$
9: **else**
10: τ_t := last element of $\mathbf{T}_{soft}$
11: **if** (Try Priority(τ_t, $\left\{\mathbf{T}_{hard} \cup \mathbf{T}_{soft}\right\} \setminus \{\tau_t\}$,n,soft)) **then**
12: $P(\tau_t) := n$
13: $\mathbf{T}_{soft} := \mathbf{T}_{soft} \setminus \{\tau_t\}$
14: **else**
15: **return** NOT POSSIBLE
16: **return** List of $\mathbf{T}_{hard} \cup \mathbf{T}_{soft}$ ordered by $P(\tau_t)$

Procedure: Try Priority($\tau_t, hp(\tau_t), priority, task\_type$)
17: $P(\tau_t) := n$
18: Assign $hp(\tau_t)$ to priorities $1, \ldots, n-1$
19: **if** (task_type==hard) **then**
20: $C_i := C_i^A$, $\forall \tau_i \in hp(\tau_t) \cup \tau_t$
21: **else**
22: $C_i := C_i^N$, $\forall \tau_i \in hp(\tau_t) \cup \tau_t$
23: **if** (τ_t is schedulable according to TDA) **then**
24: **return** true
25: **else**
26: **return** false

Hence, according to Theorem 5.7, if a feasible priority assignment exists we can use the priority assignment algorithm presented in pseudo-code in Algorithm 1 to find one. The idea is similar to OPA [Aud91]: Find a task that can take the lowest priority, i.e., it is schedulable under the assumption that all other tasks have higher priority. If such a task can be found, assign it to the lowest priority (among the tasks), remove it from the task set, and redo the process with the remaining tasks. If no suitable task is found for some priority, we return *NOT POSSIBLE*, otherwise we return a feasible priority assignment. To keep the length of the pseudo code reasonable, we do not take care of the case that either $\mathbf{T}_{hard}$ or $\mathbf{T}_{soft}$

will be empty at some point during the algorithm. In that case, only the lowest priority task of the not empty set will be tested. The main difference to OPA is that Algorithm 1 tests at most two candidates for each priority: the remaining task in $\mathbf{T}_{hard}$ and the remaining task in $\mathbf{T}_{soft}$ with the longest relative deadline, respectively. Therefore, the tasks in $\mathbf{T}_{hard}$ and $\mathbf{T}_{soft}$ are preordered according to DM. Note that regarding *Dynamic Real-Time Guarantees* it does not matter if the task with the longest deadline in $\mathbf{T}_{soft}$ or $\mathbf{T}_{hard}$ is assigned to the priority if TDA returns schedulable for both tasks. However, since tasks in $\mathbf{T}_{hard}$ are only assigned to a priority if they meet the deadline when considering only abnormal execution, trying to assign the tasks in $\mathbf{T}_{hard}$ first may result in shorter intervals with *limited timing guarantees* as well as in more tasks in $\mathbf{T}_{soft}$ that meet their deadline if abnormal execution occurs.

Theorem 5.8 (Feasible Priority Assignment)**.** *If a feasible priority assignment for a given System with Dynamic Real-Time Guarantees exists, Algorithm 1 will find a feasible assignment.*

Proof. Based on Theorem 5.7 we know that if a feasible priority assignment exists, there is also a feasible assignment where $\mathbf{T}_{hard}$ and $\mathbf{T}_{soft}$ are internally in Deadline Monotonic order. Assume that such a priority assignment S is given, e.g., an OPA where the tasks in $\mathbf{T}_{hard}$ and $\mathbf{T}_{soft}$ are reordered to be in DM ordering in the way presented in Lemma 5.5 and 5.6. To conclude the proof, we will reorder S until it has the same order as provided by Algorithm 1. An important observation is that the tasks in $\mathbf{T}_{hard}$ and $\mathbf{T}_{soft}$ always have the same internal order in S and in the priority assignment provided by Algorithm 1, as S was reordered to have DM order in both subsets, and Algorithm 1 only tries to assign the remaining tasks in $\mathbf{T}_{hard}$ and $\mathbf{T}_{soft}$ with the largest relative deadline.

Let $\tau_j \in \mathbf{T}_{hard}$ be the task in $\mathbf{T}_{hard}$ with the lowest priority in S, i.e., the task in $\mathbf{T}_{hard}$ with the longest relative deadline. Based on the interchanging argument, we try to exchange τ_j with tasks that have lower priority in S until τ_j would not be schedulable at its new priority. Hence, S remains schedulable, as for all other tasks the priority is only increased. Let T^j_{low} denote all tasks that have lower priority than τ_j after the priority of τ_j was decreased. We must examine two cases:

1. If $T^j_{low} = \emptyset$ the new position of τ_j is the lowest.
2. There are $T^j_{low} \subseteq \mathbf{T}_{soft}$ that have lower priority than τ_j. All $\tau_i \in T^j_{low}$ are schedulable, as P was schedulable and the priority of those tasks was not changed.

Now τ_j and $\tau_i \in T^j_{low}$ are in the same order as Algorithm 1 provides, as Algorithm 1 only assigns a task in $\mathbf{T}_{soft}$ to a priority if the task in $\mathbf{T}_{hard}$ cannot be assigned. In the next step, we consider the task $\tau_k \in \mathbf{T}_{hard}$ that has the lowest priority in $\mathbf{T}_{hard} \setminus \{\tau_j\}$ and decrease its priority until it would not be schedulable anymore or we would exchange it with another task in $\mathbf{T}_{hard}$. Now τ_k and all tasks with a priority lower than τ_k are in the same order as Algorithm 1 provides with the same argument. We repeat this procedure until all tasks in $\mathbf{T}_{hard}$ are in the same order as provided by Algorithm 1. □

To find a feasible priority assignment, Audsley's Algorithm [Aud91] (OPA) could be applied directly. However, in general, Algorithm 1 has a much better runtime than OPA. Let the task set contain n tasks. In the worst case, TDA has to test a pseudo-polynomial number of time points to determine if a task is schedulable on a given level. Let $TDA(n)$ be the number of tests for that level. Each of these tests has a runtime of $O(n)$ to sum up the workload of the higher priority tasks. Due to this, the time complexity needed to test the schedulability of a task at a priority level is $O(n \cdot TDA(n))$.

Hence, the time complexity is $O(n^2 \cdot n \cdot TDA(n)) = O(n^3 \cdot TDA(n))$ for OPA, as there are n priority levels and the worst case OPA has to test $O(n)$ tasks with TDA on each priority level. When Algorithm 1 is used, at most 2 tasks have to be considered for each priority level, resulting in a time complexity of $O(2n \cdot n \cdot TDA(n)) = O(n^2 \cdot TDA(n))$ to find the assignment. Ordering $\mathbf{T}_{hard}$ and $\mathbf{T}_{soft}$ according to the relative deadlines can be done in $O(n \log n)$ which is dominated by $O(n^2 \cdot TDA(n))$. Thus, when assigning priorities to give *Real-Time Service Level Guarantees* the time complexity of Algorithm 1 is a power less than the one of OPA.

5.2.4 SYSTEM MODE ANALYSIS

Contemplating a situation where abnormal execution occurs for some jobs during an interval and that this interval length Δ is known, we analyse the system mode.

limited timing guarantees

We calculate the maximum time the system provides only *limited timing guarantees* under the assumption that no further abnormal execution will occur. Let θ_b be the latest time instant at which an abnormal execution has been detected. Recall that we assume that the mode of execution (normal or abnormal) is not necessarily known when the job starts, e.g., due to fault detection with related recovery operations. Hence, we say we detect abnormal execution at the moment a job executes for more than C_i^N. However, the system mode analysis presented here also works if the execution mode of a job is known beforehand. Our objective is to find the time when the system will return to *full timing guarantees*, i.e., all jobs of all tasks will meet their deadlines.

full timing guarantees

Let θ_0 denote the latest time instant before θ_b where the processor is idle, and θ_a be the first time instant where abnormal execution is detected over $[\theta_0, \theta_b]$. Let $\theta_b - \theta_a = \Delta$, as illustrated in Figure 5.2. To provide *full timing guarantees* again after an abnormal interval, it is sufficient that the system is idle at time $\theta_f \geq \theta_b$ as no abnormal execution that happened before θ_f can affect any task that is realised at a time $t \geq \theta_f$. If we are able to show that the length of the *busy interval* for a given Δ is no more than a specific value, we know that after this interval the system will provide *full timing guarantees* again.

busy interval

We denote $\Omega(t)$ as the maximum total amount of execution time of the tasks in $\mathbf{T}$ in $[\theta_0, \theta_0 + t)$. Clearly, the interval length during which a system is busily executing is no more than the smallest value of t that satisfies $\Omega(t) \leq t$.

Theorem 5.9 (Computing $\Omega(t)$)**.** *Let Γ denote the interval $[\theta_0, \theta_0 + t)$ except $[\theta_a, \theta_b]$. An upper bound on $\Omega(t)$ can be calculated as:*

$$\Omega(t) = \Delta + F + I(t) \tag{5.1}$$

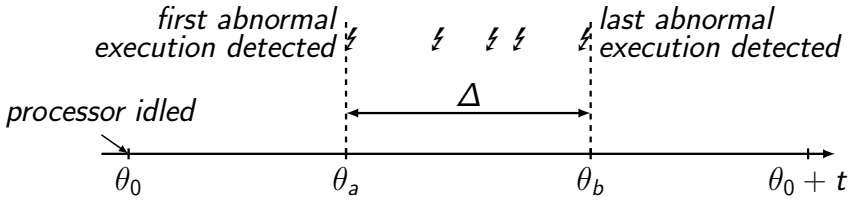


Figure 5.2: Schematic for the system mode analysis. A busy interval of length t with an interval of abnormal execution $[\theta_a, \theta_b]$ equal to length Δ. The time instants where abnormal execution is detected are marked by ↯. *Adapted from [BCH+16].*

where the terms are as described below.

- $I(t)$*: the combined workload from all jobs of all tasks executed* normally *over the interval* $[\theta_0, \theta_0 + t)$.
- F*: the work of all tasks* abnormally *executed over the interval* Γ.
- Δ*: the interval length during which* abnormal executions *are detected.*

To prove Theorem 5.9, we first derive equations for each of the aforementioned terms and show that we account for the maximum possible amount of workload. That the processor idles at θ_0 by definition directly leads to the following lemma:

Lemma 5.10. *There are at most* $\left\lceil \frac{t}{T_i} \right\rceil$ *jobs of task* τ_i *released in* $[\theta_0, \theta_0 + t)$.

Hence, $I(t) = \sum_{\tau_i \in \tau} W_i(t)$ where $W_i(t) = \left\lceil \frac{t}{T_i} \right\rceil C_i^N$. The amount of of additional execution time for a task instance in abnormal mode compared to the normal mode is at most $C_i^A - C_i^N$.

Lemma 5.11. *In* Γ*, there is at most one job of each task that executes abnormally where the additional workload for this abnormal execution, compared to jobs that execute normally, is upper bounded by* $C_i^A - C_i^N$*. Therefore,*

$$F = \sum_{\tau_i \in \tau} \left(C_i^A - C_i^N \right) \tag{5.2}$$

Proof. We prove this lemma by considering two cases:

- Additional execution in $[\theta_0, \theta_a]$: By definition of θ_a as the first time an abnormal execution is detected after the last idle time, no job is executing for more than C_i^N over $[\theta_0, \theta_a]$.
- Additional execution after θ_b: As each task may have at most one job executed at any time instant, each task has at most one job that has been partially executed and is not yet completed at time θ_b. As no further abnormal execution is detected after θ_b according to its definition, there is at most one job with abnormal execution carried out after θ_b, and the remaining workload is upper bounded by $C_i^A - C_i^N$ for every task τ_i.

Hence, we can conclude this lemma. □

Lastly we need a simple observation about Δ.

Observation 5.12. *The abnormal workload executed in the time interval $[\theta_a, \theta_b]$ is no more than Δ, regardless of whichever job is executing.*

Now we can prove Theorem 5.9.

Proof (Theorem 5.9). We have to show that we accounted for all possible workload in the time interval $[\theta_0, \theta_0 + t)$. We account for all normal executions in $I(t)$. This includes normal executions in Δ. For abnormal executions, the part up to C_i^N is also accounted for in $I(t)$, regardless if they are executed in or outside Δ, as stated in Lemma 5.10. The additional workload of $C_i^A - C_i^N$ can only be executed once outside Δ, as shown in Lemma 5.11. Thus, we get the most amount of additional workload in Δ if only additional workload due to abnormal execution is executed in Δ, which is bounded by Δ due to Observation 5.12. □

Please note that the cause for an abnormal execution that is detected at time θ_a may only occur in the interval $[\theta_0, \theta_a]$. Specifically, this is the case at least for the one job where abnormal execution is detected at θ_a. However, jobs may only be executed for a very small part of C_i^N during Δ and thus we use Δ as an upper bound. Regardless, the exact moment the cause for abnormal execution happens is not important to bound the busy interval or for *Systems with Dynamic Real-Time Guarantees* in general, since in *Systems with Dynamic Real-Time Guarantees*, all guarantees are given beforehand and no online adaptation is necessary.

5.2.5 SYSTEM MONITOR DESIGN

Up to this point, all essential analysis in terms of system scheduling is provided. However, for the industrial practice, an online monitor to reflect the system status is also important. This monitor should trigger warnings if the system can only give *limited timing guarantees* for an individual task or the whole system, and display the earliest time the task/system will return to *full timing guarantees*, i.e., the monitored task or all tasks in $\mathbf{T}_{soft}$ will meet their hard deadline. We propose to use approximation to detect the change from *full timing guarantees* to *limited timing guarantees*, and for the calculation of an upper bound of the next time instance the system will return to *full timing guarantees*.

limited timing guarantees

full timing guarantees

Since we guarantee the timing behaviour of the *timing strict* tasks offline, we only have to monitor *timing tolerable* tasks. Assume we monitor $\tau_k \in \mathbf{T}_{soft}$. For notational brevity, let $hp(\tau_k)^H := hp(\tau_k) \cap \mathbf{T}_{hard}$ and $hp(\tau_k)^S := hp(\tau_k) \cap \mathbf{T}_{soft}$, i.e., the subset of the tasks with higher priority than τ_k that are part of $\mathbf{T}_{hard}$ and $\mathbf{T}_{soft}$, respectively. Assume that for each task it is known if a job of this task is ready to be executed at the time we analyse, and that the current execution mode of the job is known as well, i.e., it is known whether it has been executed for less than C_i^N amount of time or not.

We know that the interference from tasks in $hp(\tau_k)$ and/or self-pushing (e.g., Figure 5.1) can prolong the execution of a job of τ_k. Due to this interference, multiple deadline misses for τ_k may happen once the system switches to *limited*

timing guarantees. The next time we can guarantee that τ_k meets its deadline, if no further faults occur, is the moment a lower priority task is executed, i.e., the end of the level k *busy interval*, denoted as $busy_k$. Let $|busy_k|$ be the length of $busy_k$. If $|busy_k|$ is smaller than the time the current job of τ_k has left to finish its execution, we can give *full timing guarantees* for τ_k. Otherwise τ_k may miss its deadline, we can only provide *limited timing guarantees*, and *full timing guarantees* for τ_k can be given again at the end of $busy_k$.

busy interval

Similar to Section 5.2.4, we denote the maximum total amount of execution time from $hep(\tau_k)$ in an interval from $[\theta_0, \theta_0 + t)$ as $\Omega_k(t)$ and look for the smallest value of t where $\Omega_k(t) \leq t$. To apply the the formula developed in Section 5.2.4 directly, we would have to keep track of the last time the processor executed a task in $lp(\tau_k)$ for each τ_k, and the amount of additional interference due to higher priority tasks during this interval. We use an alternative approach here by setting θ_0 to the current time and calculate the *carry in* as well as the future workload due to tasks in $hep(\tau_k)$, to not have additional, potentially high, overhead for keeping track of the interference of tasks in $hep(\tau_k)$ in $busy_k$ for each $\tau_k \in \mathbf{T}_{soft}$.

carry in

Let $I_k(t)$ denote the maximum workload due to jobs of tasks in $hep(\tau_k)$ with normal executions in $[\theta_0, \theta_0 + t]$, i.e.,

$$I_k(t) = \sum_{\tau_i \in hep(\tau_k)} W_i(t) = \sum_{\tau_i \in hep(\tau_k)} \left\lceil \frac{t}{T_i} \right\rceil C_i^N \tag{5.3}$$

Furthermore, the carry in workload from tasks in $hep(\tau_k)$ must be taken into account. We denote the carry in workload of τ_i by $G(\tau_i)$. To determine $G(\tau_i)$, we need to know how much workload remains for a job of τ_k that is currently executable, which can be estimated by keeping track of the time the job has already been running. Depending on the mode of the job, we subtract that value from C_i^N or C_i^A to get $G(\tau_i)$, ergo we also need to know if a job is currently in normal or abnormal mode. If keeping track of the time a task has been executed is too much overhead, C_i^N or C_i^A can directly be used as an upper bound on $G(\tau_i)$ depending on its mode. Since tasks in $hp(\tau_k)^H$ always meet their deadline by system design, at most one job of each task in $hp(\tau_k)^H$ can be in the system. For tasks in $hp(\tau_k)^S \cup \tau_k$ there may be carry in from more than one postponed execution and we have to sum this up with the remaining workload of a currently active job to get $G(\tau_i)$ for $\tau_i \in \mathbf{T}_{soft}$. The total carry in can be calculated as $G_k = \sum_{\tau_i \in hep(\tau_k)} G(\tau_i)$.

We have to look for the smallest t with:

$$\Omega_k(t) = \sum_{hep(\tau_k)} \left\lceil \frac{t}{T_i} \right\rceil C_i^N + \sum_{\tau_i \in hep(\tau_k)} G(\tau_i) \leq t \tag{5.4}$$

As we do not know how many jobs of τ_k will be executed before $busy_k$ ends, we create a virtual task $\tau_{k'}$ with $C_{k'} = G_k$ and virtual priority $k+1$. An upper bound on the WCRT of $\tau_{k'}$ can be calculated using Theorem 1 in [BNR+09] by Bini et al. It states that for each sporadic task τ_i in a static-priority system the WCRT R_i is upper bounded by

$$R_i \leq \frac{C_i + \sum_{j=1}^{i-1} C_j (1 - U_j)}{1 - \sum_{j=1}^{i-1} U_j} \tag{5.5}$$

In our case, we have to take the future jobs of all tasks in $hep(\tau_k)$ into account and the carry in G_k is the execution time of our virtual task. Hence, we can calculate an upper bound for the length of the level k busy period $busy_k$ as $|busy_k|^*$ by

$$|busy_k| \leq \frac{G_k + \sum_{j \in hep(\tau_k)} C_j^N (1 - U_j^N)}{1 - \sum_{j \in hep(\tau_k)} U_j^N} = |busy_k|^* \tag{5.6}$$

Note that only the tasks with a job that already started may have abnormal execution behaviour, thus we assume normal execution for all future jobs and use C_j^N and U_j^N in the formula. The related workload of abnormal executions is explicitly summed up in G_k. We can provide *full timing guarantees* for τ_k at θ_0 if $|busy_k|^* \leq D_k$. If $|busy_k|^* > D_k$ we only provide *limited timing guarantees*, and $|busy_k|^*$ is an upper bound on the time the systems needs to go back to to *full timing guarantees* for τ_k, assuming no further faults occur. It is also possible to achieve tighter bounds, e.g., by applying the results in [CHL16b; BPD15] that require sorting the tasks in $hep(\tau_k)$ by their periods.

The question remains when to check if *timing guarantees* have changed, which can be done using two general approaches. One is to check periodically, where the period of this check can be determined depending on the needed granularity during system design. Alternatively, we can check in an event-driven manner. The moment abnormal execution is detected is a natural choice for an event, as this is the only point in time a change from *full timing guarantees* to *limited timing guarantees* may happen for the affected tasks. This is an important observation, as no checks are needed while all tasks have *full timing guarantees* if no abnormal execution occurs. A change from *limited timing guarantees* back to *full timing guarantees* for a task τ_k may happen when a task with higher priority than τ_k or an instance of τ_k itself finishes.

5.2.6 EVALUATIONS

We focused on two questions. First, we determined the possible acceptance ratios for *Systems with Dynamic Real-Time Guarantees* under different scenarios in a schedulability analysis. Afterwards, we explored the behaviour of task sets with high utilization under different fault rates in system state analysis, i.e., we analyzed the percentage of time where *full timing guarantees* are provided for these task sets under different fault rates.

SCHEDULABILITY ANALYSIS

We generated random implicit-deadline task sets with a given U_{sum}^N according to the UUniFast method [BB05], applying the suggestion by Emberson et al. [ESD10] to generate the task periods according to a log-uniform distribution over two orders of magnitude. Specifically, $\log_{10} T_i$ is a uniform distribution over $[1ms - 100ms]$. The WCET in the normal mode was set according to the utilization, i.e., $C_i^N = U_i \cdot T_i$. We evaluated task sets with 5 different cardinality values, 5, 10, 20, 50 and 100 tasks, and randomly picked 30%, 40%, 50%, 60% and 70% of

these tasks to be in $\mathbf{T}_{hard}$. We calculated C_i^A for $\tau_i \in \mathbf{T}_{hard}$ according to 3 different ratios, called WCET-Factors, to simulate 3 scenarios for software-based recovery of transient faults. We assume that for complete re-execution the fault detection takes 20% of the WCET without fault detection (only one detection at the end) and for checkpointing we assumed 40% overhead but that only 20% of the job has to be re-executed. To be precise:

- Re-Execution: $C_i^A \approx 1.83 \cdot C_i^N$ as $\frac{2.2}{1.2} \approx 1.83$
- Two Re-Executions: $C_i^A \approx 2.83 \cdot C_i^N$ as $\frac{3.4}{1.2} \approx 2.83$
- Checkpointing: $C_i^A \approx 1.14 \cdot C_i^N$ as $\frac{1.6}{1.4} \approx 1.14$

We considered two different values for the relation of C_i^A to C_i^N for $\tau_i \in \mathbf{T}_{soft}$: The same value as used for $\mathbf{T}_{hard}$ or 1.0, i.e., fault detection without any kind of fault recovery. For each of these in total $5 \cdot 5 \cdot 3 \cdot 2 = 150$ settings, we analyzed 1000 randomly generated task sets for each utilization value $U_{sum}^N \in [1\%, 100\%]$ with a step size of 1%. We tested the schedulability of the task sets as a *System with Dynamic Real-Time Guarantees* using 4 static-priority orderings: Rate Monotonic (RM), Criticality Monotonic (CM), Optimal Priority Assignment (OPA) [Aud91], and the ordering provided by Algorithm 1, all tested by the schedulability test in Theorem 5.1. In these tests, we dropped the condition $U_{sum}^A \leq 1$ according to the arguments at the end of Section 5.2.2. Later in this subsection, we perform a system state analysis, examining the behaviour of task sets with high utilization under different fault rates, to support this argument. We also tested the schedulability under EDF-VD [BBD+15], the dynamic-priority state-of-the-art scheduling algorithm for mixed-criticality systems, using the schedulability test provided by Baruah et. al in [BBD+15, Section 3]. Note that, contrary to a *System with Dynamic Real-Time Guarantees*, EDF-VD does not provide any guarantees for low-criticality tasks in high-criticality mode.

In addition to the acceptance ratio, we monitored if RM, CM, or OPA were able to schedule a task set that was not schedulable by Algorithm 1. However, this case never occurred and Algorithm 1 and OPA always resulted in an identical acceptance ratio which strongly supports our claim that Algorithm 1 provides an optimal assignment (Theorem 5.8). As the two curves are identical, only one curve, labeled *Optimal Assignment* (OA), is used to represent OPA and Algorithm 1 in Figure 5.3, showing the results for sets with 10 tasks, 5 of them in $\mathbf{T}_{hard}$, and a WCET-Factor of ≈ 1.83 for both $\mathbf{T}_{hard}$ and $\mathbf{T}_{soft}$. Since the general behaviour was similar under all 150 settings, we only provide a subset of the results.

The most interesting result presented in Figure 5.3 is the comparison between EDF-VD [BBD+15] and our optimal assignment (OA). While the curve for OA starts dropping earlier than the one of EDF-VD (Utilization 52% and 61% respectively), EDF-VD drops faster. Hence, from 72% onwards, OA can schedule more task sets than EDF-VD. In this area $\sum_{\tau_i \in \mathbf{T}_{hard}} U_i^A$ can be too large to find values for the virtual deadlines in EDF-VD as those virtual deadlines are generated from the original deadlines by multiplying with a factor ≤ 1. Since OA performs an exact test considering the actual deadlines it is still able to find a feasible schedule. Similar behaviour was observed in most settings.

In Figure 5.4, we compare the optimal assignment (OA) and EDF-VD based on the three different WCET-Factors. For a WCET-Factor of 1.14, EDF-VD always

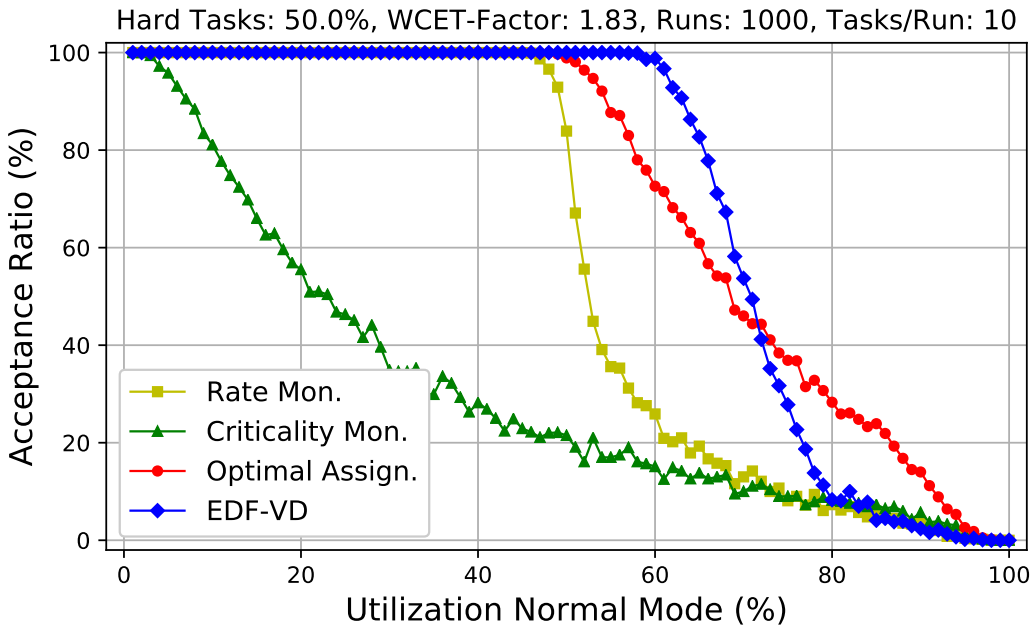


Figure 5.3: Acceptance ratio for 10 tasks per run. *Adapted from [BCH+16].*

outperforms OA. For WCET-Factors of 1.83 and 2.83, EDF-VD only performs better than OA up to a utilization of 71% and 56%, respectively. For higher utilization values OA is able to schedule more task sets than EDF-VD. The gap between EDF-VD and OA in the acceptance ratio for a given utilization seems reasonable if we consider that OA does not drop any tasks when only *limited timing guarantees* are provided, the scheduling overhead of EDF is in general larger than the overhead of static-priority scheduling, and that for EDF-VD online adaptation is necessary.

We also analyzed the schedulability as *Systems with Dynamic Real-Time Guarantees* under the Optimal Assignment (OA) considering the percentage of timing strict tasks and the size of the task set.

The effect that the percentage of timing strict tasks has on the schedulability is displayed in Figure 5.5. We considered 5 different rates for the percentage of *timing strict* tasks and only display the interesting utilization interval $[45\%, 95\%]$. When the percentage of *timing strict* tasks is higher, the acceptance rate drops earlier which was expected since we have to provide hard real-time guarantees for a higher percentage of tasks.

Furthermore, in Figure 5.6 we show the schedulability considering different set sizes, again showing the interesting utilization interval $[45\%, 75\%]$. If only the sets with 10, 20, 50 and 100 tasks are consider, the curve for the larger sets starts decreasing later but decrease faster and vice versa. Only the sets with 5 tasks behave slightly different due to the randomness of the input.

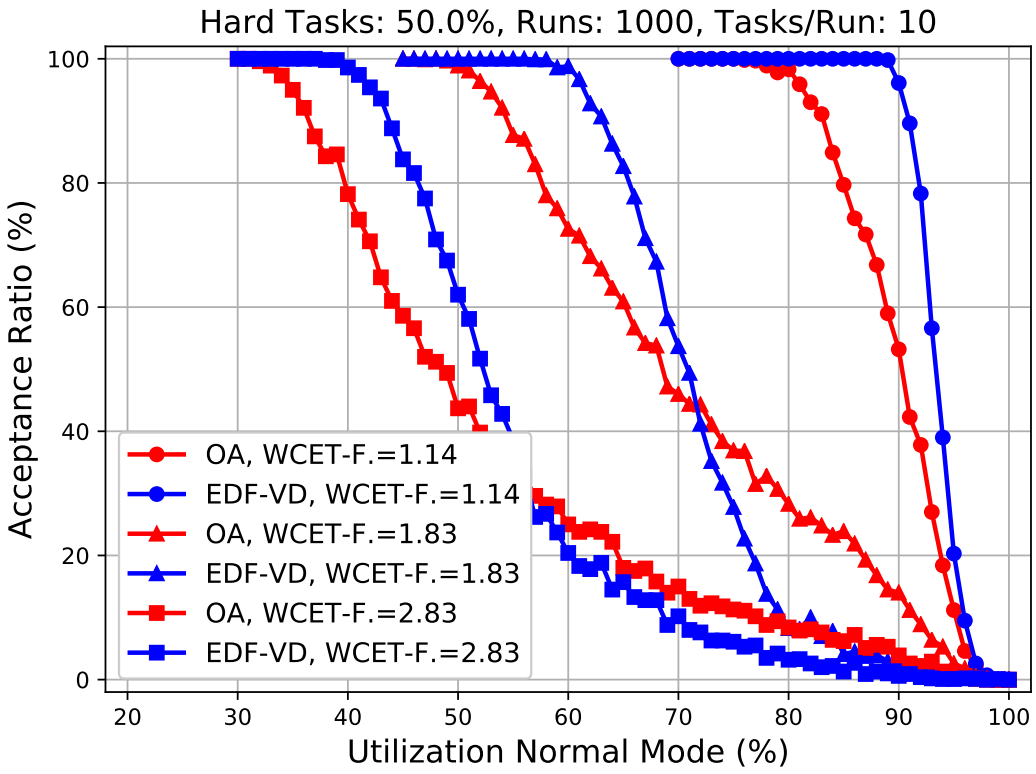


Figure 5.4: Comparison of OA and EDF-VD. *Adapted from [BCH+16].*
For a WCET-Factor of 1.14 EDF-VD is always superior to the Optimal Assignment (OA), while for WCET-Factors of 1.83 and 2.83 OA is better than EDF-VD for utilization values higher then 71% and 56% respectively.

SYSTEM STATE ANALYSIS

Figure 5.3 shows that OA has an acceptance ratio of 44.4% for task sets with 10 tasks, 50% tasks in $\mathbf{T}_{hard}$, a WCET-Factor of ≈ 1.83, and $U^N_{sum} = 70\%$, which means $U^A_{sum} \approx 128.1\%$. EDF-VD [BBD+15] achieved schedulability for 53.7% of these sets. As both algorithms only provide schedulability for roughly 50% of the task sets and the acceptance ratio is decreasing fast around 70% utilization, we say these sets have a *critical* utilization. For the first 40 of these randomly generated task sets with *critical* utilization that are schedulable according to Algorithm 1, we examined the system state, evaluating the percentage of time where *full timing guarantees* and where *limited timing guarantees* were provided.

full timing guarantees
limited timing guarantees

We used QEMU emulators under *Real-Time Executive for Multiprocessor Systems (RTEMS)* [Rte] version 4.11 where the used kernel is enhanced by patch #2772 [Che16d], enabling only one processor. The chosen board support package was *RealView Platform Baseboard Explore for Cortex-A9*. For each testing instance, the system was executed for one hour under different fault rates, i.e., on average 10^{-4}, $3 \cdot 10^{-4}$, 10^{-3}, $3 \cdot 10^{-3}$ and 10^{-2} faults per millisecond (f/ms). If an executed instance of τ_i is faulty, the corresponding WCET C^N_i becomes C^A_i. Whether the system can give *full timing guarantees* or only *limited timing guarantees* is decided by the system monitor presented in Section 5.2.5. The percentage of time the system

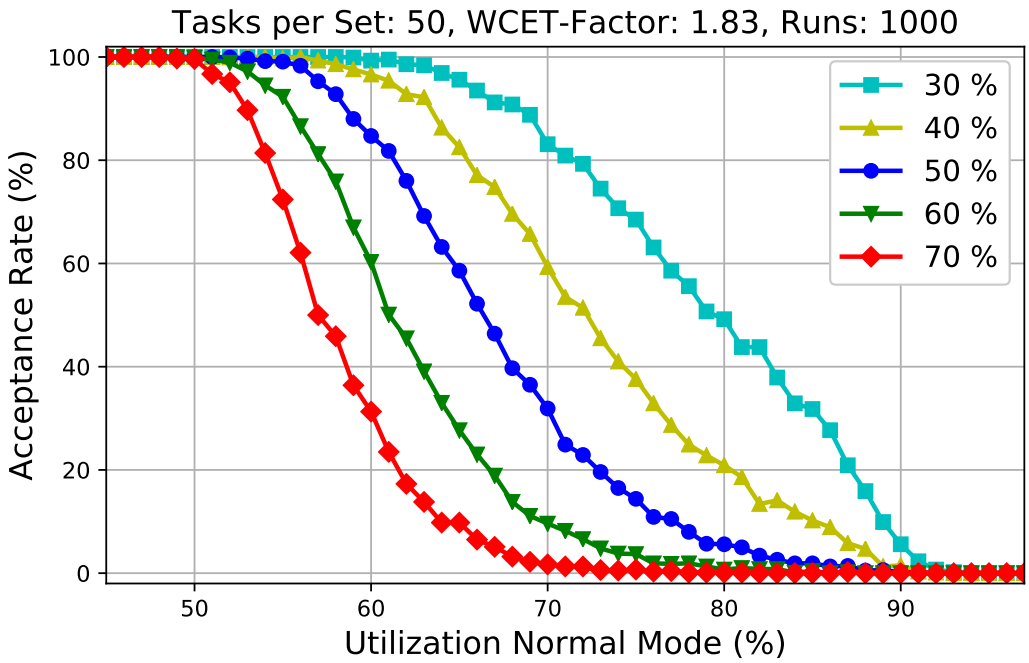


Figure 5.5: Acceptance rate for percentages of timing strict tasks. *Adapted from [BCH+16].* The acceptance rate drops earlier if the percentage of hard tasks is higher.

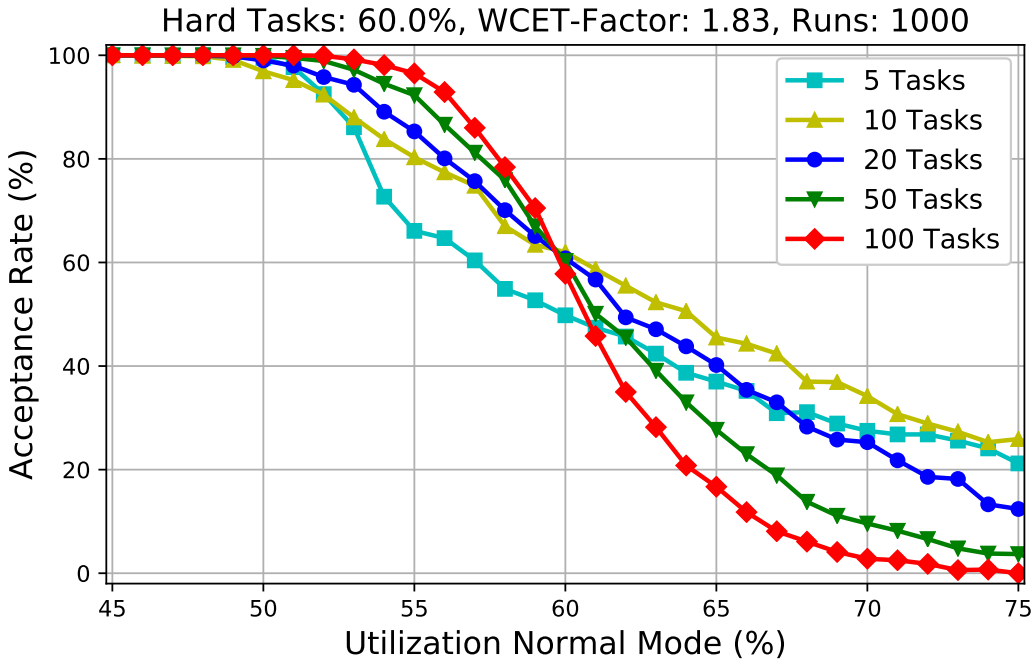


Figure 5.6: Acceptance rate for different set sizes. *Adapted from [BCH+16].* Larger task sets starts decreasing later but decrease faster. The slightly different behaviour of sets with 5 tasks is due to randomness effects.

was running with *full timing guarantees* is shown in Figure 5.7. The median of those 40 sets is colored orange, the blue box represents the interval from the first to the third quartile, and the black whiskers show the minimum and maximum.

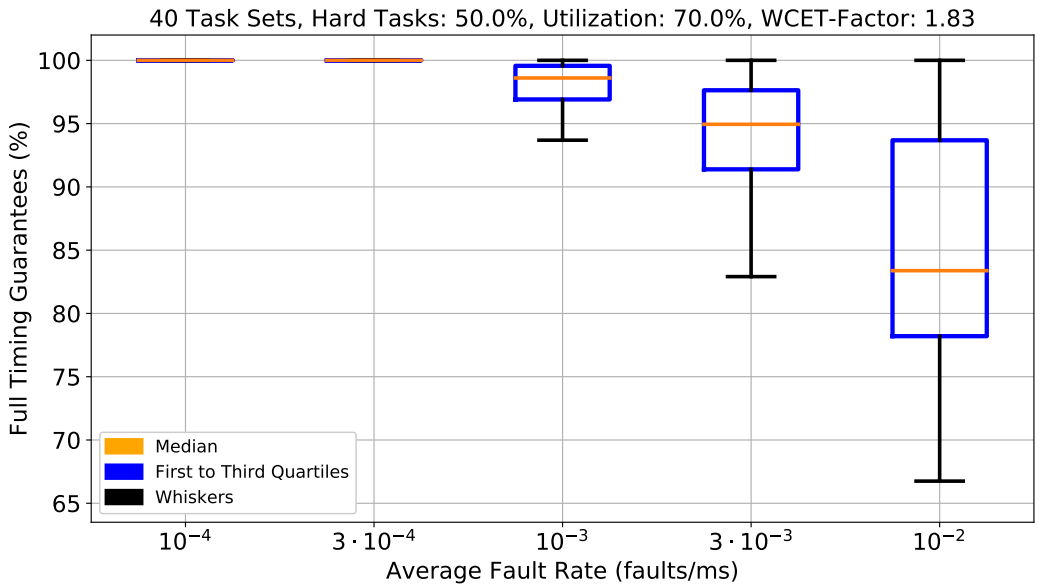


Figure 5.7: Percentage of time with *full timing guarantees* for task sets with critical utilization under different fault rates. *Adapted from [BCH+16].*

The system always provides *full timing guarantees* for fault rates of 10^{-4} and $3 \cdot 10^{-4}$ f/ms. When the fault rate is increased to 10^{-3} and $3 \cdot 10^{-3}$ the median value decreases to 98.6% and 94.9%. The third quartile is at 99.6% and 97.6%, and the first quartile is at 96.91% and 91.4%. If the fault rate is increased further to 10^{-2}, we observe 93.7%, 83.4%, and 78.2% for third quartile, median, and first quartile, respectively. This shows that even for higher fault rates under an, in general, difficult setting, *full timing guarantees* can still be provided for a reasonable percentage of time. However, for some of the task sets, the percentage of time where *full timing guarantees* can be given drops faster, as can be seen by the comparatively long lower whiskers for $3 \cdot 10^{-3}$ and 10^{-2}.

5.3 MULTIPROCESSOR SYSTEMS WITH DYNAMIC REAL-TIME GUARANTEES

After introducing *Systems with Dynamic Real-Time Guarantees* and exploring their properties in a uniprocessor environment, we extend the model to homogeneous multicore systems by introducing *Multiprocessor Systems with Dynamic Real-Time Guarantees* (MSDRTG) in Section 5.3.1 and detail the related schedulability test in Section 5.3.2. We provide two general approaches for MSDRTG, a partitioned approach in Section 5.3.3 and a semi-partitioned approach in Section 5.3.4. Afterwards, we discuss how processors suffering from abnormal execution behavior over a longer but limited interval of time, e.g., due to a high-criticality mode, intermittent faults, or processor clock speed drops resulting from overheating, can be compensated for by means of task migration, and introduce the concepts of *full compensation* and *partial compensation* in Section 5.3.5. We assess the developed techniques by means of comprehensive evaluations in Section 5.3.6. The results presented in this section appeared in *Do Nothing, but Carefully: Fault Tolerance*

Multiprocessor Systems with Dynamic Real-Time Guarantees

full compensation

partial compensation

with Timing Guarantees for Multiprocessor Systems devoid of Online Adaptation in PRDC 2018 [BSC18].

5.3.1 MULTIPROCESSOR SYSTEM MODEL

Multiprocessor Systems with Dynamic Real-Time Guarantees

Based on the general model for uncertain execution behaviour detailed in Section 2.6 and the Model of *Systems with Dynamic Real-Time Guarantees* in Section 5.2, we now define the model of *Multiprocessor Systems with Dynamic Real-Time Guarantees* (MSDRTG) as well as its respective properties, before establishing a suitable schedulability test. Our definition and the test assumes partitioned scheduling but can be extended to the semi-partitioned case later.

timing strict tasks
timing tolerable tasks

In a system $\mathbf{S}$, consider a set of n tasks $\mathbf{T} = \{\tau_1, \ldots, \tau_n\}$ with each task $\tau_i \in \mathbf{T}$ being either *timing strict*, i.e., $\tau_i \in \mathbf{T}_{hard}$, or *timing tolerable*, i.e., $\tau_i \in \mathbf{T}_{soft}$, so that $\mathbf{T}_{hard} \cap \mathbf{T}_{soft} = \emptyset$ and $\mathbf{T}_{hard} \cup \mathbf{T}_{soft} = \mathbf{T}$. The task set $\mathbf{T}$ is partitioned onto a set of m homogeneous processors, which means that $\mathbf{T}_j \cap \mathbf{T}_k = \emptyset$ for all $j \neq k$ with $j, k \in \{1, \ldots, m\}$ and $\mathbf{T}_1 \cup \mathbf{T}_2 \cup \cdots \cup \mathbf{T}_m = \mathbf{T}$. Each set of tasks $\mathbf{T}_p$ allocated to a processor p with $1 \leq p \leq m$ is identified as *subsystem* of $\mathbf{S}$ and scheduled according to a static-priority task order P_p. Let $\mathbf{T}_{p,hard}$ and $\mathbf{T}_{p,soft}$ be the subset of tasks on processor p that are in $\mathbf{T}_{hard}$ and $\mathbf{T}_{soft}$, respectively. For each $\tau_i \in \mathbf{T}$ no job can begin its execution before all previously arrived jobs released by the same task are completed, and no job of any $\tau_i \in \mathbf{T}$ must ever be aborted.

Definition 5.3 (Multiprocessor System with Dynamic Real-Time Guarantees)**.** *A system $\mathbf{S}$ with subsystems $\mathbf{T}_1, ..., \mathbf{T}_m$ is an MSDRTG if and only if each subsystem $\mathbf{T}_p$ satisfies the characteristics of a* System with Dynamic Real-Time Guarantees *in Definition 5.1, i.e., if under normal system behavior, all subsystems provide full timing guarantees, and under abnormal system behavior all affected subsystems provide at least limited timing guarantees for a bounded interval of time.*

full timing guarantees

A subsystem $\mathbf{T}_p$ provides *full timing guarantees* if hard real-time constraints are satisfied for each task $\tau_i \in \mathbf{T}_p$. More precisely, under normal system behavior, all jobs of each task $\tau_i \in \mathbf{T}_p$ meet their hard relative deadlines, i.e., $E_i^N = 0 \; \forall \tau_i \in \mathbf{T}_p$.

limited timing guarantees

In a subsystem $\mathbf{T}_p$ providing *limited timing guarantees*, service level guarantees may be downgraded for some timing tolerable tasks $\mathbf{T}_{p,soft} \subseteq \mathbf{T}$, such that all $\tau_i \in \mathbf{T}_{p,soft}$ have (at least) bounded tardiness, i.e., $0 \leq E_i^A < \gamma_i \; \forall \; \tau_i \in \mathbf{T}_{p,soft}$ for a fixed value γ_i. However, each instance of each $\tau_i \in \mathbf{T}_{p,hard}$ meets its hard relative deadline irregardless, i.e., $E_i^A = 0 \; \forall \; \tau_i \in \mathbf{T}_{p,hard}$. An MSDRTG $\mathbf{S}$ provides *full timing guarantees* if all subsystems $\mathbf{T}_p$ provide *full timing guarantees*, and *limited timing guarantees* if at least one subsystem provides only *limited timing guarantees*.

5.3.2 SCHEDULABILITY TEST

Owing to the fact that an MSDRTG $\mathbf{S}$ as defined above is composed of a set of subsystems $\{\mathbf{T}_1, \ldots, \mathbf{T}_m\}$, each of which comprises a set of tasks scheduled according to an individual static-priority order, the schedulability test for uniprocessor SDRTG proposed in Theorem 5.1 can be adapted for MSDRTGs as follows:

Theorem 5.13 (Exact Schedulability Test for MSDRTGs with Constrained Deadlines under Partitioned Scheduling). *A task system $\boldsymbol{S}$ with a given partition of the task set $\boldsymbol{T}$ into subsets $\boldsymbol{T}_1, \ldots, \boldsymbol{T}_m$, a given partition of $\boldsymbol{T}$ into $\boldsymbol{T}_{hard}$ and $\boldsymbol{T}_{soft}$, and a given priority order P_p for each subset $\boldsymbol{T}_p \in \{\boldsymbol{T}_1, \ldots, \boldsymbol{T}_m\}$ is an MSDRTG if and only if:*

1. *Each subsystem $\boldsymbol{T}_p$ can be scheduled according to TDA [LSD89] under the given priority order P_p and normal system behavior, i.e., $C_i = C_i^N \; \forall \tau_i \in \boldsymbol{T}_p$.*
2. *For each subsystem T_p all $\tau_i \in \boldsymbol{T}_{p,hard}$ can be scheduled according to TDA under the given priority order P_p and abnormal system behavior, i.e., $C_i = C_i^A \; \forall \tau_i \in \boldsymbol{T}_p$*
3. *For each subsystem $\boldsymbol{T}_p$, it holds that $U_{p,sum}^A \leq 1$.*

Since Theorem 5.13 is a direct extension of the test for uniprocessor systems in Theorem 5.1, a proof is omitted. Similar to the uniprocessor case, we omit the condition that $U_{p,sum}^A \leq 1$ in our further discussions as well as in the evaluation.

Having provided the formal specifications of an MSDRTG as well as a suitable schedulability test, we now explain how to obtain the actual task partition and priority assignment. Afterwards, we detail how the number of task sets that are feasibly schedulable as an MSDRTG can be increased by utilizing task migration in a semi-partitioned approach.

5.3.3 PARTITIONED SCHEDULING

partitioned scheduling

Adopting the *partitioned scheduling* paradigm, two distinct problems must be considered: First, the overall task set $\mathbf{T}$ must be partitioned so that each task $\tau_i \in \mathbf{T}$ is statically allocated to a specific processor. After that, a priority order must be specified for each subsystem $\mathbf{T}_p$.

For the actual partitioning, one can apply any partitioning heuristic based on any pre-sorting as detailed in Section 2.5. Multiple combinations of pre-sorting and partitioning heuristics will be compared in the evaluation in Section 5.3.6. However, whether a possible assignment of task τ_t to processor p satisfies the conditions for an MSDRTG in Theorem 5.13 requires applying the *Feasible Priority Assignment Algorithm* for (uniprocessor) SDRTGs provided in Algorithm 1, considering the previously assigned tasks T_p together with the candidate τ_t. The reason being that even if τ_t cannot be scheduled together with all previously assigned $\tau_i \in \mathbf{T}_p$ under the previous priority order P_p, the set $\mathbf{T}_p \cup \{\tau_t\}$ may be schedulable under another priority order. Hence, the priority assignment on each processor may change whenever a new task is allocated to that processor.

5.3.4 SEMI-PARTITIONED SCHEDULING

semi-partitioned scheduling

subtasks

When following a partitioned scheduling approach, it may happen that no further tasks can be added to a subsystem $\mathbf{T}_p$, but some spare capacity is left on the respective processor p. In this event, it seems sensible to allow tasks to split into so-called *subtasks*, which are executed on more than one processor, a so-called *semi-partitioned scheduling*. More precisely, we attempt to fill each processor to the

maximum capacity by assigning the largest possible task shares. As a consequence, task sets can be scheduled as an MSDRTG exhibiting a higher system utilization than feasible under partitioned techniques.

This entails additional challenges: It is necessary to decide which tasks to share, how to compute the spare capacity of a processor, i.e., how to derive the largest possible WCET of a subtask, and how to specify a subtask's deadline. In addition, it must be determined which priorities are assigned to the individual subtasks and how to ensure the correct execution order of a sequence of subtasks.

Contemplate a task $\tau_t \in \mathbf{T}$ that cannot be allocated to any processor under a given partition. In this case, a certain task τ_s, which can be either the task τ_t that could not be assigned successfully or a task that is already assigned to a processor, i.e., a $\tau_s \in \mathbf{T}_p$, may be shared between at least two processors such that τ_s is not executed by more than one processor at the same time. As a consequence of this sequential execution, a release of a subjob of the *shared task* τ_s on a processor $p+1$ must not take place before the respective subjob executed on processor p is finished. To ensure this property, it seems reasonable to execute each subjob as early as possible, i.e., as soon as it is released. For this reason, we always assign the highest priority in the subsystem to each subtask of τ_s and the subtask's relative deadline is, hence, the same as the subtask's WCET. This concept is called *C=D scheme* in the literature and was introduced by Burns et al. [BDW+12]. Owing to the fact that the remaining capacity on each processor does not suffice to execute τ_s completely, all processors maintaining one share of τ_s are filled to maximum capacity after the splitting operation. The only exception is the processor where the last subtask is assigned. In contrast to the remaining processors, another task or a subtask of another shared task can still be assigned to this processor. Since the next task instance cannot be released before the previous one is finished, the execution order of the first shared task τ_s cannot be disrupted anyway. Therefore, a subtask of τ_u that is later assigned to the same processor takes higher priority than this previously assigned last subtask of τ_s. Summarizing these considerations, a precise definition of the term *shared task* is given.

shared task

Definition 5.4 (Shared Task)**.** *A task $\tau_s \in \boldsymbol{T}$ is a shared task if its execution is not restricted to one processor and if the following conditions hold:*

- *A shared task is never executed by more than one processor at the same time, but is successively passed on to the next processor as soon as its execution budget on the current processor is exhausted.*
- *A shared task is always scheduled under the highest priority on each processor. The only exception to this rule may be the last subtask of a task. In this case, the later allocated subtask has a higher priority, since the previously assigned task segment is the last segment of the related task.*
- *Concerning all calculations involving the shared task's worst-execution time, C_s^A is used regardless of the actual system behavior.*

Each share of τ_s, denoted as a subtask of τ_s, is treated as an individual task with the specification $\tau_{s,p} = (C_{s,p}^N, C_{s,p}^A, D_s, T_s)$.

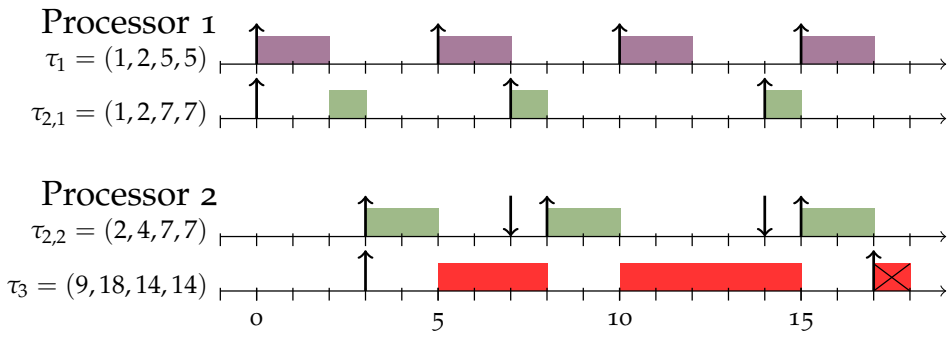


Figure 5.8: The problem of release jitter. *Adapted from [BSC18].*
Task τ_3 misses its deadline due to the release jitter of subjob $\tau_{2,2}$ on processor 2, resulting from the execution of subjob $\tau_{2,1}$ on processor 1.

Having clarified how to specify the deadlines of each subtask, how to assign its priority, and how to ensure the correct execution order of a sequence of subtasks, we go on to discuss how to compute the maximum amount of time a subtask can execute on a particular processor p, i.e., its worst-case execution time, which we assumed to be given until now. Assume that a chunk of task τ_s should be assigned to processor p and let the set of previously allocated tasks $\mathbf{T}_p$ with a priority order P_p be given. Utilizing the method proposed by Kato and Yamasaki in [KY09], it is possible to compute the maximum amount of time τ_s can be executed on a respective processor without causing a specific task $\tau_k \in \mathbf{T}_p$ to miss its deadline. The minimum value derived for any $\tau_i \in \mathbf{T}_p$ determines the worst-case execution time $C_{s,p}$ of subtask $\tau_{s,p}$ on p. Accordingly, a subtask of length $C_{s,p}$ is split off and the worst-case execution time budget of τ_s, i.e., the remaining workload to be distributed to processors with spare capacity, is reduced by $C_{s,p}$. This is repeated until either no further workload needs to be distributed or the system is deemed to be unschedulable. Please note that we always refer to the abnormal WCET C_s^A throughout the task splitting.

When a shared task τ_s is migrated from processor p to processor $p+1$, the so-called *release jitter* of each subtask must be considered as well, i.e., the difference between its best- and worst-case response time. An example is provided in Figure 5.8, outlining 3 tasks being distributed to two processors. Task τ_1 and τ_3 are statically allocated to processor 1 and 2, respectively, the first subtask $\tau_{2,1}$ of the shared task τ_2 is always executed on processor 1, and the second subtask $\tau_{2,2}$ is always executed on processor 2. Due to the additional workload produced by the higher-priority task τ_1 in the interval $[0;2]$, the first instance of $\tau_{2,1}$ has a larger response time than its second and third instance. Resulting from this, $\tau_{2,2}$ is not released exactly periodically on processor 2, but with a release jitter of 2 time units, which, in turn, leads to a deadline miss of τ_3, indicated by the cross. This can be avoided by releasing $\tau_{2,2}$ exactly periodically or sporadically, such that its maximum workload contributed by $\tau_{2,2}$ in any interval of length 14 is 4 time units and, as a consequence, τ_3 always meets its deadline.

release jitter

Owing to the fact that each subtask (except possibly the last one) of a shared task is always executed under the highest priority in the respective subsystem, its best- and worst-case response time do not differ. Other factors potentially inducing jitters - although on a smaller scale - are the time required for task preemption

and the migration time. We assume the preemption time to be sufficiently small to be neglected and that for a given subtask the related migration time is always constant. Accordingly, we can consider all subtasks as periodic tasks without any release jitter in our analysis. Otherwise, release enforcement techniques can be used as, e.g., explained in [BL92] or in Section 6.2.2.

The remaining questions is which task should be chosen as a shared task. Two diverging strategies can be pursued: either the task τ_t which could not be allocated during the task partitioning, or an already assigned task $\tau_i \in \mathbf{T}_p$. While the first case can be handled as detailed above, the latter one can be dealt with as proposed by Lakshmanan et al. [LRL09]. More precisely, the highest-priority task in $\mathbf{T}_p$ is chosen to be shared between processors. The reason is that the highest-priority task on each processor typically has a short relative deadline, whereas the task τ_t to be assigned usually has a longer deadline[2] as well as a larger WCET. Moreover, it is well known that assigning a higher priority to a task with a shorter deadline is in general favorable [LW82].

Accordingly, if the task allocation strategy is not able to assign a certain task $\tau_t \in \mathbf{T}$, we look for a processor p where τ_t can be assigned if the highest priority task τ_p^h is removed from p, i.e., τ_t can be scheduled along with $\mathbf{T}_p \setminus \left\{\tau_p^h\right\}$. Afterwards, we reassign τ_p^h, potentially dividing it into multiple subtasks and sharing it across a number of processors applying the previously explained method until the workload of τ_s is completely distributed. As soon as each $\tau_i \in \mathbf{T}$ is assigned to one (or more) processor(s), the algorithm terminates successfully. Otherwise, the task set $\mathbf{T}$ is declared to be unschedulable as an MSDRTG.

Further strategies for increasing the number of schedulable task sets can be applied as well, for example, removing previously assigned tasks τ_r from a subsystem in order to allow a feasible allocation of the currently considered task τ_t, while reconsidering τ_r later on. A comprehensive survey including conceivable approaches can be found in [DB11a].

5.3.5 COMPENSATING FAULTY PROCESSORS BY TASK MIGRATION

So far, we proposed a system model that enables dynamic timing guarantees, i.e., full timing guarantees for timing strict tasks and limited timing guarantees for timing tolerable tasks, in a multiprocessor scenario. Moreover, we suggested a number of scheduling strategies that allow such an MSDRTG to be established.

We now introduce a compensation technique for components exhibiting abnormal behavior for a limited interval of time, aiming to achieve additional tolerance with respect to, for instance, intermittent faults, mixed-criticality behaviour, or a decreased CPU clock frequency. We term a system component, i.e., a subsystem $\mathbf{T}_p$, *corrupted* if it exhibits abnormal execution behavior.

2 This depends on the order in which the tasks are partitioned. Nevertheless, even for an inverted DM order, the unassigned task τ_t most commonly has a relative deadline that is not considerably shorter than the one of the highest-priority task $\tau_p^h \in \mathbf{T}_p$.

A certain subset of tasks scheduled on the corrupted subsystem $\mathbf{T}_p$ must be migrated to other processors to satisfy their timing requirements. Since timeliness is by definition guaranteed for all tasks $\tau_i \in \mathbf{T}_{p,hard}$ under abnormal system behavior, these tasks may remain on the corrupted processor even if the subsystem exhibits abnormal behavior for an infinite time interval, as long as for each $\tau_i \in \mathbf{T}_p$ a correct result can be obtained within the respective abnormal WCET C_i^A. For all tasks $\tau_i \in \mathbf{T}_{p,soft}$, bounded tardiness is already guaranteed if $U_{p,sum}^A \leq 1$ holds. However, we remove this condition, assuming that the intervals in which the system exhibits abnormal behavior are significantly shorter than those under normal behavior. As a consequence, neither timeliness nor bounded tardiness can be ensured for any $\tau_i \in \mathbf{T}_{p,soft}$. Therefore, their migration is beneficial.

For this migration process, it is necessary to determine a specific order in which the tasks in $\mathbf{T}_p$ are migrated. However, we cannot decide which tasks to favor by means of their particular function or purpose, since all timing tolerable tasks are considered equally important in a SDRTG. Hence, we categorize the tasks in $\mathbf{T}_{p,soft}$ based on their timing properties under abnormal behaviour:

- Tasks that meet their hard deadline under abnormal system behavior anyway, denoted as $\mathbf{T}_{p,soft}^{guar}$. These can be neglected in the migration process.
- Tasks for which no timeliness but at least bounded tardiness can be guaranteed under abnormal system behavior, denoted $\mathbf{T}_{p,soft}^{bd}$.
- Tasks for which no guarantees can be given under abnormal system behavior, denoted as $\mathbf{T}_{p,soft}^{unbd}$. Note that this situation is only possible since we dropped the condition that $U_{p,sum}^A \leq 1$.

Depending on the number of corrupted subsystems as well as on their particular task sets, two levels of compensation are achievable. A corrupted processor can be *fully compensated* if the system maintains an MSDRTG after the migration process. Otherwise, it can be *partially compensated* if all $\tau_i \in \mathbf{T}_{hard}$ meet their hard deadlines under abnormal system behavior, whereas at least bounded tardiness can be guaranteed for each $\tau_i \in \mathbf{T}_{soft}$, provided that the destination processor(s) of the task migration are not affected by corruption.

Regarding the migration process, we begin with the tasks in $\mathbf{T}_{p,soft}^{unbd}$, owing to the fact that they need to be migrated to achieve both full and partial compensation. Concerning this subset, we make the following observations: Since the total utilization up to a certain priority level is an increasing function with respect to the priority, only tasks with a priority lower than any $\tau_i \in \mathbf{T}_{p,hard}$ can be in $\mathbf{T}_{p,soft}^{unbd}$. Moreover, if a task τ_j is in $\mathbf{T}_{p,soft}^{unbd}$, all tasks with a lower priority than τ_j are in $\mathbf{T}_{p,soft}^{unbd}$ as well, while each $\tau_i \in \mathbf{T}_{p,soft}^{bd}$ has a higher priority than each $\tau_j \in \mathbf{T}_{p,soft}^{unbd}$. Accordingly, $\mathbf{T}_{p,soft}^{unbd}$ can be computed easily. The tasks in $\mathbf{T}_{p,soft}^{unbd}$ are considered in deadline monotonic order during the migration process.

After $\mathbf{T}_{p,soft}^{unbd}$ has been determined, we try to assign each $\tau_j \in \mathbf{T}_{p,soft}^{unbd}$ to a non-corrupted processor that yields *full timing guarantees* after the migration process. If τ_j cannot be allocated to such a processor or no such processor exists, we search for a processor q where timeliness can still be guaranteed for all tasks $\tau_i \in \mathbf{T}_{q,hard}$, while bounded tardiness is ensured for all tasks $\tau_i \in \mathbf{T}_{q,soft} \cup \tau_j$. If at least one task in $\mathbf{T}_{p,soft}^{unbd}$ cannot be assigned either way, the corrupted processor(s) *cannot*

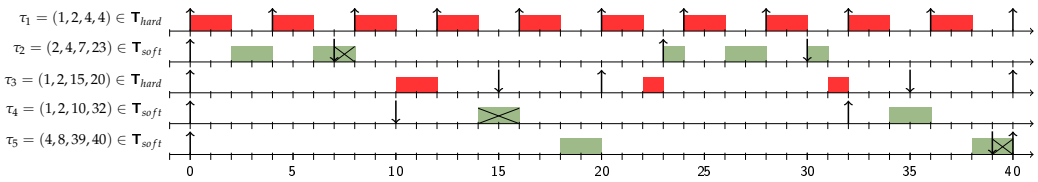


Figure 5.9: Migration example for $\mathbf{T}_p$. *Adapted from [BSC18].* $\mathbf{T}_{p,hard} = \{\tau_1, \tau_3\}$, $\mathbf{T}_{p,soft} = \{\tau_2, \tau_4, \tau_5\}$, and $U^A > 1$. At first, $\tau_5 \in \mathbf{T}^{unbd}_{p,soft}$ is migrated, resulting in $U^A \leq 1$. Afterwards, τ_2 is migrated since $\tau_2, \tau_4 \in \mathbf{T}^{bd}_{p,soft}$ and τ_2 has higher priority. As a result, τ_4 meets its deadline.

be compensated. If, in contrast, migration of at least one task $\tau_j \in \mathbf{T}^{unbd}_{p,soft}$ leads to bounded tardiness for some of the tasks in $\mathbf{T}_{q,soft} \cup \tau_j$ the corrupted processor(s) can be *partially compensated*. Otherwise, if all non-corrupted subsystems exhibit the characteristics of a *System with Dynamic Real-Time Guarantees* after the migration of $\mathbf{T}^{unbd}_{p,soft}$, we continue to migrate the tasks in $\mathbf{T}^{bd}_{p,soft}$ in deadline monotonic order until either the characteristics of an MSDRTG are restored for the system $\mathbf{S}$ (in this case, the corrupted processor(s) can be *fully compensated*) or one task $\tau_j \in \mathbf{T}^{bd}_{p,soft}$ cannot be assigned to any processor q in such a way that the properties of a *System with Dynamic Real-Time Guarantees* are maintained for $\mathbf{T}_p \cup \tau_j$. Please note that migrating a small number of tasks $\tau_j \in \mathbf{T}^{bd}_{p,soft}$ from a corrupted subsystem $\mathbf{T}_p$ to another one may already reduce the workload on $\mathbf{T}_p$ enough to provide *full compensation*, making further migration unnecessary.

partial compensation

full compensation

Concerning the actual migration, we assume that all instances of a task τ_i that is migrated from a subsystem $\mathbf{T}_p$ to a subsystem $\mathbf{T}_q$ are terminated on processor p before the migration and restarted on processor q afterwards. More precisely, the release of the first instance of τ_i on q occurs in the same moment in which its next release on p would have taken place. This indeed implies that all jobs terminated on p are lost which, at first glance, contradicts the idea of *Systems with Dynamic Real-Time Guarantees*. However, we assume that compensation techniques are utilized when the abnormal execution behaviour occurs for an unusually long interval. For instance, we assume that compared to transient faults, abnormal behaviour due to intermittent faults over a long interval of time is rare. Hence, sacrificing a few jobs can be tolerated for the benefit of system robustness.

By way of illustration, consider Figure 5.9 which portrays a corrupted subsystem under abnormal behavior with 5 distinct tasks. τ_1 and τ_3 are part of $\mathbf{T}_{hard}$. Since $U^A_{sum} \approx 1.036 > 1$, we categorize the tasks as follows: $\tau_5 \in \mathbf{T}^{unbd}_{p,soft}$ and $\tau_2, \tau_4 \in \mathbf{T}^{bd}_{p,soft}$. If τ_5 can be successfully migrated to another processor, the corrupted processor is *partially compensated*. If, moreover, τ_2 can be migrated, τ_4 meets its deadline under abnormal system behavior and the *System with Dynamic Real-Time Guarantees* property is restored for $\mathbf{T}_p$. Finally, the corrupted subsystem is *fully compensated* if τ_5 and τ_2 can be migrated, such that the system $\mathbf{S}$ exhibits the characteristics of an MSDRTG thereafter.

Although enabling full or partial compensation increases the robustness and thus the safety of an MSDRTG, a compromise must be made, since this leads to a smaller number of schedulable task sets. We will evaluate this tradeoff more thoroughly in Section 5.3.6.

Unfortunately, our proposed method is not applicable to recover from permanent faults. This follows from the fact that a system component affected by a permanent fault is entirely inoperable, for which reason not only all *timing tolerable tasks* but, in addition, all *timing strict tasks* need to be migrated to another subsystem. In this event, the migration itself leads to manifold problems for timing strict tasks, e.g., ensuring their timeliness during the migration process. Therefore, addressing this issue is beyond our scope.

timing tolerable tasks
timing strict tasks

5.3.6 EVALUATION

In the following, we discuss the results of our comprehensive evaluations, in the course of which we analyzed the schedulability of randomized synthetic task sets as MSDRTGs under different processor assignment, task splitting, and compensation techniques.

EXPERIMENT SETUP

We randomly generated implicit-deadline task sets, i.e., $D_i = T_i \; \forall \tau_i$. The number of processors m and the number of tasks n in the set differs depending on the analyzed setting, we chose $m = 4$, 8, or 16, and $n = 40$, 80, or 160. For m processors, the values of U^N_{sum} ranged from $2\% \times m$ to $100\% \times m$ with steps of $2\% \times m$. For a given m, n, and total utilization U^N_{sum}, we generated tasks according to the UUniFast method [BB05]. For each setting and each utilization value, 1000 randomly generated task sets were evaluated under 16 scheduling strategies. We also evaluated if full or partial compensation was possible for each task set, assuming that one processor was corrupted. The task periods were drawn randomly, according to a log-uniform distribution with two orders of magnitude, as suggested by Emberson et al. [ESD10], i.e., $\log_{10} T_i$ was a uniform distribution over $[1ms - 100ms]$. The WCET under normal system behavior was set according to the utilization, i.e., $C^N_i = U_i \cdot T_i$. Finally we randomly chose 50% of the tasks to be in $\mathbf{T}_{hard}$, with the remaining tasks being assigned to $\mathbf{T}_{soft}$.

Similar to the uniprocessor case, we evaluated a fault-recovery scenario, considering one re-execution, two re-executions, and checkpointing of a task, with the following ratios between C^N_i and C^A_i:

- Re-Execution: $C^A_i \approx 1.83 \cdot C^N_i$ as $\frac{2.2}{1.2} \approx 1.83$
  One fault detection with an assumed overhead of 20% at the end of the normal execution. If a fault is detected, the job is completely re-executed.
- Two Re-Executions: $C^A_i \approx 2.83 \cdot C^N_i$ as $\frac{3.4}{1.2} \approx 2.83$
  Two re-executions and two fault detections, one after the normal execution and one after the first re-execution.
- Checkpointing: $C^A_i \approx 1.14 \cdot C^N_i$ as $\frac{1.6}{1.4} \approx 1.14$
  The occurrence of a fault is tested at multiple checkpoints during the normal execution. We assumed a total overhead of 40% for the detection and that 20% of the task have to be re-executed.

We used the same ratio values, denoted *WCET-factors*, for both $\mathbf{T}_{hard}$ and $\mathbf{T}_{soft}$. As individual tasks with an abnormal utilization over 100% can, by default, not be scheduled on one processor, they were discarded during the random generation and substituted with new tasks.

EVALUATION RESULTS

In our evaluations, we tried to allocate tasks to processors for different combinations of task pre-orders and partitioning strategies. During the task pre-ordering, $\mathbf{T}_{hard}$ and $\mathbf{T}_{soft}$ were *not* sorted separately. We considered the following pre-orders:

1. Rate Monotonic Order (RM): The tasks were sorted in increasing order of their period T_i.
2. Inverted Rate Monotonic Order (IRM): Tasks with longer period T_i were allocated first.
3. Utilization Monotonic Order (UM): Tasks with higher utilization U_i^N were allocated first.

assignment strategy

We considered First-Fit (FF), Best-Fit (BF), Worst-Fit (WF), and Arbitrary-Fit (AF) as *assignment strategies*. The three pre-orders combined with the four assignment strategies led to a total of 12 basic partitioned scheduling approaches.

The results for a setup with 8 processors, 80 tasks, and a WCET-factor of 1.83, i.e., one re-execution, can be found in Figure 5.10. The labels indicate the considered pre-order and the applied assignment strategy, e.g., RM-FF for rate monotonic pre-order with first-fit assignment strategy.

In Figure 5.10(a), the assignment strategies are compared under a rate monotonic pre-order. RM-FF and RM-BF performed nearly identical with a slight advantage for RM-BF. Both acceptance ratios start dropping noticeably at $80\% \times m$, whereas RM-WF performed worst and the acceptance ratio breaks down $25\% \times m$ earlier than for RM-BF and RM-FF. This is due to the fact that under a worst-fit approach, the utilization is distributed equally while under the first-fit and best-fit strategies, the processors are filled as densely as possible. Hence, a single task with a long period and high utilization can easily lead to a situation in which no processor has sufficient remaining capacity when the worst-fit strategy is combined with a rate-monotonic pre-order. RM-AF performs slightly better than RM-WF because the randomness of the approach sometimes prevents the aforementioned worst-case scenario. Since RM-BF and RM-WF performed best and worst, they serve as reference values for all other approaches in the following subfigures of Figure 5.10.

Regarding the inverted rate-monotonic (IRM) order, IRM-FF and IRM-BF as well as IRM-WF and IRM-AF performed similar, as shown in Figure 5.10(b). While IRM-FF and IRM-BF were slightly worse than RM-BF, IRM-WF and IRM-AF accepted more task sets than RM-WF. With a utilization-monotonic (UM) pre-order, all strategies led to a nearly identical acceptance ratio, especially between RM-BF and RM-WF, as shown in Figure 5.10(c).

In addition, we examined three partitioned approaches where the tasks in $\mathbf{T}_{hard}$ and $\mathbf{T}_{soft}$ were assigned separately, namely:

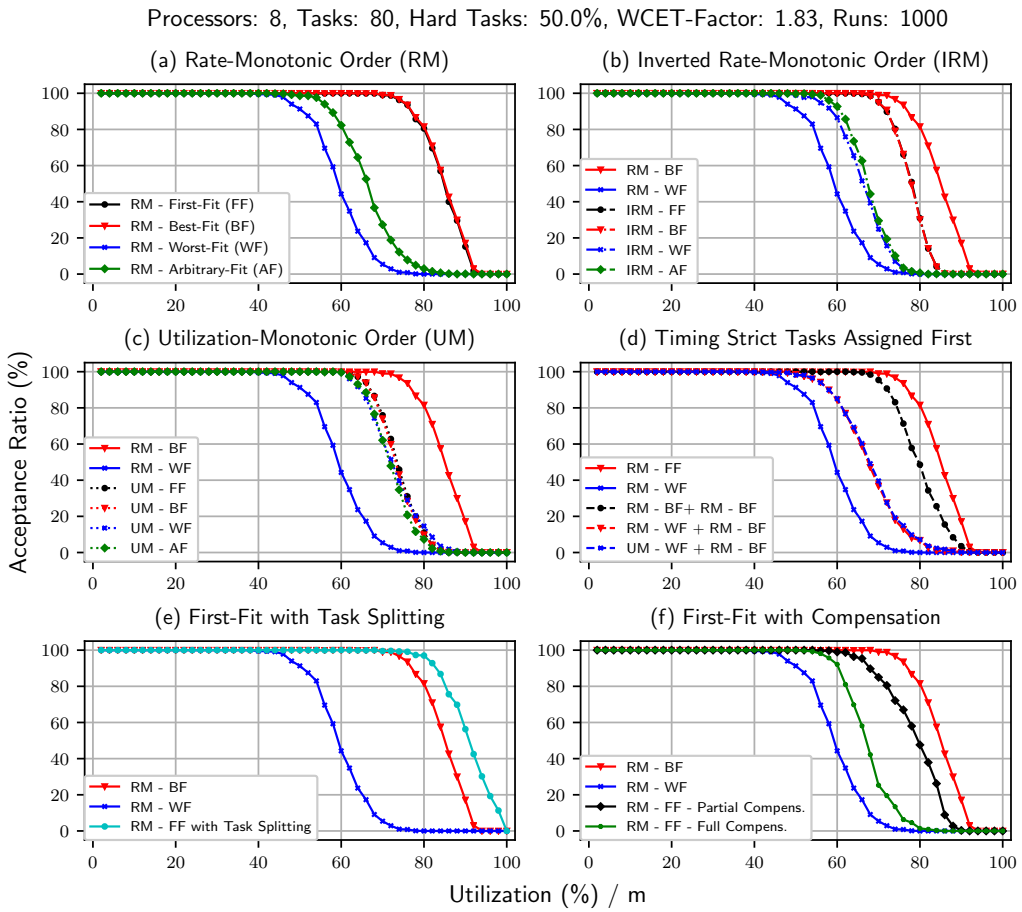


Figure 5.10: (a)-(d) Comparison of different partitioning strategies. The best and the worst strategy (RM-BF ans RM-WF) are further compared to (e) our semi-partitioned approach with highest-priority task splitting, and (f) our compensation techniques. *Adapted from [BSC18].*

1. RM-BF+RM-BF: First, the tasks in $\mathbf{T}_{hard}$, then the tasks in $\mathbf{T}_{soft}$ were partitioned using the RM-BF approach.
2. RM-WF+RM-BF: First, RM-WF was applied for $\mathbf{T}_{hard}$, and afterwards RM-BF was applied for $\mathbf{T}_{soft}$.
3. UM-WF+RM-BF: After the tasks in $\mathbf{T}_{hard}$ were assigned according to UM-WF, $\mathbf{T}_{soft}$ was assigned via RM-BF.

The results are shown in Figure 5.10(d). RM-BF+RM-BF surpasses the other two approaches but is still outperformed by RM-FF. The two other approaches performed slightly better than RM-WF.

From Figures 5.10(a)-(d), we conclude that RM-BF is superior among the considered partitioned scheduling approaches, while RM-WF performed worst. All other applied pre-orders and partitioned strategies as well as assigning $\mathbf{T}_{hard}$ and $\mathbf{T}_{soft}$ separately led to an acceptance ratio between RM-WF and RM-BF.

Aiming to analyze the benefit of semi-partitioned scheduling with respect to the schedulability, we implemented a rate-monotonic first-fit strategy with highest-priority task splitting, denoted as RM-FF-TS. Figure 5.10(e) shows that RM-FF-TS was superior to RM-FF in the evaluation, i.e., the acceptance ratio of

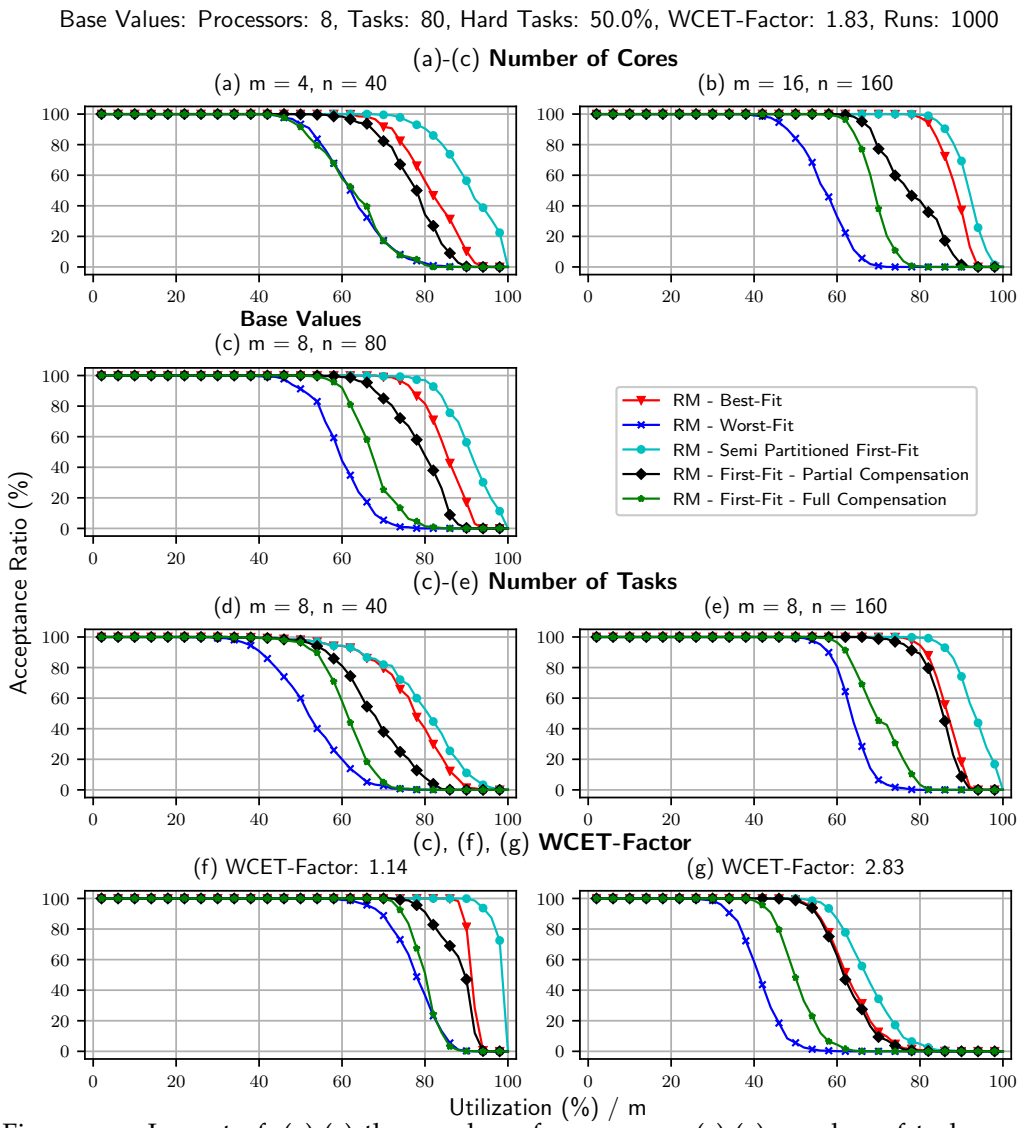


Figure 5.11: Impact of: (a)-(c) the number of processors, (c)-(e) number of tasks, and (c),(f),(g) WCET-Factor on the acceptance ratio. *Adapted from [BSC18].*

RM-FF-TS drops roughly 6% × m later than for RM-BF. Even for a utilization of 98% × m, some task sets were still schedulable as an MSDRTG.

Finally, we examined how providing full or partial compensation of one corrupted processor affects the schedulability. To be more precise, we considered all processors to be corrupted individually, and determined if the respective processor could be compensated. If *full (partial) compensation* was possible for all processors, the system provided full (partial) compensation. In Figure 5.10(f), the resulting detriment can be observed when applying RM-FF for both the initial partition and the compensation. While for full compensation the loss was nearly 20% × m, the loss for partial compensation was 10% × m. This means, *full* and *partial compensation* are often achieved for the expense of 60% × m and 70% × m system utilization in this setting.

full compensation
partial compensation

Since the relation between the considered scheduling approaches was similar in all settings, we focused on analyzing the impact of the different parameters on the schedulability. The acceptance ratios for RM-BF and RM-WF as well as

for the semi-partitioned scheduling approach and the compensation techniques are shown in Figure 5.11. We evaluated the effect of three different parameters individually:

1. **Number of processors** in Figure 5.11(a)-(c): The acceptance ratio increased, when the number of processors was increased. This result was expected, since for a constant average processor utilization and a constant average number of tasks per processor, a larger number of processors results in more possibilities to allocate the tasks. Only for RM-WF increasing the number of processors did not have a positive effect.
2. **Number of tasks** in Figure 5.11(c)-(e): Increasing the number of tasks in the systems for a constant number of processors increased the acceptance ratio as well, since the average task utilization is decreased and smaller tasks can typically be allocated easier. The gap between the acceptance ratios for systems with full and partial compensation became larger when increasing the number of tasks.
3. **WCET-Factor** in Figure 5.11(c),(f),(g): As expected, increasing the WCET-factor led to a decrease in schedulability.

Summarizing the evaluation results, it is evident that a rate-monotonic best-fit approach leads to a good acceptance ratio under partitioned scheduling when designing *Multiprocessor Systems with Dynamic Real-Time Guarantees*. If semi-partitioned scheduling techniques are considered, namely, an approach with highest priority task splitting, the acceptance ratio can be further increased. The analysis of the presented compensation techniques showed a decreased acceptance ratio, compared to the best partitioned strategy. Nevertheless, the trade-off between acceptance ratio and reliability seems reasonable, hence a longer interval of abnormal execution behaviour can be compensated using MSDRTGs.

5.4 EFFICIENTLY APPROXIMATING THE WORST-CASE DEADLINE FAILURE PROBABILITY

As detailed before, in many real-time systems, it is tolerable that at least some of the tasks in the system miss their deadline in rare situations. Regardless, these deadline misses must be quantified to ensure the system's safety. We examine the problem of determining the *worst-case deadline failure probability* of a task under uniprocessor static-priority preemptive scheduling for an uncertain execution behaviour, i.e., when each task has distinct execution modes and a related known probability distribution. The current state-of-the-art methods are either fast but imprecise analytical bounds, i.e., [CC17], or job-level convolution-based techniques that are not applicable for task sets with a reasonable size due to their runtime complexity, e.g., [MC13; DGK+02; TBE+15]. We provide a task-level convolution-based approach which exploits multinomial distributions and, compared to the traditional job-level convolution-based approaches, allows calculating the worst-case deadline failure probability with better analysis runtime, but without reducing the precision. Our approach is enhanced by a state pruning technique that significantly improves the runtime as well as the scalability without any precision loss. In addition, we propose merging equivalence classes, thus

worst-case deadline failure probability

further reducing the runtime of our analysis while the introduced precision loss can be bounded in advance. The evaluation shows that our approach is applicable for significantly larger task sets than the previously known convolution-based approaches, i.e., it can handle task sets with up to 100 tasks. The results presented in this section appeared in *Efficiently Approximating the Probability of Deadline Misses in Real-Time Systems* in ECRTS 2018 *[BPC+18]*.

5.4.1 MOTIVATION, PROBLEM DEFINITION, AND JOB-LEVEL CONVOLUTION

This section motivates the importance of the considered problem, i.e., the calculation of the worst-case deadline failure probability, and formally defines it. Afterwards, the state-of-the-art technique for convolution-based calculation is introduced, namely the traditional[3] convolution-based approach by Maxim and Cucu-Grosjean [MC13]. We finish this section by explaining the drawback of current job-level convolution-based techniques.

MOTIVATION AND PROBLEM DEFINITION

One important assumption for real-time systems is that a deadline miss, i.e., a job that does not finish its execution before its deadline, will be disastrous and thus the WCET of each task is always considered during the analysis. Nevertheless, if a job has multiple distinct execution schemes, the WCETs of those schemes may differ significantly. Examples are software-based fault-recovery techniques which rely on (at least partially) re-executing the faulty task instance, mixed-criticality systems, and a reduced CPU frequency to prevent overheating. In all these cases, it is reasonable to assume that schemes with smaller WCET are the common case while larger WCETs happen rarely.

We use the example of software-based fault-recovery in the following discussion. When such techniques are applied, the probability that a fault occurs and thus has to be corrected is very low, since otherwise hardware-based fault-recovery techniques would be applied. If re-execution may happen multiple times, the resulting execution schemes have an increased related WCET while the probability decreases drastically. Therefore, solely considering the execution scheme with the largest WCET at design time would lead to largely over-designing the system resources. Furthermore, many real-time systems can tolerate a small number of deadline misses at runtime as long as these deadline misses do not happen too frequently. This holds true especially if some of the tasks in the system only have weakly-hard or soft real-time constraints. Hence, being able to predict the probability of a deadline miss is an important property when designing real-time systems. We focus on the *worst-case deadline failure probability*[4] for a single task here, which is defined as follows:

worst-case deadline failure probability

3 We use the term *traditional convolution-based approach* when referring to the approach by Maxim and Cucu-Grosjean [MC13] to avoid confusion, since our novel approach based on multinomial distributions also uses convolution.

4 Note that the term *deadline miss probability* was used in the publication in ECRTS 2018 *[BPC+18]*. It is changed here to avoid ambiguity and to match the terminology in the survey in [DC19a].

Definition 5.5 (Worst-Case Deadline Failure Probability)**.** *Let $R_{k,j}$ be the response time of the j^{th} job of τ_k. The worst-case deadline failure probability (WCDFP) of task τ_k, denoted by Φ_k, is an upper bound on the probability that a job of τ_k is not finished before its (relative) deadline D_k, i.e.,*

$$\Phi_k = \max_j \left\{ \mathbb{P}(R_{k,j} > D_k) \right\}, \quad j = 1, 2, 3, ... \tag{5.7}$$

It was shown in [MC13] that the WCDFP of a job of a constrained- or implicit-deadline task is maximized when τ_k is released at its critical instant (see Section 2.4.1). Hence, *time-demand analysis* (TDA) [LSD89], see Eq. (2.3), can be applied to determine the worst-case response time of a task when the execution time of each job is known. This implicitly assumes that no previous job has an overrun that interferes with the analyzed job, i.e., we are searching for the probability that the first job of τ_k misses its deadline after a longer interval where all deadlines where met. Note that this is not identical to the *deadline miss rate* of a task and that the deadline miss rate may be even higher than this probability, as detailed by Chen et al. [CBC18a]. However, the approach in [CBC18a] utilizes approaches to approximate the worst-case deadline failure probability as a subroutine when calculating the deadline miss rate.

deadline miss rate

When probabilistic WCETs are considered, the WCET obtains a value in $(C_{i,1}, ..., C_{i,h})$ with a certain probability $\mathbb{P}_i(j)$ for each job of each task τ_i. Therefore, TDA for a given t is not looking for a binary decision anymore. Instead, we are interested in the probability that the accumulated workload S_t over an interval of length t is at most t. The probability that τ_k cannot finish in this interval is denoted accordingly with $\mathbb{P}(S_t > t)$. The situation where S_t is larger than t is called an *overload* for an interval of length t and hence $\mathbb{P}(S_t > t)$ is the *overload probability* at time t. According to the notation introduced in Section 2.6, $\rho_{i,t} = \lceil t/T_i \rceil$ for each task τ_i in $hp(\tau_k)$ and $\rho_{k,t} = 1$, i.e., only the first job of τ_k is considered here. Since TDA only needs to hold for one t with $0 < t \leq D_k$ to ensure that τ_k is schedulable, the probability that the test fails is upper bounded by the minimum probability among all time points at which the test could fail. As a result, the worst-case deadline failure probability Φ_k can be upper bounded by

overload probability

$$\Phi_k = \min_{0<t\leq D_k} \mathbb{P}(S_t > t) \tag{5.8}$$

The number of points considered in Eq. (2.3) and therefore in Eq. (5.8) can be reduced by only considering the *points of interest*, i.e., D_k and the release times of higher priority tasks. Nevertheless, this may still lead to a pseudo-polynomial number of points. However, since the minimum value among all these points is taken, an upper bound is still obtained when only a subset of those points is considered. Two general approaches to calculate Φ_k are known from the literature. We summarize the traditional convolution-based approach by Maxim and Cucu-Grosjean in [MC13] in the following subsection since the approach we propose is convolution-based as well. An analytical approach using the moment generating function and Chernoff bounds was introduced by Chen and Chen [CC17].

It may be easier to determine $\mathbb{P}(S_t \geq t)$ instead of $\mathbb{P}(S_t > t)$, especially when analytical bounds are used. As $\mathbb{P}(S_t \geq t) \geq \mathbb{P}(S_t > t)$ by definition, these values can be used directly when looking for an upper bound of $\mathbb{P}(S_t > t)$.

TRADITIONAL CONVOLUTION-BASED APPROACH

We use a notation similar to the one used by Maxim and Cucu-Grosjean in [MC13]. Each task is defined by a vector of the possible WCETs and the related probabilities, e.g., $\begin{pmatrix} 3 & 5 \\ 0.9 & 0.1 \end{pmatrix}$ where 3 and 5 are the WCETs and 0.9 and 0.1 are the related probabilities. The convolution of two such vectors is denoted by $\otimes$ and results in a new vector. To calculate this new vector, each element of the first vector is combined with each element of the second vector by 1) multiplying the related probabilities, and 2) summing up the related WCETs.

Example 5.2 (Convolution). $\begin{pmatrix} 3 & 5 \\ 0.9 & 0.1 \end{pmatrix} \otimes \begin{pmatrix} 5 & 6 \\ 0.8 & 0.2 \end{pmatrix} = \begin{pmatrix} 8 & 9 & 10 & 11 \\ 0.72 & 0.18 & 0.08 & 0.02 \end{pmatrix}$

Note that the summation of the probabilities is 1 for each of these vectors. The general idea of the traditional convolution-based approach [MC13] is the direct enumeration of the WCET state space[5] and the related probabilities. To this end, it considers the jobs in non-decreasing order of their arrival times. For each arriving job, the current system state which is represented by a vector of possible states, i.e., possible total WCETs and related probability, is convolved with the arriving job. This results in a new vector of possible states that represents the state space after the arrival of the job. Once all jobs that are released before a certain point in time are convolved, the probability that the workload is smaller than the next arrival time of a job is calculated. Thereafter, the jobs arriving at that time are convolved with the current states, and the probability for the next arrival time is checked, etc. This process is repeated until $t = D_k$ is reached. A small example with two tasks is detailed in Figure 5.12. The first jobs of τ_1 and τ_2 are both convolved with the initial state and the four resulting states are each convolved with the second release of τ_1 at $t = 8$. Obviously, when all jobs that are released up to any point in time are convolved, states that result in the same execution time can be combined by adding up the related probability, e.g., the states with WCET 13 and WCET 14, respectively, in Figure 5.12.

Applying the traditional convolution-based approach calculates the exact probabilities for each t in the interval of interest in one iteration. However, it can easily lead to a state explosion where the number of states is exponential in the number of jobs. To tackle this problem, Maxim and Cucu-Grosjean use a re-sampling approach, which was first proposed by Maxim et al. in [MHS+12], to reduce the number of states to a given threshold and thus reduce the runtime while only slightly decreasing the precision as shown in [MC13]. Regardless, the main problem of the traditional convolution based approach remains the state explosion.

5.4.2 THE MULTINOMIAL-BASED APPROACH

In the traditional convolution-based approach [MC13], the underlying random variable represents the execution mode of each single job. We now take a closer

5 Please note that the approach in [MC13] does not only consider probabilistic WCETs but also probabilistic periods. Since we only consider probabilistic WCETs here, we summarize accordingly.

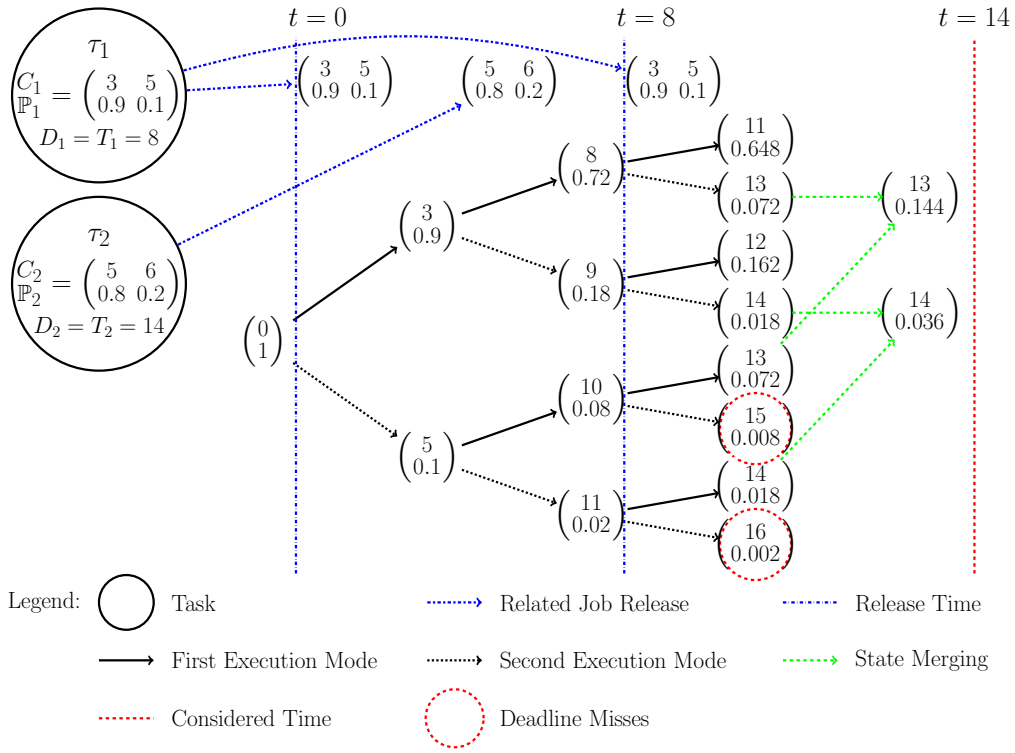


Figure 5.12: The traditional convolution-based approach. *Adapted from [BPC+18].* Assume $\mathbb{P}(S_{14} > 14)$ is determined for two tasks τ_1 and τ_2. The initial state is convolved with the two jobs released at $t = 0$ and the second job of τ_1 released at $t = 8$. Afterwards, $\mathbb{P}(S_{14})$ is determined by summing up the probabilities of the states with a workload larger than 14 (red dotted circle), leading to $\mathbb{P}(S_{14} > 14) = 0.01$. States with the same execution time can be merged (dashed green arrows). This typically happens when the related paths are permutations of each other, e.g., both paths with workload 13 have one execution of $C_{1,1}$ and one of $C_{1,2}$.

look on the related state space and show that the complexity of this approach depends on the specific definition of these random variables. Afterwards, we explain how this state space can be transformed into an equivalent space that describes the states on a task-based level by proving the invariance when considering equivalence classes for each task. As a result, we introduce our approach that is based on the multinomial distribution. We also briefly discuss the complexity of our approach compared to the traditional convolution-based approach.

THE STATE SPACE OF THE TRADITIONAL CONVOLUTION-BASED APPROACH

In the traditional convolution-based approach [MC13], $\boldsymbol{X}(t)$ is the set of the random variables representing the individual jobs released in the interval $[0, t)$ in the order of their arrival times. The notation of $\boldsymbol{X}(t)$ instead of $\boldsymbol{X}$ is necessary due to the fact that the underlying state space and thus the underlying set of random variables is dependent on the time t that is considered. Let $J(t)$ be the number of jobs released in $[0, t)$ under the critical instance of τ_k. Hence, $\boldsymbol{X}(t)$ is a set of $J(t)$

independent random variables, i.e., $\boldsymbol{X}(t)$ is the Cartesian product over those $J(t)$ variables. When aiming to simplify this computation, it is necessary to explicitly consider the random variables $\boldsymbol{X}(t)$ as well as the dependence between $\boldsymbol{X}(t)$ and the quantities S_t and C_i. For the simplicity of notation only, we assume that all jobs have a common set of h execution modes $\mathcal{M}$, i.e., $|\mathcal{M}| = h$.[6] Thus, the state space of the random variable $\boldsymbol{X}(t)$ is $\mathcal{X}(t) = \mathcal{M}^{J(t)}$. A concrete assignment of these variables is denoted $\boldsymbol{x} \in \mathcal{X}(t)$, and the portion of $\boldsymbol{x}$ that corresponds to the jobs of task τ_i is denoted $\boldsymbol{x}_i$. Each task τ_i releases $\rho_{i,t} = \lceil t/T_i \rceil$ jobs, and thus $J(t) = \sum_{\tau_i \in hep(\tau_k)} \lceil t/T_i \rceil$. Therefore, $\lceil t/T_i \rceil$ of the $J(t)$ random variables in $\boldsymbol{X}(t)$ are related to the task τ_i. Since the execution time of the j^{th} job of task τ_i depends on the related random variable $\boldsymbol{X}_{i,j}(t)$, we denote it $C_i(\boldsymbol{X}_{i,j}(t))$. Linking the total workload S_t to the random variables, from Eq. (2.3) we get:

$$S_t = S_t(\boldsymbol{X}(t)) = C_k(\boldsymbol{X}_{k,1}(t)) + \sum_{\tau_i \in hp(\tau_k)} \sum_{j=1}^{\rho_{i,t}} C_i(\boldsymbol{X}_{i,j}(t)) \tag{5.9}$$

Based on Eq. (5.9), we denote the exact expression for the probability of an overload at time t as

$$\mathbb{P}(S_t(\boldsymbol{X}(t)) > t) = \sum_{\boldsymbol{x} \in \mathcal{X}(t)} \mathbb{P}(\boldsymbol{X}(t) = \boldsymbol{x}) \mathbb{1}_{\{S_t(\boldsymbol{x}) > t\}} \tag{5.10}$$

where $\mathbb{1}_{\{\text{expression}\}}$ is the *indicator function* which evaluates to 1 if the expression is true, and to 0 otherwise. Since the execution modes of the jobs are assumed to be independent, the joint probability mass $\mathbb{P}(\boldsymbol{X}(t))$ factorizes over the jobs. The probability of each execution mode per job is fully determined by its corresponding task:

$$\mathbb{P}(\boldsymbol{X}(t) = \boldsymbol{x}) = \prod_{\tau_i \in hp(\tau_k)} \prod_{j=1}^{\rho_{i,t}} \mathbb{P}_i(\boldsymbol{x}_{i,j}(t)) \tag{5.11}$$

Each factor $\mathbb{P}_i(x)$ is the probability mass of any job of task τ_i, being in some state $x \in \mathcal{M}$. Note that Eq. (5.10) is exactly the quantity computed by the traditional convolution-based approach [MC13]. Hence, it stems from the state space $\mathcal{X}(t) = \mathcal{M}^{J(t)}$ that is exponential in the total number of jobs. Nevertheless, we leverage the independence of job modes to compute $\mathbb{P}(S_t(\boldsymbol{X}(t))) \geq t)$ over a different state space, which is the key insight of our method.

INVARIANCE AND EQUIVALENCE CLASSES

In Eq. (5.11), for any fixed task τ_i, the expression $\prod_{j=1}^{\rho_{i,t}} \mathbb{P}_i(\boldsymbol{x}_{i,j})$ is determined by the number of jobs for each state in $\mathcal{M}$. As an example, consider an arbitrary task τ_i with two distinct execution states, i.e., $\mathcal{M} = \{C_{i,1}, C_{i,2}\}$, and suppose that $\boldsymbol{x}_i = (C_{i,1}, C_{i,2}, C_{i,1}, C_{i,2})$, $\boldsymbol{x}'_i = (C_{i,1}, C_{i,1}, C_{i,2}, C_{i,2})$, and $\boldsymbol{x}''_i = (C_{i,2}, C_{i,1}, C_{i,1}, C_{i,2})$. The resulting probability is identical in all three cases, i.e., $\mathbb{P}_i(\boldsymbol{x}_i) = \mathbb{P}_i(\boldsymbol{x}'_i) = \mathbb{P}_i(\boldsymbol{x}''_i)$. We formalize this property subsequently:

6 If a task has less than h (or even only one) execution modes, dummy modes with probability 0 can ensure this condition. Alternatively, $\mathcal{M}_i$ and h_i can be defined based on the actual number of execution modes of τ_i.

Lemma 5.14 (Probability Permutation Invariance). *Let τ_i be a task with a set of distinct execution modes $\mathcal{M}$, let $\rho_{i,t}$ be the number of jobs of τ_i released up to time t, and let $\boldsymbol{x}_i \in \mathcal{M}^{\rho_{i,t}}$ be the random vector that represents the execution mode of all jobs which belong to task τ_i. The probability mass $\mathbb{P}_i$ is permutation invariant with respect to $\boldsymbol{x}_i$, i.e.,*

$$\forall\, \boldsymbol{x}_i \in \mathcal{M}^{\rho_{i,t}} : \forall \sigma \in \mathbb{S}_{\rho_{i,t}} : \mathbb{P}_i(\boldsymbol{x}_i) = \mathbb{P}_i(\sigma(\boldsymbol{x}_i)) \tag{5.12}$$

where $\mathbb{S}_n$ contains all permutations of n objects.

Proof. The lemma follows directly from the independence of job-wise execution modes, hence $\mathbb{P}_i(\boldsymbol{x}_i) = \prod_{j=1}^{\rho_{i,t}} \mathbb{P}_i(\boldsymbol{x}_{i,j})$, and from the commutativity of the multiplication. □

Until now, we considered a single task τ_i, but Lemma 5.14 holds *for all* tasks simultaneously. Recall that the random modes of all tasks are represented by $\boldsymbol{X}(t)$. Let $\boldsymbol{X}_i(t)$ represent the random modes of the jobs of task τ_i, i.e., $\boldsymbol{X}_i(t)$ is the subset of random variables in $\boldsymbol{X}(t)$ that relate to the random modes of τ_i. Applying the permutation invariance to each $\boldsymbol{X}_i(t)$, we derive a partition on $\mathcal{X}(t)$ into equivalence classes:

Definition 5.6 (Execution Mode Equivalence Classes). *For any $\boldsymbol{x} \in \mathcal{X}(t)$, its equivalence class $[\![\boldsymbol{x}]\!]$ with respect to permutation invariance is given by* *equivalence class*

$$[\![\boldsymbol{x}]\!] = \{\boldsymbol{x}' \in \mathcal{X}(t) \mid \forall \tau_i \in hep(\tau_k) : \exists \sigma \in \mathbb{S}_{\rho_{i,t}} : \boldsymbol{x}_i = \sigma(\boldsymbol{x}'_i)\} \tag{5.13}$$

Based on this definition, the statement $\forall \boldsymbol{x}' \in [\![\boldsymbol{x}]\!] : \mathbb{P}(\boldsymbol{x}) = \mathbb{P}(\boldsymbol{x}')$ is a straightforward corollary of Lemma 5.14. The equivalence relation in Definition 5.6 is established by an equivalent occurrence of execution modes for each task. Therefore, each equivalence class has a canonical representative, which is given by a tuple $\boldsymbol{\ell} \in \bigotimes_{\tau_i \in hep(\tau_k)} \{1, 2, \ldots, \rho_{i,t}\}^{|\mathcal{M}|}$, that for each task contains the number of jobs for all execution modes. For convenience we let $[\![\boldsymbol{\ell}]\!]$ address the set of all $\boldsymbol{x}$ in the same equivalence class and rephrase Eq. (5.10) accordingly.

Lemma 5.15 (Class-based Overload Probability). *For any set of execution modes $\mathcal{M}$, let $\mathcal{L}(t) = \bigotimes_{\tau_i \in hep(\tau_k)} \{0, 1, 2, \ldots, \rho_{i,t}\}^{|\mathcal{M}|}$. Then,*

$$\mathbb{P}(S_t(\boldsymbol{X}(t)) \geq t) = \sum_{\boldsymbol{\ell} \in \mathcal{L}(t)} \prod_{\tau_i \in hep(\tau_k)} \frac{\rho_{i,t}! \prod_{j=1}^{|\mathcal{M}|} \mathbb{P}_i(j)^{\boldsymbol{\ell}_{i,j}}}{\prod_{x \in \mathcal{M}} \boldsymbol{\ell}_{i,x}!} \mathbb{1}_{\{S_t([\![\boldsymbol{\ell}]\!]) \geq t\}} \tag{5.14}$$

where $\boldsymbol{\ell}_{i,j}$ denotes the number of jobs of task τ_i which are in the j-th execution mode, and $S_t([\![\boldsymbol{\ell}]\!])$ denotes the execution time for some arbitrary $\boldsymbol{x} \in [\![\boldsymbol{\ell}]\!]$.

Proof. For all members of the class $[\![\boldsymbol{x}]\!]$, each task has the same number of jobs in the same state. Hence, iterating over the set $\mathcal{L}(t) = \bigotimes_{\tau_i \in hep(\tau_k)} \{0, 1, 2, \ldots, \rho_{i,t}\}^{|\mathcal{M}|}$ corresponds to iterating over all such count vectors, which is in turn the same as iterating over all equivalence classes $[\![\boldsymbol{x}]\!]$. Each class $[\![\boldsymbol{\ell}]\!]$ contains all state permutations for all jobs of each task. For each task τ_i, this is equivalent to the well-known combinatorial problem of determining the number of ways how $\rho_{i,t}$

objects can be placed into $|\mathcal{M}|$ bins, given by the corresponding multinomial coefficient. Combining those for all tasks, we get

$$|[\![\boldsymbol{\ell}]\!]| = \prod_{\tau_i \in hep(\tau_k)} \binom{\rho_{i,t}}{\boldsymbol{\ell}_{i,1}\ \boldsymbol{\ell}_{i,2}\ \dots\ \boldsymbol{\ell}_{i,|\mathcal{M}|}} = \prod_{\tau_i \in hep(\tau_k)} \frac{\rho_{i,t}!}{\prod_{x \in \mathcal{M}} \boldsymbol{\ell}_{i,x}!} \tag{5.15}$$

Combining these facts results in

$$\sum_{\boldsymbol{x} \in \mathcal{X}(t)} \mathbb{P}(\boldsymbol{X}(t) = \boldsymbol{x}) = \sum_{\boldsymbol{\ell} \in \mathcal{L}(t)} |[\![\boldsymbol{\ell}]\!]| \mathbb{P}(\boldsymbol{X}(t) = [\![\boldsymbol{\ell}]\!]) \tag{5.16}$$

Observing that $\mathbb{P}(\boldsymbol{X}(t) = [\![\boldsymbol{\ell}]\!]) = \prod_{j=1}^{|\mathcal{M}|} \mathbb{P}_i(j)^{\boldsymbol{\ell}_{i,j}}$ implies the lemma. □

DETAILING THE MULTINOMIAL-BASED APPROACH

Now, we combine the findings in this section into an algorithm that more efficiently calculates $\mathbb{P}(S_t > t)$, i.e., the probability of an overload for a length t. For simplicity of presentation, we refer to the overload probability *at time t* and the state space *at time t*, implicitly assuming that both the probability and the state space is calculated over the interval $[0, t)$ considering the critical instant of τ_k. The traditional convolution-based approach determines this probability by successively calculating the probability for all other points of interest in the interval $[0, t)$. Nevertheless, the probability for t is evaluated based on the resulting states after all jobs in $[0, t)$ are convolved. Hence, with respect to t, the intermediate states are not considered.

We utilize this insight to calculate the vector representing the possible states at time t more efficiently. Lemma 5.14 shows that the overload probability of a state for a concrete variable assignment $\boldsymbol{x} \in \mathcal{X}(t)$ is identical to the probability of all permutations of $\boldsymbol{x}$, i.e., the related equivalence class. This allows to consider the jobs in $J(t)$ in any order. Furthermore, Lemma 5.15 shows that all assignments that are part of the same equivalence class result in the same value for S_t. Considering one specific task τ_i, those assignments differ regarding the order in which the execution modes happen but not regarding the total number of executions in a given mode. However, if the jobs are convolved in the non-decreasing order of their arrival times, this results in a large number of unnecessary states that are merged later on. For example, in Figure 5.12 the state space can be reduced if the second job of τ_1 is convolved before the job of τ_2, since the resulting merged state space after the convolution of the two jobs of τ_1 only has 3 states that represent the number of executions in each mode. Therefore, to reduce the state space as much as possible, we consider the jobs in an order that is based on the tasks they are related to, i.e., first all $\rho_{1,t}$ jobs of τ_1 are considered, then all $\rho_{2,t}$ jobs of τ_2, etc. However, if the jobs are just reordered and then convolved, this still leads to a large of number states that are merged later on. Regardless, the number of states is already significantly lower than in the traditional convolution-based approach.

Fortunately, if the number of jobs a task releases in the examined interval is known, all possible combinations and the related probabilities can be calculated directly using the multinomial distribution. To be more precise, assume a

given task τ_i as well as a given number of releases $\rho_{i,t}$ in an interval of length t and let $\ell_{i,j}$ be the number of executions in mode $j \in \{1,...,h\}$. We know that $\ell_{i,j} \in \{0,1,...,\rho_{i,t}\}$ and $\sum_{j=1}^{h} \ell_{i,j} = \rho_{i,t}$, resulting in $\binom{\rho_{i,t}+h-1}{h-1}$ possible combinations of $\ell_{i,1},...,\ell_{i,h}$ where $\binom{a}{b} = \frac{a!}{b!(a-b)!}$ is the binomial coefficient. For each combination, the related probability is

$$\frac{\rho_{i,t}!}{\ell_{i,1}!\ell_{i,2}!...\ell_{i,h}!}\mathbb{P}_i(1)^{\ell_{i,1}} \cdot \mathbb{P}_i(2)^{\ell_{i,2}} \cdot ... \cdot \mathbb{P}_i(h)^{\ell_{i,h}} \tag{5.17}$$

where $\frac{\rho_{i,t}!}{\ell_{i,1}!\ell_{i,2}!...\ell_{i,h}!}$ determines the number of possible paths for the related equivalence classes and $\mathbb{P}_i(1)^{\ell_{i,1}} \cdot \mathbb{P}_i(2)^{\ell_{i,2}} \cdot ... \cdot \mathbb{P}_i(h)^{\ell_{i,h}}$ is the probability of one of these paths. The total workload of the $\rho_{i,t}$ jobs of τ_i is calculated for each of these combinations based on the related values of $\ell_{i,1}$ to $\ell_{i,h}$. The $\binom{\rho_{i,t}+h-1}{h-1}$ states represent the equivalence classes of τ_i and the related probabilities. After calculating these representatives for each task, the overload probability can be determined by convolving them and adding up the overload probabilities of the resulting state space. A concrete example where each task has two possible execution modes is given in Figure 5.13. Note that based on Lemma 5.14, the states representing the tasks, and therefore the tasks themselves, can be convolved in any order.

In fact, for the specific time point t, the job-based state space of the traditional convolution-based approach has been transferred into a task-based space state with identical properties regarding the overload probability. To visualize the different approaches, the traditional convolution-based approach constructs a tree based on the jobs (see Figure 5.12), where each level represents the state of the system after the related job is convolved and the number of children on a given level depends on the number of possible modes of the related job. Contrarily, the multinomial-based approach constructs a tree based on the tasks (see Figure 5.13) and the number of children on each level depends on the number of jobs the related task releases. If the nodes on the $J(t)^{th}$ level of the binary tree are merged as show in Figure 5.12, the number of states on that level is identical to the number of states on the k^{th} level of the tree resulting from our approach. While the state space of our reformulation is still large, it enables pruning strategies and other state reduction strategies which are not suitable for the traditional approach. These strategies are explained in Section 5.4.3.

COMPLEXITY DISCUSSION AND COMPARISON

When considering the complexity of the multinomial-based approach for τ_k over an interval $[0,t)$ under the critical instance of τ_k, both the number of tasks that are contributing to the workload in the interval, i.e., $\rho_{i,t}$ for the higher priority tasks, and the total number of jobs in the interval $J(t)$ have to be considered. The number of multinomial coefficients depends on $\rho_{i,t}$ and the number of possible execution states h for each task and can be calculated as $\binom{\rho_{i,t}+h-1}{h-1}$. This is also called the h-simplex of the $\rho_{i,t}^{th}$ component. The convolution of these states over all tasks leads to a total number of states of $\prod_{i=1}^{k}\binom{\rho_{i,t}+h-1}{h-1}$.

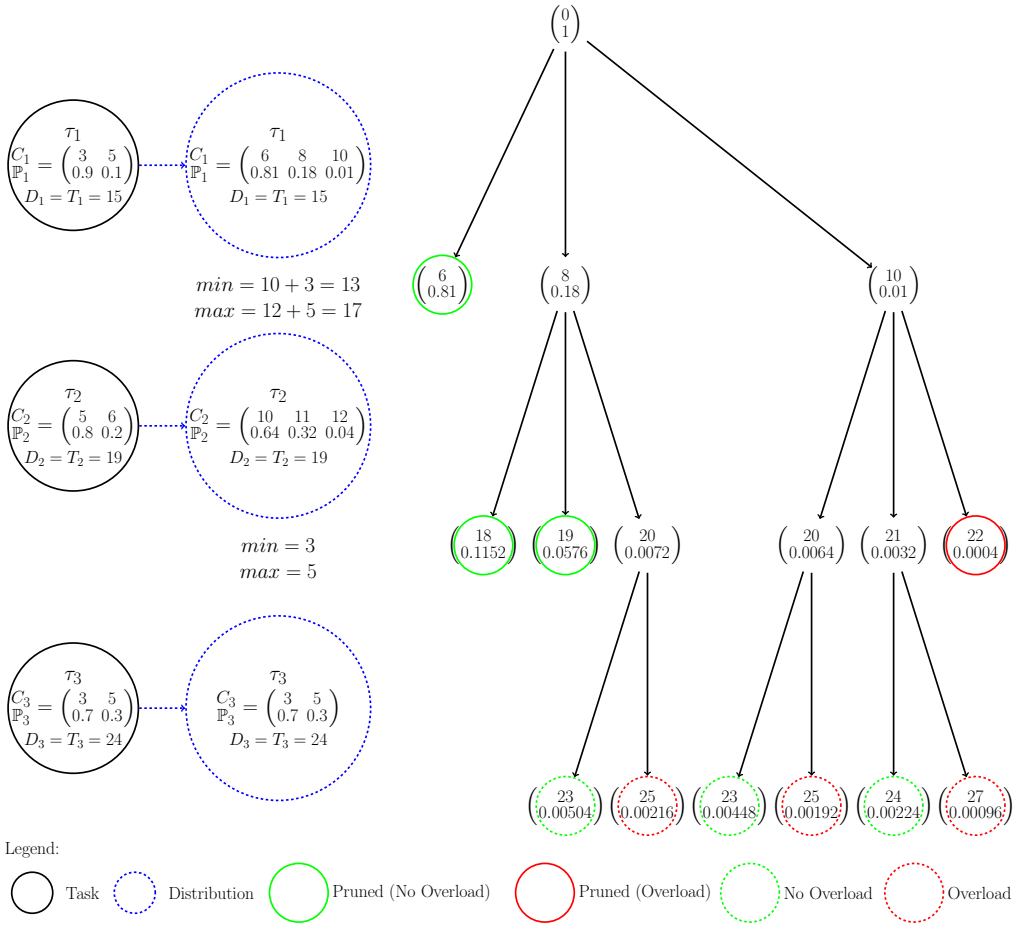


Figure 5.13: The multinomial-based approach. *Adapted from [BPC+18].*
The number of children depends on the number of jobs of the related task. Note that nodes can be ignored in further steps if they never lead to an overload (green solid circles) or if they always lead to an overload (red solid circle). The overload probability at $t = 24$ is calculated by adding up the related probabilities (dashed and solid red) which results in a deadline miss probability of 0.00574.

The traditional convolution-based approach considers each job individually with h possible outcomes and, therefore, leads to $h^{J(t)}$ states, i.e., it is exponential in the number of jobs. Hence, without state merging, it is not feasible for input sets with a reasonable cardinality. However, as an integral part of the process, the traditional convolution-based approach also calculates the deadline miss probability at all possible points of interest in the interval, i.e., at each point in time a job is released. Moreover, states can be merged when they have the same related workload, e.g., states resulting from a permutation of the same number of abnormal executions of a given task. Lemma 5.14 directly implies that when convolution is used in combination with merging states, the final number of states for the convolution-based approach at time t is identical to the number of states created by the multinomial-distribution-based approach. However, while our approach creates only necessary states, the traditional convolution-based approach not only creates unnecessary states but also requires additional overhead for state merging after each step. Therefore, when considering a single point in

time, our approach is significantly faster than the traditional convolution-based approach with task merging.

Nevertheless, since our approach needs to consider all points of interest individually, if the number of such points increases due to the number of tasks, the traditional convolution-based could have advantages. We were, however, not able to observe this behaviour in our evaluation since both our multinomial-based approach as well as the traditional convolution-based approach with state merging were only rarely able to provide results for task sets with a cardinality of 10 or larger. Hence, for our approach, runtime optimizations are provided in the next section. Note that the number of tasks that is feasible depends on the actual setting and that the period range is the most important parameter since it relates to the number of jobs, an effect that will be evaluated in Section 5.4.4.

5.4.3 RUNTIME IMPROVEMENT

We introduce two strategies to improve the runtime efficiency of the multinomial-based approach. The first one prunes the state space, i.e., it discards states directly, if the impact on the overload probability can be determined without considering the remaining tasks. This reduces the runtime without sacrificing any precision. The second strategy combines execution mode equivalence classes with very low probability when creating the task representations to reduce the size of the state space beforehand. This leads to an increase in the approximated overload probabilities, but the resulting error can be bounded for each task under consideration and therefore also for the total error of the derived overload probability. Note that both techniques can be combined.

PRUNING THE STATE SPACE

Our multinomial-based approach calculates the probabilities for each interval individually, a property already used when transferring the state space from job-based to task-based. For convenience, assume that the representatives of the tasks are convolved according to the task index. Recall that the state space can be seen as a rooted tree where each node on the j^{th} row represents a possible state after the convolution of the first j tasks, and that we are only interested in the nodes on the k^{th} (and last) layer, i.e., the states after all task representations are convolved. Such a tree is displayed in Figure 5.13. The general concept of pruning is to remove a state R if the resulting subtree, i.e., the subtree with root R, has no further impact on the evaluation on the k^{th} layer, which means that either *all* states on the k^{th} layer in the subtree with root R evaluate to an overload, or for *all* states on the k^{th} layer in the subtree with root R, the resulting workload is less than or equal to the interval length. In the first case, the state is discarded and the related probability is added to the overload probability for time t. In the second case, the state is discarded immediately. This is done by checking the boundary conditions. To this end, we list the minimum and maximum execution time each task can contribute to the total workload up to time t. On the i^{th} layer, the minimum and maximum workload that can be contributed by the remaining

tasks, denoted as $C_{\min_i}$ and $C_{\max_i}$, is the sum of the minimum and maximum values of the remaining tasks. Let $\mathbb{P}(\text{discard})$ be a variable accounting for the overload probability of discarded states, initialized with 0. For each state Q created by the convolution of τ_i with the previous state space, let $C(Q)$ be the related total workload. We evaluate the following two conditions:

1. $C(Q) + C_{\max_i} \leq t$: All paths in the subtree rooted at Q leads to states where no overload is detected at time t, since the branch related to the maximum cumulative workload in this subtree does not lead to an overload. Therefore, Q can directly be discarded. In the example in Figure 5.13, those states are marked with a solid green circle.
2. $C(Q) + C_{\min_i} > t$: All paths in the subtree rooted at Q result in an overload at time t, since the branch related to the minimum cumulative workload in this subtree results in an overload. Hence, Q can be discarded and $\mathbb{P}(\text{discard})$ is increased by the probability of Q. In Figure 5.13, those states are marked with a solid red circle.

Obviously, all created states can only fulfill one of these conditions but not both since $C(Q) + C_{\min_i} \leq C(Q) + C_{\max_i}$. If Q fulfills neither, the state is added to the representation of $\tau_1, ..., \tau_i$. That state pruning does not change the calculated probability follows directly from the observations that the total probability of a subtree is equal to the probability of the root, and from the fact that the total workload of each branch is always smaller than the maximum workload (larger than the minimum workload, respectively). Hence, a proof is omitted. Note that the order in which the tasks are considered has no impact on the applicability of the pruning technique.

Similar techniques cannot easily be exploited for job-level convolution-based approaches, e.g., Maxim and Cucu-Grosjean [MC13], Diaz et al. [DGK+02], or Tanasa et al. [TBE+15], since one major difference is that it calculates the overload probability of all values successively. To be more precise, it considers the critical instant of τ_k at time 0 and calculates the deadline miss probability for all intervals $[0, t)$ with $0 < t \leq D_k$, and the result at time t depends on the result at time t' if $t' < t$. This can be visualized by a directed rooted tree where each level is created according to a job's arrival time and represents exactly one job, i.e., the height of the tree depends on the number of considered jobs (see Figure 5.12). The nodes on each layer represent the state space after the convolution of the related job. One important property of this approach is that the probability of a deadline miss is calculated on each layer. Hence, pruning a state, i.e., removing a state and the branches resulting from it, can only be done if those branches have no impact on the probability on *all* following layers, i.e, a state R at time t_a can only be pruned if all branches of the subtree with root R will either lead to an overload at t_b for all $t_b \in (t_a, D_k]$, or to no overload at t_b for all $t_b \in (t_a, D_k]$. This cannot be determined by evaluating the overload condition for any single time point $t_b \in (t_a, D_k]$. For instance, assume that $C(Q) + C_{\min_{t_b}} > t_b$ for a $t_b \in (t_a, D_k]$, where $C_{\min_{t_b}}$ is the minimum workload created by jobs released in the interval $[t_a, t_b)$. Let t_{b-1} and t_{b+1} be, with respect to t_b, the previous and next considered points in time in the convolution based approach. We observe that τ_k may have no overload at t_{b-1}, if the minimum workload of the job released at t_{b-1} is smaller than $t_b - t_{b-1}$. Similar arguments can be made to create a case

# $C_{i,2}$ jobs	0	1	2	3	4	5	6	7	8	9	10
Total C_i	10	11	12	13	14	15	16	17	18	19	20
Probability	0.78	0.2	0.023	0.0016	$7.0 \cdot 10^{-05}$	$2.2 \cdot 10^{-06}$	$4.63 \cdot 10^{-08}$	$6.8 \cdot 10^{-10}$	$6.53 \cdot 10^{-12}$	$3.72 \cdot 10^{-14}$	$9.5 \cdot 10^{-17}$
# $C_{i,2}$ jobs	0	1	2	3	4	5	6 or 7			8, 9, or 10	
Total C_i	10	11	12	13	14	15	17			20	
Probability	0.78	0.2	0.023	0.0016	$7.0 \cdot 10^{-05}$	$2.2 \cdot 10^{-06}$	$4.701 \cdot 10^{-08}$			$6.564711 \cdot 10^{-12}$	

Table 5.1: Multinomial distribution and state merging. *From [BPC+18].*
Distribution of τ_i with $C_{i,1} = 1$, $C_{i,2} = 2$, $\mathbb{P}_i(1) = 0.975$, $\mathbb{P}_i(2) = 0.025$ for 10 releases. The upper part details the distribution before and the lower part after merging equivalence classes.

with no overload at t_{b+1} and for the cases where τ_k has no overload at t_b if $C_{\max_{t_b}}$ is considered.

UNION OF EXECUTION MODE EQUIVALENCE CLASSES

The general concept of this runtime improvement technique is to reduce the state space when creating the representation for the individual tasks by unifying equivalence classes with low probability. In contrast to the pruning technique, this obviously results in a loss of precision when approximating the worst-case deadline failure probability for a given point in time. However, if done carefully, the precision loss can be upper bounded by a constant. We introduce the concept based on the example in Table 5.1, detailing the release of 10 jobs in the interval of interest for a task τ_i with two execution modes that have a WCET of $C_{i,1} = 1$ and $C_{i,2} = 2$, with related probabilities $\mathbb{P}_i(1) = 0.975$ and $\mathbb{P}_i(2) = 0.025$. In the upper half, the original equivalence classes are displayed, i.e., one for each possible number of jobs (0 to 10) in mode 2, together with their total WCET and their (rounded) related probability. After the introduction of the concept, we will explain how the approach can be generalized.

The probability decreases rapidly when the number of jobs that are executed in the second mode increases. Such distributions are common when considering probabilistic execution times for real-time systems. The reason is that if the execution mode with larger WCET has a comparatively high probability, classical non-probabilistic worst-case response time analysis considering the larger WCET must be utilized to ensure timeliness for relatively common cases. Since the probability of the equivalence classes decreases, the impact of those classes on the overload probability over the given interval decreases as well. The number of states that are created in our approach, and therefore its runtime, can be reduced by unifying some of these highly unlikely equivalence classes. To guarantee a safe approximation, i.e., the resulting overload probability is only increased, we define the merge of a set of equivalence class as follows:

Definition 5.7 (Union of Task Equivalence Classes)**.** *For a given interval of interest* $[0, t)$*, let* $\mathcal{C} = \{[\![\boldsymbol{x}_i]\!], [\![\boldsymbol{x}_i']\!], [\![\boldsymbol{x}_i'']\!], \ldots\}$ *be a set of* $|\mathcal{C}| = q$ *equivalence classes of task* τ_i*. For each class* $[\![\boldsymbol{x}_i]\!] \in \mathcal{C}$*, let* $\mathbb{P}_i([\![\boldsymbol{x}_i]\!])$ *and* $C_i([\![\boldsymbol{x}_i]\!])$ *denote its probability and the related total worst-case execution time, respectively. Furthermore, let* $[\![\boldsymbol{x}_i^{\max}]\!] \in \mathcal{C}$ *be the equivalence class with the highest total WCET, i.e.,* $[\![\boldsymbol{x}_i^{\max}]\!] = \arg\max_{[\![\boldsymbol{x}_i]\!] \in \mathcal{C}} C_i([\![\boldsymbol{x}_i]\!])$.

When we union all classes in $\mathcal{C} = \{[\![\boldsymbol{x}_1]\!], ..., [\![\boldsymbol{x}_q]\!]\}$*, they are replaced by one new class* $[\![\boldsymbol{x}_i^{\mathcal{C}}]\!] = \bigcup_{[\![\boldsymbol{x}_i]\!] \in \mathcal{C}} [\![\boldsymbol{x}_i]\!]$ *that has the following characteristics:*

1. $C_i([\![\boldsymbol{x}_i^{\mathcal{C}}]\!]) = C_i([\![\boldsymbol{x}_i^{\max}]\!])$
2. $\mathbb{P}_i([\![\boldsymbol{x}_i^{\mathcal{C}}]\!]) = \sum_{[\![\boldsymbol{x}_i]\!] \in \mathcal{C}} \mathbb{P}_i([\![\boldsymbol{x}_i]\!])$

As shown in Table 5.1, when merging the equivalence classes for 6 and 7 executions of mode 2, the probability of the newly created class is the summation of their probabilities and the related WCET is the maximum among those two classes, here the WCET of the class with 7 executions in mode 2. We now show that merging a set of equivalence classes leads to a bounded error of the overload probability.

Lemma 5.16 (Unifying Equivalence Classes Leads to a Bounded Maximum Error)**.** *For task τ_i let $\mathcal{C} = \{[\![\boldsymbol{x}_i']\!], [\![\boldsymbol{x}_i'']\!], \ldots\}$ be a set of $|\mathcal{C}| = q$ equivalence classes for the interval of interest $[0, t)$. If $\mathcal{C}$ is merged into $[\![\boldsymbol{x}_i^{\mathcal{C}}]\!]$ according to Definition 5.7, the probability of overload can only increase and the error is bounded by $(\sum_{[\![\boldsymbol{x}_i]\!] \in \mathcal{C}} |[\![\boldsymbol{x}_i]\!]| \mathbb{P}_i([\![\boldsymbol{x}_i]\!])) - |[\![\boldsymbol{x}_i^{\max}]\!]| \mathbb{P}_i([\![\boldsymbol{x}_i^{\max}]\!])$.*

This follows from Eq. (5.14), Eq. (5.16), and the fact that any $\mathcal{C}$ in which no class $[\![\boldsymbol{x}_i]\!]$ triggers the indicator function $\mathbb{1}_{\{S_t([\![\boldsymbol{x}]\!]) > t\}}$ does not introduce any error. Hence, if at least $[\![\boldsymbol{x}_i^{\max}]\!]$ triggers $\mathbb{1}_{\{S_t([\![\boldsymbol{x}]\!]) > t\}}$ the maximum probability increase happens if all other classes did not trigger $\mathbb{1}_{\{S_t([\![\boldsymbol{x}]\!]) > t\}}$ before the unification, but do afterwards. Since the process can be repeated for all tasks, we conclude:

Theorem 5.17 (Bounded Overall Increase of the Overload Probability)**.** *If equivalence classes of tasks with respect to the interval $[0, t)$ are merged, the total increase of the overload probability for this interval is increased by the summation of the individual overload probability increase of the tasks.*

We are now able to calculate the overloaded probability over $[0, t)$ with a bounded total error while reducing the considered states. Assume a value b for the allowed maximum error and a set of n tasks to be given. The maximum error is bounded by b if for each task the error is bounded by b/n. This can be achieved by ordering the related states in decreasing order of probability, traversing them in this order while summing up the probabilities of each state, and keeping all states until the summation is larger than $1 - b/n$. Afterwards, the remaining states are unified into one according to Definition 5.7.

So far we considered a setting similar to Table 5.1, where the workload increases as the probability decreases. However, this is not necessarily the case, for example, when a task has two execution modes with an equal probability. Nevertheless, the approach based on Theorem 5.17 can still be exploited directly, since the union of equivalence classes is agnostic to the workloads and related probabilities as long as the total probability of the combined equivalence classes is less than b/n.

5.4.4 EVALUATION

The main focus of the evaluation was to determine if the proposed multinomial-based approach can provide good results with a reasonable analysis runtime, especially considering the scalability with respect to the number of tasks for

reasonable settings. For a given utilization U_{sum} and a number of tasks, we generated random implicit-deadline task sets with one execution mode according to the UUniFast method [BB05]. As suggested by Emberson et al. [ESD10], the periods of those tasks were generated according to a log-uniform distribution with two orders of magnitude, i.e., $10ms - 1000ms$. We only considered tasks with two distinct execution modes in the evaluation, called normal and abnormal execution mode, and hence $\mathcal{M} = \{N, A\}$. The normal execution mode is considered to have a (significantly) higher probability. The WCET in the normal mode was set according to the utilization, i.e., $C_{i,N} = U_i \cdot T_i$, and the WCET in abnormal mode was calculated as $C_{i,A} = f \cdot C_{i,N}$ for all tasks in the set.

We used a fixed setting, defined by U_{sum}, f, and $\mathbb{P}_i(A)$, tracking the resulting worst-case deadline failure probability and runtime related parameters. In each setting, the worst-case deadline failure probability for the lowest-priority task under Rate Monotonic scheduling was determined. In our evaluations, we considered the following approaches where the **bold** name indicates how the approach is referred to:

1. **Convolution:** The *traditional convolution-based approach* [MC13].
2. **Conv. Merge:** The *traditional convolution-based approach* [MC13] with state merging.
3. **Multinomial:** Our multinomial-based approach from Sec. 5.4.2.
4. **Pruning:** Our approach combined with the pruning technique.
5. **Unify:** Our approach combined with the pruning technique and reducing the complexity by union of equivalence classes.
6. **Approx:** Approximation of **Pruning** by only considering the deadline of τ_k and the last releases of higher-priority tasks, inspired from the literature, e.g., [CHL15b; BB04; BCH15; CC17].
7. **Chernoff:** The analytical approach using *Chernoff bounds* by Chen and Chen [CC17].
8. **Hoeffding:** The analytical approach from [BPC+18][7] based on the *Hoeffding's inequality*.
9. **Bernstein:** The analytical approach from [BPC+18][7] based on the *Bernstein inequalities*.

To allow runtime comparisons, all approaches were implemented in the same programming language, Python 2.7, and executed on the same machine, a 12 core Intel Xeon *X*5650 with 2.67 GHz and 20 GB RAM. For the analytical bounds, in contrast to the work by Chen and Chen [CC17], all releases of higher-priority tasks were considered since the bounds have a lower runtime than our approach.

We randomly generated tasks sets with a normal-mode utilization of $U_{sum} = 70$, setting $f = 2$ and $\mathbb{P}_i(A) = 0.025$ for all tasks. Hence, $\mathbb{P}_i(N) = 0.975$. **Convolution** usually did not deliver a result for a cardinality of 5 due to an out-of-memory

7 The approaches based on the Hoeffding's inequality and on the Bernstein inequalities were presented in the same work as the task-level convolution-based approach but are not part of this thesis. Hence, we refer to the paper here.

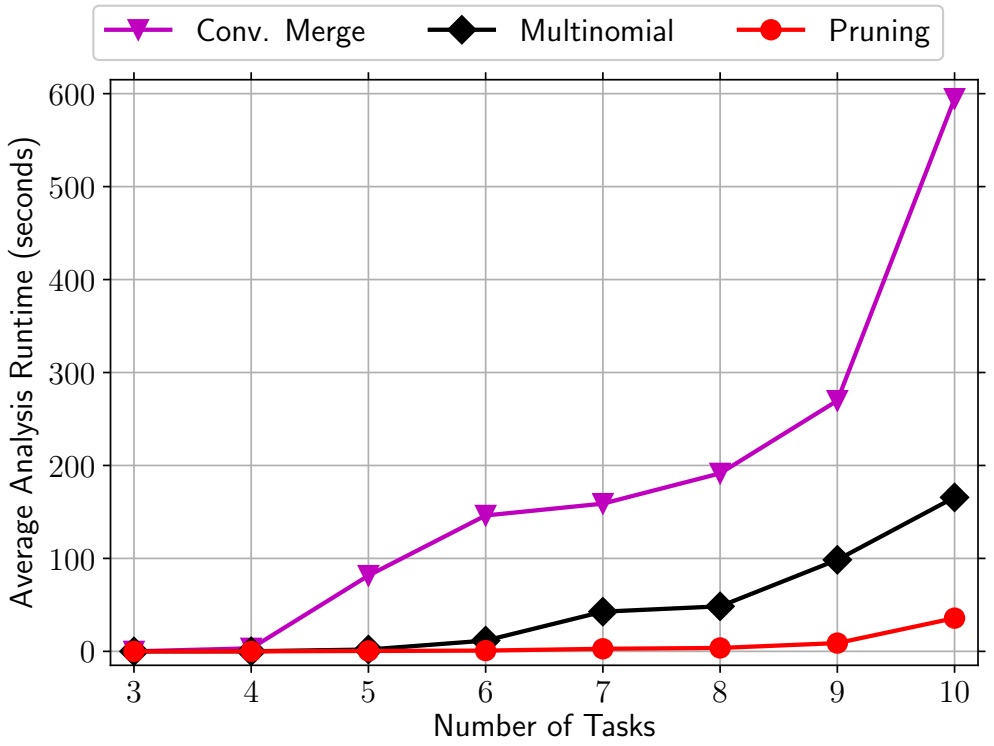


Figure 5.14: Average runtime of **Conv. Merge**, **Multinomial**, and **Pruning** with respect to the task set cardinality.

error. In some cases, results could not even be provided for 3 tasks, since, for instance, 38 jobs already lead to $2^{38} = 274877906944$ states for D_k in **Convolution**. For **Conv. Merge** and **Multinomial**, a setting with 10 tasks was often not feasible already. However, the results for **Conv. Merge**, **Multinomial**, and **Pruning** were always identical (if **Conv. Merge** and **Multinomial** derived results), showing that our pruning technique drastically decreases the runtime of the analysis and increases the scalability without any precision loss. We analyzed the average runtime of these three approaches for small task sets, i.e., 20 sets each from 3 to 10 tasks, which is shown in Figure 5.14. It shows that for 7 or more tasks per set **Multinomial** was approximately 2.5 to 4 times faster than **Conv. Merge**, while **Pruning** was approximately 15 to 200 times faster than **Conv. Merge**. Note that **Conv. Merge** and **Multinomial** only returned results[8] for 18 of the 20 sets for $n = 9$, and 13 of the 20 sets for $n = 10$, hence we did not evaluate **Conv. Merge** and **Multinomial** for larger sets.

Figure 5.15 displays the average runtime of the analysis with respect to the cardinality for the remaining approaches. To analyze the scalability, the cardinality of the task sets ranged from 5 to 35 in steps of 5. For a cardinality from 5 to 20 tasks, we evaluated 20 task sets. For a cardinality from 25 to 35 tasks, due to the high runtime, 5 task sets were analyzed. For 5 and 10 tasks, **Conv. Merge** is displayed for comparison. **Bernstein** and **Hoeffding** are orders of magnitude faster than the other approaches which are similar with respect to the related runtime. The difference between **Approx** and **Pruning** stems from a different

8 The averages runtime for all 3 approaches is only calculated over these sets, although **Pruning** always delivered a result.

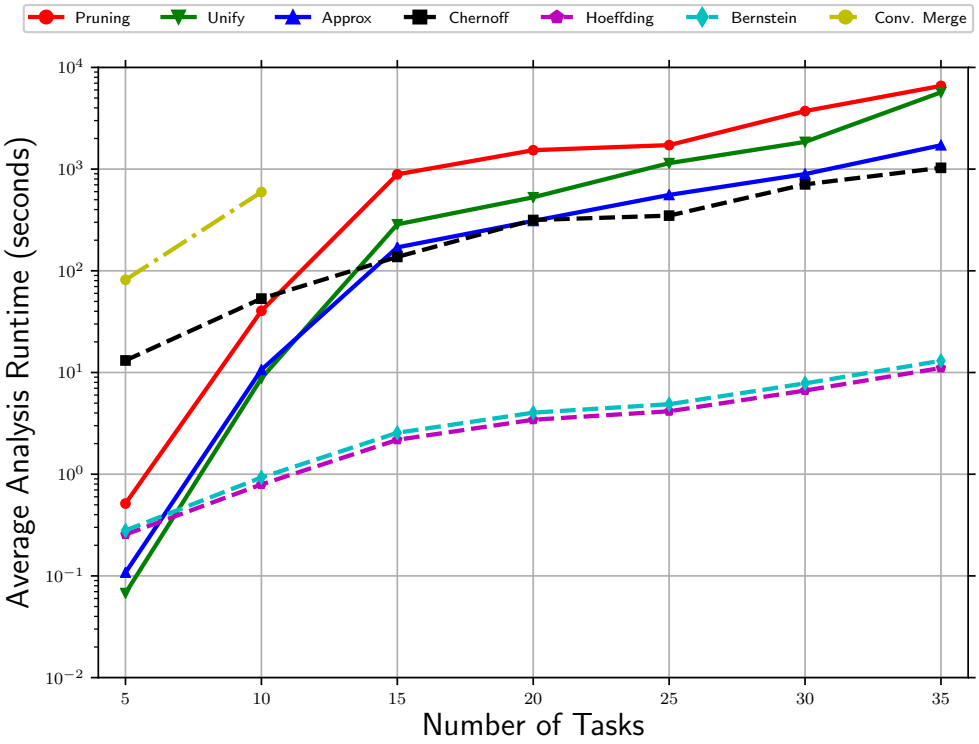


Figure 5.15: Average runtime with respect to task set cardinality. *Adapted from [BPC+18].*

number of tested time points, i.e., for **Approx** this number depends on the number of tasks, and for **Pruning** on the number of jobs, while the runtime for an individual time point does not differ largely.

We also examined the precision of the considered approaches. However, statistical information for derived worst-case deadline failure probabilities is unfortunately not meaningful. For example, for task sets with 15 tasks, the derived worst-case deadline failure probability in our evaluations under **Pruning** ranged from $3.0 \cdot 10^{-39}$ to $6.1 \cdot 10^{-5}$. Therefore, comparing the average values or other statistical parameters does not yield much information. In addition, comparing relative values is problematic if the probability is low. Hence, we show a small sample of 5 task sets with roughly similar probabilities in Figure 5.17(a). These are the first 5 randomly generated task sets with a worst-case deadline failure probability larger than 10^{-6}. This selection is only done to increase the readability of the figure. We observed an in general similar relative behaviour among (nearly) all the evaluated task sets. We see that the error of **Bernstein** and **Hoeffding** is large compared to **Chernoff**, i.e., by several orders of magnitude, while the three approaches based on the multinomial distribution result in similar values, roughly one order of magnitude better than **Chernoff**. We also conducted experiments with different probabilistic distributions which lead to in general identical results.

Figure 5.17(b) compares the worst-case deadline failure probability of the three multinomial-distribution based approaches more closely. **Unify** performs very similar to **Pruning**, i.e., the error is in the magnitude of 10^{-9}. This is significantly smaller than the predefined value for the *allowed error* for **Unify** of 10^{-6} set in the evaluation. The reason is that: 1) execution mode equivalence classes are only merged for some of the tasks, while the maximum error for each task

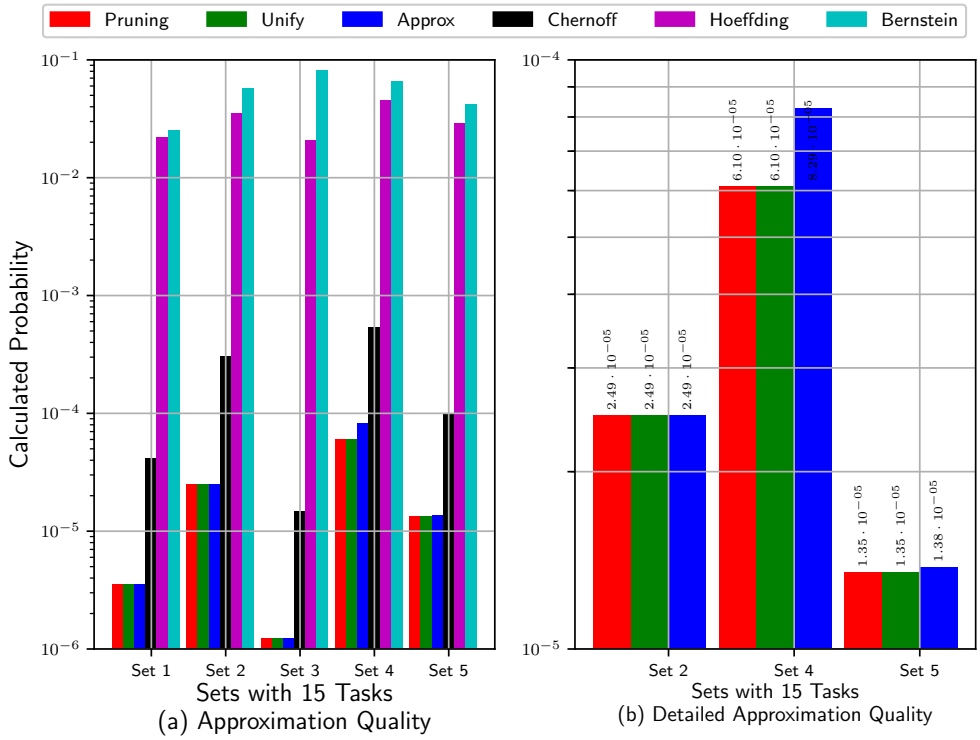


Figure 5.16: (a) Approximation quality for 5 sets with 15 tasks. (b) Detailed approximation quality for the multinomial-based approaches. *Adapted from [BPC+18].*

may already be significantly smaller than 10^{-6}, and 2) the presented worst-case analysis regarding the precision loss is pessimistic. For **Approx** the error for Set 4 and Set 5 is in the magnitude of 10^{-5} and 10^{-7}, respectively, since only a subset of the points of interest is considered. In some rare cases an even larger relative difference could be observed.

Intuitively, it seems especially helpful to use the union of equivalence classes if the periods of tasks differ largely, e.g., in automotive applications where periods often range from 1 to 1000 ms [KZH15], since in this setup tasks with a short period may release a large number of jobs. To evaluate the impact of the period range on the runtime, we considered period ranges from 1 order of magnitude to 3 orders of magnitude in steps of 0.5, i.e., from $[10, 100]$ to $[10, 10000]$. The runtime evaluation is shown in Figure 5.17.[9] It shows that the task-level convolution-based approaches, i.e., **Pruning**, **Unify**, and **Approx**, are more sensitive to an increased period range than the analytical bounds. The runtime increase for **Pruning** stems from the larger number of considered test points and from the larger number of jobs per task. On the other hand, the runtime of **Approx** increases more slowly since only the number of considered jobs increases but not the number of test points, since **Approx** only examines the last release of each higher-priority task. Furthermore, **Unify** is faster than **Approx** which means that the union of a potentially large number of equivalence classes has a larger impact than only considering a small number of time points. **Conv. Merge** only delivered

9 Note that the evaluations presented in Figure 5.17 are performed on a different machine, an Intel Core i7 – 8550U at 1.8GHz with 16GB Ram and 8MB L3 cache. Hence, the resulting runtimes in Figure 5.17 are not directly comparable to the runtimes in the other figures.

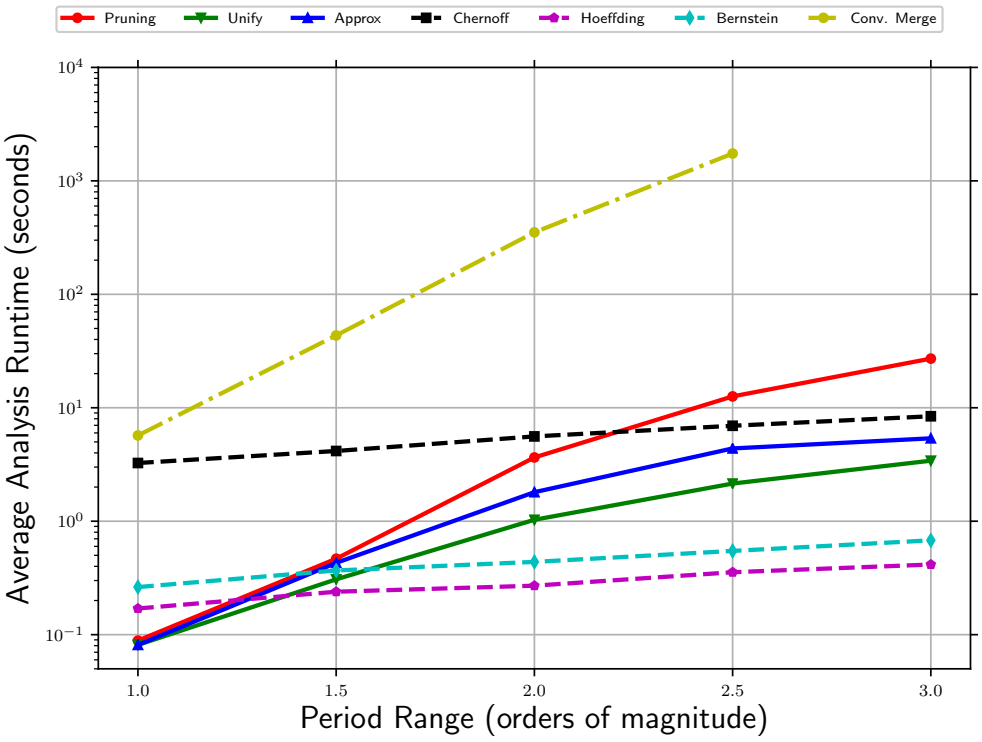


Figure 5.17: The impact of the considered period range on the runtime. We considered 1 order of magnitude, i.e., $[10, 100]$, to 3 orders of magnitude, i.e., $[10, 10000]$.

results for 18 of the task sets with a maximum period range of 2 magnitudes, and for 15 of the task sets with a period range of 2.5 magnitudes. For 3 orders of magnitudes, only in 1 case a result was delivered. The related datapoint is therefore omitted. The average runtime increase of **Conv. Merge** compared to **Pruning** was approximately 64, 93, 96, and 138 times, i.e., it increased with the period range. We note that the average values for all approaches are calculated over all task sets where the related approach delivered a result. The reason is that the sets where **Conv. Merge** did not deliver results are the sets where the runtime of the other approaches is largest, and removing these task sets from the evaluation would therefore result in an unrealistically small average runtime for all approaches. We also note that the actual runtimes for **Conv. Merge**, **Pruning**, **Unify**, and **Approx** differed by approximately 2 orders of magnitude, depending on the specific task set. This difference also becomes larger with the period range. The reason is that under the same setting the number of jobs that have to be considered differs largely depending on the smallest and largest period in the task set.

Most importantly, the provided task-level convolution-based approaches are even able to deliver results for large task sets, since the time needed to evaluate a single point in time still remains in the scale of minutes. We evaluated task sets with 75 and 100 tasks, where on average one time point was evaluated in 621.6 and 791.1 seconds, respectively. Therefore, when a given task set needs to be analyzed, the approach can be used directly, especially since it is highly parallelizable due to the fact that different points in time can be analyzed completely individually. Hence, we suggest to first run *Hoeffding's* as well as *Bernstein's* bounds since they

have a small runtime even for large task sets. If a sufficiently low worst-case deadline failure probability cannot be guaranteed from these bounds, we propose to run the multinomial-based approach with equivalence class union in parallel on multiple machines by partitioning the time points equally.

5.5 CONCLUSION

Systems with Dynamic Real-Time Guarantees

We provided *Systems with Dynamic Real-Time Guarantees* to model real-time task sets in an uncertain execution environment. While previous models focused on one specific area with uncertain behaviour, e.g., fault tolerance or mixed-criticality, our model was designed with a focus on general applicability for systems with an uncertain execution behaviour. *Systems with Dynamic Real-Time Guarantees* provide *dynamic timing guarantees*, i.e., *full timing guarantees* and *limited timing guarantees*, depending on the execution mode of the tasks in the system without the necessity of any online adaption in case of abnormal execution behaviour. Instead, static-priority scheduling is exploited and all guarantees are provided offline. In the course of our exploration, we defined the model of *Systems with Dynamic Real-Time Guarantees*, presented a schedulability test as well as a way to find an optimal assignment of static-priorities for such systems, and showed how to monitor the system state online. The model was later extended to *Multiprocessor Systems with Dynamic Real-Time Guarantees*, exploring both partitioned and semi-partitioned scheduling strategies. Moreover, we introduced the concept of *full and partial compensation* to enhance the system reliability should one or more processors suffer from abnormal execution behavior during a longer interval of time, e.g., due to intermittent faults.

full timing guarantees

limited timing guarantees

Multiprocessor Systems with Dynamic Real-Time Guarantees

full compensation

partial compensation

The evaluations provided good support for our claim that uncertain and faulty execution environments can be reasonably handled without any online adaptation if certain properties can be provided offline. We showed that, if the fault rate and task settings are given, the percentage of time where the system only provides *limited timing guarantees* can be approximated. For a concrete system, this can be used to decide whether additional adaptation might be needed. The evaluations further showed that reasonable acceptance ratios can be achieved for partitioned and semi-partitioned scheduling. Not least, we found that improving the system robustness by applying the proposed compensation techniques entails a tolerable trade-off between acceptance ratio and reliability.

worst-case deadline failure probability

Furthermore, we provided a novel way to calculate the *worst-case deadline failure probability* of constrained-deadline sporadic real-time tasks on uniprocessor platforms, where time points are considered individually. Our approach convolves the equivalence classes of a task represented by the values of the related multinomial distribution. The runtime of this approach can be improved by the detailed pruning technique without any precision loss. Furthermore, we presented an approximation that unifies equivalent classes with a bounded loss of precision. We demonstrated the effectiveness in the evaluations, specifically showing that our approach scales reasonably even for large task sets.

6

SELF-SUSPENSION AND ITS APPLICATIONS IN MULTIPROCESSOR SYNCHRONIZATION

Self-suspension behaviour has become increasingly important for many real-time applications, due to 1) cloud offloading in the *Internet of Things* era [ARS18], 2) the interactions with external devices, such as GPUs [LLZ+15], I/O devices [KSS+07], and accelerators [BA05], 3) suspension-aware protocols for multiprocessor resource synchronization [Raj90; Bra13], etc. However, introducing suspension delays may negatively impact real-time schedulability, particularly given that such delays can be quite lengthy in many scenarios.

Two self-suspension models are studied in the literature. They are applicable in different scenarios and have a high tradeoff between flexibility and accuracy:

- The *dynamic self-suspension* model allows a job of task τ_i to suspend itself at any moment before it finishes as long as the maximum self-suspension time S_i is not violated. It can be utilized when only limited information about the suspension behavior is known. It has a higher flexibility, but results in more pessimistic analyses and designs of scheduling policies if the suspending pattern can be defined precisely. *dynamic self-suspension*
- The *segmented self-suspension* model characterizes the lengths of the computation segments and suspension intervals as an array, composed of $m_i + 1$ computation segments that are separated by m_i suspension intervals. It has a lower flexibility, but the self-suspending structure can be exploited by the scheduling algorithms for better scheduling decisions. However, such a concrete segmented pattern is only achievable if the structure of the program is well designed and the execution pattern is determinable. *segmented self-suspension*

We first consider the *segmented self-suspension* model, where the scheduler design problem is $\mathcal{NP}$-hard in the strong sense as shown by Ridouard et al. [RRC04]. Chen [Che16a] showed that deciding whether a segmented self-suspension task set is schedulable by a static-priority scheduling policy is $co\mathcal{NP}$-hard in the strong sense. Approximation algorithms can resolve this computational complexity issue in $\mathcal{NP}$-hard scheduling problems. In real-time systems, approximations with *speedup or resource augmentation factors* ensure a bounded gap between the derived solution and the optimal solution for $\mathcal{NP}$-hard problems. *segmented self-suspension* *speedup factor*

A promising approach is *fixed-relative-deadline* (FRD) scheduling that was introduced by Chen and Liu [CL14]. In FRD scheduling, all subtasks are assigned individual relative deadlines and the subjobs are scheduled under dynamic-priority scheduling or static-priority scheduling with release time enforcement for the subjobs. Hence, setting the deadlines for the subjobs is the most challenging problem regarding the performance of an FRD approach. In Section 6.1, we consider one-segmented self-suspension, i.e., $m_i = 1$, and propose *Shortest* *fixed-relative-deadline* *SEIFDA*

Execution Interval First Deadline Assignment (SEIFDA) that assigns the deadlines in increasing order of the execution interval length, which is defined as $D_i - S_i$ for each task τ_i. In contrast to previous approaches, SEIFDA's assignment strategy is not agnostic to the deadlines assigned for the other tasks but takes them into account, which results in a substantial performance gain compared to the FRD assignment strategies from the literature. Furthermore, we show that SEIFDA has a speedup factor of 3, which is identical to the best known speedup factor of another FRD approach, i.e., of equal-deadline assignment (EDA) [CL14].

multiprocessor resource sharing

resource-oriented partitioned scheduling

semi-partitioned scheduling

One prominent reason for self-suspension behaviour is *multiprocessor resource sharing*. Since the shared resources are usually the schedulability bottleneck in modern multiprocessor platforms, the concept of *resource-oriented partitioned scheduling* (ROP) was proposed by Huang et al. [HYC16] in 2016. The main idea of ROP is to focuses on the *shared resources* instead of the *computing tasks*. Therefore, each shared resource is assigned to one designated *synchronization* processor while the non-critical sections are executed on the application processors, leading to a *semi-partitioned scheduling* since the critical sections are (potentially) migrated. This focus on the resource access results in short response times for the critical sections, while from the perspective of the non-critical sections the execution of the critical sections appears as self-suspension. Utilizing SEIFDA in the design of a *resource oriented partitioned* scheduling (ROP) in Section 6.2 promises good schedulability due to the increased schedulability of SEIFDA compared to other self-suspension scheduling algorithms. Furthermore, we use a release enforcement technique for the task migration to avoid a decrease in schedulability resulting from release jitter. By combining four different approaches for scheduling non-critical sections and two approaches for scheduling critical sections, we explore 8 different algorithms and their effectiveness compared to state-of-the-art multiprocessor synchronization scheduling algorithms in the evaluation. In addition, we show that RM together with PCP under ROP with release enforcement has a speedup factor of 6 with respect to a necessary scheduling condition, improving the best previously known speedup factor result in the literature.

dynamic self-suspension

hybrid self-suspension

Both SEIFDA as well as its application in a ROP with release enforcements have a good performance, both *theoretically* and *empirically*. However, since SEIFDA is only applicable for the segmented self-suspension model, the provided ROP is also restricted to it. The only other model considered in the real-time systems research community is the *dynamic self-suspension* model. Scheduling algorithms and schedulability tests for the dynamic self-suspension model could directly be applied in a ROP. However, while the segmented model is very precise and over-restrictive, the dynamic model is imprecise and over flexible, which leads to an over-pessimistic analysis when more information than the total WCET and the maximum suspension length of tasks is available. Hence, we try to bridge this gap by introducing multiple *hybrid self-suspension* models in Section 6.3. These models provide different tradeoffs between flexibility and precision that can be achieved based on the information that is known for the considered task set. Compared to the dynamic self-suspension model, all hybrid models have at least one additional parameter m_i that predefines the number of self-suspension intervals. However, instead of assuming one specific execution/suspension pattern, a task is seen as a set of (potentially unknown) possible execution/suspension patterns. The hybrid

models provide several options to model the tasks, depending on whether the execution/suspension pattern of a job can be known when it arrives:

- *Pattern-oblivious Models*: It is assumed that the number of self-suspension intervals of a job of task τ_i is known to be at most m_i. Two submodels are considered: In the first one, *individual upper bounds* on the WCET of the individual computation segments and on the maximum suspension time of the suspension intervals are given. In the second one, *multiple execution paths* are known, i.e., each task is described by a set of specific execution/suspension patterns. These patterns are known offline and can be utilized when designing the scheduling policy, but it is not possible to determine which specific pattern will be executed when a new job arrives. *pattern-oblivious*
- *Pattern-clairvoyant Model*: The individual execution/suspension pattern of each job is of τ_i is known offline and also at the moment the job arrives. *pattern-clairvoyant*

We show how these models can be applied to FRD by carefully examining the special case that each task has only one self-suspension interval, i.e., $m_i = 1$, considering SEIFDA. The evaluation shows that, compared to the dynamic self-suspension task model, the hybrid self-suspension task models can achieve different degrees of improvement, depending on the knowledge about the execution/suspension patterns.

6.1 ONE-SEGMENTED SELF-SUSPENSION

For scheduling *segmented self-suspension* task sets, Chen and Liu [CL14] as well as Huang and Chen [HC16] proposed to use release time enforcement, called *fixed-relative-deadline* (FRD), under dynamic-priority scheduling and static-priority scheduling, respectively. An FRD scheduler assigns a separate relative deadline to each computation segment of a task. Thus, the relative deadline assignment policies become critical to the performance of FRD scheduling. Chen and Liu [CL14] introduced a rather simple assignment policy, namely equal-deadline assignment (EDA) that assigns the same relative deadline to each computation segment of a self-suspending task and uses EDF to schedule the computation segments. They showed that EDA yields a better resource augmentation bound than traditional job-level or task-level static-priority scheduling algorithms. While the study in [CL14] assumed one self-suspension interval per task, EDA has bounded *speedup factors* for multiple self-suspension intervals under both EDF and static-priority scheduling as well, as shown in [HC16]. Regardless, its deadline assignment policy is rather straightforward and the potential of FRD scheduling is not fully exploited under EDA. One main concern is that the deadlines of each task are set without considering the other tasks. *segmented self-suspension* *fixed-relative-deadline* *speedup factor*

To tackle this problem, we propose the *Shortest Execution Interval First Deadline Assignment* (SEIFDA) that assigns the deadlines in increasing order of the execution interval length, i.e., the relative deadline minus the self-suspension time. We focus on one-segmented self-suspension task systems, i.e., a job of a task can suspend at most once and therefore $m_i = 1$. When considering task τ_k, SEIFDA greedily chooses any feasible deadline based on the interference from the $k-1$

tasks with already assigned deadlines, assuming that the shorter computation segment of task τ_k has a short relative deadline. This results in several strategies for the deadline selection.

We take a closer look at FRD scheduling in Section 6.1.1 and introduce a general schedulability test for FRD scheduling in Section 6.1.2. To ease the presentation and the implementation of the algorithm, we prove that it is sufficient to consider the case where the first computation segment has a WCET that is not larger than the WCET of the second computation segment in Section 6.1.3, since otherwise the computation segments can be swapped before the deadline assignment and swapped back afterwards. The concept of SEIFDA is presented in Section 6.1.4. We provide three related deadline assignment strategies in Section 6.1.5 which are proven to be incomparable in Section 6.1.6, i.e., they do not dominate each other. Afterwards, we show that SEIFDA has a speedup factor of 3 for each of these strategies in Section 6.1.7. Moreover, we introduce an approximated schedulability test in Section 6.1.8 to achieve a reasonable runtime. In addition, Section 6.1.9 presents a generalized mixed integer linear programming (MILP) that can be formulated based on the tolerable loss in the schedulability test defined by the users. Our evaluation in Section 6.1.10 shows that the resulting FRD scheduling algorithms yield significantly better performance than existing schedulers for such task systems. We assume implicit-deadline tasks for the simplicity of presentation, while our approach can be applied to constrained-deadline tasks as well with minor modifications. The results presented in this section appeared in *Uniprocessor Scheduling Strategies for Self-Suspending Task Systems* in RTNS 2016 *[BHC+16]*.

6.1.1 FIXED-RELATIVE-DEADLINE (FRD) STRATEGIES

fixed-relative-deadline

As we adopt a *fixed-relative-deadline* (FRD) strategy, we introduce the concept in detail, assuming two computation segments. For each $\tau_i \in \mathbf{T}$, an FRD policy assigns relative deadlines $D_{i,1}$ and $D_{i,2}$ for the executions of the first subtask and the second subtask of τ_i, respectively. Based on these relative deadlines, each subjob has its own absolute deadline assigned when a job arrives. Specifically, when a job of τ_i arrives at time t,

- the first subjob, i.e., the first computation segment, has the release time t and its absolute deadline is $t + D_{i,1}$,
- the suspension has to be finished before $t + D_{i,1} + S_i$, and
- the second subjob, i.e., the second computation segment, *is enforced* to be released at time $t + D_{i,1} + S_i$ and its absolute deadline is $t + D_{i,1} + S_i + D_{i,2}$.

The subjobs are scheduled using these relative deadlines with EDF as the underlying scheduling policy.

An FRD assignment is feasible if the WCRT of the first (second, respectively) computation segment of task τ_i is no more than $D_{i,1}$ ($D_{i,2}$, respectively). Moreover, an FRD scheduling policy has to ensure that $D_{i,1} + D_{i,2} + S_i \leq D_i$ to secure the feasibility of the resulting schedule. We assume *implicit-deadline* tasks and, hence, $D_{i,1} + D_{i,2} + S_i = T_i$. Otherwise, we can always increase $D_{i,2}$ by setting it to $T_i - S_i - D_{i,1}$ without jeopardizing the schedulability of the task set.

implicit-deadline

EXISTING FRD APPROACHES

Two existing FRD deadline assignments were discussed in [CL14]

- Proportional (proportional relative deadline assignment):
  $D_{i,1} = \frac{C_{i,1}}{C_{i,1}+C_{i,2}} \cdot (T_i - S_i)$; $D_{i,2} = \frac{C_{i,2}}{C_{i,1}+C_{i,2}} \cdot (T_i - S_i)$
- EDA (equal relative deadline assignment):
  $D_{i,1} = D_{i,2} = \frac{T_i - S_i}{2}$

While Proportional seems very reasonable and EDA seems very pessimistic, it was shown in [CL14] that Proportional does not yield a constant speedup factor. The reason is that the aggressive relative deadline assignment greedily sets $D_{i,1}$ as $\frac{C_{i,1}}{C_{i,1}+C_{i,2}} \cdot (T_i - S_i)$ without considering interference from other tasks.

6.1.2 SCHEDULABILITY TEST FOR FRD

The schedulability tests for FRD that are introduced in [CL14] are mainly for EDA and more general schedulability tests are not provided. Therefore, we first introduce a general schedulability test for FRD. We use *demand bound functions* (DBF) to calculate the maximum cumulative execution time requirement of a task over a given interval $[t_0, t_0 + t)$ when the arrival time of the computation segments is within this interval. For simplicity of presentation, we set t_0 to 0 for the illustrative example used in this section. The concrete DBF for an FRD scheduling policy depends only on the value of $D_{i,1}$ as $D_{i,2} = T_i - S_i - D_{i,1}$ as discussed before.

demand bound function

One intuitive way to formulate the DBFs of a task for an FRD scheduling policy is to represent it as a *generalized multiframe task* (GMF) [BCG+99]. In the GMF task model, task τ_i is represented by a 3-tuple of vectors $(\overrightarrow{C_i}, \overrightarrow{D_i}, \overrightarrow{T_i})$ where $\overrightarrow{C_i}$, $\overrightarrow{D_i}$, and $\overrightarrow{T_i}$ are vectors of identical length, representing the WCETs, relative deadlines, and interarrival times of the frames, respectively. We use superscripts when referring to the terms in GMF tasks. For one-segmented self-suspension, these vectors have a length of 2, resulting in two frames depending on the values of $D_{i,1}$, $D_{i,2}$, and S_i. The j^{th} frame of a task τ_i has the WCET, relative deadline, and interarrival time of the $(j \ mod \ 2)^{th}$ frame. For a one-segment self-suspending task, the resulting GMF has two frames: $\tau_i = \{(C_{i,1}, D_i^1, T_i^1), (C_{i,2}, D_i^2, T_i^2)\}$. As the second computation segment is released after the suspension interval, $D_i^1 = D_{i,1}$ and $T_i^1 = D_{i,1} + S_i$. Moreover, for the second frame $T_i^2 = T_i - T_i^1 = T_i - D_{i,1} - S_i$ and $D_i^2 = D_{i,2} = T_i - D_{i,1} - S_i$. Now we formulate the DBFs for the case where the segment released at time 0 is represented by the first frame as dbf_i^1 and by the second frame as dbf_i^2 in Eq. (6.1) and Eq. (6.2), respectively.

generalized multiframe task

If the first computation segment is released at 0, the segment has to be finished at $t = D_{i,1}$ while the second segment has to be finished at $t = T_i$. This pattern repeats periodically and is formalized in Eq. (6.1):

$$dbf_i^1(t, D_{i,1}) = \left\lfloor \frac{t + (T_i - D_{i,1})}{T_i} \right\rfloor C_{i,1} + \left\lfloor \frac{t}{T_i} \right\rfloor C_{i,2} \tag{6.1}$$

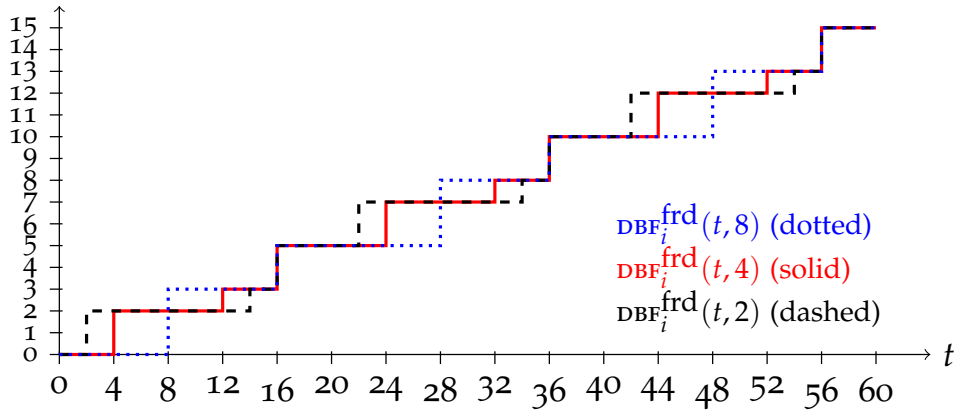


Figure 6.1: An example of $\text{DBF}_i^{\text{frd}}(t, D_{i,1})$ for different values of $D_{i,1}$, where $C_{i,1} = 2$, $C_{i,2} = 3$, $S_i = 4$, and $T_i = 20$. *Adapted from [BHC+16].*

If the second computation segment is released at 0, it has to be finished at time $t = D_{i,2}$, the behavior is identical to releasing the first segment at time $-(D_{i,1} + S_i)$. Hence, the first segment has to be finished at time $T_i - S_i$. This pattern repeats periodically and is formalized in Eq. (6.2):

$$dbf_i^2(t, D_{i,1}) = \left\lfloor \frac{t + (D_{i,1} + S_i)}{T_i} \right\rfloor C_{i,2} + \left\lfloor \frac{t + S_i}{T_i} \right\rfloor C_{i,1} \tag{6.2}$$

The exact DBF for τ_i under an FRD assignment is the maximum of the two possible arrival patterns:

$$\text{DBF}_i^{\text{frd}}(t, D_{i,1}) = \max(dbf_i^1(t, D_{i,1}), dbf_i^2(t, D_{i,1})) \tag{6.3}$$

We use the DBF in Eq. (6.3) for an exact schedulability test:

exact test

Theorem 6.1 (Exact Schedulability Test for FRD)**.** *An FRD schedule is feasible if and only if*

$$\sum_{\tau_i \in \boldsymbol{T}} \text{DBF}_i^{frd}(t, D_{i,1}) \leq t \qquad \forall t \geq 0 \tag{6.4}$$

Proof. This follows directly from Theorem 1 in [BCG+99], i.e., the schedulability condition for generalized multiframe task systems under EDF using demand bound functions. □

How the DBF differs depending on the setting for $D_{i,1}$ is detailed in Figure 6.1 for a task with $C_{i,1} = 2$, $C_{i,2} = 3$, $S_i = 4$, and $T_i = 20$ for three different settings of $D_{i,1}$, i.e., $D_{i,1} = 2$ (black, dashed), 4 (red, solid), and 8 (blue, dotted). For example, with $D_{i,1} = 4$ we get $D_{i,2} = 12$. We consider two cases, depending on whether the computation segment released at time 0 is $C_{i,1}$ or $C_{i,2}$, and take the maximum of both cases as the maximum possible workload. If $C_{i,1}$ is released at 0, the DBF equals 0 in the interval $[0, 4)$, as no workload has to be finished up until this point.

The maximum workload after $t = 4$ is at least 2, as $C_{i,1}$ has to be finished, and at $t = 20$ it is 5, as both $C_{i,1}$ and $C_{i,2}$ have to be finished. When $C_{i,2}$ starts at $t = 0$ it must be finished at 12, and thus the total workload in $[0, 12)$ is 3. In this case, $C_{i,1}$ is released at $t = 12$ with absolute deadline 16, followed by the suspension interval, thus the workload is 3 in $[12, 16)$ and 5 in $[16, 20)$ if $C_{i,2}$ is released first. The red line in Figure 6.1 is the maximum of both cases in $[0, 20)$. As the task is released periodically with period 20, the DBF is also periodic with period 20. Note that each period has only 3 jump points, since the jump at T_i by $dbf_i^1(t, D_{i,1})$ is already covered by the jump of $dbf_i^2(t, D_{i,1})$ at $T_i - S_i$.

In addition to the exact schedulability test, we introduce two necessary conditions for the schedulability of one-segmented self-suspending task sets. One for the schedulability under an FRD assignment and one for any arbitrary scheduling algorithm. This allows us to compare our approach to the best possible result any FRD scheduling algorithm and any scheduling algorithm could provide.

Lemma 6.2 (Necessary Condition for FRD). *If there exists an FRD schedule to feasibly schedule* $\boldsymbol{T}$*, then* *necessary test*

$$\sum_{\tau_i \in \boldsymbol{T}} \text{DBF}_i^{\textit{frd-nece}}(t) \leq t, \qquad \forall t \geq 0 \tag{6.5}$$

where

$$\text{DBF}_i^{\textit{frd-nece}}(t) = \left(\left\lfloor \frac{t + S_i}{T_i} \right\rfloor \right) (C_{i,1} + C_{i,2}) \tag{6.6}$$

Proof. This was proved in Lemma 1 in [CL14] with a slightly different formulation of the equation. □

We now provide a necessary condition for any arbitrary scheduling algorithm for implicit-deadline one-segment self-suspension task sets, assuming that $C_{i,1}$, $C_{i,2}$, and S_i are given. The following equation lower bounds the workload in the current period

$$\text{G}_i(t) = \begin{cases} 0 & \text{if } 0 \leq t < T_i - S_i \\ C_i^{max} & \text{if } T_i - S_i \leq t < T_i \end{cases} \tag{6.7}$$

When considering multiple releases, Eq. (6.7) lower bounds the workload in the last period, which is started but not finished yet, i.e., in the interval $\left[\left\lfloor \frac{t}{T_i} \right\rfloor \cdot T_i, t \right]$. Combining this with the workload contributed in each completed period which is $C_{i,1} + C_{i,2}$, we get a lower bound over a given time interval of length t:

$$\text{DBF}_i^{\text{nece}}(t) = \left\lfloor \frac{t}{T_i} \right\rfloor (C_{i,1} + C_{i,2}) + \text{G}_i \left(t - \left\lfloor \frac{t}{T_i} \right\rfloor \cdot T_i \right) \tag{6.8}$$

Hence, we get the following necessary condition:

Lemma 6.3 (General Necessary Condition for One-Segmented Self-Suspension). *If task set* $\boldsymbol{T}$ *can be feasibly scheduled, then* *necessary test*

$$\sum_{\tau_i \in \boldsymbol{T}} \text{DBF}_i^{\textit{nece}}(t) \leq t, \qquad \forall t \geq 0 \tag{6.9}$$

Proof. Obviously $C_{i,1} + C_{i,2}$ have to be scheduled after a complete interval of length T_i, independent from the concrete scheduling policy. We now have to show that Eq. (6.7) is a lower bound on the possible workload distributions over one period.[1] We consider the two cases $C_{i,1} \geq C_{i,2}$ and $C_{i,1} < C_{i,2}$. If $C_{i,1} \geq C_{i,2}$ and we release $C_{i,1}$ at time $t_0 = 0$, the first subjob has to be finished before $T_i - S_i$, as S_i and the execution of $C_{i,2}$ still have to happen before T_i. If $C_{i,1} < C_{i,2}$, we release $C_{i,1}$ at $-S_i - C_{i,1}$ and thus $C_{i,2}$ has to be finished before $T_i - S_i$ independent from the scheduling policy. As both the release patterns and the DBF are periodic with period T_i this concludes the proof. □

6.1.3 TASK SET TRANSFORMATION

Before presenting our solution, we first examine some characteristics of the demand bound function $\text{DBF}_i^{\text{frd}}(t, D_{i,1})$. This provides an important transformation of task τ_i to simplify the following presentation. Since all the step functions in Eq. (6.1) and Eq. (6.2) have a period T_i, it is clear that $\text{DBF}_i^{\text{frd}}(t, D_{i,1})$ is in general periodic with at most four individual increasing points in one period.

Suppose that we are interested in time t with $\ell T_i \leq t < (\ell + 1)T_i$ where ℓ is a non-negative integer. For $dbf_i^1(t)$, we know that

- $dbf_i^1(t) = \ell(C_{i,1} + C_{i,2})$ when $\ell T_i \leq t < \ell T_i + D_{i,1}$;
- $dbf_i^1(t) = \ell(C_{i,1} + C_{i,2}) + C_{i,1}$ when $\ell T_i + D_{i,1} \leq t < (\ell + 1)T_i$.

For $dbf_i^2(t)$, we know that

- $dbf_i^2(t) = \ell(C_{i,1} + C_{i,2})$ when $\ell T_i \leq t < \ell T_i + (T_i - S_i - D_{i,1}) = \ell T_i + D_{i,2}$;
- $dbf_i^2(t) = \ell(C_{i,1} + C_{i,2}) + C_{i,2}$ when $\ell T_i + D_{i,2} \leq t < \ell T_i + T_i - S_i$;
- $dbf_i^2(t) = (\ell + 1)(C_{i,1} + C_{i,2})$ when $\ell T_i + T_i - S_i \leq t < (\ell + 1)T_i$.

Therefore, $dbf_i^2(t) \geq dbf_i^1(t)$ if $(t \mod T_i) > T_i - S_i$. Moreover, the following two properties follow directly from the definition:

Lemma 6.4. *If $C_{i,1} \leq C_{i,2}$ and $D_{i,1} \geq (T_i - S_i)/2$, then*

$$\forall t \geq 0, \qquad \text{DBF}_i^{frd}(t, D_{i,1}) \geq \text{DBF}_i^{frd}(t, T_i - S_i - D_{i,1}).$$

Lemma 6.5. *If $C_{i,1} \geq C_{i,2}$ and $D_{i,1} \leq (T_i - S_i)/2$, then*

$$\forall t \geq 0, \qquad \text{DBF}_i^{frd}(t, D_{i,1}) \geq \text{DBF}_i^{frd}(t, T_i - S_i - D_{i,1}).$$

These two lemmas suggest assigning a shorter relative deadline to the shorter computation segment with C_i^{min} for each task τ_i. However, it is notationally inconvenient to distinguish two cases, depending on whether $C_{i,1}$ is smaller or not. Fortunately, the notational complication can be easily handled by swapping $C_{i,1}$ and $C_{i,2}$ if $C_{i,1} > C_{i,2}$.

1 The remaining proof is identical to the proof of Lemma 2 in [CL14]. Since our condition is stronger, we include the proof for completeness.

Lemma 6.6. *Suppose that $C_{i,1} > C_{i,2}$ for a task τ_i. We can create a corresponding task τ_i^* with the same parameters as τ_i but $C_{i,1}$ and $C_{i,2}$ are swapped in task τ_i^*. If $D_{i,1} \geq (T_i - S_i)/2$, then*

$$\forall t \geq 0, \qquad \text{DBF}_i^{frd}(t, D_{i,1}) = \text{DBF}_{i^*}^{frd}(t, T_i - S_i - D_{i,1})$$

where $\text{DBF}_{i^*}^{frd}(t, T_i - S_i - D_{i,1})$ is the demand bound function of task τ_i^* by setting the relative deadline of the first computation segment in task τ_i^* (i.e., execution time $C_{i,2}$) to $T_i - S_i - D_{i,1}$.

Proof. This follows from inspecting the corresponding demand bound functions, as they are identical. □

Hence, *we will implicitly assume that* $C_{i,1} \leq C_{i,2}$. If $C_{i,2} < C_{i,1}$, we can simply reorder them before proceeding to the relative deadline assignment of task τ_i and swap them, together with the assigned deadlines, back after the assignment. Based on Lemma 6.6 and the discussions earlier, this does not result in any additional restriction, but simplifies the presentation flow.

6.1.4 GREEDY APPROACH

Although EDA only greedily assigns the relative deadline, it was already shown in [CL14] that the following condition

$$\text{DBF}_i^{\text{frd}}(t, (T_i - S_i)/2) \leq \text{DBF}_i^{\text{frd-nece}}(t) \tag{6.10}$$

holds for any $t \geq 0$. Hence, the idea behind EDA was to keep this constant factor by setting $D_{i,1}$ to $(T_i - S_i)/2$. Chen and Liu [CL14] showed that EDA has an arbitrary speedup factor of 3. Nevertheless, EDA has certain drawbacks:

1. By default, it cannot handle task sets where for any task τ_i in the set $C_i^{max} > (T_i - S_i)/2$.
2. Assigning $D_{i,1}$ to $(T_i - S_i)/2$ is pretty aggressive and therefore not necessarily a good option since the demand bound function for setting $D_{i,1}$ to $(T_i - S_i)/2$ may have a large jump too early. Hence, EDA was chosen as one of the examples where the too-aggressive enforcement led to a good speedup factor but to a bad performance in Section 4.5.3.

A large early jump in the DBF can be observed in Figure 6.1, where $\text{DBF}_i^{\text{frd}}(t, 8)$ is the EDA deadline setting. Contrary to the other two examples, the EDA setting has only two jumps and the first jump at time $t = 8$ is the latest jump, but $t = 8$ is also the earliest moment in time where a demand of 3 can be reached. Whether a later time for the first jump or a smaller first jump is beneficial depends on the task set under consideration. However, the problem of a large early demand increases with the difference between $C_{i,1}$ and $C_{i,2}$. Furthermore, EDA does not allow for a small increase of the demand earlier, to achieve the benefit that the larger portion of the demand has to be finished later. However, to allow such a benefit, an algorithm must consider the deadline setting of other tasks and cannot

Algorithm 2 Shortest Execution Interval First Deadline Assignment (SEIFDA)

Input: $\mathbf{T}$ of n one-segment self-suspension sporadic implicit-deadline tasks
1: re-index (sort) tasks such that $T_i - S_i \leq T_j - S_j$ for $i < j$;
2: **for** $k = 1$ to n **do**
3: **if** $\exists x \in \left(C_{k,1}, \frac{T_k - S_k}{2}\right]$ where Eq. (6.11) holds **then**
4: chose x^* from these values according to the deadline assignment strategy;
5: set $D_{k,1} \leftarrow x^*$, and $D_{k,2} \leftarrow T_k - S_k - x^*$;
6: **else**
7: **return** "no feasible FRD schedule is found";
8: **return** return the relative deadline assignment for each task τ_i in $\mathbf{T}$;

chose deadlines based on a predefined criterion that is independent from the other tasks.

SEIFDA

Therefore, we propose the following algorithm called *Shortest Execution Interval First Deadline Assignment* (SEIFDA): First, we re-index (sort) the given n tasks such that $T_i - S_i \leq T_j - S_j$ for $i < j$. Then, we iteratively assign their relative deadlines under FRD scheduling, starting from task τ_1 to task τ_n. Suppose that the relative deadlines $D_{i,1}$ and $D_{i,2}$ of all tasks $\tau_i \in \{\tau_1, \tau_2, \ldots, \tau_{k-1}\}$ have been already assigned.

Note that, based on Section 6.1.3, we only have to consider $C_{k,1} \leq C_{k,2}$ for the deadline assignment. If $C_{k,1} > C_{k,2}$ we swap $C_{k,1}$ and $C_{k,2}$ before the deadline assignment, swap them back after the assignment, and swap the respective deadlines as well. As shown in Lemma 6.4, if a feasible FRD assignment exists we can always assign the deadline of $C_{k,1}$ to a $D_{k,1}$ with $D_{k,1} \leq (T_k - S_k)/2$. To be more precise, if at least one x in the range of $(C_{k,1}, (T_k - S_k)/2]$ with

$$\mathrm{DBF}_k^{\mathrm{frd}}(t, x) + \sum_{i=1}^{k-1} \mathrm{DBF}_i^{\mathrm{frd}}(t, D_{i,1}) \leq t, \quad \forall t \geq 0 \tag{6.11}$$

exists, then we greedily assign $D_{k,1}$ to one of these x values. The pseudocode of Algorithm SEIFDA is presented in Algorithm 2.

6.1.5 RELATIVE DEADLINES SELECTION FOR τ_k

Algorithm 2 provides a framework for assigning the relative deadlines for FRD scheduling. However, the question remains which value x should be chosen if Eq. (6.11) holds for multiple values. Due to the greedy strategy, the relative deadlines are fixed once they are assigned, i.e., they cannot be changed later in the process if a different choice would have been more beneficial. Suppose that x^* is the chosen value of x when considering task τ_k. We propose the following assignment strategies:

- Minimum x (denoted minD): x^* is the minimum x so that Eq. (6.11) holds.
- Maximum x (denoted maxD): x^* is the maximum x so that Eq. (6.11) holds.
- Proportionally-Bounded-Min x (denoted PBminD): x^* is set to the minimum x so that both Eq. (6.11) and $x \geq \frac{C_{k,1}}{C_{k,1}+C_{k,2}}(T_k - S_k)$ hold.

The resulting demand bound functions differ, depending on how we assign $D_{k,1}$ and, as a result, $D_{k,2}$ in Algorithm SEIFDA. We now show that EDA is a special case of and dominated by SEIFDA-maxD.

Theorem 6.7. *If a task set* $\mathbf{T}$ *is schedulable by Algorithm EDA, the task set* $\mathbf{T}$ *is also schedulable by Algorithm SEIFDA-maxD.*

Proof. EDA assigns $D_{i,1} = D_{i,2} = (T_i - S_i)/2$ for all tasks $\tau_i \in \mathbf{T}$. If $\mathbf{T}$ is schedulable by Algorithm EDA, then

$$\sum_{\tau_i \in \mathbf{T}} \text{DBF}_i^{\text{frd}}(t, (T_i - S_i)/2) \leq t \quad \forall t \geq 0$$

Algorithm SEIFDA-maxD assigns the maximum $x \in \left(C_{k,1}, \frac{T_k - S_k}{2}\right]$ that satisfies Eq. (6.11) when assigning the relative deadlines for task τ_k. Therefore, the algorithm assigns $D_{k,1} = (T_k - S_k)/2$ for all tasks $\tau_k \in \mathbf{T}$ if EDA is feasible. Hence, Algorithm SEIFDA-maxD scheduled all task sets that are feasible by EDA with an identical deadline assignment. □

6.1.6 SEIFDA-MAXD AND SEIFDA-MIND

In the previous subsection, we showed that SEIFDA-maxD dominates EDA. It would be interesting to have such a relation between SEIFDA-minD and EDA or between SEIFDA-maxD and SEIFDA-minD. We show that such a relation does not exist by creating one task set that is schedulable by SEIFDA-maxD but not by SEIFDA-minD (Table 6.1, Figure 6.2) and one that is schedulable by SEIFDA-minD but not by SEIFDA-maxD (Table 6.2, Figure 6.3).

For the task set in Table 6.1, SEIFDA-minD assigns $D_{1,1} = 5$, hence $D_{1,2} = 15$, resulting in steps at 5 and 20 for $\text{DBF}_1^{\text{frd}}(t, 5)$, periodically repeated with a period of 25. This leads to $D_{2,1} \in [25 + \varepsilon; 30]$ as possible values. Independent from the actually assigned value, (in Figure 6.2 we assume 26) the second job of $C_{1,1}$ misses its deadline at $t = 30$ as the total workload is $2 \cdot C_{1,1} + C_{1,2} + C_{2,1} = 30 + \varepsilon > 30$. However, the EDA is feasible as $D_{1,1} = 10 \Rightarrow D_{1,2} = 10$ and the second release of $C_{i,1}$ is feasible with the absolute deadline of 35.

For the task set in Table 6.2, SEIFDA-minD assigns $D_{1,1} = \varepsilon$, hence $D_{1,2} = 20 + \varepsilon$. For $\text{DBF}_1^{\text{frd}}(t, \varepsilon)$ the steps are at ε, $20 + \varepsilon$, and $20 + 2\varepsilon$. With $D_{2,1} = 10 + 2\varepsilon$ this leads to a schedulable task set as shown by the DBF in Figure 6.3. If SEIFDA-maxD is used $D_{1,1} = D_{1,2} = 10 + \varepsilon$. This leads to a deadline miss for $C_{2,1}$ no matter which deadline is assigned (in Figure 6.2 we assume $C_{2,1} = 20$) as the total workload in the interval $[0, 20]$ is $20 + 2\varepsilon$ if $C_{1,2}$ and $C_{2,1}$ are both released at time 0. This directly leads to the following theorem:

Theorem 6.8. *SEIFDA-minD does not dominate SEIFDA-maxD and SEIFDA-maxD does not dominate SEIFDA-minD.*

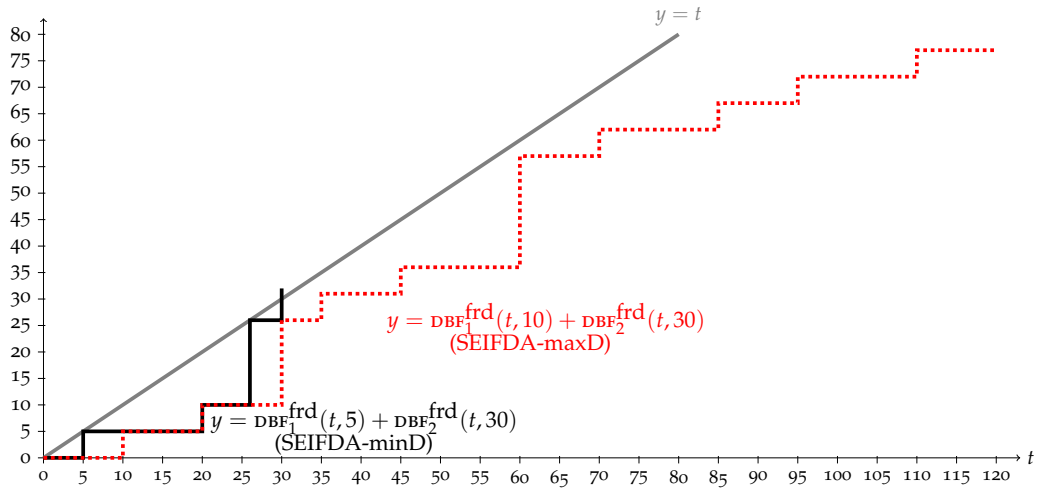


Figure 6.2: SEIFDA-maxD and SEIFDA-minD do not dominate each other (part 1): Schedulability test for SEIFDA-maxD (red) and SEIFDA-minD (black) for the task set in Table 6.1, $\varepsilon = 1$. *Adapted from [BHC+16].*

					SEIFDA-minD		SEIFDA-maxD	
Task	$C_{i,1}$	$C_{i,2}$	S_i	T_i	$D_{i,1}$	$D_{i,2}$	$D_{i,1}$	$D_{i,2}$
τ_1	5	5	5	25	5	15	10	10
τ_2	15+ε	15+ε	940	1000	↯	↯	30	30

Table 6.1: SEIFDA-maxD and SEIFDA-minD do not dominate each other (part 1): An example for comparing SEIFDA-maxD and SEIFDA-minD, where $0 < \varepsilon \leq 1$. A ↯ denotes that SEIFDA-minD does not find a feasible value for $D_{2,1}$ and thus $D_{2,2}$ is not assigned either. *Adapted from [BHC+16].*

6.1.7 SPEEDUP FACTOR OF SEIFDA

Based on the assumption that $C_{i,1} \leq C_{i,2}$, the following lemma provides inequalities between $\text{DBF}_i^{\text{frd}}(t, D_{i,1})$ and the necessary conditions when $t \geq (T_i - S_i)/2$.

Lemma 6.9. *Suppose that $D_{i,1}$ is assigned with $0 < D_{i,1} \leq (T_i - S_i)/2$. For any time $t \geq (T_i - S_i)/2$, we get*

$$\text{DBF}_i^{frd}(t, D_{i,1}) \leq 2\text{DBF}_i^{nece}(t) \quad \textit{if } T_i - S_i \leq t < T_i + D_{i,1} \tag{6.12}$$

$$\text{DBF}_i^{frd}(t, D_{i,1}) \leq \text{DBF}_i^{nece}(2t) \quad \textit{otherwise} \tag{6.13}$$

Proof. We consider all cases for $t \geq (T_i - S_i)/2$:

- If $(T_i - S_i)/2 \leq t < T_i - S_i$, we know that $\text{DBF}_i^{\text{frd}}(t, D_{i,1}) \leq C_{i,2} = \text{DBF}_i^{\text{nece}}(T_i - S_i) \leq \text{DBF}_i^{\text{nece}}(2t)$.
- If $T_i - S_i \leq t < T_i + D_{i,1}$, we get $\text{DBF}_i^{\text{frd}}(t, D_{i,1}) = C_{i,1} + C_{i,2} \leq 2\text{DBF}_i^{\text{nece}}(t)$.
- If $T_i + D_{i,1} \leq t \leq (3T_i - S_i)/2$, the result is $\text{DBF}_i^{\text{frd}}(t, D_{i,1}) \leq C_{i,1} + 2C_{i,2} = \text{DBF}_i^{\text{nece}}(2T_i - S_i) \leq \text{DBF}_i^{\text{nece}}(2t)$.
- If $(3T_i - S_i)/2 < t < 2T_i + D_{i,1}$, we get $\text{DBF}_i^{\text{frd}}(t, D_{i,1}) \leq 2\,(C_{i,1} + C_{i,2})$ $\leq 2C_{i,1} + 3C_{i,2} = \text{DBF}_i^{\text{nece}}(3T_i - S_i) \leq \text{DBF}_i^{\text{nece}}(2t)$.
- If $2T_i + D_{i,1} \leq t$, we know $\text{DBF}_i^{\text{frd}}(t, D_{i,1}) \leq \left(\left\lfloor \frac{t}{T_i} \right\rfloor + 1\right)(C_{i,1} + C_{i,2})$ $\leq \text{DBF}_i^{\text{nece}}(t + 2T_i) \leq \text{DBF}_i^{\text{nece}}(2t)$.

□

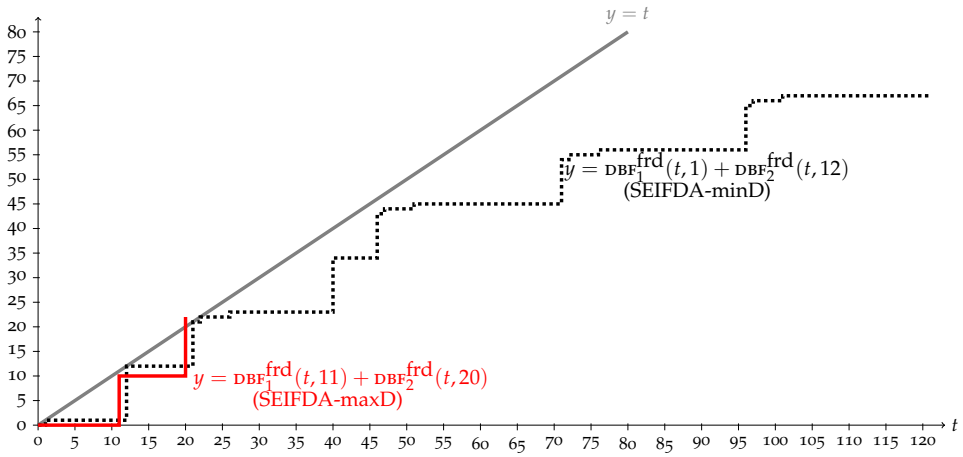


Figure 6.3: SEIFDA-maxD and SEIFDA-minD do not dominate each other (part 2): Schedulability test for SEIFDA-maxD (red) and SEIFDA-minD (black) for the task set in Table 6.2, $\varepsilon = 1$. *Adapted from [BHC+16].*

					SEIFDA-minD		SEIFDA-maxD	
Task	$C_{i,1}$	$C_{i,2}$	S_i	T_i	$D_{i,1}$	$D_{i,2}$	$D_{i,1}$	$D_{i,2}$
τ_1	ε	10	5-2ε	25	ε	20+ε	10+ε	10+ε
τ_2	10+ε	10+ε	960	1000	10+2ε	30-2ε	↯	↯

Table 6.2: SEIFDA-maxD and SEIFDA-minD do not dominate each other (part 2): An example for comparing SEIFDA-maxD and SEIFDA-minD, where $0 < \varepsilon \leq 1$. A ↯ denotes that SEIFDA-maxD does not find a feasible value for $D_{2,1}$ and thus $D_{2,2}$ is not assigned either. *Adapted from [BHC+16].*

We can now show that SEIFDA has a *speedup factor* of 3. *speedup factor*

Theorem 6.10. *The arbitrary speedup factor of SEIFDA by adopting the schedulability test in Theorem 6.1 is* 3.

Proof. Assume that the task set $\mathbf{T}$ cannot be feasibly scheduled by SEIFDA. We show that $\mathbf{T}$ is also not schedulable by any algorithm at speed $\frac{1}{3}$. Recall that the tasks are indexed such that $T_i - S_i \leq T_j - S_j$ if $i \leq j$. Let $\mathbf{T}' = \{\tau_1, \tau_2, \ldots, \tau_k\}$ be the subset of $\mathbf{T}$ such that task set $\mathbf{T}'$ cannot be feasibly scheduled by SEIFDA, and $\mathbf{T}' \setminus \{\tau_k\}$ can be feasibly schedule by SEIFDA, according to Theorem 6.1.

If k is 1, $C_{1,1} + C_{1,2} > T_1 - S_1$ must hold and the arbitrary speedup factor is 1, since $\mathbf{T}$ is by definition not schedulable by any algorithm at the original system speed. Hence, we focus on $k \geq 2$. By the assumption that $\mathbf{T}' \setminus \{\tau_k\}$ can be feasibly scheduled by SEIFDA under the schedulability test in Theorem 6.1, we know

$$\sum_{i=1}^{k-1} \mathrm{DBF}_i^{\mathrm{frd}}(t, D_{i,1}) \leq t \qquad \forall t \geq 0 \tag{6.14}$$

where $D_{i,1}$ is the relative deadline for $C_{i,1}$ under SEIFDA.

When we intend to assign the relative deadlines for task τ_k, the infeasibility of SEIFDA for $\mathbf{T}'$ under the schedulability test in Theorem 6.1 implies that

$$\exists t \geq 0 \qquad \mathrm{DBF}_k^{\mathrm{frd}}\left(t, \frac{T_k - S_k}{2}\right) + \sum_{i=1}^{k-1} \mathrm{DBF}_i^{\mathrm{frd}}(t, D_{i,1}) > t \tag{6.15}$$

That means, at least setting $D_{k,1}$ to $(T_k - S_k)/2$ cannot pass the schedulability test in Theorem 6.1. For notational brevity, we set $D_{k,1}$ to $(T_k - S_k)/2$ for the rest of the proof. This indicates that SEIFDA fails to derive a feasible FRD schedule when assigning $D_{k,1}$ to $(T_k - S_k)/2$.

Suppose that t^* is a certain t such that the condition in Eq. (6.15) holds. By the fact that $\text{DBF}_k^{\text{frd}}(t, D_{k,1}) = 0$ when $t < D_{k,1}$ and the assumption that the FRD schedule of $\mathbf{T}' \setminus \{\tau_k\}$ is feasible (see Eq. (6.14)), we know that t^* must be no less than $D_{k,1}$, which is $(T_k - S_k)/2$. Since $T_i - S_i \leq T_k - S_k$ for $i = 1, 2, \ldots, k$, we also know that $t^* \geq (T_i - S_i)/2$, i.e., the conditions in Lemma 6.9 are applicable. We further partition the task set $\mathbf{T}'$ into two subsets:

- $\mathbf{T}'_1 = \{\tau_i \in \mathbf{T}' \mid T_i - S_i \leq t^* < T_i + D_{i,1}\}$, and
- $\mathbf{T}'_2 = \mathbf{T}' \setminus \mathbf{T}'_1$.

This means that for task τ_i in $\mathbf{T}'_1$, we can use the condition in Eq. (6.12) by Lemma 6.9 and for task τ_i in $\mathbf{T}'_2$, we can use the condition in Eq. (6.13) by Lemma 6.9. From the above discussions, we know

$$\begin{aligned} t^* &< \sum_{\tau_i \in \mathbf{T}'_1} \text{DBF}_i^{\text{frd}}(t^*, D_{i,1}) + \sum_{\tau_i \in \mathbf{T}'_2} \text{DBF}_i^{\text{frd}}(t^*, D_{i,1}) \\ &\leq \sum_{\tau_i \in \mathbf{T}'_1} 2\text{DBF}_i^{\text{nece}}(t^*) + \sum_{\tau_i \in \mathbf{T}'_2} \text{DBF}_i^{\text{nece}}(2t^*) \end{aligned} \tag{6.16}$$

By dividing both sides by t^*, we get

$$1 < 2 \sum_{\tau_i \in \mathbf{T}'_1} \frac{\text{DBF}_i^{\text{nece}}(t^*)}{t^*} + 2 \sum_{\tau_i \in \mathbf{T}'_2} \frac{\text{DBF}_i^{\text{nece}}(2t^*)}{2t^*} \tag{6.17}$$

Since $\mathbf{T}'_1 \cup \mathbf{T}'_2$ is $\mathbf{T}'$ and $\mathbf{T}'_1 \cap \mathbf{T}'_2$ is $\emptyset$, we know

$$y = \sum_{\tau_i \in \mathbf{T}'_1} \frac{\text{DBF}_i^{\text{nece}}(t^*)}{t^*} \leq \sum_{\tau_i \in \mathbf{T}'} \frac{\text{DBF}_i^{\text{nece}}(t^*)}{t^*} \tag{6.18}$$

$$\begin{aligned} z &= \sum_{\tau_i \in \mathbf{T}'_2} \frac{\text{DBF}_i^{\text{nece}}(2t^*)}{2t^*} = \sum_{\tau_i \in \mathbf{T}'} \frac{\text{DBF}_i^{\text{nece}}(2t^*)}{2t^*} - \sum_{\tau_i \in \mathbf{T}'_1} \frac{\text{DBF}_i^{\text{nece}}(2t^*)}{2t^*} \\ &\leq \sum_{\tau_i \in \mathbf{T}'} \frac{\text{DBF}_i^{\text{nece}}(2t^*)}{2t^*} - \sum_{\tau_i \in \mathbf{T}'_1} \frac{\text{DBF}_i^{\text{nece}}(t^*)}{2t^*} \\ &= \sum_{\tau_i \in \mathbf{T}'} \frac{\text{DBF}_i^{\text{nece}}(2t^*)}{2t^*} - y/2 \end{aligned} \tag{6.19}$$

Hence, we know that $\sum_{\tau_i \in \mathbf{T}'} \frac{\text{DBF}_i^{\text{nece}}(2t^*)}{2t^*} \geq z + y/2$ and $\sum_{\tau_i \in \mathbf{T}'} \frac{\text{DBF}_i^{\text{nece}}(t^*)}{t^*} \geq y$. Since $1 < 2y + 2z$ in Eq. (6.17), we know that either $\sum_{\tau_i \in \mathbf{T}'} \frac{\text{DBF}_i^{\text{nece}}(t^*)}{t^*} > 1/3$ or $\sum_{\tau_i \in \mathbf{T}'} \frac{\text{DBF}_i^{\text{nece}}(2t^*)}{2t^*} > 1/3$. The reason is that either $y > 1/3$ or $z + y/2 > 1/3$ holds which can be calculated using the intersection $z + y/2 = y$, i.e., $z = y/2$, and $1 < 2y + 2z = 3y$. Therefore, the arbitrary speedup factor is 3. □

6.1.8 APPROXIMATED TEST AND TIME COMPLEXITY

The schedulability test in Theorem 6.1 is a necessary and sufficient test that requires *exponential time complexity*. To achieve a better runtime, we have to ensure that we do not have to test for all $t \geq 0$. It is known that we only have to test at the t values where the demand bound function $\text{DBF}_i^{\text{frd}}(t)$ actually changes. These time points are

exponential time complexity

$$\mathbf{\Psi}_i = \{D_{i,1} + \ell T_i, T_i - S_i - D_{i,1} + \ell T_i, T_i - S_i + \ell T_i | \ell \in \mathbb{N}^0\} \tag{6.20}$$

where $\mathbb{N}^0$ is the set of non-negative integers. This means, the test in Theorem 6.1 is equivalent to

$$\forall \tau_i \in \mathbf{T}, \quad \forall t \in \mathbf{\Psi}_i, \quad \sum_{\tau_i \in \mathbf{T}} \text{DBF}_i^{\text{frd}}(t) \leq t \tag{6.21}$$

One may further constrain ℓ to be at most H/T_i, where H is the *hyperperiod*, i.e, the least common multiple of the periods of the tasks in $\mathbf{T}$. However, the time complexity remains exponential.

hyperperiod

To reduce the time-complexity, we can utilize *approximated demand bound functions* [CKT02; CC11]. Our general approach is to use the exact demand bound function for g periods of a task, where g is a user-defined (positive) integer, and use a linear approximation to upper bound the DBF after the given number of periods. Similar to the construction of the exact DBFs, we use one approximated DBF for the case where $C_{i,1}$ is released at $t = 0$ in Eq. (6.22a), and one for the case where $C_{i,2}$ is released at $t = 0$ in Eq. (6.22b), and take the maximum of both values in Eq. (6.23).

approximated demand bound function

$$\widehat{dbf}_i^1(t, D_{i,1}) = \begin{cases} dbf_i^1(t, D_{i,1}) & \text{if } t < gT_i \\ U_i t - D_{i,1}U_{i,1} + C_{i,1} & \text{otherwise.} \end{cases} \tag{6.22a}$$

$$\widehat{dbf}_i^2(t, D_{i,1}) = \begin{cases} dbf_i^2(t, D_{i,1}) & \text{if } t < gT_i - S_i \\ U_i(t + S_i) + C_{i,2}\frac{D_{i,1}}{T_i} & \text{otherwise.} \end{cases} \tag{6.22b}$$

$$\widehat{\text{DBF}}_i^{\text{frd}}(t, D_{i,1}) = \max(\widehat{dbf}_i^1(t, D_{i,1}), \widehat{dbf}_i^2(t, D_{i,1})) \tag{6.23}$$

As the proofs in this subsection are rather technical and straight forward we merely provide the ideas here, while the complete proofs are in the Appendix.

Theorem 6.11. *The function* $\widehat{\text{DBF}}_i^{frd}(t, D_{i,1})$ *in Eq.* (6.23) *is a safe upper bound of* $\text{DBF}^{frd}(t, D_{i,1})$ *for any* $t \geq 0$ *and a specified* $D_{i,1} \leq (T_i - S_i)/2$. *Therefore, if* $\sum_{\tau_i \in \mathbf{T}} U_i \leq 1$ *and*

$$\sum_{\tau_i \in \mathbf{T}} \widehat{\text{DBF}}_i^{frd}(t, D_{i,1}) \leq t \qquad \forall t \geq 0$$

then the resulting FRD schedule is feasible. Moreover, this schedulability test can be done in $O(g|\mathbf{T}|^2)$ *time complexity.*

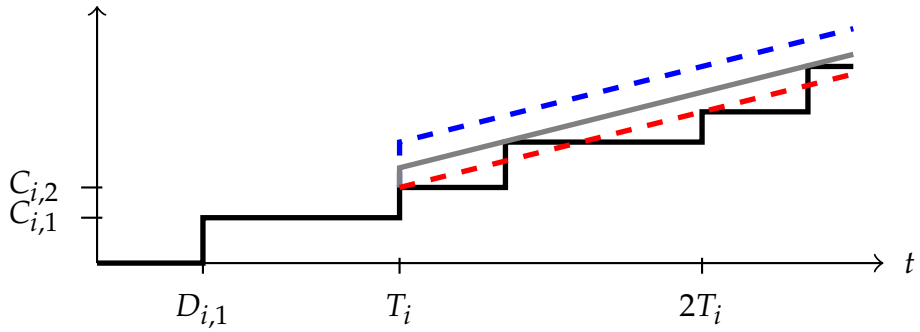


Figure 6.4: The linearized DBF for Eq. (6.22a) with $g = 1$. We use exact steps identical to Eq. (6.1) up to gT_i and linearization after gT_i. We show the linearization of the original, exact curve (black) without adjustment (red dashed, does not work), linearization after jumping by $C_{i,1}$ at gT_i (blue dashed, over approximation), and after adjusting by $D_{i,1}U_{i,1}$ at gT_i (gray, tight).

Proof. The first part of the proof, to show that Eq. (6.22a) is an over approximation of Eq. (6.1) and that Eq. (6.22b) is an over approximation of Eq. (6.2), can be done by inspecting the corresponding values at the non-linear points of Eq. (6.1) and Eq. (6.2), respectively, for $t > gT_i - S_i$, as illustrated in Figure 6.4. This directly leads to the conclusion that Eq. (6.23) is an over approximation of Eq. (6.3). A more detailed discussion is in the Appendix.

To analyze the time complexity, we only have to perform the schedulability tests at the points in time where $\sum_{\tau_i \in \mathbf{T}} \widehat{\text{DBF}}_i^{\text{frd}}(t, D_{i,1})$ changes discontinuously. Each task τ_i has exactly 3 jump points in each of the g periods when $\widehat{\text{DBF}}_i^{\text{frd}}(t, D_{i,1})$ (Eq. (6.23)) is used which leads to $3g$ discrete jump points at $\ell T_i + D_{i,1}$, $\ell T_i + T_i - S_i - D_{i,1}$, and $t = \ell T_i + T_i - S_i$ with $\ell = 0, 1, 2, \ldots, g-1$ for each $\tau_i \in \mathbf{T}$. The reason is that the jump of Eq. (6.22a) at $(l+1)T_i$ is to the same value as the jump of Eq. (6.22b) at $t = \ell T_i + T_i - S_i$. Let $\mathbf{P}$ be the set of all these $3g|\mathbf{T}|$ jump points of all $\tau_i \in \mathbf{T}$ and let t^* be the maximum of the points in $\mathbf{P}$. It is easy to see that $\sum_{\tau_i \in \mathbf{T}} \widehat{\text{DBF}}_i^{\text{frd}}(t, D_{i,1})$ is a linear function for $t > t^*$. Due to the condition $\sum_{i=1}^{n} U_i \leq 1$, this means that we have $\sum_{\tau_i \in \mathbf{T}} \widehat{\text{DBF}}_i^{\text{frd}}(t, D_{i,1}) \leq t$ for all $t > t^*$. Hence, we have to test whether $\sum_{i=1}^{n} U_i \leq 1$ and we have to check all the time points where $\sum_{\tau_i \in \mathbf{T}} \widehat{\text{DBF}}_i^{\text{frd}}(t, D_{i,1})$ is not linear, i.e., all points in $\mathbf{P}$ which are $3g|\mathbf{T}|$ points in total. As each test has to calculate the workload up to the tested point for each of the $|\mathbf{T}|$ tasks, the time complexity is $O(g|\mathbf{T}|^2)$. □

In Theorem 6.11, we proved that a linear approximation of the demand bound functions in Eq. (6.3) can be calculated in $O(g|\mathbf{T}|^2)$ where $g \in \mathbb{N}^0$ is given and $|\mathbf{T}|$ is the number of tasks in the set. We now examine the quality of the approximation with respect to the given g, by providing an upper bound on the ratio between over approximation and exact value.

Theorem 6.12. *For a given integer $g \geq 1$*

$$\widehat{\text{DBF}}_i^{frd}(t, D_{i,1}) \leq \left(1 + \frac{1}{g}\right) \text{DBF}_i^{frd}(t, D_{i,1}) \qquad \forall t \geq 0 \tag{6.24}$$

Proof. We know that both the exact DBFs in Eq. (6.1) and Eq. (6.2) are step functions with two steps per period, resulting in two intervals with the same value. We have to compare their value over this interval to the maximum value the approximated DBF takes over this interval. For example, we compare the values of $dbf_i^1(gT_i, D_{i,1})$ with $\widehat{dbf}_i^1(gT_i + D_{i,1}, D_{i,1})$ and $dbf_i^1(gT_i + D_{i,1}, D_{i,1})$ with $\widehat{dbf}_i^1((g+1)T_i, D_{i,1})$ to conclude for Eq. (6.1) compared to Eq. (6.22a). The details can be found in the Appendix. □

This shows that we can use Eq. (6.23) to formulate Algorithm 2 as an approximation scheme for finding FRD solutions. The needed quality guarantee of $1 + \frac{1}{g}$ (in the schedulability test) follows directly from Theorem 6.12.

6.1.9 MIXED INTEGER LINEAR PROGRAMMING

In this section, we provide a programming under logical conditions to assign the relative deadlines of the computation segments, which can be rephrased as a *mixed integer linear programming* (*MILP*). We utilize the schedulability test in Theorem 6.11 by assuming that $g \geq 1$ is given as an integer. Moreover, let $\mathbf{L}$ be $\{0, 1, 2, \ldots, g-1\}$ for notational brevity. We can formulate the studied problem as the following programming under logical constraints:

MILP

$$\text{find a feasible solution} \tag{6.25a}$$

s.t.

$$0 \leq D_{i,1} \leq \frac{T_i - S_i}{2}, \qquad \forall \tau_i \in \mathbf{T} \tag{6.25b}$$

$$b_{i,j}^{3\ell+1} = \widehat{\text{DBF}_i^{\text{frd}}}(\ell T_i + D_{i,1}, D_{j,1}), \tag{6.25c}$$

$$b_{i,j}^{3\ell+2} = \widehat{\text{DBF}_i^{\text{frd}}}((\ell+1)T_i - S_i - D_{i,1}, D_{j,1}), \tag{6.25d}$$

$$b_{i,j}^{3\ell+3} = \widehat{\text{DBF}_i^{\text{frd}}}((\ell+1)T_i - S_i, D_{j,1}), \tag{6.25e}$$

$$\text{(Eqs. (6.25c), (6.25d), (6.25e)) } \forall \tau_i \in \mathbf{T}, , \tau_j \in \mathbf{T}, \ell \in \{\mathbf{L}\})$$

$$\sum_{\tau_j \in \mathbf{T}} b_{i,j}^{3\ell+1} \leq \ell T_i + D_{i,1}, \tag{6.25f}$$

$$\sum_{\tau_j \in \mathbf{T}} b_{i,j}^{3\ell+2} \leq (\ell+1)T_i - S_i - D_{i,1} \tag{6.25g}$$

$$\sum_{\tau_j \in \mathbf{T}} b_{i,j}^{3\ell+3} \leq (\ell+1)T_i - S_i \tag{6.25h}$$

$$\text{(Eqs. (6.25f) (6.25g) (6.25h) } \forall \tau_i \in \mathbf{T}, \ell \in \{\mathbf{L}\})$$

$D_{i,1}$ and $b_{i,j}^h$ are variables that can be assigned to real numbers. The variable $b_{i,j}^{3\ell+1}$ is the approximate demand bound function $\widehat{\text{DBF}_i^{\text{frd}}}(t, D_{j,1})$ of task τ_j when $t = \ell T_i + D_{i,1}$. Similarly, the variable $b_{i,j}^{3\ell+2}$ is the approximate demand bound function $\widehat{\text{DBF}_i^{\text{frd}}}(t, D_{j,1})$ of τ_j when $t = \ell T_i + D_{i,2} = (\ell+1)T_i - S_i - D_{i,1}$. The approximate demand bound function $\widehat{\text{DBF}_i^{\text{frd}}}(t, D_{j,1})$ of task τ_j is represented by

the variable $b_{i,j}^{3\ell+3}$ when $t = \ell T_i + T_i - S_i = (\ell+1)T_i - S_i$. Therefore, the condition in Eq. (6.25f), Eq. (6.25g), and Eq. (6.25h) is identical to the inequality $\sum_{\tau_j \in \mathbf{T}} \widehat{\mathrm{DBF}}_i^{\mathrm{frd}}(t, D_{j,1}) \leq t$ when t is $\ell T_i + D_{i,1}$, $(\ell+1)T_i - S_i - D_{i,1}$, and $(\ell+1)T_i - S_i$ for every task τ_i in $\mathbf{T}$ and $\ell = 0, 1, 2, \ldots, g-1$.

Hence, by Theorem 6.11, the above programming can be used to search a feasible relative deadline assignment $D_{i,1}$ for $\tau_i \in \mathbf{T}$. The constraints except Eq. (6.25c), Eq. (6.25d), and Eq. (6.25e) (due to the logical conditions inherited from Eq. (6.22a) and Eq. (6.22b)), are linear functions with respect to the variables. By adopting the well-known Big-M Method, each of the logical conditions in Eq. (6.25c), Eq. (6.25d), and Eq. (6.25e) can be expressed using several linear constraints and several binary variables. As a result, the above programming can be implemented as an MILP.

Please note that the presented *mixed integer linear programming* is a special case of the MILP that was developed by Peng and Fisher [PF16] in parallel and has been published in RTCSA 2016.

6.1.10 EVALUATION

We conducted experiments using synthesized task sets to compare the proposed approaches in comparison to other approaches from the literature. The metric to compare the results is the *acceptance ratio* with respect to the task set utilization. We generated 100 task sets with a cardinality of 10 tasks for each of the analyzed utilization levels that ranged from 0% to 100% with steps of 5%.

For each task set, we first generated a set of sporadic implicit-deadline tasks with cardinality 10, adopting the UUniFast method [BB05] to generate a set of utilization values with the given goal. We used the approach suggested by Emberson et al. [ESD10] and generated the task periods according to a log-uniform distribution with two orders of magnitude. To be precise, $\log_{10} T_i$ is a uniform distribution over $[1ms - 100ms]$. The execution time was accordingly set to $C_i = T_i U_i$ and the relative deadline was set to the task periods, i.e., $D_i = T_i$. We converted them to self-suspending tasks by generating the suspension lengths of the tasks according to a uniform distribution in either of three ranges depending on the self-suspension length:

- Short suspension: $[0.01(T_i - C_i), 0.1(T_i - C_i)]$
- Moderate suspension: $[0.1(T_i - C_i), 0.3(T_i - C_i)]$
- Long suspension: $[0.3(T_i - C_i), 0.6(T_i - C_i)]$

We then generated $C_{i,1}$ as a percentage of C_i, according to a uniform distribution, and set $C_{i,2}$ accordingly.

We first analyzed the acceptance rate of SEIFDA, considering the three assignment strategies minD, maxD, and PBminD with respect to $g \in \{1, 2, 3, 5\}$ for the different settings of the suspension length, and compared it to the MILP approach in Eq. (6.25) with $g = 1$. Figure 6.5 displays these results for SEIFDA-minD. The three subfigures show that SEIFDA-minD-1 already does not lose much compared

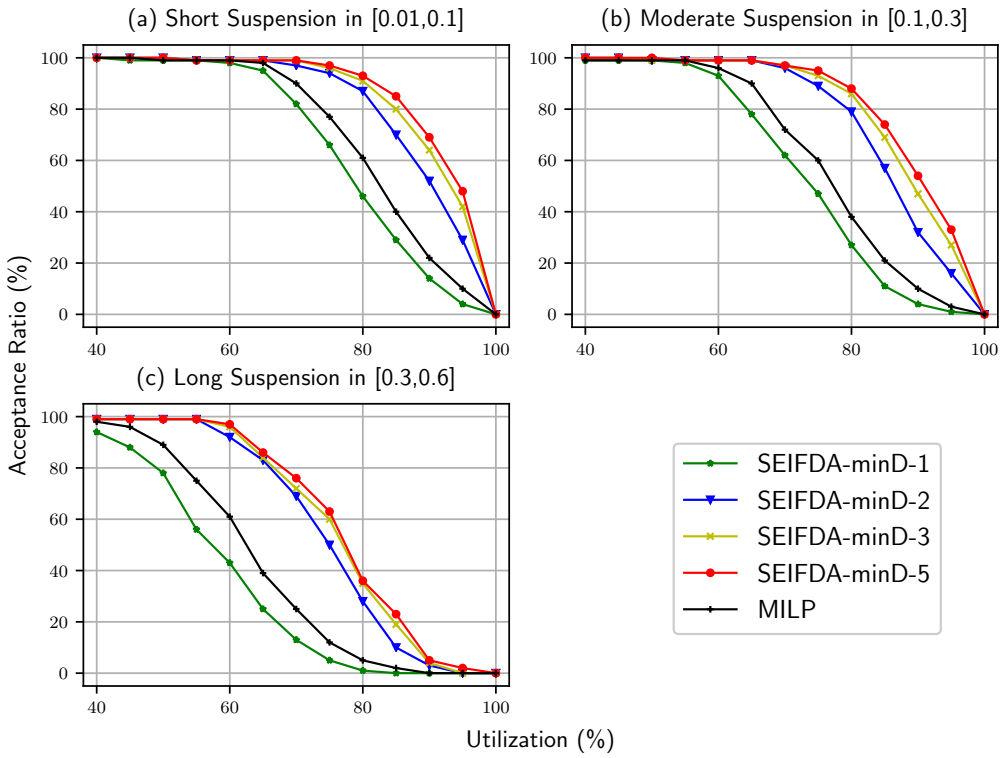


Figure 6.5: Impact of the g value for SEIFDA-minD.
We considered different lengths of the suspension intervals and also compared to the MILP in Eq. (6.25) with $g = 1$. *Adapted from [BHC+16].*

to the MILP with $g = 1$ while SEIFDA-minD-2, SEIFDA-minD-3, and SEIFDA-minD-5 deliver far better results. Also, the gap between SEIFDA-minD-2 and SEIFDA-minD-5 is relatively small. While the MILP with $g = 1$ performs better than SEIFDA-minD-1, the number of variables and constraints grows quadratically with respect to g in our MILP implementation by using the Big-M Method while SEIFDA is linear with respect to g.

Furthermore, SEIFDA-minD, SEIFDA-maxD, and SEIFDA-PBminD were compared. The performance for $g = 2$ and $g = 5$ is detailed in Figure 6.6. It shows that SEIFDA-minD and SEIFDA-PBminD are close to each other, and that SEIFDA-PBminD performs better than SEIFDA-minD in most cases, i.e., only for a long suspension length SEIFDA-minD performs slightly better for some values. SEIFDA-minD and SEIFDA-PBminD both clearly perform better than SEIFDA-maxD. Even SEIFDA-minD-2 and SEIFDA-PBminD-2 outperform SEIFDA-maxD-5 most of the time.

In addition, SEIFDA-maxD-5 and SEIFDA-PBminD-5 were compared with the following scheduling approaches:

- *SCEDF*: the suspension-oblivious approach by converting suspension time into computation time.
- EDA: The Equal-Deadline Assignment under linear demand bound approximations in Theorem 8 in [CL14].
- MILP: The proposed MILP in Section 6.1.9. Gurobi [Gur], a state-of-the-art MILP solver, is used to solve Eq. (6.25).

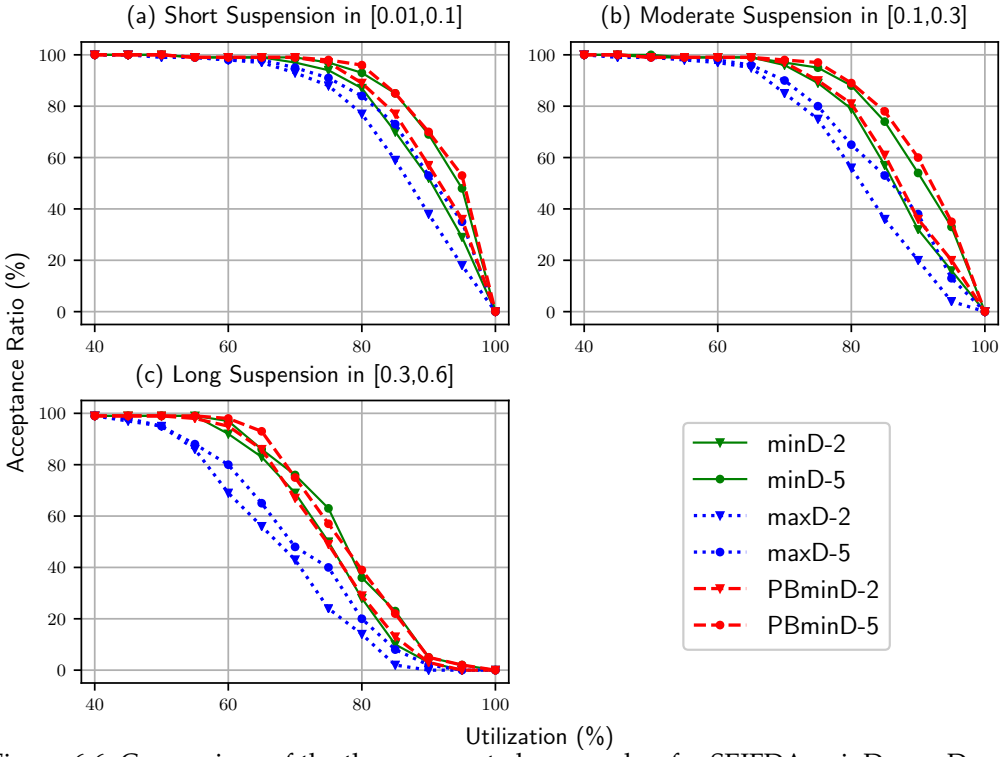


Figure 6.6: Comparison of the three presented approaches for SEIFDA: minD, maxD, and PBminD for g-values 2 and 5 considering different lengths of the suspension intervals. *Adapted from [BHC+16].*

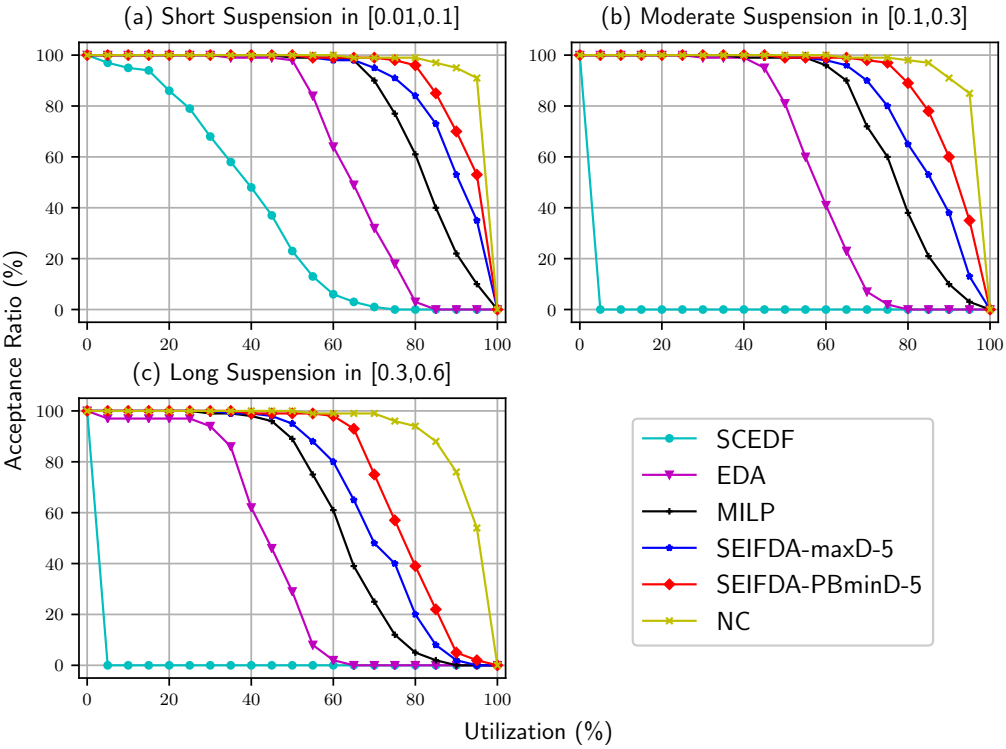


Figure 6.7: Comparison of SEIFDA-maxD-5 and SEIFDA-PBminD-5 with suspension-oblivious EDF (SCEDF), EDA, the MILP in Eq. (6.25) with $g = 1$, and the necessary condition (NC) for arbitrary algorithms (Lemma 6.3) considering different lengths of the suspension intervals. *Adapted from [BHC+16].*

- NC: The necessary condition in Lemma 6.3. We compared to the necessary condition to evaluate how much we may lose to a theoretical optimal algorithm in the worst case.

We chose SEIFDA-PBminD-5 and SEIFDA-maxD-5 to cover the performance range of the SEIFDA-Algorithm. The results are shown in Figure 6.7. EDA is clearly outperformed by the MILP, SEIFDA-PBminD-5, and SEIFDA-maxD-5. While NC does not decrease much when the suspension length is increased, the gap between SEIFDA-PBminD-5 and the necessary condition becomes larger with an increasing suspension length.

6.2 RESOURCE-ORIENTED PARTITIONING

shared resource

The unavoidable blocking time induced by *resource sharing* jeopardizes the timing correctness of *real-time systems*. In uniprocessor systems, mutual exclusion and synchronization based on *priority inheritance* techniques have been well studied. Known protocols like the *Priority Ceiling Protocol* (PCP) [Raj90], the *Priority Inheritance Protocol* (PIP) [Raj90], and the *Stack Resource Policy* (SRP) [Bak91] have been accepted standards for nearly 30 years. Both PCP under static-priority scheduling as well as SRP under dynamic-priority scheduling prevent deadlocks and result in the minimal number of 1 priority-inversion blockings (*pi-blockings*) for work-conserving scheduling algorithms in uniprocessor systems with acceptable implementation overhead.

Priority Ceiling Protocol
Stack Resource Policy
pi-blocking

multiprocessor resource sharing

In *multiprocessor systems, resource sharing* and synchronization lead to additional synchronization overhead, since tasks may not only *share resources* with tasks on the same processor but also with tasks that are running on other processors. While in the uniprocessor scenario the performance of a resource sharing protocol primarily depends on the number of *pi*-blockings and the implementation overhead, in a multiprocessor scenario one must also consider what happens when a job is blocked by a job that is executed on a different processor, and how resource access is prioritized for tasks on multiple processors. Existing protocols can be classified into

suspension-based protocols

- *suspension-based protocols* where a job suspends itself from the processor when waiting for a resource allocated to a job on another processor, and

spin-based protocols

- *spin-based protocols* where a job continues to be executed on the processor in a busy waiting manner until either it gets access to the requested resource or is preempted by a higher-priority job on the same processor.

Furthermore, the performance of multiprocessor resource sharing protocols highly depend on the task partition (under partitioned scheduling) and the task priority order (under global scheduling).

partitioned scheduling

We consider *partitioned scheduling* since it performs reasonably well in practice due to its low runtime overhead. Furthermore, uniprocessor resource sharing protocols can be extended to the multiprocessor case under partitioned scheduling more easily. The existing multiprocessor resource sharing protocols like the Multiprocessor Priority Ceiling Protocol (MPCP) (with suspension locks) by Rajkumar [Raj90] and the Multiprocessor resource sharing Protocol (MrsP) (with

spin locks) by Burns and Wellings [BW13] ensure mutual exclusion under the assumption that the task partition is given. However, it has been shown by Brandenburg and Anderson [BA10] that in the worst case the number of *pi-blockings* can be lower bounded by the number of processors. This means that the advantages of partitioned scheduling, reducing the multiprocessor scheduling problem to uniprocessor subproblems, can be outweighed by the synchronization overhead if the partition is not done carefully. Some heuristics to find good task partitions have been proposed [LNR09; NNB10; WB13a] although without theoretical analysis regarding speedup factors.

pi-blocking

resource-oriented partitioned scheduling

As the shared resources are usually the bottlenecks, *resource-oriented partitioned scheduling* (ROP) was proposed by Huang et al. [HYC16] in 2016 and is adopted in this section. This alternative approach changes the perspective and focuses on the *shared resources* instead of the *computing tasks*. The idea of ROP is to first assign each shared resource to one designated *synchronization processor* while the non-critical sections will be executed on other *application processors*, decoupled from the critical sections. This focus on the resource access allows keeping the response times of the *critical sections* as short as possible. When considering the *non-critical sections*, the worst-case response time of those critical sections can be seen as suspension time of *self-suspending tasks* on a uniprocessor due to the use of partitioned scheduling. ROP focuses on the strategy of task and resource partitioning and is in general comparable with any suspension-based locking protocols extended from the uniprocessor PCP (i.e., DPCP in [RSL88]), SRP, FIFO (i.e., DFLP in [Bra13]), and priority-based non-preemptive scheduling.

synchronization processor

application processor

critical section

non-critical section

Huang et al. [HYC16] provided a general exploration of ROP under *static-priority* scheduling and assumed that the execution pattern of a task is not known and can change from one job to another, i.e., similar to the *dynamic self-suspension* task model. We analyze how the schedulability under ROP can be improved using release enforcement for the task migration under the assumption that the tasks can be modelled according to the *segmented self-suspension* model. To this end, we explain how scheduling algorithms and analyses for segmented self-suspension tasks, e.g., [HC16; GH98] and SEIFDA in Section 6.1, can be jointly applied with ROP. We restrict ourselves to a very fundamental yet challenging special case where each task has one non-nested critical section. A brief comment regarding this restriction is given in Section 6.2.7.

dynamic self-suspension

segmented self-suspension

We first introduce the concept of ROP as introduced by Huang et al. in [HYC16] in Section 6.2.1. Afterwards, we explain the advantages of release enforcement in such a ROP scenario in Section 6.2.2. In Section 6.2.3, schedulability tests for ROP in combination with the considered algorithms are detailed. The resource and task allocation is considered in Section 6.2.4. The examination result is a family of possible algorithms, which adopt *static-priority* scheduling for the processors that execute *critical sections*, and either *static-priority* or *dynamic-priority* scheduling for the processors that execute only *non-critical sections*. In Section 6.2.5, we show that RM together with PCP under ROP with release enforcement has a speedup factor of 6 with respect to a necessary scheduling condition, improving the best previously known result of $11 - 6/(m+1)$ by Huang et al. [HYC16]. We explore 8 different algorithms, combining four different approaches for scheduling non-critical sections and two approaches for scheduling critical sections, and

their effectiveness compared to state-of-the-art multiprocessor synchronization scheduling algorithms and their analyses in the evaluations in Section 6.2.6. Note that we do not intend to compare the performance of the locking protocols here and refer the readers to [YWB15] for a survey and detailed comparisons of the global scheduling protocols. However, we show that a simple combination of release enforcement and ROP can take care of task partitioning and priority assignment directly and yield good performance both *theoretically* and *empirically*. For comparing with the other lock-based protocols for partitioned or semi-partitioned scheduling, reasonable task partitioning or priority assignments have to be provided. The results presented in this section appeared in *Release Enforcement in Resource-Oriented Partitioned Scheduling for Multiprocessor Systems* in RTNS 2017 *[BCH+17]*.

6.2.1 RESOURCE-ORIENTED PARTITION

resource-oriented partitioned scheduling

The main idea of *resource-oriented partitioned scheduling* is to separate the critical and non-critical sections by migrating all critical sections that access the same resource to one dedicated *synchronization processor*. After that, the schedulability of the critical sections and the non-critical sections can be analyzed individually. Therefore, executing the critical section of a task on the synchronization processor can be considered as if the task suspends itself from its *application processor*. Furthermore, migrating all critical sections for a given resource to the same processor allows to directly utilize uniprocessor resource sharing techniques like *PCP* and *SRP*. The general ROP approach introduced by Huang et al. in [HYC16] consists of the following steps:

Priority Ceiling Protocol
Stack Resource Policy

synchronization processor
application processor
critical section
non-critical section

1. The m processors are partitioned into m^R *synchronization processors* for critical sections and $m^C = m - m^R$ *application processors* for non-critical sections.
2. For each shared resource, the related *critical sections* are assigned to one designated *synchronization* processor.
3. The *non-critical sections* of each task are statically allocated onto a designated *application* processor. Note that both critical and non-critical sections of a task may still be allocated to the same processor as the remaining capacity on the synchronization processors can be used to execute non-critical sections.

When a job enters a critical section, it suspends itself on its *application processor*. The job returns to the ready queue of the application processor after its critical section is executed on the *synchronization processor* related to the requested resource. As a task may be executed on more than one processor, resource-oriented partitioned scheduling has additional overheads when compared to *partitioned scheduling*, which are similar to *semi-partitioned scheduling*. The critical points in the design of an algorithm based on ROP are:

partitioned scheduling
semi-partitioned scheduling

1. the *number of synchronization processors*,
2. with respect to the critical sections, the *partition of the shared resources* on those synchronization processors,
3. with respect to the non-critical sections, the *partition of the sporadic real-time tasks* with suspension behaviour onto the application processors,

4. the *assigned base priorities* for the sporadic tasks, and
5. the *moment* where the critical section of a task is requested on the synchronization processor.

The original ROP by Huang et. al [HYC16] only considered the first four points and assumed that task migration is possible at any time, due to the use of the more general dynamic self-suspension model.

6.2.2 RELEASE ENFORCEMENT

critical section

From the perspective of the *application processor*, executing the *critical section* of a task on the *synchronization processor* can be considered as if the task suspends itself. From the perspective of the *synchronization processor* executing a given set of shared resources, the critical sections migrating can be modeled as incoming sporadic tasks. However, if the task migration happens directly when $C_{k,1}$ finishes its execution, *release jitter* resulting from the time difference between the *best-case* and the *worst-case response time* of $C_{k,1}$ has to be taken into account. Details can, for instance, be found in [YCH17] for the original DPCP. This jitter introduces pessimism into the WCRT analysis of the critical section and can be removed by enforcing the critical sections to be periodic. This can be achieved by *migrating* the task τ_k to the synchronization processor at the fixed time $t_{k,1}^{migr} \geq R_k(C_{k,1})$ after the task is released on the application processor. This does not necessarily mean that the task migration must be enforced to be periodic as well. It is sufficient if the critical section of a job arriving at time t_k^{arr} is only considered by the scheduler on the synchronization processor at time $t_k^{arr} + t_{k,1}^{migr}$. The same problems occur for the second non-critical sections and can again be solved by migrating back the task according to the WCRT of the critical section. We directly use $R_k(A_k)$ on the synchronization processor as suspension time, i.e., the task is migrated back after exactly $R_k(A_k)$ time units. Hence, due to the release enforcement for both migrations, there is no release jitter for $C_{k,2}$ as well.

release jitter
worst-case response time
task migration

This approach is also called *phase modification* (PM) in [BL92; SL96] and *static offset* in [GH98]. It differs from other enforcement strategies with similar names, i.e., the release guard by Sun and Liu [SL96], in which the release time of A_k ($C_{k,2}$, respectively) of task τ_k must be at least T_k time units apart from A_k ($C_{k,2}$, respectively) of the previous job of τ_k, and the period enforcer introduced by Rajkumar [Raj91], which has been shown incomparable with all existing analyses regarding suspension-based locking protocols by Chen and Brandenburg [CB17].

Under release enforcement, the timing analysis is equivalent to *end-to-end* deadline analysis, e.g., [BL92], or the *static-offset* static-priority uniprocessor analysis, e.g., [GH98]. However, our focus is on partitioning and scheduling the tasks. Hence, we apply existing safe timing analysis and scheduling algorithms based on the literature on self-suspension, detailed in Section 6.2.3. Other approaches to assign relative deadlines and validate $R_k(C_{k,1}) + R_k(A_k) + R_k(C_{k,2}) \leq T_k$ under release enforcement can be applied as well.

6.2.3 SCHEDULABILITY TESTS UNDER RELEASE ENFORCEMENT

For resource-oriented partitioned scheduling, the schedulability on each processor can be analyzed individually. While the actual mappings of tasks and resources is detailed in Section 6.2.4, this subsection focuses on the scheduling decisions for the individual processors and the schedulability tests. Therefore, we assume a mapping of shared resources and tasks onto processors to be given. We first examine the scheduling regarding critical sections and the resulting *worst-case response time* on a *synchronization processor*, determining the maximum suspension time S_k for τ_k regarding the application processors. Afterwards, we use this suspension time S_k to analyze the schedulability on the *application processor*. In the course of this, we set individual deadlines $D_{k,1}$ and $D_{k,2}$ for the first and second computation segments, respectively. Under the release enforcement, a constrained-deadline task τ_k is schedulable by a scheduling algorithm if

worst-case response time
synchronization processor
application processor

1. $D_{k,1} + S_k + D_{k,2} \leq T_k$,
2. $R_k(C_{k,1}) \leq D_{k,1}$,
3. $R_k(C_{k,2}) \leq D_{k,2}$, and
4. $R_k(A_k) \leq S_k$.

Since they in our approach are utilized for the preplanned migration, $D_{i,1}$, S_i, and $D_{i,2}$ are also set for static-priority scheduling. For the simplicity of presentation, we assume that task migration takes no time. Task τ_i migrates to its synchronization processor $D_{i,1}$ time units after a job of task τ_i arrives and migrates back to its application processor $S_i + D_{i,1}$ time units after a job of task τ_i arrives.

We first assume that each processor is either a synchronization processor or an application processor, and consider the case that the critical and the non-critical sections are scheduled on the same processor afterwards. We always examine the schedulability of task τ_k under the assumption that the schedulability of the tasks that are previously assigned on the same processor is already assured.

When looking at task τ_k, the set of tasks placed on the same synchronization processor as the critical section and the set of tasks placed on the same application processor as the non-critical section are not necessarily identical. Furthermore, the priority ordering of tasks on a synchronization processor is not necessarily the same as on an application processor. Hence, we introduce the following notation:

- for *critical sections*: $hps(\tau_k)$ denotes the tasks with higher priority than τ_k on the synchronization processor, i.e., under static-priority scheduling, and
- for *non-critical sections*: $hpa(\tau_k)$ denotes the tasks with higher priority than τ_k on the application processor if static-priority scheduling is used.

critical section
non-critical section

Note that we only consider dynamic-priority scheduling for non-critical sections while critical sections are always scheduling using static-priority scheduling.

CRITICAL SECTION RESPONSE TIME ANALYSIS

critical section
synchronization processor
Rate Monotonic
Time Demand Analysis
worst-case response

For each *synchronization processor*, the task priorities are assigned according to *Rate Monotonic*. Due to release enforcement, the inter-arrival time of a task τ_i on the synchronization processor is at least T_i. Hence, the extended *Time Demand Analysis* (TDA) in Eq. (4.1) can safely be used to test whether the *worst-case response* time of A_k is no more than D_k:

$$\exists t,\ 0 < t \leq T_k \text{ and } B_k + A_k + \sum_{\tau_i \in hps(\tau_k)} \left\lceil \frac{t}{T_i} \right\rceil A_i \leq t \tag{6.26}$$

If Eq. (6.26) holds for some values of t, we use the minimum $t^*_{k,s}$ among those values as the maximum suspension time S_k. The value of B_k is calculated depending on the resource sharing policy, i.e., Eq. (2.17) for NPP and Eq. (2.18) for PCP.

SCHEDULING ANALYSIS FOR NON-CRITICAL SECTIONS

non-critical section
application processor

To determine if τ_k is schedulable under the scheduling policy on the *application processor*, we validate whether $R_k(C_{k,1}) + S_k + R_k(C_{k,2}) \leq T_k$ after $S_k = R_k(A_k)$ was determined beforehand. We assign individual deadlines $D_{k,1} \geq R_k(C_{k,1})$ and $D_{k,2} \geq R_k(C_{k,2})$ for $C_{k,1}$ and $C_{k,2}$, respectively, with $D_{k,1} + S_k + D_{k,2} = D_k$. While those deadlines are not necessary when scheduling tasks under static-priority, they are used for both static- and dynamic-priority to determine the point in time where the migration takes place. To be precise, the migration to the synchronization processor happens at $\theta_a + D_{k,1}$ and the migration back to the application processor at $\theta_a + D_{k,1} + S_k$ where θ_a is the jobs arrival time. When considering τ_k, we assume that $D_{i,1}$ and $D_{i,2}$ are already assigned for all tasks τ_i that are already allocated to the processor. As shown in [HC16] for static-priority and in Section 6.1.2 for dynamic-priority, those tasks can be modeled as *general multiframe* (GMF) tasks [BCG+99] with two frames. According to the notation in Section 6.1.2, τ_i is represented by two 3-tuples $\tau_i = \{(C^1_i, D^1_i, T^1_i), (C^2_i, D^2_i, T^2_i)\}$, representing two alternately released subtasks. The computation time for the GMF subtasks is the same as for the computation segments, i.e., $C^1_i = C_{i,1}$ and $C^2_i = C_{i,2}$. As the second computation segment is released after the suspension interval, we know that $D^1_i = D_{i,1}$ and $T^1_i = D_{i,1} + S_i$. Moreover, $T^2_i = T_i - T^1_i = T_i - D_{i,1} - S_i$ and $D^2_i = D_{i,2} = T_i - D_{i,1} - S_i$.

generalized multiframe task

Rate Monotonic
execution interval monotonic

We consider two assignment orders, namely *Rate Monotonic* and *execution interval monotonic* (EIM) were tasks are ordered according to $T_i - S_i$ (see Section 6.1.4). These two orders are each combined with static-priority scheduling as well as dynamic priority scheduling according to SEIFDA as introduced in Section 6.1.

Static-Priority - FRD - Rate Monotonic (FP-RM): The task priorities on each application processor are assigned in RM order. We determine $R_k(C_{k,1}) = t^*_{k,1}$ and $R_k(C_{k,2}) = t^*_{k,2}$ as the minimum t such that

$$0 < t \leq T_i \ \text{ and } \ C_{k,1} + \sum_{\tau_i \in hpa(\tau_k)} W_i(t) \leq t \tag{6.27}$$

holds, where $W_i(t)$ is the maximum interference of τ_i over the interval $[0,t)$. Takada and Sakamura showed in [TS97] that $W_i(t)$ can be calculated as the maximum $\max\left\{E_i^h(t)\right\}$ of the interference patterns $E_i^h(t)$ for $h \in \{1,2\}$ where $E_i^h(t) = \sum_{j=h}^{h+l+1} C_i^{\{j \bmod 2\}}$ and l is the minimum integer with $\sum_{j=h}^{h+l+1} T_i^{\{j \bmod 2\}} \geq t$. Note that the interference from $hpa(\tau_k)$ is identical to that based on the static offset analysis in [GH98]. Similarly, $t_{k,2}^*$ is determined using $C_{k,2}$ in Eq. (6.27).

If $t_{k,1}^* + t_{k,2}^* < T_k - S_k$, the remaining slack can be freely distributed when assigning the deadlines $D_{k,1}$ and $D_{k,2}$. We use an equal density assignment with respect to $t_{k,1}^*$ and $t_{k,2}^*$. To be precise, we first consider $\frac{C_{k,1}}{D_{k,1}+S_k} = \frac{C_{k,2}}{D_{k,2}}$ and if $D_{k,1}$ is less than $t_{k,1}^*$, we set $D_{k,1} = t_{k,1}^*$ and adjust $D_{k,2}$ accordingly. Similar, we set $D_{k,2} = t_{k,2}^*$ if $D_{k,2} < t_{k,2}^*$.

Static-Priority - Execution Interval Monotonic (FP-EIM): The only difference to FP-RM is that tasks are considered and prioritized in increasing order according to their execution interval $T_i - S_i$.

Earliest Deadline First - Fixed Relative Deadlines - EIM (EDF-EIM): The first dynamic-priority approach we consider is to use SEIFDA as introduced in Section 6.1 on the individual processors.[2] For each task, individual relative deadlines are assigned for the two computation segments using the Proportionally-Bounded-Min approach and approximated DBFs as provided in Section 6.1.5. Afterwards, the subjobs are scheduled accordingly using EDF.

EDF - FRD - RM (EDF-RM): The only difference to EDF-EIM is that tasks are assigned in RM instead of EIM order.

IMPROVEMENTS FOR NON-CRITICAL SECTIONS

Allowing non-critical sections to be executed on the synchronization processors may increase the schedulability as it utilizes otherwise unused capacities. We apply static-priority scheduling for the non-critical sections, even if dynamic-priority scheduling is used on the processors that are only used for non-critical sections, and always assign each critical section with a higher priority than all non-critical sections on the same processor.

6.2.4 RESOURCE AND TASK ALLOCATION

Assume a given number of $m^R \geq 1$ synchronization processors and $m^C = m - m^R$ application processors. We proceed with the following three steps, which are a revised version of the algorithm by Huang et al. [HYC16] as different tests and scheduling algorithms are applied:

synchronization processor
Worst-Fit Decreasing

(1) **Assign tasks to synchronization processors**: The resources are assigned to the given number of synchronization processors using *Worst-Fit Decreasing* (WFD) based on the resource utilization, i.e., the resources are ordered non-increasingly according to $U^{\mathcal{R}_q}$.

2 The renaming to EDF-EIM is to match the terminology we use here, since we focus on the general strategies and not on the specific algorithms.

(2) **Calculate WCRT on synchronization processors**: For each task τ_k, we calculate $R_k(A_k)$ using Eq. (6.26), considering the m^R processors individually. The blocking time B_k is calculated according to either PCP (Eq. (2.18)) or NPP (Eq. (2.17)), depending on the synchronization protocol. The WCRT $R_k(A_k)$ is the suspension time S_k for each task τ_k.

application processor

(3) **Assign tasks to application processors**: The tasks are sorted in increasing order according to the scheduling approach for the application processors, i.e., either by $T_i - S_i$ if EDF-EIM or FP-EIM is used or according to T_i if EDF-RM or FP-RM is used. The non-critical sections of the tasks are assigned to the application processors according to the first-fit approach, using the related schedulability test. If the schedulability condition in Section 6.2.3 holds, the deadlines are set accordingly and the non-critical sections of the task are assigned to the processor. If a task is not schedulable on any application processor, we try to assign it to the synchronizations processors.

The key factor in this algorithm is the setting of m^R. One has to consider a general trade-off between 1) longer suspension times if m^R is small, and 2) longer worst-case response times of the non-critical sections if m^R is large. Since the above assignment algorithm works for any $m^R \leq m$, trying all possible settings of m^R increases the time complexity only by a factor of m. Hence, the algorithm is executed for all sensible numbers of synchronization processors, i.e., $m^R \in \{1, ..., min(m, r)\}$. We show that setting $m^R = \max\left\{\left\lfloor 6 \sum_{\tau_i \in \tau} U_i^A \right\rfloor, 1\right\}$ leads to a speedup factor of 6 for FP-RM together with PCP and release enforcement in the next section. However, we point out that an algorithm that sets $m^R = \max\left\{\left\lfloor 6 \sum_{\tau_i \in \tau} U_i^A \right\rfloor, 1\right\}$ greedily results in a *potential pitfall* with significant performance loss, since such a setting of m^R is an enforcement technique that is *too strong and applied at an early stage of the algorithm*, which is problematic as stated in Observation 6 in Section 4.5.3.

6.2.5 SPEEDUP FACTORS

speedup factor

This section shows that release enforcement, together with resource-oriented partitioned scheduling, leads to a *speedup factor* of 6. We start with the necessary scheduling conditions for a task set to be feasible by any multiprocessor scheduling algorithm, shown by Huang et al. in Lemma 3 in [HYC16].

Lemma 6.13 (Necessary Condition, Lemma 3 in [HYC16]). *Any implicit-deadline task system τ that is feasible upon a platform comprised of m processors must satisfy the following conditions*

$$U^C + U^{\mathcal{R}} \leq m \tag{6.28}$$

$$U_k \leq 1 \qquad \forall \tau_k \in \boldsymbol{T} \tag{6.29}$$

$$\max_{\tau_i \in ld_{\mathcal{R}_q}(\tau_k)} A_i + A_k + \sum_{\tau_j \in sd_{\mathcal{R}_q}(\tau_k)} \left\lfloor \frac{T_k}{T_j} \right\rfloor A_j \leq T_k \qquad \forall \tau_k \in \boldsymbol{T} \tag{6.30}$$

where $ld_{\mathcal{R}_q}(\tau_k)$ and $sd_{\mathcal{R}_q}(\tau_k)$ are the sets of tasks that access the same shared resource R as τ_k but with longer $(T_i > T_k)$ and shorter or the same $(T_i \leq T_k)$ periods, respectively.

Lemma 6.14. *Following the necessary condition in Eq. (6.30), the blocking time B_k of a task τ_k derived under resource-oriented partitioned scheduling with PCP must be upper bounded by its period T_k when the tasks are prioritized by using RM.*

Due to Eq. (6.30), $U^{\mathcal{R}_q} \leq 1$ must hold for each resource $\mathcal{R}_q \in \mathcal{R}$. The proof of Lemma 6.14 was provided by Huang et al. in [HYC16] as part of the proof of their Lemma 5. For the speedup analysis, we first provide the following lemma regarding the WCRT of a computation segment E_k if the utilization on the related processor is low enough.

Lemma 6.15. *Let $hp^*(E_k)$ be the periodically arriving computation segments with higher priority than E_k on the same processor. Suppose the WCRT analysis for a computation segment E_k is to find the minimum $t > 0$ where*

$$E_k + \sum_{\tau_i \in hp^*(E_k)} \left\lceil \frac{t}{T_i} \right\rceil E_i = t \tag{6.31}$$

If $T_i \leq T_k$ for all tasks $\tau_i \in hp^*(E_k)$ and

$$\left(\sum_{\tau_i \in hp^*(\tau_k)} \frac{E_i}{T_i} \right) + \frac{E_k}{T_k} = Y \leq 0.5 \tag{6.32}$$

then $R_k(E_k)$ under Eq. (6.31) is at most $T_k \cdot Y$.

Proof. Suppose V_i is $\frac{E_i}{T_i}$. We must consider two cases:

$$T_i \leq \frac{T_k}{2} \qquad \Rightarrow \left\lceil \frac{t}{T_i} \right\rceil E_i \leq T_k V_i \qquad \forall\, 0 < t \leq \frac{T_k}{2} \tag{6.33}$$

$$\frac{T_k}{2} < T_i \leq T_k \qquad \Rightarrow \left\lceil \frac{t}{T_i} \right\rceil E_i = E_i \leq T_k V_i \qquad \forall\, 0 < t \leq \frac{T_k}{2} \tag{6.34}$$

Therefore, we know that for all t with $0 < t \leq \frac{T_k}{2}$:

$$E_k + \sum_{\tau_i \in hp^*(E_k)} \left\lceil \frac{t}{T_i} \right\rceil E_i \leq E_k + \sum_{\tau_i \in hp^*(E_k)} V_i T_k = Y T_k \tag{6.35}$$

This means that Eq (6.31) holds when $t = Y T_k$. □

Note that we apply Lemma 6.15 for analyzing $R_k(C_{k,1})$, $S_k = R_k(A_k)$, and $R_k(C_{k,2})$ by putting different formulas in Eq. (6.31) and Eq. (6.32). For the simplicity of presentation in the following statements, we will implicitly assume that the task set can be feasibly scheduled on the original platform and therefore the necessary conditions in Lemmas 6.13 and 6.14 hold. Moreover, our goal here is to prove the schedulability in a specific setting, i.e., when the platform speed is 6. Hence, *for the remaining proofs in this section, all execution times, blocking times, utilization values, and analyses are based on the platform after speeding up by* 6.

Lemma 6.16. *If the two conditions that $S_k + T_k(U_k^C + 2\sum_{\tau_i \in hpa(\tau_k)} U_i^C) \leq T_k$ and that $U_k^C + \sum_{\tau_i \in hpa(\tau_k)} U_i^C \leq 0.5$ hold, then the WCET of task τ_k (under release enforcement, RM-P, and resource-oriented partitioned scheduling) is*

$$R_k(\tau_k) \leq S_k + T_k \left(U_k^C + 2 \sum_{\tau_i \in hpa(\tau_k)} U_i^C \right) \tag{6.36}$$

Proof. Under RM-P and release enforcement, $R_k(C_{k,1})$, i.e., the *offset* to release the critical section to its synchronization processor, is the minimum $t > 0$ such that Eq. (6.27) holds. Since $W_i(t)$ defined for Eq. (6.27) is less than or equal to $\left\lceil \frac{t}{T_i} \right\rceil (C_{i,1} + C_{i,2})$ for task τ_i, we can safely approximate $R_k(C_{k,1})$ as the minimum $t > 0$ with $C_{k,1} + \sum_{\tau_i \in hpa(\tau_k)} \left\lceil \frac{t}{T_i} \right\rceil C_i = t$. We can conclude from the assumption that $\frac{C_{k,1}}{T_k} + \sum_{\tau_i \in hpa(\tau_k)} U_i^C \leq U_k^C + \sum_{\tau_i \in hpa(\tau_k)} U_i^C \leq 0.5$, that Eq. (6.32) holds with $E_k = C_{k,1}$ for τ_k and $E_i = C_i$ for $\tau_i \in hpa(\tau_k)$. Hence, by applying Lemma 6.15,

$$R_k(C_{k,1}) \leq C_{k,1} + T_k \sum_{\tau_i \in hpa(\tau_k)} U_i^C = D_{k,1} \tag{6.37}$$

Similarly, $R_k(C_{k,2}) \leq C_{k,2} + T_k \sum_{\tau_i \in hpa(\tau_k)} U_i^C = D_{k,2}$. As a result,

$$R_k(\tau_k) \leq S_k + T_k(U_k^C + 2 \sum_{\tau_i \in hpa(\tau_k)} U_i^C)$$

if $S_k + T_k(U_k^C + 2\sum_{\tau_i \in hpa(\tau_k)} U_i^C) \leq T_k$. □

Lemma 6.17. *Under FP-RM-PCP with release enforcement on a platform with m homogeneous processors of speed 6 and the number of synchronization processors set to* $m^R = \max\left\{\left\lfloor 6 \sum_{\tau_i \in \tau} U_i^A \right\rfloor, 1\right\}$*, the maximum response time* S_k *of a task on a synchronization processor is at most*

$$S_k \leq \begin{cases} \left(\frac{1}{6} + \sum_{\tau_i \in \tau} U_i^A\right) T_k & \text{if } m^R = 1 \\ 0.5 T_k & \text{if } m^R \geq 2 \end{cases} \tag{6.38}$$

when the resources are packed according to the worst fit heuristic.

Proof. By Lemma 6.14, when RM is used together with PCP for scheduling, we know that $B_k/T_k \leq 1/6$ after speeding up.

When m^R is 1, we know that $\sum_{\tau_i \in \tau} U_i^A < 1/3$ and all critical sections in $\mathbf{T}$ are assigned to one processor. Due to the release enforcement, the WCRT S_k of the critical section of task τ_k is the minimum t such that Eq. (6.26) holds, i.e., $B_k + A_k + \sum_{\tau_i \in hps(\tau_k)} \left\lceil \frac{t}{T_i} \right\rceil A_i = t$. Since $A_k/T_k + \sum_{\tau_i \in hps(\tau_k)} U_i^A \leq \sum_{\tau_i \in \tau} U_i^A \leq 1/3$ and $B_k/T_k \leq 1/6$, we know that $Y = \frac{B_k + A_k}{T_k} + \sum_{\tau_i \in hps(\tau_k)} U_i^A \leq 0.5$, i.e., the condition in Eq. (6.32) holds when RM is used for prioritizing the critical sections, $E_k = A_k + B_k$ for task τ_k, and $E_i = A_i$ for task τ_i in $hps(\tau_k)$. Hence, $S_k \leq Y T_k \leq (1/6 + \sum_{\tau_i \in \tau} U_i^A) T_k$ when m^R is 1 due to Lemma 6.15.

For the rest of the proof, we focus on $m^R \geq 2$. We only need to prove that the total resource utilization of the critical sections on any of the m^R synchronization processors is $\leq 1/3$. In this case, since $\frac{B_k + A_k}{T_k} + \sum_{\tau_i \in hps(\tau_k)} U_i^A \leq 1/6 + 1/3 = 0.5$, the same response time analysis used above, when m^R is 1, can directly be applied to conclude $S_k \leq 0.5 T_k$. We first consider how the resources are packed to the m^R synchronization processors at the platform with a speed of 6. Suppose that we are now assigning the q-th resource $\mathcal{R}_q$. By definition, $q \leq r$. Let $U^{\wp_\ell}$ denote the resource utilization on a synchronization processor $\wp_\ell$. Before assigning R^q to any of the m^R synchronization processors, there is one synchronization processor

with the minimum utilization so far, denoted as $\wp_j$. Due to the *worst-fit* strategy, $U^{\wp_\ell} \geq U^{\wp_j}$ for any synchronization processor $\wp_\ell$.

We show that the utilization of the resources assigned to $\wp_j$ (after assigning $\mathcal{R}_q$ to $\wp_j$) is always $\leq \frac{2}{6} = \frac{1}{3}$, i.e., $U^{\wp_j} + U^{\mathcal{R}_q} \leq \frac{1}{3}$, for a platform with speed 6. Assume for contradiction that $U^{\wp_j} + U^{\mathcal{R}_q} > \frac{1}{3}$. Hence, $U^{\wp_\ell} + U^{\mathcal{R}_q} > \frac{1}{3}$, i.e., $U^{\wp_\ell} > \frac{1}{3} - U^{\mathcal{R}_q}$ for any synchronization processor $\wp_\ell \in m^R$. Combining the above information, we get

$$\begin{aligned}
\sum_{i=1}^{q} U^{\mathcal{R}_i} &= U^{\mathcal{R}_q} + \sum_{i=1}^{q-1} U^{\mathcal{R}_i} = U^{\mathcal{R}_q} + \sum_{\ell} U^{\wp_\ell} \\
&> U^{\mathcal{R}_q} + m^R \left(\frac{1}{3} - U^{\mathcal{R}_q}\right) = \frac{1}{3} + (m^R - 1)\left(\frac{1}{3} - U^{\mathcal{R}_q}\right) \\
&\overset{\dagger}{\geq} \frac{1}{3} + (m^R - 1)\frac{1}{6} = \frac{m^R + 1}{6} \\
&= \frac{1}{6} \times \left(\left\lfloor 6 \sum_{\tau_i \in \tau} U_i^A \right\rfloor + 1\right) \overset{*}{>} \sum_{\tau_i \in \tau} U_i^A = \sum_{i=1}^{r} U^{\mathcal{R}_i} \geq \sum_{i=1}^{q} U^{\mathcal{R}_i}
\end{aligned}$$

where $\overset{\dagger}{\geq}$ is due to $m^R \geq 2$ and $U^{\mathcal{R}_q} \leq 1/6$ as the platform speed is 6, and $\overset{*}{>}$ is due to the fact $\lfloor x \rfloor > x - 1$. Therefore, we reach a contradiction. As a result, the total resource utilization of the critical sections on any of the m^R synchronization processors is $\leq 1/3$, and $S_k \leq 0.5T_k$ for any task τ_k when $m^R \geq 2$. □

Based on these results, we can now prove a speedup factor of 6.

Theorem 6.18. *The speedup factor of the proposed resource-oriented partitioned scheduling algorithm is 6 if the Priority Ceiling Protocol is used to schedule the critical sections on the synchronization processors when $m \geq 2$, the worst-fit approach is used to assign the critical sections to the synchronization processors, and the non-critical sections are assigned in rate-monotonic order.*

Proof. Suppose that the input task set $\mathbf{T}$ can be feasibly scheduled on m uni-speed processors. We show that in this case the task set is also schedulable by the *resource-oriented partitioned scheduling* on m processors with speed $s = 6$. We assume a special setting of m^R with $m^R = \max\left\{\left\lfloor 6 \sum_{\tau_i \in \tau} U_i^A \right\rfloor, 1\right\}$ and $m^C = m - m^R$ in the analysis. We need to show that $R_k(\tau_k) \leq T_k$ for any task τ_k in $\mathbf{T}$. Since RM is used for the priority assignment on the synchronization processors, and the tasks are assigned to the application processors in RM order, Lemma 6.16 and Lemma 6.17 can be implicitly applied if the required utilization condition can be satisfied. Two cases have to be considered:

Case 1: $\mathbf{m^R \geq 2}$. That is, $\sum_{\tau_i \in \tau} U_i^A \geq \frac{2}{6}$. Moreover, the necessary condition in Eq. (6.28) leads to the following inequality after speeding up with a factor of 6:

$$\sum_{\tau_i \in \tau} (6U_i^C + 6U_i^A) \leq m \Rightarrow m^R + \sum_{\tau_i \in \tau} 6U_i^C \leq m^C + m^R \Rightarrow \sum_{\tau_i \in \tau} U_i^C \leq \frac{m^C}{6}$$

Hence, when we assign task τ_k to an application processor, there must be an application processor with utilization $\leq \frac{1}{6}$ due to the pigeon hole principle. Let

this processor be $\wp_j$ and let the set of the tasks that are already assigned on this processor be $hpa^j(\tau_k)$. Therefore, we know that $\sum_{\tau_i \in hpa^j(\tau_k)} U_i^C \leq 1/6$ and $U_k^C + \sum_{\tau_i \in hpa^j(\tau_k)} U_i^C \leq 1/3$. By Lemma 6.17, $S_k \leq 0.5T_k$, and by Lemma 6.16, we know that

$$R_k(\tau_k) \leq S_k + \Big(U_k^C + 2 \sum_{\tau \in hpa^j(\tau_k)} U_i^C\Big) T_k \leq T_k \tag{6.39}$$

Case 2: $\mathbf{m^R = 1}$. This means that $\sum_{\tau_i \in \tau} U_i^A < \frac{2}{6}$. From Lemma 6.17, we know $S_k \leq (1/6 + \sum_{\tau_i \in \tau} U_i^A) T_k$. We consider two subcases 1) $m = 2$ and 2) $m \geq 3$. When m is 2, one processor is used for synchronization and another processor is used for non-critical sections. The necessary condition in Eq. (6.28) after speeding up with a factor of 6 leads to:

$$U_k^C + \sum_{\tau_i \in hpa(\tau_k)} U_i^C \leq \sum_{\tau_i \in \tau} U_i^C \leq \frac{m}{6} - \sum_{\tau_i \in \tau} U_i^A = \frac{1}{3} - \sum_{\tau_i \in \tau} U_i^A \tag{6.40}$$

Therefore, using Lemma 6.16 due to $U_k^C + \sum_{\tau_i \in hpa(\tau_k)} U_i^C \leq 1/3 < 0.5$ when $m = 2$, results in

$$\begin{aligned} R_k(\tau_k) \leq & S_k + \Big(U_k^C + 2 \sum_{\tau \in hpa(\tau_k)} U_i^C\Big) T_k \\ \leq & \Big(\frac{1}{6} + \sum_{\tau_i \in \tau} U_i^A + \frac{2}{3} - 2 \sum_{\tau_i \in \tau} U_i^A\Big) T_k \leq T_k \end{aligned} \tag{6.41}$$

When m is at least 3, due to the pigeon hole principle, there exists an application processor $\wp_j$ in the $m^C = m - 1$ application processors with utilization less than or equal to $(\sum_{i=1}^{k-1} U_i^C)/(m-1) \leq (-U_k^C + \sum_{\tau_i \in \tau} U_i^C)/(m-1)$ before assigning τ_k. Let such a processor be $\wp_j$ and the set of the tasks that are already assigned on this processor be $hpa^j(\tau_k)$. Hence,

$$\begin{aligned} & U_k^C + 2 \sum_{\tau_i \in hpa^j(\tau_k)} U_i^C \leq U_k^C + \frac{-2U_k^C + \sum_{\tau_i \in \tau} 2U_i^C}{m-1} \\ \overset{*}{\leq} & \; U_k^C \left(1 - \frac{2}{m-1}\right) + \frac{\frac{2m}{6} - 2\sum_{\tau_i \in \tau} U_i^A}{m-1} \overset{\dagger}{\leq} \frac{1}{2} - \frac{2\sum_{\tau_i \in \tau} U_i^A}{m-1} \end{aligned} \tag{6.42}$$

where $\overset{*}{\leq}$ is due to $\sum_{\tau_i \in \tau} U_i^C + U_i^A \leq \frac{m}{6}$ after speeding up and $\overset{\dagger}{\leq}$ is due to $0 < U_k^C \leq \frac{1}{6}$ and $m \geq 3$. Similarly,

$$\begin{aligned} & U_k^C + \sum_{\tau_i \in hpa^j(\tau_k)} U_i^C \leq U_k^C + \frac{-U_k^C + \sum_{\tau_i \in \tau} U_i^C}{m-1} \\ \leq & \; U_k^C \left(1 - \frac{1}{m-1}\right) + \frac{\frac{m}{6} - \sum_{\tau_i \in \tau} U_i^A}{m-1} \overset{\ddagger}{\leq} \frac{4}{12} = \frac{1}{3} < 0.5 \end{aligned} \tag{6.43}$$

where $\overset{\ddagger}{\leq}$ is due to the fact that the function is monotonically decreasing with respect to m. Therefore, when using Lemma 6.16 due to $U_k^C + \sum_{\tau_i \in hpa^j(\tau_k)} U_i^C < 0.5$ in Eq. (6.43), the condition in Eq. (6.42) for $m \geq 3$ leads to

$$\begin{aligned} & R_k(\tau_k) \leq S_k + \Big(U_k^C + 2 \sum_{\tau \in hpa^j(\tau_k)} U_i^C\Big) T_k \\ \leq & \Big(\frac{1}{6} + \sum_{\tau_i \in \tau} U_i^A + \frac{1}{2} - \frac{2\sum_{\tau_i \in \tau} U_i^A}{m-1}\Big) T_k \\ \leq & \Big(\frac{2}{3} + \frac{1}{m-1}\Big((m-3) \sum_{\tau_i \in \tau} U_i^A\Big)\Big) T_k \\ \leq & \Big(\frac{2}{3} + \frac{(m-3)\frac{2}{6}}{m-1}\Big) T_k \leq T_k \end{aligned} \tag{6.44}$$

As a result, at a speed of 6, we can always find an application processor to assign task τ_k so that it meets its deadline. □

6.2.6 EVALUATION

We conducted evaluations for $m = 4$, 8, and 16 processors. Depending on m, we generate 100 task sets for each utilization level, from $5\% \cdot m$ to $100\% \cdot m$, in steps of $5\% \cdot m$. The cardinality of each task set is $10 \times m$. The distribution of periods is within one order of magnitude, i.e., from 1ms to 10ms. All tasks have implicit deadlines, i.e., $D_i = T_i$. We applied the suggestion of Emberson et al. [ESD10] and generated the task periods according to a *log-uniform distribution*. The overall ratio of non-critical to critical-sections depends on $\alpha \in \{5, 10, 20\}$. For example, if $\alpha = 5$ and $U_{sum} = 120\%$, we get $U^{\mathcal{R}} = 120\% \times \frac{1}{5+1} = 20\%$ and $U^C = 120\% \times \frac{5}{5+1} = 100\%$. Therefore, the larger α is, the smaller is the critical section. In each utilization step, the `Randfixedsum` method [ESD10] is adopted twice to generate two sets of utilization values with the given goals of total critical-sections utilization and total non-critical-sections utilization. Those values are combined ensuring that $U_i^A + U_i^C \leq 1$ for every task τ_i. The WCETs of the non-critical sections and critical section of task τ_i are set accordingly, i.e., $C_i = T_i U_i^C$ and $A_i = T_i U_i^A$, and $C_{i,1}$ is drawn uniformly from $[0, C_i]$, setting $C_{i,2} = C_i - C_{i,1}$. Each critical section was randomly assigned to one of the r resources under a uniform distribution.

Multiple resource sharing protocols were compared based on the *acceptance ratio*. We evaluated the following approaches, using the RM order and priority assignment if not mentioned otherwise, where the color and linestyle are related to the curve in Figure 6.8.

- LP-GFP-FMLP [BLB+07] (black, dashed): a linear-programming-based (LP) analysis for global static-priority scheduling using the Flexible Multiprocessor Locking Protocol (FMLP) [BLB+07].
- LP-PFP-DPCP [Bra13] (red, dashed): LP based analysis for partitioned static-priority scheduling and DPCP [RSL88]. Tasks are assigned using WFD as proposed in [Bra13].

- LP-PFP-MPCP [Bra13] (magenta, dashed): LP based analysis for partitioned static-priority scheduling using MPCP [Raj90]. Tasks are partitioned according to WFD as proposed in [Bra13].
- GS-MSRP (blue, dashed) [WB13b]: the Greedy Slacker (GS) partitioning heuristic with the spin-based locking protocol MSRP [GLN01] under Audsley's Optimal Priority Assignment.
- LP-EE-vpr (NC) [AR14] (cyan, dashed): a necessary scheduling condition for LP-EE-vpr.
- gEDF-vpr (NC) [AE10] (green, dashed): a necessary scheduling condition for gEDF-vpr.
- LP-GFP-PIP (cyan, solid): LP based global static-priority scheduling using the Priority Inheritance Protocol (PIP) [EA09].
- MrsP (magenta, solid): the Multiprocessor resource sharing Protocol [BW13] with the Synchronization-Aware Partitioning Algorithm [LNR09].
- ROP-PCP (black, solid): the ROP in [HYC16] using PCP.
- FP-RM-PCP (blue, solid): the proposed ROP with RM assignment and PCP on each synchronization processor.
- FP-EIM-PCP (red, solid): the proposed ROP with static-priority EIM assignment and PCP on each synchronization processor.
- EDF-EIM-PCP (green, solid): the proposed ROP with SEIFDA FRD deadline assignment in EIM order, scheduled under EDF on the application processors, and PCP under RM on each synchronization processor.

We evaluated our proposed ROP approaches in all 8 combinations, i.e., FP or EDF, RM or EIM, and PCP or NPP. We only present the FP and the EDF approach that (in general) leads to the best performance, i.e., FP-EIM-PCP and EDF-EIM-PCP, as well as FP-RM-PCP since it provides a speedup factor of 6. In all cases, the approaches using the PCP and NPP performed similarly. For our approaches and ROP-PCP, we used approximated demand bound functions, where the linear approximation starts from the third period (see Section 6.1.8).

The results of our evaluations are shown in Figure 6.8. We analyzed the effect of the three parameters individually by changing:

1. $m = r \in \{4, 8, 16\}$ in Fig. 6.8 (a)-(c),
2. r for a fixed m, i.e., $r \in \{4, 8, 16\}$ and $m = 8$, in Fig. 6.8 (c)-(e), and
3. $\alpha \in \{5, 10, 20\}$ in Fig. 6.8 (c),(f),(g).

In general, ROP-RM-PCP (black, solid) is outperformed by both FP-EIM-PCP (red, solid) and EDF-EIM-PCP (green, solid). While EDF-EIM-PCP clearly outperforms FP-EIM-PCP for most settings, there are some settings where FP-EIM-PCP and EDF-EIM-PCP are really close and there are even cases where RM-EIM-PCP deems more task sets schedulable than EDF-EIM-PCP. LP-GFP-PIP (cyan, solid), FP-RM-PCP (blue, solid), and LP-GFP-FMLP (black, dashed) generally behave similarly and mostly outperform all other approaches beside ROP-RM-PCP, FP-EIM-PCP, and EDF-EIM-PCP. This is not surprising, as LP-GFP-FMLP and

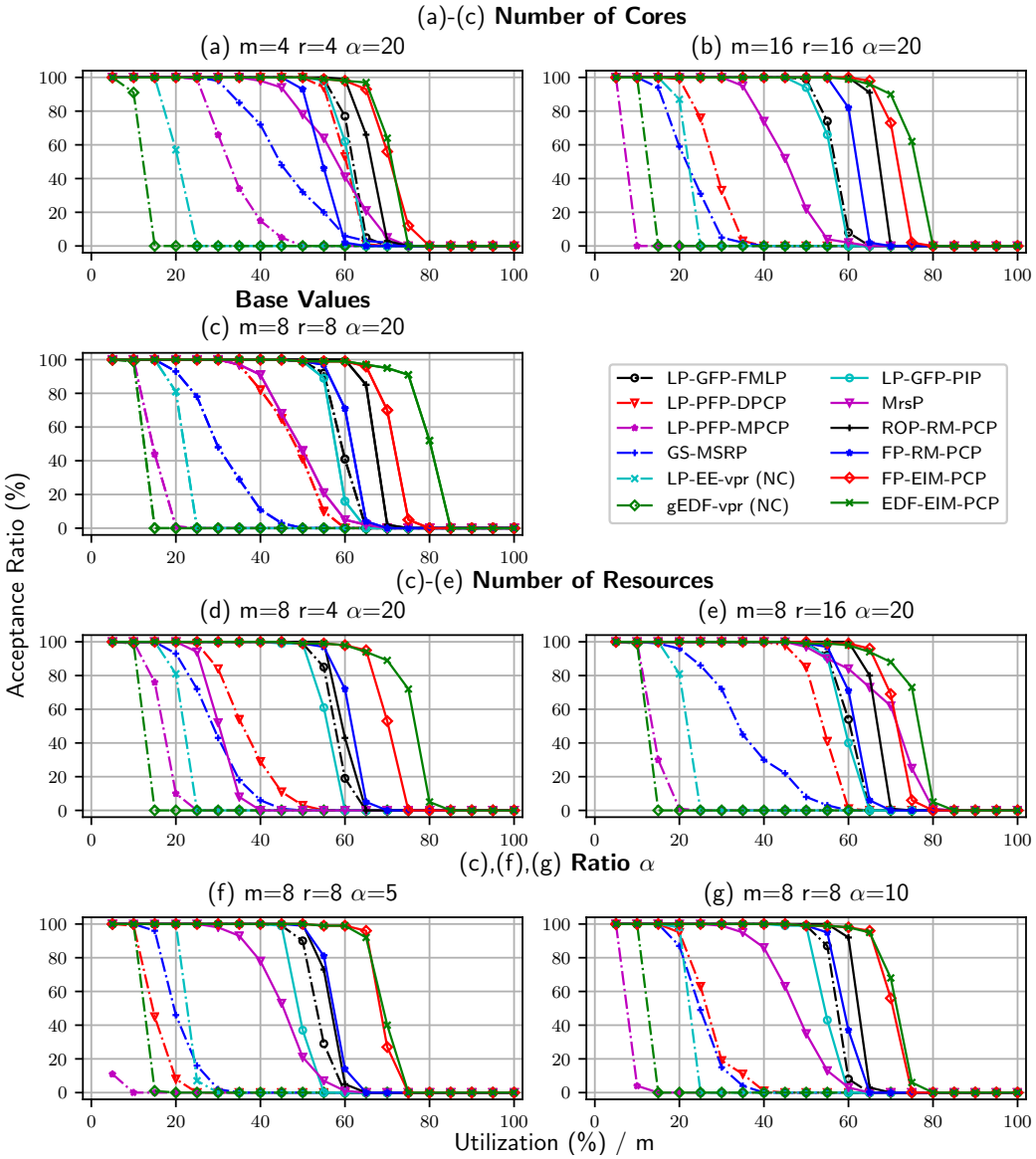


Figure 6.8: Comparison of different approaches under different parameter settings.

LP-GFP-PIP are the best locking protocols under global scheduling, according to the empirical study by Yang et al. in [YWB15]. RM-EIM-PCP outperforms FP-RM-PCP due to the fact that tasks with shorter execution intervals are normally harder to schedule than tasks with longer execution intervals, and the execution interval does not necessarily increase with the period. MrsP (magenta, solid) and LP-PFP-DPCP (red, dashed) have a very wide range regarding their acceptance ratio and no general trend can be determined. LP-PFP-MPCP (magenta, dashed), LP-EE-vpr (cyan, dashed), gEDF-vpr (green, dashed), and GS-MSRP (blue, dashed) are clearly outperformed and therefore not further discussed.

Fig. 6.8 (a)-(c), $\mathbf{m = r \in \{4, 8, 16\}}$: For LP-GFP-PIP, LP-GPF-FMLP, ROP-RM-PCP, and FP-EIM-PCP the number of processors m does not have much impact. MrsP and LP-PFP-DPCP perform better if $m = 4$. While MrsP has similar performance for $m = 8, 16$, the acceptance ratio of LP-PFP-DPCP drops significantly for $m = 16$. EDF-EIM-PCP performs comparable with FP-EIM-PCP for $m = 4$ but has a better

acceptance ratio for $m = 16$ while the gap is even larger for $m = 8$. One possible explanation is that EDF-EIM-PCP only performs better than FP-EIM-PCP on the application processors, and that when $m = 4$ the schedulability is dominated by the critical sections where EDF-EIM-PCP and FP-EIM-PCP perform the same, since both approaches use PCP under RM on the synchronization processors.

Fig. 6.8 (c)-(e), $\mathbf{r} \in \{4, 8, 16\}$ and $\mathbf{m} = 8$: The ratio of r to m seems to not have much effect on LP-GFP-PIP, LP-GFP-FMLP, and FP-EIM-PCP while ROP-RM-PCP performs worse for $r = 4$ and $m = 8$. The acceptance ratio of EDF-EIM-PCP is similar for $r = 4$ and $r = 16$ and better for $r = 8$. The most interesting part is the acceptance ratio of MrsP, which is worse than LP-GFP-PIP for $r = 4$ and $r = 8$ but performs almost as good as EDF-EIM-PCP for $r = 16$. Moreover, LP-PFP-DPCP performs better if the number of resources is larger.

Fig. 6.8 (c),(f),(g), $\alpha \in \{5, 10, 20\}$: A higher value of α, and therefore a smaller percentage of critical section utilization, results in a larger acceptance ratio. FP-EIM-PCP and EDF-EIM-PCP perform similarly for $\alpha = 5$ and $\alpha = 10$, while for $\alpha = 20$, EDF-EIM-PCP clearly outperforms FP-EIM-PCP. The most likely reason is that EDF-EIM-PCP only performs better than FP-EIM-PCP on the application processors while the performance on the synchronization processors is the same, since in both cases PCP under RM is used, and that when the critical section utilization is high, i.e., for $\alpha = 5$ in Fig. 6.8(f) and $\alpha = 10$ in Fig. 6.8(g), the critical section has an even higher impact on the schedulability.

6.2.7 MULTIPLE CRITICAL SECTIONS

While task sets where each task has only one critical section are rare, this restricted scenario allows to evaluate the applicability of the concept and to show its possible gain. The general concept of release enforcement can directly be extended to multiple sections. The worst-case response time of the non-critical sections on the application processors can be determined by modelling higher-priority tasks with multiple critical sections as generalized multiframe tasks. However, for more than two non-critical sections a good assignment of the relative deadlines on the application processors remains an open problem. One possibility is to use proportional relative deadline assignment for tasks with multiple critical sections, especially when the majority of tasks have one critical section where SEIFDA can be applied. One practical scenario where all tasks have only one critical section are Open-MP task sets, where synchronization among tasks is always performed at the end of a task [SGW+17].

6.3 HYBRID SELF-SUSPENSION MODELS

When considering self-suspension behaviour, two concrete models have been studied in the literature: the *dynamic* and the *segmented* self-suspension (sporadic) task model. In the dynamic self-suspension model, a task τ_i is specified like an ordinary sporadic task that has the worst-case self-suspension time S_i as an additional parameter. A job of task τ_i can suspend itself at any moment, several times

if necessary, before it finishes, as long as the total self-suspension time of the job is not more than S_i. By contrast, the *segmented* self-suspension model defines an interleaved execution and self-suspension pattern $(C_{i,1}, S_{i,1}, C_{i,2}, S_{i,2}, ..., S_{i,m_i}, C_{i,m_i+1})$ for any job of a task τ_i, which is composed of $m_i + 1$ computation segments that are separated by m_i suspension intervals, where $C_{i,j}$ is the worst-case execution time of a computation segment, and $S_{i,j}$ is the maximum length of a self-suspension interval. These two models are applicable in different scenarios with a high tradeoff between *flexibility* and *accuracy*:

dynamic self-suspension

- The *dynamic self-suspension* model only requires limited information about the suspension behavior. It has a higher flexibility but is very imprecise, which results in pessimistic analyses and designs of scheduling policies if more information regarding the suspension behaviour is known.

segmented self-suspension

- The *segmented self-suspension* model has a lower flexibility, but allows the scheduling algorithms to exploit the self-suspending structure, possibly resulting in a better scheduling decision and a more precise analysis. However, such a concrete segmented pattern is only achievable if the structure of the program is well designed and the execution pattern is determinable.

To summarize, the dynamic self-suspension model is very flexible but inaccurate, while the segmented self-suspension model is very restrictive but very accurate. This shows that there is a large gap between these two widely-adopted self-suspension task models. Hence, we propose several *hybrid self-suspension* task models which can potentially fill this gap. They are more flexible than the segmented self-suspension task model and less pessimistic than the dynamic self-suspension task model, therefore achieving different levels of tradeoff between flexibility and precision. Compared to the dynamic self-suspension model, all hybrid models have an additional parameter m_i that predefines the number of self-suspension intervals. However, instead of assuming one concrete execution/suspension pattern, a task is seen as a set of (potentially unknown) possible execution/suspension patterns. They provide several options to model the tasks, depending on whether the execution/suspension pattern of a job is known when it arrives to the system:

hybrid self-suspension

- *Pattern-oblivious Models*: The concrete execution pattern of a job is unknown at runtime. This model only assumes that the number of self-suspension intervals of a job of task τ_i is at most m_i. However, all possible execution paths may be known offline. We specifically explore two cases:
  - *Individual Upper Bounds:* While the specific individual execution paths are unknown, an upper bound can be determined on the WCET time for each of the computation segmented and on the maximum suspension time of each suspension interval.
  - *Multiple Paths:* Each task τ_i is specified by a set of p execution/suspension patterns, which describe the possible execution paths.
- *Pattern-clairvoyant Model*: The individual execution/suspension pattern of each job is of τ_i is known the moment the job arrives.

The individual models are introduced in Section 6.3.1. We show how these models can be applied by carefully examining the special case that each task has only one

self-suspension interval, i.e., $m_i = 1$. The applicability of FRD and extensions of demand bound functions for the different hybrid self-suspension task models are exemplified based on SEIFDA in Sections 6.3.2 to 6.3.4. Afterwards, we formalize the schedulability test and examine the demand bound functions of the individual hybrid models in Section 6.3.5. Our approaches are shown to be effective in terms of schedulability in the evaluation in Section 6.3.6. The evaluation shows that, compared to the dynamic self-suspension task model and the segmented self-suspension task model (that enforces the execution upper bounds on the computation segments), the hybrid self-suspension task models can achieve different degrees of improvement, depending on the knowledge about the execution/suspension patterns. The results presented in this section appeared in *Hybrid Self-Suspension Models in Real-Time Embedded Systems* in RTCSA 2017 *[BHC17]*.

6.3.1 HYBRID SELF-SUSPENSION TASK MODELS

hybrid self-suspension

In the *hybrid self-suspension task models*, we assume that in addition to S_i, the number of self-suspension intervals m_i is known for each task. Hence, the execution of each job of τ_i is composed of at most $m_i + 1$ *computation segments* separated by m_i *suspension intervals*, similar to the segmented self-suspension model. The summation of the execution times of the computation segments of a job of task τ_i is at most C_i, while the summation of the lengths of the self-suspension intervals of a job of task τ_i is at most S_i. All these values are positive for self-suspending tasks. The proposed hybrid models are:

dynamic self-suspension

- *more precise* and less flexible than the traditional *dynamic self-suspension* task model, where m_i is not considered, and

segmented self-suspension

- *more flexible and less precise* than the traditional *segmented self-suspension* task model, where the WCET of each of the $m_i + 1$ computation segments and the maximum suspension time for each of the m_i suspension intervals is assumed to be given by a fixed value.

In our description, we assume that the tasks will be scheduled by an FRD scheduling strategy as detailed in Section 6.1.1. We do not assume that each task in the task set must be a self-suspending task. If a task has no self-suspension behavior, i.e., S_i and m_i are both 0, such an ordinary (non-suspending) sporadic task should still be scheduled by using its original deadline. Hence, for the simplicity of presentation, we do not consider these tasks here. Although we focus on one self-suspension interval in our analyses, Huang and Chen [HC16] showed that FRD is a valid approach for multiple self-suspension intervals.

For the hybrid self-suspension task models to be applicable, we assume that each task can generally be described by a set of p disjunct execution/suspension patterns similar to the patterns used in the segmented self-suspension model. At runtime, each job is executed according to one of these specific patterns. Hence, different jobs of a task may have different execution/suspension patterns.

The proposed hybrid models provide several options depending on

1. the number of possible execution/suspension patterns,
2. the information that can be derived for each of these patterns, and

3. whether the execution pattern of a job can be determined at the moment a job arrives to the system.

Suppose that a job of task τ_i is released at time θ_a. If the (high-level) execution/suspension pattern of the job cannot be identified at time θ_a, we call the scenario *pattern-oblivious*. If the pattern can be identified, e.g., by checking (some of) the known input values at the moment a job arrives (potentially with approximations), we call the scenario *pattern-clairvoyant*. Clearly, pattern-clairvoyant approaches only work if the overhead for the identification is neglectable. We consider the following scenarios:

pattern-oblivious

pattern-clairvoyant

- *Pattern-oblivious*: Depending on the knowledge on the execution times of the computation segments and the self-suspension time of the suspension intervals, we analyze the following two subcases:
  - *Individual Upper Bounds*: We assume the upper bounds on the execution time of the *j-th* computation segment to be known, i.e., $C_{i,j}$ is no more than the individually specified $C_{i,j}^{max}$ for each $j = 1, 2, \ldots, m_i + 1$. In addition, the suspension of a job of task τ_i takes place at most m_i times and the j-th suspension is for at most $S_{i,j}^{max}$ amount of time for each $j = 1, 2, \ldots, m_i$. Moreover, we assume that the WCET of task τ_i is at most C_i^{max} while the maximum suspension time is at most S_i^{max}. Specifically, according to this definition $\sum_{j=1}^{m_i+1} C_{i,j}^{max} \geq C_i^{max}$ and $\sum_{j=1}^{m_i} S_{i,j}^{max} \geq S_i$. This scenario covers a special case where $C_{i,j}^{max}$ is set to C_i^{max} for each j, i.e., no specific information about the segments is known. This approach is directly applicable if the individual bounds S_i^{max}, C_i^{max}, and m_i are known, but further information about the internal structure of the task is not available.
  - *Multiple Paths*: A task τ_i is directly described by the p different execution paths with known execution/suspension patterns, in which a task τ_i can suspend at most m_i times. In this case, all possible paths are precisely known, i.e., as precise as in the segmented model, but the system is not able to identify which path will be executed at the moment a job arrives. For the special case $m_i = 1$ this results in a set of p triples: $\left\{(C_{i,1}^1, S_i^1, C_{i,2}^1), \ldots, (C_{i,1}^p, S_i^p, C_{i,2}^p)\right\}$.
- *Pattern-clairvoyant*: Each job of τ_i has an individual execution/suspension pattern and the pattern that will be executed is known when the job arrives. We assume such an identification takes negligible time.[3] This is more precise than the two models above. If all the jobs of task τ_i have the same pattern, then the model becomes the segmented self-suspension task model.

Table 6.3 provides a summary of the flexibility and the accuracy of different self-suspension task models.

To examine the hybrid models more carefully, we explain how FRD strategies, namely SEIFDA, can be extended from the segmented self-suspension model to hybrid self-suspension task models when each task has at most one suspension

3 It is also possible to include it into the first computation segment in all the paths. However, our analysis has to be revised and adjusted to give the identification process the highest priority.

suspension model	flexibility	accuracy
dynamic	very flexible (high)	inaccurate (low), *over flexible*
pattern-oblivious (hybrid)	less flexible than dynamic (medium to high)	applicable in most cases for known m_i (low to medium)
pattern-clairvoyant (hybrid)	less flexible than pattern-oblivious (medium to low)	more accurate than pattern-oblivious (medium to high)
segmented	very restrictive (low)	only applicable for fixed patterns (high), *over restrictive*

Table 6.3: High-level comparison of the dynamic, hybrid, and segmented self-suspension model. *Adapted from [BHC17].*

					Hybrid Self-Suspension Model								
Given Task Parameters					IUB		MP		SSSD		PDAB, Bias 2		
$T_i = D_i = 30$	$C_{i,1}$	$C_{i,2}$	C_i	S_i	$D_{i,1}$	$D_{i,2}$	$D_{i,1}$	$D_{i,2}$	$D_{i,1}$	$D_{i,2}$	Ratio	$D_{i,1}$	$D_{i,2}$
τ_i^1	2	3	5	5	8	14	8	17	8	17	10/15	12	13
τ_i^2	4	3	7	8				14	14	8	12.6/9.4	11	11
τ_i^3	2	7	9	7				15	8	15	5.1/17.9	7.1	15.9
max	4	7	9	8									

Table 6.4: Example deadline assignments under FRD for the hybrid self-suspension models as presented in Sec. 6.3.2. *Adapted from [BHC17].*

interval, by considering multiple execution/suspension patterns instead of a single execution/suspension pattern. This examination shows how determinable knowledge about $C_{i,1}$, $C_{i,2}$, and S_i, can be included into the algorithm and its analysis. We first consider different FRD strategies for the pattern-oblivious scenarios, presented in Section 6.3.2 and Section 6.3.3. After that, pattern-clairvoyant scenarios are discussed in Section 6.3.4.

As a running example, we use the task τ_i detailed in Table 6.4 with three execution patterns (paths). The execution patterns are denoted τ_i^1, τ_i^2, and τ_i^3. While the period $T_i = 30$ is identical for all execution patterns, $C_{i,1}$, $C_{i,2}$, $C_i = C_{i,1} + C_{i,2}$, and S_i depend on the specific execution pattern. We assume that for each job of task τ_i one of these three execution patterns is executed.

6.3.2 PATTERN-OBLIVIOUS: INDIVIDUAL UPPER BOUNDS

We assume to know individual upper bounds (IUB) of the execution time for computation segment, i.e., $C_{i,j}^p \leq C_{i,j}^{max}$ for each execution pattern p with $j \in \{1,2\}$, and the maximum suspension time[4] $S_i^{max} = \max\{S_i^p\}$. Let the maximum total WCET among all patterns be $C_i^{max} = \max\left\{C_{i,1}^p + C_{i,2}^p \mid \text{p is a possible execution pattern}\right\}$. Note that to apply this (most basic) hybrid model, no explicit knowledge about the individual execution/suspension patterns is needed, as long as C_i^{max}, $C_{i,1}^{max}$, $C_{i,2}^{max}$, and S_i^{max} can be determined.

We construct the two resulting DBFs for the case where $C_{i,1}$ is released at t_0 and for the case where $C_{i,2}$ is released at t_0 in Eq. (6.46) and Eq. (6.47), respectively. If $C_{i,1}$ is released at t_0, the DBF is periodic with period T_i, and C_i^{max} is the workload

4 For $m_i > 1$, one would consider $S_{i,j}^{max}$ individually for each suspension interval and S_i^{max} is defined similar to C_i^{max}, i.e., as maximum over the summation for the possible patterns.

in every full period, i.e., in every period but the last one. Note that it is possible that there are 0 full periods before the time t that is analyzed. To take care of the workload in the last period, which is the only period that has started before the analyzed time t but did not finish at time t, we define G_i^I to sum up the workload inside one period as:

$$G_i^I(t, D_{i,1}) = \begin{cases} 0 & \text{if } 0 \leq t < D_{i,1} \\ C_{i,1}^{max} & \text{if } D_{i,1} \leq t < T_i \end{cases} \tag{6.45}$$

If $C_{i,1}$ is released at t_0, the corresponding demand bound function is

$$dbf_i^{I,1}(t, D_{i,1}) = \left\lfloor \frac{t}{T_i} \right\rfloor C_i^{max} + G_i^I \left(t - \left\lfloor \frac{t}{T_i} \right\rfloor T_i, D_{i,1} \right) \tag{6.46}$$

The first part determines the maximum demand of the released jobs for completed periods, i.e., both computation segments are finished, while the second part adds a computation segment $C_{i,1}$ if needed.

If the first computation segment of task τ_i, released after or at t_0, is from $C_{i,2}$, we have to consider $C_{i,2}^{max}$ at $D_{i,2}$ and the first release of $C_{i,1}$ happens at $D_{i,2}$. Hence, the corresponding DBF is

$$dbf_i^{I,2}(t, D_{i,1}) = \begin{cases} 0 & \text{if } 0 \leq t < D_{i,2} \\ C_{i,2}^{max} + dbf_i^{I,1}(t - D_{i,2}, D_{i,1}) & \text{if } t \leq D_{i,2} \end{cases} \tag{6.47}$$

where $D_{i,2}$ is $T_i - S_i - D_{i,1}$. Therefore, the DBF for the pattern-oblivious model with individual upper bounds directly follows:

Lemma 6.19. *The DBF of τ_i for the pattern-obvious model with individual upper bounds under an FRD assignment is:*

$$dbf_i^I(t, D_{i,1}) = \max(dbf_i^{I,1}(t, D_{i,1}), dbf_i^{I,2}(t, D_{i,1})) \tag{6.48}$$

Considering Eq. (6.48), it is not difficult to show that $D_{i,1}$ should be no more than $(T_i - S_i)/2$ if $C_{i,1}^{max} \leq C_{i,2}^{max}$ and vice versa, and we can apply SEIFDA.

Figure 6.9 shows the above functions for task τ_i as listed in Table 6.4. The values of $C_{i,1}^{max}$, $C_{i,2}^{max}$, C_i^{max} and S_i^{max} are calculated as the maximum of the 3 execution/suspension patterns. As they are independent from the deadline assignment, they are listed as *Given Task Parameters* in Table 6.4. For IUB, the value of $D_{i,1}$ is chosen using SEIFDA under the given strategy and $D_{i,2}$ is set accordingly, i.e., to $D_{i,2} = T_i - S_i^{max} - D_{i,1}$. For the example in Table 6.4 under IUB, we assume the deadline assignment strategy sets $D_{i,1} = 8$, and, hence, with $S_i^{max} = 8$, we get $D_{i,2} = 30 - 8 - 8 = 14$. The resulting $dbf_i^{I,1}(t, D_{i,1})$, $dbf_i^{I,2}(t, D_{i,1})$, and $dbf_i^I(t, D_{i,1})$ are shown in Figure 6.9.

6.3.3 PATTERN-OBLIVIOUS: MULTIPLE PATHS

We assume to know the specific set of p triples of WCETs and maximum suspension times $\left\{(C_{i,1}^1, S_i^1, C_{i,2}^1), \ldots, (C_{i,1}^p, S_i^p, C_{i,2}^p)\right\}$, where each triple describes a

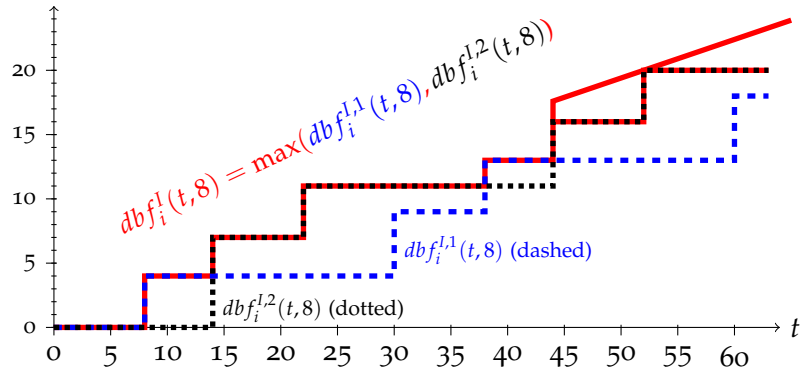


Figure 6.9: Demand bound function for individual upper bounds (IUB): dbf_i^I for task τ_i in Table 6.4 with linear approximation (solid line) for $g = 1$, i.e., after $t = g \cdot T_i + D_{i,2} = 44$. *Adapted from [BHC17].*

possible execution/suspension pattern. However, when a job arrives in the system at time t, it is unknown which path will be executed. When adopting FRD for such a scenario, we use a fixed $D_{i,1}$ across all execution paths. The second computation segment of the job always has an absolute deadline of $t + T_i$. If the first computation segment can meet its deadline, the second computation segment is released at time $t + D_{i,1} + S_i^j$ for the j-th execution path.

For the deadline assignment of τ_i, first $C_{i,1}^{max}$, $C_{i,2}^{max}$, and S_i^{max} are calculated. The actual deadline assignment is based on these values. Especially, they are used to calculate the minimum value for $D_{i,1}$ if PBminD is used as assignment strategy for SEIFDA. $G_i^{MP}(t, D_{i,1})$ is defined identically to the related function in Eq. (6.45) for the case when individual upper bounds are used, as $D_{i,1}$ is identical for all execution patterns.

We consider two cases. If the first computation segment of task τ_i released after or at t_0 is from $C_{i,1}$, the corresponding demand bound function $dbf_i^{MP,1}(t, D_{i,1})$ is identical to $dbf_i^{I,1}(t, D_{i,1})$ in Eq. (6.46), as $D_{i,1}$ is the same for all patterns:

$$dbf_i^{MP,1}(t, D_{i,1}) = \left\lfloor \frac{t}{T_i} \right\rfloor C_i^{max} + G_i^{MP}\left(t - \left\lfloor \frac{t}{T_i} \right\rfloor T_i, D_{i,1}\right) \tag{6.49}$$

While $dbf_i^{MP,1}(t, D_{i,1})$ is independent from the executed path, the DBF for the case where the second computation segment is released at t_0 depends on the related pattern. If the first computation segment of task τ_i that is released after or at t_0 is from $C_{i,2}^j$, i.e., from version j, the corresponding DBF is

$$dbf_{i,j}^{MP,2}(t, D_{i,1}) = \begin{cases} 0 & \text{if } 0 \le t < D_{i,2}^j \\ C_{i,2}^j + dbf_i^{MP,1}(t - D_{i,2}^j, D_{i,1}) & \text{if } t \le D_{i,2}^j \end{cases} \tag{6.50}$$

where $D_{i,2}^j$ is $T_i - S_i^j - D_{i,1}$. From the above discussions when deriving Eq. (6.49) and Eq. (6.50), the DBF directly follows.

Lemma 6.20. *The DBF of τ_i for the pattern-obvious model with multiple paths under an FRD assignment is as follows:*

$$dbf_i^{MP}(t, D_{i,1}) = \max\left(dbf_i^{MP,1}(t, D_{i,1}), \max_{j \in \{1,\dots p\}} dbf_{i,j}^{MP,2}(t, D_{i,1})\right) \tag{6.51}$$

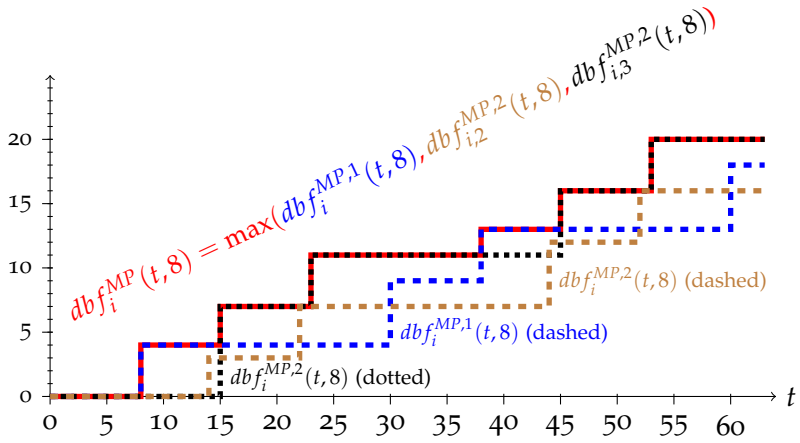


Figure 6.10: Demand bound function for multiple paths (MP): dbf_i^{MP} for τ_i in Table 6.4. For each path i, the DBF when $C_{i,2}$ is released at 0 is considered individually. $dbf_{i,1}^{MP,2}$ is omitted as it is strictly smaller than $dbf_{i,2}^{MP,2}$. *Adapted from [BHC17].*

Based on the demand bound function in Eq. (6.51), the same approach as in Sec. 6.3.2 using SEIFDA can be applied. Note that in Table 6.4 for pattern-oblivious multiple paths (MP), the value of $D_{i,2}$ differs due to the different suspension intervals of τ_i^1, τ_i^2, and τ_i^3. This leads to a tighter DBF as the jump to 7 happens at $t = 15$ instead of $t = 14$ in Figure 6.10.

6.3.4 PATTERN-CLAIRVOYANT

For this model, we assume that for a task τ_i which is described by a set of p triples $\left\{(C_{i,1}^1, S_i^1, C_{i,2}^1), \ldots, (C_{i,1}^p, S_i^p, C_{i,2}^p)\right\}$ of possible execution patterns we know which of these patterns will be executed at the moment the job arrives to the system. We first present the corresponding demand bound functions when $D_{i,1}^j$ and $D_{i,2}^j$ with $D_{i,1}^j + S_i^j + D_{i,2}^j = T_i$ are already assigned for $j = 1, 2, \ldots, p$ before considering the individual deadline assignment. We implicitly assume $D_{i,1}^j + S_i^j + D_{i,2}^j = T_i$.

Let $C_{i,1}^{max} = \max_{j\in\{1,\ldots p\}}\left\{C_{i,1}^j\right\}$ and $C_i^{max} = \max_{j\in\{1,\ldots p\}}\left\{C_{i,1}^j + C_{i,2}^j\right\}$. To calculate the workload in the period that started before t and did not finish at t, i.e., the last release before t, let $G_i^{clair}(t)$ be defined as:

$$G_i^{clair}(t) = \max_{j\in\{1,\ldots p\}}\left\{\begin{array}{ll} 0 & \text{if } t < D_{i,1}^j \\ C_{i,1}^j & \text{if } D_{i,1}^j \leq t < T_i \end{array}\right\} \tag{6.52}$$

We have to consider the maximum $C_{i,1}^j$ for each t in $0 \leq t < T_i$, as the relative deadline for the first segment of τ_i^j depends on the concrete execution/suspension pattern and is independent from the deadlines of other patterns for the same task. Again, we consider two general release patterns depending on the segment that

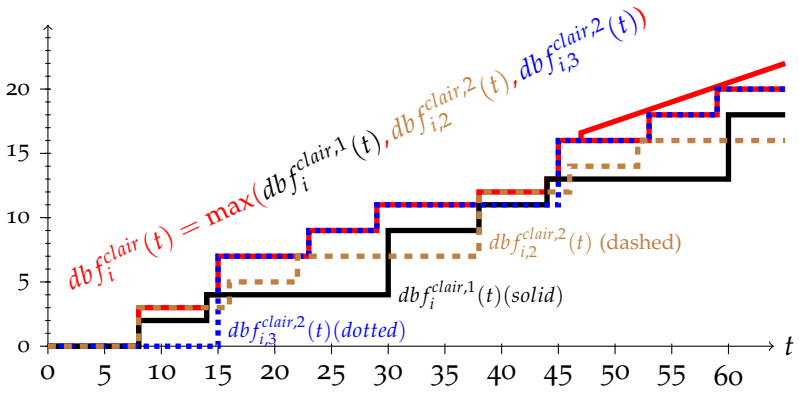


Figure 6.11: Demand bound function for shorter segment, shorter deadline (SSSD): $dbf_i^{clair}(t)$ for τ_i in Table 6.4, approximation for $g = 1$ (red straight line) after $t = g \cdot T_i + D_{i,2}^{max} = 30 + 17 = 47$. Since $dbf_{i,1}^{clair,2}(t) \leq dbf_{i,3}^{clair,2}(t)\ \forall t$, the curve $dbf_{i,1}^{clair,2}(t)$ is omitted, but $D_{i,2}^2$ is $D_{i,2}^{max}$ in the approximation. *Adapted from [BHC17].*

is released at $t = 0$. If the first computation segment of task τ_i released after or at t_0 is from $C_{i,1}$, the corresponding demand bound function $dbf_i^{clair,1}(t)$ is

$$dbf_i^{clair,1}(t) = \left\lfloor \frac{t}{T_i} \right\rfloor C_i^{max} + G_i^{clair} \left(t - \left\lfloor \frac{t}{T_i} \right\rfloor T_i \right) \tag{6.53}$$

If the first computation segment of task τ_i released at or after t_0 is from $C_{i,2}^j$, the corresponding demand bound function is

$$dbf_{i,j}^{clair,2}(t) = \begin{cases} 0 & \text{if } 0 \leq t < D_{i,2}^j \\ C_{i,2}^j + dbf_i^{clair,1}(t - D_{i,2}^j) & \text{if } t \leq D_{i,2}^j \end{cases} \tag{6.54}$$

where $D_{i,2}^j$ is $T_i - S_i^j - D_{i,1}^j$. Again, the DBF follows directly.

Lemma 6.21. *The DBF of τ_i for the pattern-clairvoyant model under an FRD assignment is as follows:*

$$dbf_i^{clair}(t) = \max \left(dbf_i^{clair,1}(t), \max_{j \in \{1,\ldots p\}} dbf_{i,j}^{clair,2}(t) \right) \tag{6.55}$$

So far, we provided the DBF assuming that $D_{i,1}^j$ and $D_{i,2}^j$ are assigned for $j = 1, 2, \ldots, p$. Now, we discuss how to assign these relative deadlines. Since the scheduler is assumed to be clairvoyant and the executed pattern is known at the arrival time of a job, we could calculate FRDs for each of the patterns using SEIFDA and schedule the jobs with specific deadlines calculated specifically for each execution/suspension pattern. However, this leads to a combinatorial explosion if the number of execution/suspension patterns is large. Hence, we instead present the two following heuristics.

SHORTER SEGMENT, SHORTER DEADLINE (SSSD)

Instead of considering all patterns individually, this approach allows to assign the deadlines of one segment for all patterns of a task at the same time. This has

the advantage that if the deadline of one of the segments is known, the deadline for the other segment can be determined directly since $D_{i,1}^j + S_i^j + D_{i,2}^j = T_i$. As a result of Lemma 6.4 and Lemma 6.5, we know that one should always assign the shorter deadline D_i^{short} to the shorter computation segment. A proper D_i^{short} can be found using SEIFDA directly after ordering the tasks according to $T_i - \max_{j\in\{1,\dots,p\}} \left\{S_i^j\right\}$ in increasing order. We assign a constant relative deadline D_i^{short} with $0 < D_i^{short} \leq (T_i - S_i)/2$ to the shorter computation segments of all possible patterns. This means that if $C_{i,1}^j \leq C_{i,2}^j$, then $D_{i,1}^j$ is set to D_i^{short} and $D_{i,2}^j$ is set to $T_i - S_i^j - D_i^{short}$. If $C_{i,1}^j > C_{i,2}^j$, then $D_{i,2}^j$ is set to D_i^{short} and $D_{i,1}^j$ is set to $T_i - S_i^j - D_i^{short}$. Figure 6.11 shows the demand bound function of task τ_i in Table 6.4 under SSSD. Note in Table 6.4, D_i^{short} is always assigned to the smaller computation segment while the other deadline depends on S_i^j. This results in 3 different $dbf_{i,j}^{clair,2}(t)$. In Figure 6.11 only $dbf_{i,1}^{clair,2}(t)$ and $dbf_{i,3}^{clair,2}(t)$ are shown as $dbf_{i,1}^{clair,2}(t) \leq dbf_{i,3}^{clair,2}(t)\ \forall t$.

PROPORTIONAL DEADLINE WITH A BIAS (PDAB)

This heuristic assigns the relative deadlines proportionally to the required execution time, since choosing the proportionally bounded minimum led to the best performance in the evaluation in Section 6.1.10. However, it is known that a proportional deadline assignment can result in reduced schedulability if one of the computation segments is significantly shorter than the other [CL14]. To avoid that an arbitrarily short relative deadline is assigned to an arbitrarily short computation segment, we introduce a constant bias D_i^{bias} for the shorter computation segments. In addition, the relative deadline of the shorter computation segment of the j-th execution path of task τ_i must be no more than $(T_i - S_i^j)/2$. Therefore, when $C_{i,1}^j \leq C_{i,2}^j$, the relative deadline $D_{i,1}^j$ is set to $\min\left\{(T_i - S_i^j)/2, D_i^{bias} + (T_i - S_i^j)\frac{C_{i,1}^j}{C_{i,1}^j + C_{i,2}^j}\right\}$, and hence $D_{i,2}^j$ is set to $T_i - S_i^j - D_{i,1}^j$. When $C_{i,1}^j > C_{i,2}^j$, then $D_{i,2}^j$ is set to $\min\left\{(T_i - S_i^j)/2, D_i^{bias} + (T_i - S_i^j)\frac{C_{i,2}^j}{C_{i,1}^j + C_{i,2}^j}\right\}$ and thus $D_{i,1}^j$ is set to $T_i - S_i^j - D_{i,2}^j$. The example in Table 6.4 has a bias of 2. Note that for pattern 2 the case happens where $D_{i,1}$ is set to $(T_i - S_i^j)/2$ due to a too large bias increase. A proper D_i^{bias} can be found by using SEIFDA directly after ordering the tasks in increasing order according to $T_i - \max_{j\in\{1,\dots,p\}} \left\{S_i^j\right\}$.

6.3.5 SCHEDULABILITY TESTS AND EXAMINATION OF THE DEMAND BOUND FUNCTIONS

With the same argument as for Theorem 6.1, we can replace $\text{DBF}_i^{\text{frd}}(t, D_{i,1})$ with $dbf_i^I(t, D_{i,1})$ from Eq. (6.48), with $dbf_i^{MP}(t, D_{i,1})$ from Eq. (6.51), or with $dbf_i^{clair}(t)$ from Eq. (6.55), depending on the adopted hybrid self-suspension model.

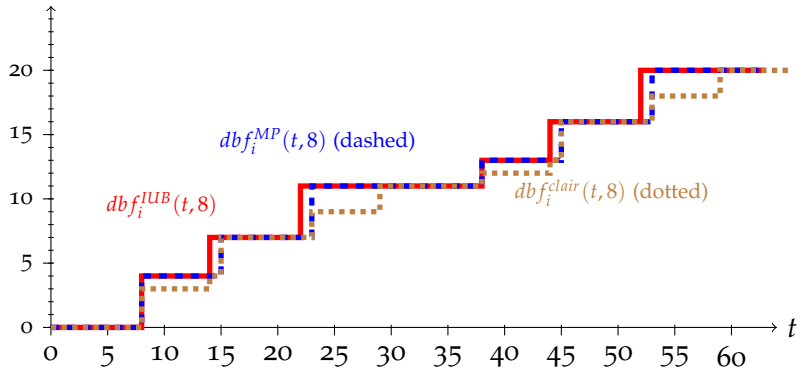


Figure 6.12: Comparison of the related DBFs for the different hybrid models. *Adapted from [BHC17].*

Theorem 6.22. *An FRD schedule under a deadline assignment policy $\mathcal{A}$ is feasible if*

$$\sum_{\tau_i \in \boldsymbol{T}} dbf_i^{\mathcal{A}}(t, D_{i,1}) \leq t \qquad \forall t \geq 0 \tag{6.56}$$

where $dbf_i^{\mathcal{A}}(t, D_{i,1})$ is defined by the adopted hybrid self-suspension strategy. That is, $dbf_i^{\mathcal{A}}(t, D_{i,1})$ can be either $dbf_i^{I}(t, D_{i,1})$ from Eq. (6.48), $dbf_i^{MP}(t, D_{i,1})$ from Eq. (6.51), or $dbf_i^{clair}(t)$ from Eq. (6.55),

Proof. This follows directly from Theorem 6.1, Lemma 6.19, Lemma 6.20, and Lemma 6.21. □

To tackle the problem of combinatorial explosion that results from the multiple jump points in the DBF, a linear approximation as described in Section 6.1.8 can be utilized. We again take a linear approximation after g completed jobs of each tasks, i.e., from $t = g \cdot T_i + D_{i,2}^{max}$, where the slope is given by the task utilization U_i. To get a safe upper bound, the maximum of lines with slope U_i through all jump points in the next period is taken. Examples of this approximation are shown in Figure 6.9 and Figure 6.11 by the red straight line. This again leads to a $1 + \frac{1}{g}$ approximation of the DBFs. The proof is similar to the one presented in Section 6.1.8 and is therefore omitted.

Since the different hybrid self-suspension models are assumed to have access to different amounts of information, the related demand bound functions become tighter when more information can be used as shown in Figure 6.12. Hence, $dbf_i^{clair}(t,8) \leq dbf_i^{MP}(t,8) \leq dbf_i^{IUB}(t,8)$, as the clairvoyant approach can use more information than MP, which in turn can use more information than IUB.

6.3.6 EVALUATION

To show the achievable tradeoffs and that the proposed hybrid models effectively utilize the available information, we conducted evaluations for synthesized task sets, where we compared the proposed approaches with methods for the dynamic self-suspension model based on the *acceptance ratio* (in percent) with respect to

the task set utilization. For each utilization level in a range from 5% to 100% with steps of 5%, we generated 100 task sets with a cardinality of 10 tasks.

We adopted the UUniFast method [BB05] to generate sets with a given total utilization. The task periods were in *log-uniform distribution*, as suggested by Emberson et al. [ESD10], with a period range of one or two orders of magnitude i.e, $[10ms - 100ms]$ or $[10ms - 1000ms]$, respectively. We accordingly set $C_i = T_i U_i$ and created implicit deadline task sets, i.e., $D_i = T_i$. We converted them to self-suspending tasks where the suspension lengths of the tasks were randomly chosen according to a uniform distribution in one of three ranges:

- short suspension: $[0.01(T_i - C_i), 0.1(T_i - C_i)]$
- moderate suspension: $[0.1(T_i - C_i), 0.3(T_i - C_i)]$
- long suspension: $[0.3(T_i - C_i), 0.6(T_i - C_i)]$

Each self-suspension task consisted of two paths:

- One path was randomly chosen to have the largest WCET C_i. The WCET of the remaining path was adjusted by multiplying it with a uniformly-distributed random variable in $[0.8, 1]$.
- One path was randomly chosen to have the largest suspension time, equal to S_i. The worst-case suspension time of the remaining path was adjusted by multiplying it with a uniformly-distributed random variable in $[0.8, 1]$.
- For each path, we generated $C_{i,1}$ as a percentage of its WCET, according to a uniform distribution, and set $C_{i,2}$ accordingly.

We consider a discrete time model in the evaluation, i.e., all task parameters were rounded up to integers. We evaluated the following approaches:

- *SCEDF*: the suspension-oblivious approach by converting suspension time into computation time.
- *PASS-OPA*: The state-of-the-art approach for static-priority scheduling of dynamic self-suspending tasks presented in [HCZ+15]. Each interfering job is considered by running the path with the maximum cumulative execution time, i.e., C_i^{max}. Each task analyzed is considered as the task running through the path with the maximum cumulative computation and suspension time, i.e., $max_{1 \leq j \leq p}\{C_{i,1}^j + S_i^j + C_{i,2}^j\}$.
- *Oblivious-IUB*: The approach in Section 6.3.2.
- *Oblivious-MP*: The approach in Section 6.3.3.
- *Clairvoyant-SSSD*: The approach in Section 6.3.4.
- *Clairvoyant-PDAB*: The approach in Section 6.3.4.

The DBFs were approximated with $g = 2$ in all calculations. For *Oblivious-IUB* and *Oblivious-MP*, SEIFDA-PBminD was used in the deadline assignment, as SEIFDA-PBminD is usually the best deadline assignment strategy, according to the experimental results in Section 6.1.10. For *Clairvoyant-SSSD* and *Clairvoyant-PDAB*, we used SEIFDA-minD since a proportional lower bound is already part of the assignment in *Clairvoyant-PDAB*.

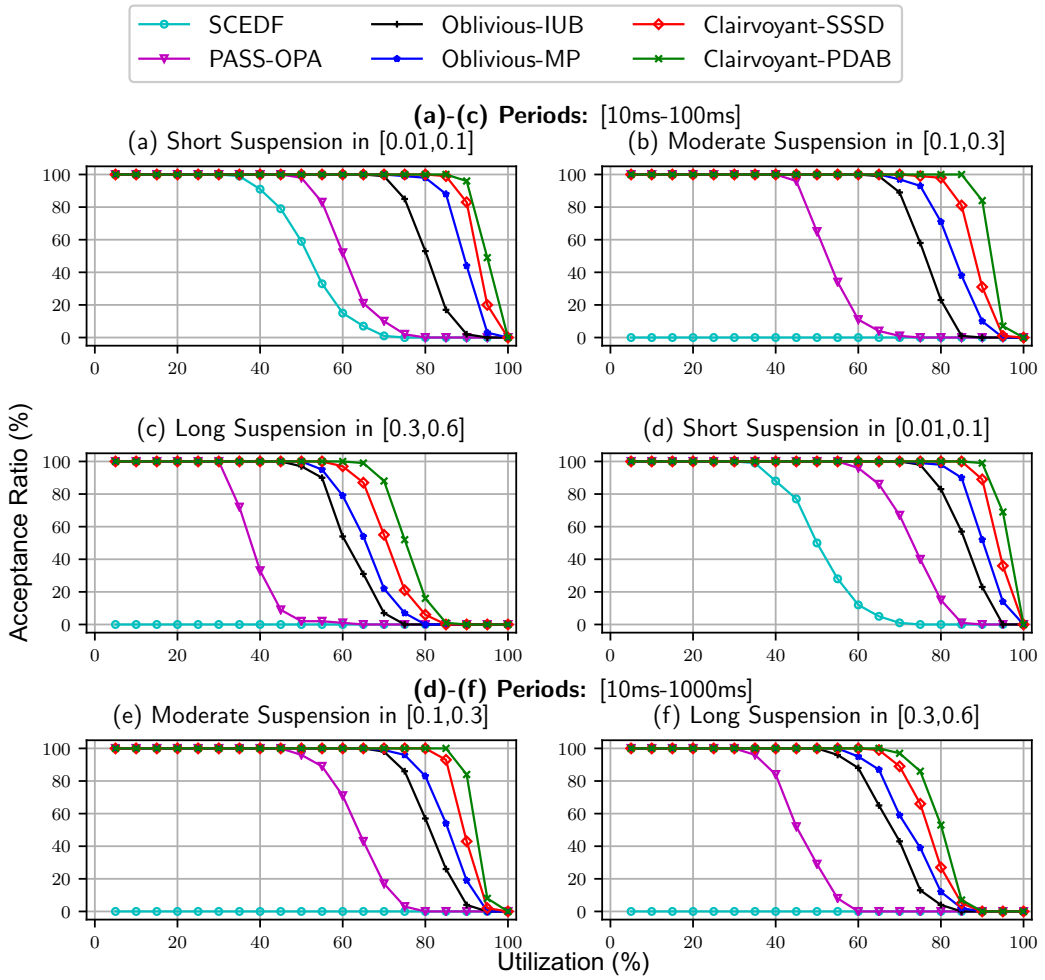


Figure 6.13: Comparison of the hybrid self-suspension models with approaches for the dynamic self-suspension model, considering different suspension length and periods in $[10ms - 100ms]$ in (a)-(c) or $[10ms - 1000ms]$ in (d)-(f). Approximated DBFs with $g = 2$ are used. *Adapted from [BHC17].*

For periods in $[10ms, 100ms]$ (Figure 6.13(a)-(c)), we observe that the presented approaches achieve a significantly better acceptance ratio than *PASS-OPA*, which always clearly outperforms *SCEDF*. When more information about the task system is used, the acceptance increases, i.e., *Clairvoyant* approaches use more information than *Oblivious-MP*, which uses more information than *Oblivious-IUB*. For the *Clairvoyant* approaches, *Clairvoyant-PDAB* is almost always better than *Clairvoyant-SSSD*. The acceptance ratio is higher for longer periods in $[10ms, 1000ms]$ (Figure 6.13(d)-(f)). However, the results generally exhibit similar behaviour, therefore further discussion is omitted.

This evaluation shows that carefully using all available information of the execution/suspension patterns, in both the self-suspension model and the scheduling algorithms, results in a significant advantage with regards to schedulability. Thus, instead of focusing only on the segmented and dynamic self-suspension model, the presented models and scheduling strategies should be used if possible.

6.4 CONCLUSION

We investigated uniprocessor scheduling for *one-segmented self-suspending task systems*. We consider *fixed-relative-deadline* (FRD) scheduling and provided a general FRD schedulability test for dynamic-priority scheduling based on *demand bound functions*. Afterwards, we introduced a new FRD scheduling approach called *Shortest Execution Interval First Deadline Assignment* (*SEIFDA*) that allows multiple deadline assignment strategies. SEIFDA yields a significantly better performance than existing approaches, as shown in the evaluation, and has a *speedup factor* of 3, i.e., the best known for segmented self-suspension.

segmented self-suspension
fixed-relative-deadline
demand bound function
SEIFDA
speedup factor

One important cause of self-suspension behaviour is *multiprocessor resource sharing*. We provided several *resource-oriented partitioned scheduling* (ROP) strategies, using both static-priority and dynamic-priority scheduling. In contrast to the initial work on ROP by Huang et al. [HYC16], our approaches use *release enforcement* to ensure that *release jitter* does not have to be considered when analyzing the schedulability. We modeled the non-critical sections as segmented self-suspending tasks and apply the related state-of-the-art uniprocessor techniques to find a feasible partition. We showed that one of our approaches, namely FP-RM-PCP, has a *speedup factor* of 6 compared to the optimal schedule, improving the best previously known result. In the evaluations, two of our approaches, namely FP-EIM-PCP and EDF-EIM-PCP (which is based on SEIFDA), outperformed the resource sharing protocols known from the literature. This shows the effectiveness of our approaches and the resource-oriented partitioned scheduling approach in general, both *theoretically* and *empirically*.

multiprocessor resource sharing
resource-oriented partitioned scheduling
release enforcement
speedup factor

However, one of the disadvantages of the proposed ROP is the fact that the underlying segmented self-suspension model is very restrictive and therefore may not be applicable. While ROP can also utilize the dynamic self-suspension model, this model is over flexible and therefore may lead to a pessimistic analysis. To bridge this gap between the two models, we proposed multiple *hybrid self-suspension* models that utilize additional information about the considered tasks and carefully examined a special case where the jobs in the system suspend themselves at most once. Depending on the available knowledge about the execution/suspension patterns, we designed *pattern-oblivious* approaches, that use the information of the patterns *offline* but not *online*, and *pattern-clairvoyant* approaches, which use the the information *both offline and online*. We explained how to design FRD scheduling strategies based on SEIFDA that utilize the offline patterns and develop different scheduling strategies, depending on the applicable hybrid self-suspension task model. Empirically, our newly developed approaches are shown effective in terms of acceptance ratio compared to the state-of-the-art scheduling strategies for the dynamic self-suspension task model. To the best of our knowledge, this is the first result for a hybrid self-suspension task model. We strongly believe that these results open a new dimension for suspension-aware real-time embedded systems. For example, the dynamic self-suspension task model has been widely used for analyzing the multiprocessor synchronization protocols, e.g., [Bra13]. If the number of suspension intervals is small, our conclusion shows that quantifying the execution/suspension patterns can potentially help improve the schedulability significantly.

hybrid self-suspension
pattern-oblivious
pattern-clairvoyant

7 CONCLUSIONS AND OUTLOOK

This dissertation examines *real-time* systems and focuses on realistic task and system models, scheduling algorithms and schedulability tests, and their theoretical performance evaluation. This chapter first recapitulates the contributions which are provided in this dissertation in Section 7.1. In Section 7.2 it is examined whether these contributions support the dissertation hypothesis. Afterwards, an outlook at possible future work is given in Section 7.3. In Section 7.4 the dissertation is concluded with some final remarks and an outlook.

7.1 SUMMARY OF THE CONTRIBUTIONS

The contributions of this dissertation are summarized according to the chapters where they are detailed.

7.1.1 SPEEDUP FACTORS AND UTILIZATION BOUNDS

theoretical evaluation method

parametric utilization bound

semi-harmonic task set

speedup factor

Chapter 4 primarily considers *theoretical evaluation methods* that compare the worst-case behaviour of scheduling algorithms or schedulability tests, focusing on *speedup factors* and *parametric utilization bounds*. It first showed that *parametric utilization bounds* can drastically increase the utilization bounds compared to the not parameterized state-of-the-art. To be precise, large improvements were provided on the *utilization bound* for non-preemptive Rate Monotonic scheduling, considering the blocking factor of the task set as an additional parameter, and on the *utilization bound* for preemptive Rate Monotonic scheduling when considering *automotive systems* where the task periods are chosen from $\{1, 2, 5, 10, 20, 50, 100, 200, 1000\}$ ms, i.e., the periods are *semi-harmonic*. Afterwards, *speedup factors* for Deadline Monotonic scheduling compared to Earliest Deadline First scheduling were considered, examining both preemptive and non-preemptive scheduling as well as implicit-, constrained-, and arbitrary-deadline task sets. These examinations provided the rather surprising result that in these cases linear-time sufficient schedulability tests for Deadline Monotonic scheduling have the same speedup factors as exact schedulability tests with exponential time complexity.

This raised the questions whether and in which situations speedup factors or utilization bounds are a reasonable tool to compare the performance of scheduling algorithms and schedulability tests, and several misinterpretations or misunderstandings from the real-time systems literature were detailed. Therefore, perspectives on how to understand and utilize speedup factors and utilization bounds were given, pointing out 8 observations regarding their improper usage. Specifically, it was discussed why these metrics often lack the power to

discriminate between the performance of different scheduling algorithms and schedulability tests, even in situations where the performance differs largely in empirical evaluations. An algorithm or test with a worse speedup factor or bound may even perform much better in empirical evaluations and in practice, since theoretical methods only consider, potentially practically irrelevant, corner cases.

parametric augmentation function

Resulting from this, *parametric augmentation functions* were proposed as a possible solution, which describe theoretical comparisons not with a single value but with a vector of values that detail the augmentation function based on these values. This allows to exclude corner cases if they are not relevant in the considered setting and to establish regions of dominance between scheduling algorithms or schedulability tests. An example on how such an examination can be performed has been provided as well.

7.1.2 UNCERTAIN EXECUTION BEHAVIOUR

uncertain execution behaviour
worst-case execution time

Chapter 5 considers the situation where the periodic or the sporadic task model is not able to correctly describe the system due to an *uncertain execution behaviour* regarding the *worst-case execution time* (WCET) of the tasks. Since for such systems considering the absolute worst-case scenario in the analysis would lead to an intolerable increase in hardware costs, a reasonable tradeoff between hardware costs and timing guarantees must be found. The provided results based on the observation that even if all tasks in a systems have real-time constraints, some tasks are usually *more important* for the system stability while others are *not so important*. Hence, rare deadline misses can be tolerated for the latter while for the *more important* timeliness must always be guaranteed. An important example for systems with such properties are *mixed-criticality systems* [Veso7], where the mainstream research on has been criticized lately [ENN+15; EN16]. Specifically, low-criticality tasks should not be abandoned, and systems should return to low-criticality mode after a sufficient amount of time.

mixed-criticality systems

Systems with Dynamic Real-Time Guarantees

To answer this criticism, *Systems with Dynamic Real-Time Guarantees* were introduced, both for uniprocessor and multiprocessor scenarios, providing a more suitable model for systems with an uncertain execution behaviour. During runtime, a *System with Dynamic Real-Time Guarantees* provides either *full timing guarantees* if all jobs meet their deadline or *limited timing guarantees* if only the jobs of the *more important tasks* are guaranteed to meet their deadline while *not so important tasks* have bounded tardiness. All these guarantees are given offline using static-priority scheduling without *online adaptation*. The approach provides a reasonable performance compared to the state-of-the-art for mixed-criticality systems that needs online adaptation and allows to drop not so important tasks. The approach is extended to partitioned and semi-partitioned multiprocessor systems, providing reasonable acceptance ratios. A task migration technique is presented that allows to compensate processors with an abnormal execution behaviour over a certain time period, e.g., due to overheating or intermittent faults.

full timing guarantees
limited timing guarantees

worst-case deadline failure probability

In addition, a novel approach to over-approximate the *worst-case deadline failure probability* of a task under static-priority scheduling in uncertain execution

environments has been introduced using task-level convolution that is based on multinomial distributions. Like job-level convolution-based approaches, the approach is more precise than analytical bounds, but contrary to them it is scalable to large task sets due to multiple runtime improvement techniques that cannot be applied to job-level convolution-based approaches.

7.1.3 SELF-SUSPENSION

fixed-relative-deadline
SEIFDA
resource-oriented partitioned scheduling

In Chapter 6, a *fixed-relative-deadline* scheduling algorithm called *Shortest Execution Interval First Deadline Assignment* (SEIFDA) for the one-*segmented self-suspension* model is introduced and shown to outperform the state-of-the-art, both *theoretically* and *empirically*. SEIFDA is utilized in the design of a *resource-oriented partitioned* scheduling with release enforcement for multiprocessor resource sharing, again outperforming the state-of-the-art both *theoretically* and *empirically*.

hybrid self-suspension
pattern-clairvoyant
pattern-oblivious

In addition, the gap between the over flexible dynamic and the over precise segmented self-suspension model is examined. It is bridged by introducing multiple *hybrid self-suspension* models, which assume a self-suspending task to be specified by a set of possible execution patterns that are known offline. These models have different tradeoffs between flexibility and precision that can be achieved based on the information that is known for the considered task set. Their applicability depends on the additional information known compared to the dynamic self-suspension model and on whether this is known offline and online, so-called *pattern-clairvoyant* models, or only offline, so-called *pattern-oblivious* models. The evaluation shows that this information can be utilized to significantly increase the performance regarding schedulability compared to the dynamic self-suspension model.

7.2 EXAMINATION OF THE DISSERTATION HYPOTHESIS

The question remains whether the contributions support the hypothesis:

> *Realistic scheduling models and analyses are essential for guaranteeing timing correctness in advanced real-time systems while ensuring that the system resources necessary to provide these guarantees are not over-provisioned.*

The exploration of speedup factors in Chapter 4 showed that overvaluing their meaning and focusing on a good speedup factor during the design of an algorithm can lead to serious performance drawbacks. Hence, avoiding the detailed misconceptions helps to achieve algorithms and analyses that perform better in realistic situations. One possible reason for such performance drawbacks is that countermeasures are taken to guarantee the performance of the algorithm in a corner case or in a scenario with limited practical relevance. A similar problem was determined for utilization bounds that often also result from corner cases. Hence, the underlying model can be seen as too general, and therefore too imprecise and unrealistic for an evaluated scenario where such cases are excluded. To achieve a more realistic evaluation based on such metrics while keeping the advantage of worst-case guarantees, it has been proposed to provide

parametric augmentation bounds, i.e., to include additional parameters into the evaluated function to achieve better values when certain conditions are met. This allows to provide tighter individual results for subsets of the evaluated general model. The possible gain of such an approach was exemplified in the provided parametric utilization bounds for non-preemptive Rate Monotonic scheduling and for automotive task systems. It was shown that such a parametrized analysis can result in a tighter theoretical analysis and hence a more realistic assessment of the performance of the scheduling algorithm, thus avoiding to provide system resources that are not necessary in the considered setting.

The Systems with Dynamic Real-Time Guarantees introduced in Chapter 5 provide a general way to model uncertain execution behaviour. While they are not limited to mixed-criticality systems, they explicitly take criticism of the mixed-criticality research into account that was voiced by systems engineers who stated that the current model and analysis does not meet their expectations. Specifically, tasks are not abandoned, the system can return to the initial system state, and no online adaptation is necessary. Otherwise, considering the mixed-criticality model, jobs with decreased priority could result in a backlog that is not analyzed, a return to low-criticality mode may be problematic since this situation is never considered in the analysis, or online adaptation may cause a deadline miss due to the high overhead. In such a situation an analysis that determines a task set to be schedulable may in fact be optimistic since it does not consider these system characteristics. However, these problems are avoided in Systems with Dynamic Real-Time Guarantees since the backlog cannot affect the timeliness of the more critical tasks and no online adaptation is performed. Hence, providing a realistic system model that considers these scenarios ensures that the timeliness of the system can be guaranteed in the analysis.

When evaluating the worst-case deadline failure probability in an uncertain execution environment, an analysis should be both precise and scalable. However, precise job-level convolution-based approaches are only applicable for small task sets. Contrarily, analytical approaches are faster but may lead to a large over-estimation that cannot be quantified. The task-level convolution provided in Chapter 5 is scalable to large task sets and also allows to achieve a better runtime by sacrificing a bounded amount of precision. Hence, it enables to precisely quantify the worst-case deadline failure probability for systems with a realistic number of tasks while avoiding over provisioning of system resources to ensure that the worst-case deadline failure probability is below a certain threshold.

The algorithms introduced in Chapter 6, namely SEIFDA and the resource-oriented partitioned scheduling, show the possible performance gain when a method considers a specific situation. In this case it is considered that the number of suspension intervals or the number of critical sections is 1. Such a restriction models a specific scenario more realistically than the general model that assumes no bounds on these parameters. While the resulting algorithms are not applicable to the general scenario, the performance gain allows to reduce the system resources necessary for this special case. One possible application of one-segmented self-suspension is offloading, where a part of the task is executed on a remote device, e.g., to save energy. Furthermore, Open-MP task sets, where synchronization among tasks is always performed at the end of a task, are a

possible direct application for multiprocessor resource sharing when all tasks have one critical section.

The hybrid self-suspension models introduced in Chapter 6 allow to bridge the gap between the dynamic and the segmented self-suspension model with different tradeoffs between accuracy and flexibility. Applying the dynamic model is safe but can result in largely over-estimated system resources when the tasks have a more specific structure. Furthermore, it is often an unrealistic assumption that the number of suspension intervals cannot be bounded while both the worst-case execution time as well as the total suspension time can be determined precisely, especially when considering real-time systems where tasks usually have a specified structure. On the other hand, applying the segmented self-suspension model may jeopardize the timing correctness when tasks do not precisely match the model. The hybrid models fill this gap by enabling the use of the accessible information, resulting in a more realistic model of the examined tasks than the dynamic or the segmented self-suspension model. The evaluation of the hybrid self-suspension models and the related schedulability tests show that they achieve a better performance than schedulability tests for the dynamic model and therefore potentially allow to use a system with less resources than determined by an analysis under the dynamic self-suspension model.

Summarizing the aforementioned, the findings in this dissertation support the hypothesis that realistic models and analyses allow to improve the analysis precision in advanced real-time systems without jeopardizing the timing correctness. They are therefore essential to achieve timing guarantees while reducing the system resources that are required. Such a gain may result from removing pessimism by modelling the system at hand more precisely, thus enabling the access to additional information, or by excluding scenarios that are not relevant for the considered system in the design of an algorithm and in the analysis.

7.3 FUTURE WORK

Resulting from the observations and conclusions regarding *speedup factors* and *utilization bounds* in Chapter 4, it would be interesting to examine in which scenarios *parametric augmentation functions* are able to improve the theoretical understanding of scheduling algorithms and schedulability tests. However, while they are a potentially powerful tool, applying them to improve speedup factors and utilization bounds in practical scenarios may be difficult.

After providing a new, scalable analysis technique to over-approximate the worst-case deadline failure probability under static-priority scheduling in Chapter 5, exploring how such techniques can be extended to dynamic-priority scheduling approaches like EDF is a logical next step. Nevertheless, the resulting scenario is significantly more difficult, since analysis cannot easily be restricted to a short time interval like in the static-priority case. Therefore, improved analysis techniques and new ideas are necessary to tackle the resulting increase in computational complexity.

The SEIFDA algorithm introduced in Chapter 6 is limited to one-segmented self-suspension. Extending it to multiple self-suspension intervals and determine

its performance both for self-suspending task sets as well as in a resource-oriented partitioned scheduling or under the hybrid self-suspension model therefore is a reasonable next step to fully utilize the potential of this algorithm. Furthermore, it is interesting to investigate how hybrid models can be utilized in resource-oriented partitioned scheduling algorithm or in other multiprocessor resource sharing protocols.

7.4 FINAL REMARKS AND OUTLOOK

This dissertation provides realistic models and analyses for real-time systems with a focus on practical relevance and applicability. It shows how considering a specific setting can improve the results that are achievable compared to the general setting, since it reduces the pessimism in modeling and analysis. Investigating interesting and practically relevant special cases may also help to ease the transfer of academic results into industrial practice. Furthermore, examining a specific but restricted scenario can result in a better understanding of the underlying problem and may help to avoid flaws when the more complex general setting is examined. Moreover, algorithms that handle the restricted setting efficiently can potentially be extended to more general settings. However, such an analysis should be general enough to be applied to related scenarios, especially when the considered scenario is strongly restricted or only relevant for a limited set of practical applications. Nevertheless, it seems meaningful to consider important restricted scenarios in research more often, especially when general settings are too far from industrial applications or extremely complex.

The provided task-level convolution-based approach to calculate the worst-case deadline failure probability assumes periodic or sporadic tasks under static-priority scheduling. While it cannot directly be applied to dynamic-priority scheduling, extensions to cover other static-priority scenarios for uniprocessor scheduling like non-preemptive scheduling, limited preemptive scheduling, or resource sharing under PCP seem relatively straightforward. Instead of examining such extensions individually, an interesting idea is to determine a set of criteria that must be fulfilled to apply the provided analysis. Afterwards, for a problem at hand it can be determined whether the analysis is applicable based on these criteria. Otherwise, a (potentially more pessimistic) schedulability test that fulfills the criteria can be used. One example for a similar concept are the three conditions that must be met to ensure that a schedulability test is OPA compatible [DB09]. Subsequently, similar criteria could be explored for a possible extension to dynamic-priority scheduling.

The examination of resource-oriented partitioned scheduling shows the potential improvement when resource sharing is not considered from a task-centric but from a resource-centric point of view. However, the currently provided solution is limited to one resource access and to tasks where the non-critical sections can be modelled by the one-segmented self-suspension model. While this allowed to provide good solutions for this special case, an evaluation of the more general setting is necessary. Nevertheless, a focus on removing one individual restriction seems questionable. The reason is that one of the shortcomings of most real-time

resource sharing protocols is that they try to solve one specific problem, i.e., resource sharing, while ignoring related problems like task allocation or priority ordering which highly impact the performance. Therefore, it seems interesting to not only tackle the individual restrictions but to take a more holistic point of view. For instance, when assuming non-nested critical sections and that a large share of the tasks has only one critical section, it seems possible to assign the fixed relative deadlines for tasks with multiple critical sections first according to the proportional scheme, and afterwards assign the deadlines for tasks with only one critical section using SEIFDA. However, such a solution depends on the applicability of the assumptions and otherwise alternative solutions must be explored. Furthermore, it is important to decide in which situations a job that is waiting for a shared resource should be suspended and in which situation the job should spin on the processor. Here resource-oriented partitioned scheduling has the advantage that the blocking time of a task on the synchronization processor can be bounded relatively easily. Moreover, due to the use of PCP, the WCRT on the synchronization processor can be relatively short for tasks with a short period, making spinning a potentially good solution for these tasks. Hence, combining these and other ideas into a more holistic resource-oriented partitioned scheduling that provided different treatments for tasks based on certain criteria seems to be an interesting research direction.

BIBLIOGRAPHY

[ASL04] Tarek F. Abdelzaher, Vivek Sharma, and Chenyang Lu. "A Utilization Bound for Aperiodic Tasks and Priority Driven Scheduling." In: *IEEE Trans. Computers* 53.3 (2004), pp. 334–350.

[ARS18] Arun Adiththan, S. Ramesh, and Soheil Samii. "Cloud-assisted control of ground vehicles using adaptive computation offloading techniques." In: *2018 Design, Automation & Test in Europe Conference & Exhibition, DATE 2018, Dresden, Germany, March 19-23, 2018*. IEEE, 2018, pp. 589–592.

[AG08] Sebastian Altmeyer and Gernot Gebhard. "WCET Analysis for Preemptive Scheduling." In: *8th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis, Prague, Czech Republic, July 1, 2008*. Vol. 8. OASICS. 2008.

[AM11] Sebastian Altmeyer and Claire Maiza. "Cache-related preemption delay via useful cache blocks: Survey and redefinition." In: *Journal of Systems Architecture - Embedded Systems Design* 57.7 (2011), pp. 707–719.

[ABJ01] Björn Andersson, Sanjoy K. Baruah, and Jan Jonsson. "Static-Priority Scheduling on Multiprocessors." In: *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS 2001), London, UK, 2-6 December 2001*. IEEE Computer Society, 2001, pp. 193–202.

[AB08] Björn Andersson and Konstantinos Bletsas. "Sporadic Multiprocessor Scheduling with Few Preemptions." In: *20th Euromicro Conference on Real-Time Systems, ECRTS 2008, 2-4 July 2008, Prague, Czech Republic, Proceedings*. IEEE Computer Society, 2008, pp. 243–252.

[AE10] Björn Andersson and Arvind Easwaran. "Provably good multiprocessor scheduling with resource sharing." In: *Real-Time Systems* 46.2 (2010), pp. 153–159.

[AR14] Björn Andersson and Gurulingesh Raravi. "Real-time scheduling with resource sharing on heterogeneous multiprocessors." In: *Real-Time Systems* 50.2 (2014), pp. 270–314.

[AT06] Björn Andersson and Eduardo Tovar. "Multiprocessor Scheduling with Few Preemptions." In: *12th IEEE Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2006), 16-18 August 2006, Sydney, Australia*. IEEE Computer Society, 2006, pp. 322–334.

[AT09] Björn Andersson and Eduardo Tovar. "The utilization bound of non-preemptive rate-monotonic scheduling in Controller Area Networks is 25%." In: *IEEE Fourth International Symposium on Industrial Embedded Systems - SIES*. 2009, pp. 11–18.

[Aud91] Neil C. Audsley. *Optimal Priority Assignment And Feasibility Of Static Priority Tasks With Arbitrary Start Times*. 1991.

[AE13] Philip Axer and Rolf Ernst. "Stochastic response-time guarantee for non-preemptive, fixed-priority scheduling under errors." In: *The 50th Annual Design Automation Conference 2013, DAC '13, Austin, TX, USA, May 29 - June 07, 2013*. 2013, 172:1–172:7.

[Bak06] Theodore P. Baker. "An Analysis of Fixed-Priority Schedulability on a Multiprocessor." In: *Real-Time Systems* 32.1-2 (2006), pp. 49–71.

[Bak03] Theodore P. Baker. "Multiprocessor EDF and Deadline Monotonic Schedulability Analysis." In: *Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS 2003), 3-5 December 2003, Cancun, Mexico*. IEEE Computer Society, 2003, pp. 120–129.

[Bak91] Theodore P. Baker. "Stack-based Scheduling of Realtime Processes." In: *Real-Time Systems* 1 (1991), pp. 67–99.

[BB07] Theodore P. Baker and Sanjoy K. Baruah. "Schedulability analysis of multiprocessor sporadic task systems." In: *Journal of Embedded Computing* (Jan. 2007).

[BC07a] Theodore P. Baker and Michele Cirinei. "Brute-Force Determination of Multiprocessor Schedulability for Sets of Sporadic Hard-Deadline Tasks." In: *Principles of Distributed Systems, 11th International Conference, OPODIS 2007, Guadeloupe, French West Indies, December 17-20, 2007. Proceedings*. Vol. 4878. Lecture Notes in Computer Science. Springer, 2007, pp. 62–75.

[Bar15a] Sanjoy K. Baruah. "Federated Scheduling of Sporadic DAG Task Systems." In: *IEEE International Parallel and Distributed Processing Symposium, IPDPS*. 2015, pp. 179–186.

[Bar16a] Sanjoy K. Baruah. "Schedulability Analysis for a General Model of Mixed-Criticality Recurrent Real-Time Tasks." In: *IEEE Real-Time Systems Symposium, RTSS*. 2016, pp. 25–34.

[Bar16b] Sanjoy K. Baruah. "Schedulability analysis of mixed-criticality systems with multiple frequency specifications." In: *International Conference on Embedded Software, EMSOFT*. 2016, 24:1–24:10.

[Bar07] Sanjoy K. Baruah. "Techniques for Multiprocessor Global Schedulability Analysis." In: *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS 2007), 3-6 December 2007, Tucson, Arizona, USA*. IEEE Computer Society, 2007, pp. 119–128.

[Bar05] Sanjoy K. Baruah. "The Limited-Preemption Uniprocessor Scheduling of Sporadic Task Systems." In: *17th Euromicro Conference on Real-Time Systems (ECRTS 2005), 6-8 July 2005, Palma de Mallorca, Spain, Proceedings*. IEEE Computer Society, 2005, pp. 137–144.

[Bar15b] Sanjoy K. Baruah. "The federated scheduling of constrained-deadline sporadic DAG task systems." In: *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition, DATE*. 2015, pp. 1323–1328.

[Bar15c] Sanjoy K. Baruah. "The federated scheduling of systems of conditional sporadic DAG tasks." In: *Proceedings of the 15th International Conference on Embedded Software (EMSOFT)*. 2015.

[BB08a] Sanjoy K. Baruah and Theodore P. Baker. "Global EDF Schedulability Analysis of Arbitrary Sporadic Task Systems." In: *20th Euromicro Conference on Real-Time Systems, ECRTS 2008, 2-4 July 2008, Prague, Czech Republic, Proceedings*. IEEE Computer Society, 2008, pp. 3–12.

[BB08b] Sanjoy K. Baruah and Theodore P. Baker. "Schedulability analysis of global edf." In: *Real-Time Systems* 38.3 (2008), pp. 223–235.

[BBD+11] Sanjoy K. Baruah, Vincenzo Bonifaci, Gianlorenzo D'Angelo, Alberto Marchetti-Spaccamela, Suzanne van der Ster, and Leen Stougie. "Mixed-Criticality Scheduling of Sporadic Task Systems." In: *Algorithms - ESA 2011 - 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings*. Vol. 6942. Lecture Notes in Computer Science. Springer, 2011, pp. 555–566.

[BBD+15] Sanjoy K. Baruah, Vincenzo Bonifaci, Gianlorenzo D'Angelo, Haohan Li, Alberto Marchetti-Spaccamela, Suzanne van der Ster, and Leen Stougie. "Preemptive Uniprocessor Scheduling of Mixed-Criticality Sporadic Task Systems." In: *J. ACM* 62.2 (2015), 14:1–14:33.

[BBD+12] Sanjoy K. Baruah, Vincenzo Bonifaci, Gianlorenzo D'Angelo, Haohan Li, Alberto Marchetti-Spaccamela, Suzanne van der Ster, and Leen Stougie. "The Preemptive Uniprocessor Scheduling of Mixed-Criticality Implicit-Deadline Sporadic Task Systems." In: *24th Euromicro Conference on Real-Time Systems, ECRTS*. 2012, pp. 145–154.

[BBM+09] Sanjoy K. Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Sebastian Stiller. "Implementation of a Speedup-Optimal Global EDF Schedulability Test." In: *21st Euromicro Conference on Real-Time Systems, ECRTS*. 2009, pp. 259–268.

[BBM+10] Sanjoy K. Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Sebastian Stiller. "Improved multiprocessor global schedulability analysis." In: *Real-Time Systems* 46.1 (2010), pp. 3–24.

[BB11a] Sanjoy K. Baruah and Alan Burns. "Implementing Mixed Criticality Systems in Ada." In: *Reliable Software Technologies - Ada-Europe 2011 - 16th Ada-Europe International Conference on Reliable Software Technologies, Edinburgh, UK, June 20-24, 2011. Proceedings*. Vol. 6652. Lecture Notes in Computer Science. Springer, 2011, pp. 174–188.

[BBD11] Sanjoy K. Baruah, Alan Burns, and Robert I. Davis. "Response-Time Analysis for Mixed Criticality Systems." In: *Proceedings of the 32nd IEEE Real-Time Systems Symposium, RTSS 2011, Vienna, Austria, November 29 - December 2, 2011*. IEEE Computer Society, 2011, pp. 34–43.

[BBG16] Sanjoy K. Baruah, Alan Burns, and Zhishan Guo. "Scheduling Mixed-Criticality Systems to Guarantee Some Service under All Non-erroneous Behaviors." In: *28th Euromicro Conference on Real-Time Systems, ECRTS 2016, Toulouse, France, July 5-8, 2016*. IEEE Computer Society, 2016, pp. 131–138.

[BCG+99] Sanjoy K. Baruah, Deji Chen, Sergey Gorinsky, and Aloysius K. Mok. "Generalized Multiframe Tasks." In: *Real-Time Systems* 17.1 (1999), pp. 5–22.

[BCP+96] Sanjoy K. Baruah, N. K. Cohen, C. Greg Plaxton, and Donald A. Varvel. "Proportionate Progress: A Notion of Fairness in Resource Allocation." In: *Algorithmica* 15.6 (1996), pp. 600–625.

[BF07a] Sanjoy K. Baruah and Nathan Fisher. "Global Deadline-Monotonic Scheduling of Arbitrary-Deadline Sporadic Task Systems." In: *Principles of Distributed Systems, 11th International Conference, OPODIS 2007, Guadeloupe, French West Indies, December 17-20, 2007. Proceedings*. 2007, pp. 204–216.

[BF08] Sanjoy K. Baruah and Nathan Fisher. "Global Fixed-Priority Scheduling of Arbitrary-Deadline Sporadic Task Systems." In: *Distributed Computing and Networking, 9th International Conference, ICDCN 2008, Kolkata, India, January 5-8, 2008.* Vol. 4904. Lecture Notes in Computer Science. Springer, 2008, pp. 215–226.

[BF06] Sanjoy K. Baruah and Nathan Fisher. "The Partitioned Multiprocessor Scheduling of Deadline-Constrained Sporadic Task Systems." In: *IEEE Trans. Computers* 55.7 (2006), pp. 918–923.

[BF05] Sanjoy K. Baruah and Nathan Fisher. "The Partitioned Multiprocessor Scheduling of Sporadic Task Systems." In: *RTSS*. 2005, pp. 321–329.

[BF07b] Sanjoy K. Baruah and Nathan Fisher. "The partitioned dynamic-priority scheduling of sporadic task systems." In: *Real-Time Systems* 36.3 (2007), pp. 199–226.

[BMR90] Sanjoy K. Baruah, Aloysius K. Mok, and Louis E. Rosier. "Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor." In: *Proceedings of the Real-Time Systems Symposium - 1990, Lake Buena Vista, Florida, USA, December 1990*. IEEE Computer Society, 1990, pp. 182–190.

[BRH90] Sanjoy K. Baruah, Louis E. Rosier, and Rodney R. Howell. "Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor." In: *Real-Time Systems* 2.4 (1990), pp. 301–324.

[BV08] Sanjoy K. Baruah and Steve Vestal. "Schedulability Analysis of Sporadic Tasks with Multiple Criticality Specifications." In: *20th Euromicro Conference on Real-Time Systems, ECRTS 2008, 2-4 July 2008, Prague, Czech Republic, Proceedings*. IEEE Computer Society, 2008, pp. 147–155.

[Bau05] Robert C. Baumann. "Radiation-induced soft errors in advanced semiconductor technologies." In: *IEEE Transactions on Device and Materials Reliability* 5.3 (2005), pp. 305–316.

[BAMCG16] Slim Ben-Amor, Dorin Maxim, and Liliana Cucu-Grosjean. "Schedulability analysis of dependent probabilistic real-time tasks." In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems, RTNS 2016, Brest, France, October 19-21, 2016*. 2016, pp. 99–107.

[BC07b] Marko Bertogna and Michele Cirinei. "Response-Time Analysis for Globally Scheduled Symmetric Multiprocessor Platforms." In: *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS 2007), 3-6 December 2007, Tucson, Arizona, USA*. IEEE Computer Society, 2007, pp. 149–160.

[BCL05a] Marko Bertogna, Michele Cirinei, and Giuseppe Lipari. "Improved Schedulability Analysis of EDF on Multiprocessor Platforms." In: *17th Euromicro Conference on Real-Time Systems (ECRTS 2005), 6-8 July 2005, Palma de Mallorca, Spain, Proceedings*. IEEE Computer Society, 2005, pp. 209–218.

[BCL05b] Marko Bertogna, Michele Cirinei, and Giuseppe Lipari. "New Schedulability Tests for Real-Time Task Sets Scheduled by Deadline Monotonic on Multiprocessors." In: *Principles of Distributed Systems, 9th International Conference, OPODIS 2005, Pisa, Italy, December 12-14, 2005, Revised Selected Papers*. Vol. 3974. Lecture Notes in Computer Science. Springer, 2005, pp. 306–321.

[BXM+11] Marko Bertogna, Orges Xhani, Mauro Marinoni, Francesco Esposito, and Giorgio C. Buttazzo. "Optimal Selection of Preemption Points to Minimize Preemption Overhead." In: *23rd Euromicro Conference on Real-Time Systems, ECRTS 2011, Porto, Portugal, 5-8 July, 2011*. IEEE Computer Society, 2011, pp. 217–227.

[BL92] Riccardo Bettati and Jane W.-S. Liu. "End-to-End Scheduling to Meet Deadlines in Distributed Systems." In: *ICDCS*. 1992, pp. 452–459.

[BB05] Enrico Bini and Giorgio C. Buttazzo. "Measuring the Performance of Schedulability Tests." In: *Real-Time Systems* 30.1-2 (2005), pp. 129–154.

[BB04] Enrico Bini and Giorgio C. Buttazzo. "Schedulability Analysis of Periodic Fixed Priority Systems." In: *IEEE Trans. Computers* 53.11 (2004), pp. 1462–1473.

[BBB01] Enrico Bini, Giorgio C. Buttazzo, and Giuseppe M. Buttazzo. "A Hyperbolic Bound for the Rate Monotonic Algorithm." In: *13th Euromicro Conference on Real-Time Systems (ECRTS 2001), 13-15 June 2001, Delft, The Netherlands, Proceedings*. 2001, pp. 59–66.

[BNR+09] Enrico Bini, Thi Huyen Chau Nguyen, Pascal Richard, and Sanjoy K. Baruah. "A Response-Time Bound in Fixed-Priority Scheduling with Arbitrary Deadlines." In: *IEEE Trans. Computers* 58.2 (2009), pp. 279–286.

[BPD15] Enrico Bini, Andrea Parri, and Giacomo Dossena. "A Quadratic-Time Response Time Upper Bound with a Tightness Property." In: *2015 IEEE Real-Time Systems Symposium, RTSS 2015, San Antonio, Texas, USA, December 1-4, 2015*. IEEE Computer Society, 2015, pp. 13–22.

[BBS15] Alessandro Biondi, Giorgio C. Buttazzo, and Stefano Simoncelli. "Feasibility Analysis of Engine Control Tasks under EDF Scheduling." In: *27th Euromicro Conference on Real-Time Systems, ECRTS 2015, Lund, Sweden, July 8-10, 2015*. IEEE Computer Society, 2015, pp. 139–148.

[BNB16] Alessandro Biondi, Marco Di Natale, and Giorgio C. Buttazzo. "Performance-Driven Design of Engine Control Tasks." In: *7th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2016, Vienna, Austria, April 11-14, 2016*. IEEE Computer Society, 2016, 45:1–45:10.

[BNB15] Alessandro Biondi, Marco Di Natale, and Giorgio C. Buttazzo. "Response-time analysis for real-time tasks in engine control applications." In: *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems, ICCPS 2015, Seattle, WA, USA, April 14-16, 2015*. ACM, 2015, pp. 120–129.

[BS18] Alessandro Biondi and Youcheng Sun. "On the ineffectiveness of 1/m-based interference bounds in the analysis of global EDF and FIFO scheduling." In: *Real-Time Systems* 54.3 (2018), pp. 515–536.

[Ble07] Konstantinos Bletsas. "Worst-case and Best-case Timing Analysis for Real-time Embedded Systems with Limited Parallelism." PhD thesis. Dept of Computer Science, University of York, 2007.

[BA05] Konstantinos Bletsas and Neil C. Audsley. "Extended Analysis with Reduced Pessimism for Systems with Limited Parallelism." In: *11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2005), 17-19 August 2005, Hong Kong, China*. IEEE Computer Society, 2005, pp. 525–531.

[BLB+07] Aaron Block, Hennadiy Leontyev, Björn B. Brandenburg, and James H. Anderson. "A Flexible Real-Time Locking Protocol for Multiprocessors." In: *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007), 21-24 August 2007, Daegu, Korea*. IEEE Computer Society, 2007, pp. 47–56.

[BM12] Vincenzo Bonifaci and Alberto Marchetti-Spaccamela. "Feasibility Analysis of Sporadic Real-Time Multiprocessor Task Systems." In: *Algorithmica* 63.4 (2012), pp. 763–780.

[Bos91] Bosch. *Controller Area Network Specification 2.0*. 1991.

[Bra14a] Björn B. Brandenburg. "Blocking Optimality in Distributed Real-Time Locking Protocols." In: *LITES* 2 (2014), 01:1–01:22.

[Bra13] Björn B. Brandenburg. "Improved analysis and evaluation of real-time semaphore protocols for P-FP scheduling." In: *19th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2013, Philadelphia, PA, USA, April 9-11, 2013*. IEEE Computer Society, 2013, pp. 141–152.

[Bra11] Björn B. Brandenburg. "Scheduling and locking in multiprocessor real-time operating systems." PhD thesis. University of North Carolina at Chapel Hill, 2011.

[Bra14b] Björn B. Brandenburg. "The FMLP+: An Asymptotically Optimal Real-Time Locking Protocol for Suspension-Aware Analysis." In: *Euromicro Conference on Real-Time Systems (ECRTS)*. 2014, pp. 61–71.

[BA10] Björn B. Brandenburg and James H. Anderson. "Optimality Results for Multiprocessor Real-Time Locking." In: *Real-Time Systems Symposium (RTSS)*. 2010, pp. 49–60.

[BA13] Björn B. Brandenburg and James H. Anderson. "The OMLP family of optimal multiprocessor real-time locking protocols." In: *Design Autom. for Emb. Sys.* 17.2 (2013), pp. 277–342.

[BCB+08] Björn B. Brandenburg, John M. Calandrino, Aaron Block, Hennadiy Leontyev, and James H. Anderson. "Real-Time Synchronization on Multiprocessors: To Block or Not to Block, to Suspend or Spin?" In: *Proceedings of the 14th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2008, April 22-24, 2008, St. Louis, Missouri, USA*. IEEE Computer Society, 2008, pp. 342–353.

[BG16] Björn B. Brandenburg and Mahircan Gul. "Global Scheduling Not Required: Simple, Near-Optimal Multiprocessor Real-Time Scheduling with Semi-Partitioned Reservations." In: *2016 IEEE Real-Time Systems Symposium, RTSS 2016, Porto, Portugal, November 29 - December 2, 2016*. IEEE Computer Society, 2016, pp. 99–110.

[BFR71] Paul Bratley, Michael Florian, and Pierre Robillard. "Scheduling with earliest start and due date constraints." In: *Naval Research Logistics Quarterly* 18.4 (1971), pp. 511–519.

[BLV07] Reinder J. Bril, Johan J. Lukkien, and Wim F. J. Verhaegh. "Worst-Case Response Time Analysis of Real-Time Tasks under Fixed-Priority Scheduling with Deferred Preemption Revisited." In: *19th Euromicro Conference on Real-Time Systems, ECRTS'07, 4-6 July 2007, Pisa, Italy, Proceedings*. IEEE Computer Society, 2007, pp. 269–279.

[BLV09] Reinder J. Bril, Johan J. Lukkien, and Wim F. J. Verhaegh. "Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption." In: *Real-Time Systems* 42.1-3 (2009), pp. 63–119.

[BCH15] Georg von der Brüggen, Jian-Jia Chen, and Wen-Hung Huang. "Schedulability and Optimization Analysis for Non-preemptive Static Priority Scheduling Based on Task Utilization and Blocking Factors." In: *27th Euromicro Conference on Real-Time Systems, ECRTS 2015, Lund, Sweden, July 8-10, 2015*. 2015, pp. 90–101.

[BCH+16] Georg von der Brüggen, Kuan-Hsun Chen, Wen-Hung Huang, and Jian-Jia Chen. "Systems with Dynamic Real-Time Guarantees in Uncertain and Faulty Execution Environments." In: *2016 IEEE Real-Time Systems Symposium, RTSS 2016, Porto, Portugal, November 29 - December 2, 2016*. 2016, pp. 303–314.

[BHC+16] Georg von der Brüggen, Wen-Hung Huang, Jian-Jia Chen, and Cong Liu. "Uniprocessor Scheduling Strategies for Self-Suspending Task Systems." In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems, RTNS 2016, Brest, France, October 19-21, 2016*. 2016, pp. 119–128.

[BCD+17] Georg von der Brüggen, Jian-Jia Chen, Robert I. Davis, and Wen-Hung Huang. "Exact speedup factors for linear-time schedulability tests for fixed-priority preemptive and non-preemptive scheduling." In: *Information Processing Letters* 117 (2017).

[BHC17] Georg von der Brüggen, Wen-Hung Huang, and Jian-Jia Chen. "Hybrid self-suspension models in real-time embedded systems." In: *23rd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2017, Hsinchu, Taiwan, August 16-18, 2017*. 2017, pp. 1–9.

[BUC+17] Georg von der Brüggen, Niklas Ueter, Jian-Jia Chen, and Matthias Freier. "Parametric utilization bounds for implicit-deadline periodic tasks in automotive systems." In: *Proceedings of the 25th International Conference on Real-Time Networks and Systems, RTNS 2017, Grenoble, France, October 04 - 06, 2017*. 2017, pp. 108–117.

[BCH+17] Georg von der Brüggen, Jian-Jia Chen, Wen-Hung Huang, and Maolin Yang. "Release enforcement in resource-oriented partitioned scheduling for multiprocessor systems." In: *Proceedings of the 25th International Conference on Real-Time Networks and Systems, RTNS 2017, Grenoble, France, October 04 - 06, 2017*. 2017, pp. 287–296.

[BPC+18] Georg von der Brüggen, Nico Piatkowski, Kuan-Hsun Chen, Jian-Jia Chen, and Katharina Morik. "Efficiently Approximating the Probability of Deadline Misses in Real-Time Systems." In: *30th Euromicro Conference on Real-Time Systems, ECRTS 2018, July 3-6, 2018, Barcelona, Spain*. 2018, 6:1–6:22.

[BSC18] Georg von der Brüggen, Lea Schönberger, and Jian-Jia Chen. "Do Nothing, But Carefully: Fault Tolerance with Timing Guarantees for Multiprocessor Systems Devoid of Online Adaptation." In: *23rd IEEE Pacific Rim International Symposium on Dependable Computing, PRDC 2018, Taipei, Taiwan, December 4-7, 2018*. 2018, pp. 1–10.

[BLO+95] Almut Burchard, Jörg Liebeherr, Yingfeng Oh, and Sang Hyuk Son. "New Strategies for Assigning Real-Time Tasks to Multiprocessor Systems." In: *IEEE Trans. Computers* 44.12 (1995), pp. 1429–1442.

[BBT15] Artem Burmyakov, Enrico Bini, and Eduardo Tovar. "An exact schedulability test for global FP using state space pruning." In: *Proceedings of the 23rd International Conference on Real Time Networks and Systems, RTNS 2015, Lille, France, November 4-6, 2015*. ACM, 2015, pp. 225–234.

[Bur94] Alan Burns. "Preemptive Priority-Based Scheduling: An Appropriate Engineering Approach." In: *Advances in Real-Time Systems, chapter 10*. Prentice Hall, 1994, pp. 225–248.

[BB11b] Alan Burns and Sanjoy K. Baruah. "Timing Faults and Mixed Criticality Systems." In: *Dependable and Historic Computing - Essays Dedicated to Brian Randell on the Occasion of His 75th Birthday*. Vol. 6875. Lecture Notes in Computer Science. Springer, 2011, pp. 147–166.

[BB13] Alan Burns and Sanjoy K. Baruah. "Towards A More Practical Model for Mixed Criticality Systems." In: *Proc. WMC, RTSS*. 2013, pp. 1–6.

[BB06] Alan Burns and Gordon Baxter. "Time Bands in Systems Structure." In: *Structure for Dependability: Computer-Based Systems*. Jan. 2006, pp. 74–88.

[BD18] Alan Burns and Robert I. Davis. "A Survey of Research into Mixed Criticality Systems." In: *ACM Comput. Surv.* 50.6 (2018), 82:1–82:37.

[BDW+12] Alan Burns, Robert I. Davis, P. Wang, and Fengxiang Zhang. "Partitioned EDF scheduling for multiprocessors using a C=D task splitting scheme." In: *Real-Time Systems* 48.1 (2012), pp. 3–33.

[BW13] Alan Burns and Andy J. Wellings. "A Schedulability Compatible Multiprocessor Resource Sharing Protocol - MrsP." In: *Euromicro Conference on Real-Time Systems (ECRTS)*. 2013, pp. 282–291.

[But11] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications, Third Edition*. Real-Time Systems Series. Springer, 2011. ISBN: 978-1-4614-0675-4.

[BBY13] Giorgio C. Buttazzo, Marko Bertogna, and Gang Yao. "Limited Preemptive Scheduling for Real-Time Systems. A Survey." In: *IEEE Trans. Industrial Informatics* 9.1 (2013), pp. 3–15.

[BBB14] Giorgio C. Buttazzo, Enrico Bini, and Darren Buttle. "Rate-adaptive tasks: Model, analysis, and design issues." In: *Design, Automation & Test in Europe Conference & Exhibition, DATE 2014, Dresden, Germany, March 24-28, 2014*. European Design and Automation Association, 2014, pp. 1–6.

[But12] Darren Buttle. "Real-time in the prime time." Keynote talk at the Euromicro Conference on Real-Time Systems (ECRTS). 2012.

[CKT02] Samarjit Chakraborty, Simon Künzli, and Lothar Thiele. "Approximate Schedulability Analysis." In: *IEEE Real-Time Systems Symposium*. 2002, pp. 159–168.

[Che16a] Jian-Jia Chen. "Computational Complexity and Speedup Factors Analyses for Self-Suspending Tasks." In: *RTSS*. IEEE Computer Society, 2016, pp. 327–338.

[Che16b] Jian-Jia Chen. "Federated scheduling admits no constant speedup factors for constrained-deadline DAG task systems." In: *Real-Time Systems* 52.6 (2016), pp. 833–838.

[Che16c] Jian-Jia Chen. "Partitioned Multiprocessor Fixed-Priority Scheduling of Sporadic Real-Time Tasks." In: *28th Euromicro Conference on Real-Time Systems, ECRTS 2016, Toulouse, France, July 5-8, 2016*. IEEE Computer Society, 2016, pp. 251–261.

[CBC+18] Jian-Jia Chen, Nikhil Bansal, Samarjit Chakraborty, and Georg von der Brüggen. "Packing Sporadic Real-Time Tasks on Identical Multiprocessor Systems." In: *29th International Symposium on Algorithms and Computation, ISAAC 2018, December 16-19, 2018, Jiaoxi, Yilan, Taiwan*. Vol. 123. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018, 71:1–71:14.

[CB17] Jian-Jia Chen and Björn B. Brandenburg. "A Note on the Period Enforcer Algorithm for Self-Suspending Tasks." In: *LITES* 4.1 (2017), 01:1–01:22.

[CBH+17a] Jian-Jia Chen, Georg von der Brüggen, Wen-Hung Huang, and Robert I. Davis. "On the Pitfalls of Resource Augmentation Factors and Utilization Bounds in Real-Time Scheduling." In: *29th Euromicro Conference on Real-Time Systems, ECRTS 2017, June 27-30, 2017, Dubrovnik, Croatia*. 2017, 9:1–9:25.

[CBH+17b] Jian-Jia Chen, Georg von der Brüggen, Wen-Hung Huang, and Cong Liu. "State of the art for scheduling and analyzing self-suspending sporadic real-time tasks." In: *23rd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2017, Hsinchu, Taiwan, August 16-18, 2017*. IEEE Computer Society, 2017, pp. 1–10.

[CBS+18] Jian-Jia Chen, Georg von der Brüggen, Junjie Shi, and Niklas Ueter. "Dependency Graph Approach for Multiprocessor Real-Time Synchronization." In: *2018 IEEE Real-Time Systems Symposium, RTSS 2018, Nashville, TN, USA, December 11-14, 2018*. IEEE Computer Society, 2018, pp. 434–446.

[CBU18] Jian-Jia Chen, Georg von der Brüggen, and Niklas Ueter. "Push Forward: Global Fixed-Priority Scheduling of Arbitrary-Deadline Sporadic Task Systems." In: *30th Euromicro Conference on Real-Time Systems, ECRTS 2018, July 3-6, 2018, Barcelona, Spain*. Vol. 106. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018, 8:1–8:24.

[CC11] Jian-Jia Chen and Samarjit Chakraborty. "Resource Augmentation Bounds for Approximate Demand Bound Functions." In: *Proceedings of the 32nd IEEE Real-Time Systems Symposium, RTSS 2011, Vienna, Austria, November 29 - December 2, 2011*. 2011, pp. 272–281.

[CHH+19] Jian-Jia Chen, Tobias Hahn, Ruben Hoeksma, Nicole Megow, and Georg von der Brüggen. "Scheduling Self-Suspending Tasks: New and Old Results." In: *31st Euromicro Conference on Real-Time Systems, ECRTS 2019, July 9-12, 2019, Stuttgart, Germany*. Vol. 133. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019, 16:1–16:23.

[CHL15a] Jian-Jia Chen, Wen-Hung Huang, and Cong Liu. "Evaluate and Compare Two Utilization-Based Schedulability-Test Frameworks for Real-Time Systems." In: *CoRR* (2015). URL: https://arxiv.org/abs/1505.02155.

[CHL16a] Jian-Jia Chen, Wen-Hung Huang, and Cong Liu. "k2Q: A Quadratic-Form Response Time and Schedulability Analysis Framework for Utilization-Based Analysis." In: *Real-Time Systems Symposium, RTSS*. 2016, pp. 351–362.

[CHL16b] Jian-Jia Chen, Wen-Hung Huang, and Cong Liu. "k2Q: A Quadratic-Form Response Time and Schedulability Analysis Framework for Utilization-Based Analysis." In: *2016 IEEE Real-Time Systems Symposium, RTSS*. 2016, pp. 351–362.

[CHL15b] Jian-Jia Chen, Wen-Hung Huang, and Cong Liu. "k2U: A General Framework from k-Point Effective Schedulability Analysis to Utilization-Based Tests." In: *2015 IEEE Real-Time Systems Symposium, RTSS 2015, San Antonio, Texas, USA, December 1-4, 2015*. IEEE Computer Society, 2015, pp. 107–118.

[CL14] Jian-Jia Chen and Cong Liu. "Fixed-Relative-Deadline Scheduling of Hard Real-Time Tasks with Self-Suspensions." In: *Proceedings of the IEEE 35th IEEE Real-Time Systems Symposium, RTSS 2014, Rome, Italy, December 2-5, 2014*. IEEE Computer Society, 2014, pp. 149–160.

[CNH16] Jian-Jia Chen, Geoffrey Nelissen, and Wen-Hung Huang. "A Unifying Response Time Analysis Framework for Dynamic Self-Suspending Tasks." In: *ECRTS*. IEEE Computer Society, 2016, pp. 61–71.

[CNH+19] Jian-Jia Chen, Geoffrey Nelissen, Wen-Hung Huang, Maolin Yang, Björn B. Brandenburg, Konstantinos Bletsas, Cong Liu, Pascal Richard, Frédéric Ridouard, Neil C. Audsley, Raj Rajkumar, Dionisio de Niz, and Georg von der Brüggen. "Many suspensions, many problems: a review of self-suspending tasks in real-time systems." In: *Real-Time Systems* 55.1 (2019), pp. 144–207.

[Che16d] Kuan-Hsun Chen. *#2772 ticket: Enhancement for more general real-time model*. http://devel.rtems.org/ticket/2772. 2016. URL: http://devel.rtems.org/ticket/2772.

[CBC18a] Kuan-Hsun Chen, Georg von der Brüggen, and Jian-Jia Chen. "Analysis of Deadline Miss Rates for Uniprocessor Fixed-Priority Scheduling." In: *24th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2018, Hakodate, Japan, August 28-31, 2018*. IEEE Computer Society, 2018, pp. 168–178.

[CBC16] Kuan-Hsun Chen, Georg von der Brüggen, and Jian-Jia Chen. "Overrun Handling for Mixed-Criticality Support in RTEMS." In: *WMC 2016*. Proceedings of WMC 2016. Porto, Portugal, 2016.

[CBC18b] Kuan-Hsun Chen, Georg von der Brüggen, and Jian-Jia Chen. "Reliability Optimization on Multi-Core Systems with Multi-Tasking and Redundant Multi-Threading." In: *IEEE Trans. Computers* 67.4 (2018), pp. 484–497.

[CC17] Kuan-Hsun Chen and Jian-Jia Chen. "Probabilistic schedulability tests for uniprocessor fixed-priority scheduling under soft errors." In: *12th IEEE International Symposium on Industrial Embedded Systems, SIES 2017, Toulouse, France, June 14-16, 2017*. 2017, pp. 1–8.

[CUB+19] Kuan-Hsun Chen, Niklas Ueter, Georg von der Bruggen, and Jian-Jia Chen. "Efficient Computation of Deadline-Miss Probability and Potential Pitfalls." In: *Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy, March 25-29, 2019*. IEEE, 2019, pp. 896–901.

[Dav16] Robert 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)*. July 2016.

[DB11a] Robert I. Davis and Alan Burns. "A survey of hard real-time scheduling for multiprocessor systems." In: *ACM Comput. Surv.* 43.4 (2011), 35:1–35:44.

[DB11b] Robert I. Davis and Alan Burns. "Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems." In: *Real-Time Systems* 47.1 (2011), pp. 1–40.

[DB09] Robert I. Davis and Alan Burns. "Priority Assignment for Global Fixed Priority Pre-Emptive Scheduling in Multiprocessor Real-Time Systems." In: *Proceedings of the 30th IEEE Real-Time Systems Symposium, RTSS 2009, Washington, DC, USA, 1-4 December 2009*. IEEE Computer Society, 2009, pp. 398–409.

[DB08] Robert I. Davis and Alan Burns. "Response Time Upper Bounds for Fixed Priority Real-Time Systems." In: *Proceedings of the 29th IEEE Real-Time Systems Symposium, RTSS 2008, Barcelona, Spain, 30 November - 3 December 2008*. IEEE Computer Society, 2008, pp. 407–418.

[DBB+15] Robert I. Davis, Alan Burns, Sanjoy K. Baruah, Thomas Rothvoß, Laurent George, and Oliver Gettings. "Exact comparison of fixed priority and EDF scheduling based on speedup factors for both pre-emptive and non-pre-emptive paradigms." In: *Real-Time Systems* 51.5 (2015), pp. 566–601.

[DBB+07] Robert I. Davis, Alan Burns, Reinder J. Bril, and Johan J. Lukkien. "Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised." In: *Real-Time Systems* 35.3 (2007), pp. 239–272.

[DC19a] Robert I. Davis and Liliana Cucu-Grosjean. "A Survey of Probabilistic Schedulability Analysis Techniques for Real-Time Systems." In: *LITES* 6.1 (2019), 04:1–04:53.

[DC19b] Robert I. Davis and Liliana Cucu-Grosjean. "A Survey of Probabilistic Timing Analysis Techniques for Real-Time Systems." In: *LITES* 6.1 (2019), 03:1–03:60.

[DCB+16] Robert I. Davis, Liliana Cucu-Grosjean, Marko Bertogna, and Alan Burns. "A review of priority assignment in real-time systems." In: *Journal of Systems Architecture - Embedded Systems Design* 65 (2016), pp. 64–82.

[DFP+14] Robert I. Davis, Timo Feld, Victor Pollex, and Frank Slomka. "Schedulability tests for tasks with Variable Rate-dependent Behaviour under fixed priority scheduling." In: *20th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2014, Berlin, Germany, April 15-17, 2014*. IEEE Computer Society, 2014, pp. 51–62.

[DGC10] Robert I. Davis, Laurent George, and Pierre Courbin. "Quantifying the Sub-optimality of Uniprocessor Fixed Priority Non-Pre-emptive Scheduling." In: *International Conference on Real-Time and Network Systems (RTNS'10)* (2010).

[DRB+09a] Robert I. Davis, Thomas Rothvoß, Sanjoy K. Baruah, and Alan Burns. "Exact quantification of the sub-optimality of uniprocessor fixed priority pre-emptive scheduling." In: *Real-Time Systems* 43.3 (2009), pp. 211–258.

[DRB+09b] Robert I. Davis, Thomas Rothvoß, Sanjoy K. Baruah, and Alan Burns. "Quantifying the Sub-optimality of Uniprocessor Fixed Priority Pre-emptive Scheduling for Sporadic Tasksets with Arbitrary Deadlines." In: *Real-Time Networks and Systems Conference*. 2009.

[DTG+15] Robert I. Davis, Abhilash Thekkilakattil, Oliver Gettings, Radu Dobrin, and Sasikumar Punnekkat. "Quantifying the Exact Sub-optimality of Non-preemptive Scheduling." In: *2015 IEEE Real-Time Systems Symposium, RTSS 2015, San Antonio, Texas, USA, December 1-4, 2015*. IEEE Computer Society, 2015, pp. 96–106.

[Der74] Michael L. Dertouzos. "Control Robotics: The Procedural Control of Physical Processes." In: *IFIP Congress*. 1974, pp. 807–813.

[DL78] Sudarshan K. Dhall and C. L. Liu. "On a Real-Time Scheduling Problem." In: *Operations Research* 26.1 (1978), pp. 127–140.

[DGK+02] José Luis Díaz, Daniel F. García, Kanghee Kim, Chang-Gun Lee, Lucia Lo Bello, José María López, Sang Lyul Min, and Orazio Mirabella. "Stochastic Analysis of Periodic Real-Time Systems." In: *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02), Austin, Texas, USA, December 3-5, 2002*. 2002, pp. 289–300.

[DLB+18] Zheng Dong, Cong Liu, Soroush Bateni, Kuan-Hsun Chen, Jian-Jia Chen, Georg von der Brüggen, and Junjie Shi. "Shared-Resource-Centric Limited Preemptive Scheduling: A Comprehensive Study of Suspension-Based Partitioning Approaches." In: *IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2018, 11-13 April 2018, Porto, Portugal*. IEEE Computer Society, 2018, pp. 164–176.

[EA09] Arvind Easwaran and Björn Andersson. In: *Real-Time Systems Symposium (RTSS)*. 2009.

[EY17] Pontus Ekberg and Wang Yi. "Fixed-Priority Schedulability of Sporadic Tasks on Uniprocessors is NP-Hard." In: *2017 IEEE Real-Time Systems Symposium, RTSS 2017, Paris, France, December 5-8, 2017*. 2017, pp. 139–146.

[EY12] Pontus Ekberg and Wang Yi. "Outstanding Paper Award: Bounding and Shaping the Demand of Mixed-Criticality Sporadic Tasks." In: *24th Euromicro Conference on Real-Time Systems, ECRTS 2012, Pisa, Italy, July 11-13, 2012*. IEEE Computer Society, 2012, pp. 135–144.

[EY15] Pontus Ekberg and Wang Yi. "Uniprocessor Feasibility of Sporadic Tasks with Constrained Deadlines Is Strongly coNP-Complete." In: *27th Euromicro Conference on Real-Time Systems, ECRTS 2015, Lund, Sweden, July 8-10, 2015*. 2015, pp. 281–286.

[ESD10] Paul Emberson, Roger Stafford, and Robert I. Davis. "Techniques for the synthesis of multiprocessor tasksets." In: *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010)*. 2010, pp. 6–11.

[Eri14] Jeremy P. Erickson. "Managing Tardiness Bounds and Overload in Soft Real-Time Systems." PhD thesis. University of North Carolina at Chapel Hill, 2014.

[EN16] Rolf Ernst and Marco Di Natale. "Mixed Criticality Systems - A History of Misconceptions?" In: *IEEE Design & Test* 33.5 (2016), pp. 65–74.

[ENN+15] Alexandre Esper, Geoffrey Nelissen, Vincent Nélis, and Eduardo Tovar. "How realistic is the mixed-criticality real-time system model?" In: *Proceedings of the 23rd International Conference on Real Time Networks and Systems, RTNS 2015, Lille, France, November 4-6, 2015*. ACM, 2015, pp. 139–148.

[FBD+18] Timo Feld, Alessandro Biondi, Robert I. Davis, Giorgio C. Buttazzo, and Frank Slomka. "A survey of schedulability analysis techniques for rate-dependent tasks." In: *Journal of Systems and Software* 138 (2018), pp. 100–107.

[Fis07] Nathan Fisher. *The multiprocessor real-time scheduling of general task systems*. Citeseer, 2007.

[FB06] Nathan Fisher and Sanjoy K. Baruah. "Global static-priority scheduling of sporadic task systems on multiprocessor platforms." In: (Jan. 2006).

[FBB06] Nathan Fisher, Sanjoy K. Baruah, and Theodore P. Baker. "The Partitioned Scheduling of Sporadic Tasks According to Static-Priorities." In: *18th Euromicro Conference on Real-Time Systems, ECRTS'06, 5-7 July 2006, Dresden, Germany, Proceedings*. IEEE Computer Society, 2006, pp. 118–127.

[GLN01] Paolo Gai, Giuseppe Lipari, and Marco Di Natale. "Minimizing Memory Utilization of Real-Time Task Sets in Single and Multi-Processor Systems-on-a-Chip." In: *Real-Time Systems Symposium (RTSS)*. 2001, pp. 73–83.

[GNL+03] Paolo Gai, Marco Di Natale, Giuseppe Lipari, Alberto Ferrari, Claudio Gabellini, and Paolo Marceca. "A comparison of MPCP and MSRP when sharing resources in the Janus multiple-processor on a chip platform." In: *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2003), May 27-30, 2003, Toronto, Canada*. IEEE Computer Society, 2003, p. 189.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. ISBN: 0-7167-1044-7.

[GGL13] Gilles Geeraerts, Joël Goossens, and Markus Lindström. "Multiprocessor schedulability of arbitrary-deadline sporadic tasks: complexity and antichain algorithm." In: *Real-Time Systems* 49.2 (2013), pp. 171–218.

[GMR95] Laurent George, Paul Muhlethaler, and Nicolas Rivierre. *Optimality and non-preemptive real-time scheduling revisited*. Research Report RR-2516. INRIA, 1995.

[GRS96] Laurent George, Nicolas Rivierre, and Marco Spuri. *Preemptive and Non-Preemptive Real-Time UniProcessor Scheduling*. Research Report. INRIA, 1996.

[GFB03] Joël Goossens, Shelby Funk, and Sanjoy K. Baruah. "Priority-Driven Scheduling of Periodic Task Systems on Multiprocessors." In: *Real-Time Systems* 25.2-3 (2003), pp. 187–205.

[Gra69] Ronald L. Graham. "Bounds on Multiprocessing Timing Anomalies." In: *SIAM Journal of Applied Mathematics* 17.2 (1969), pp. 416–429.

[GES+11] Nan Guan, Pontus Ekberg, Martin Stigge, and Wang Yi. "Effective and Efficient Scheduling of Certifiable Mixed-Criticality Sporadic Task Systems." In: *Proceedings of the 32nd IEEE Real-Time Systems Symposium, RTSS 2011, Vienna, Austria, November 29 - December 2, 2011*. IEEE Computer Society, 2011, pp. 13–23.

[GSY+09] Nan Guan, Martin Stigge, Wang Yi, and Ge Yu. "New Response Time Bounds for Fixed Priority Multiprocessor Scheduling." In: *IEEE Real-Time Systems Symposium (RTSS)*. 2009, pp. 387–397.

[GB15] Zhishan Guo and Sanjoy K. Baruah. "Uniprocessor EDF scheduling of AVR task systems." In: *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems, ICCPS 2015, Seattle, WA, USA, April 14-16, 2015*. ACM, 2015, pp. 159–168.

[Gur] *Gurobi Optimization Inc.* http://www.gurobi.com. 2016.

[GH98] José C. Palencia Gutiérrez and Michael González Harbour. "Schedulability Analysis for Tasks with Static and Dynamic Offsets." In: *Proceedings of the 19th IEEE Real-Time Systems Symposium, Madrid, Spain, December 2-4, 1998*. IEEE Computer Society, 1998, pp. 26–37.

[HS92] Leslie A. Hall and David B. Shmoys. "Jackson's Rule for Single-Machine Scheduling: Making a Good Heuristic Better." In: *Math. Oper. Res.* 17.1 (1992), pp. 22–35.

[HDK+17] Arne Hamann, Dakshina Dasari, Simon Kramer, Michael Pressler, and Falk Wurst. "Communication Centric Design in Complex Automotive Embedded Systems." In: *29th Euromicro Conference on Real-Time Systems, ECRTS 2017, June 27-30, 2017, Dubrovnik, Croatia*. Vol. 76. LIPIcs. 2017, 10:1–10:20.

[HQE14] Zain Alabedin Haj Hammadeh, Sophie Quinton, and Rolf Ernst. "Extending typical worst-case analysis using response-time dependencies to bound deadline misses." In: *2014 International Conference on Embedded Software, EMSOFT 2014, New Delhi, India, October 12-17, 2014*. ACM, 2014, 10:1–10:10.

[HT97] Ching-Chih Han and Hung-Ying Tyan. "A better polynomial-time schedulability test for real-time fixed-priority scheduling algorithm." In: *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS '97), December 3-5, 1997, San Francisco, CA, USA*. IEEE Computer Society, 1997, pp. 36–45.

[HFB+18] Tim Harde, Matthias Freier, Georg von der Brüggen, and Jian-Jia Chen. "Configurations and Optimizations of TDMA Schedules for Periodic Packet Communication on Networks on Chip." In: *Proceedings of the 26th International Conference on Real-Time Networks and Systems, RTNS 2018, Chasseneuil-du-Poitou, France, October 10-12, 2018*. ACM, 2018, pp. 202–212.

[HCB+18] Nils Hölscher, Kuan-Hsun Chen, Georg von der Brüggen, and Jian-Jia Chen. "Examining and Supporting Multi-Tasking in EV3OSEK." In: *OSPERT 2018* (2018), p. 25.

[Hor74] W. A. Horn. "Some simple scheduling algorithms." In: *Naval Research Logistics Quarterly* 21.1 (1974), pp. 177–185.

[HV95] Rodney R. Howell and Muralidhar K. Venkatrao. "On Non-Preemptive Scheduling of Recurring Tasks Using Inserted Idle Times." In: *Inf. Comput.* 117.1 (1995), pp. 50–62.

[HLK11] Pi-Cheng Hsiu, Der-Nien Lee, and Tei-Wei Kuo. "Task synchronization and allocation for many-core real-time systems." In: *International Conference on Embedded Software, (EMSOFT)*. 2011, pp. 79–88.

[HGS+14] Pengcheng Huang, Georgia Giannopoulou, Nikolay Stoimenov, and Lothar Thiele. "Service adaptions for mixed-criticality systems." In: *19th Asia and South Pacific Design Automation Conference, ASP-DAC 2014, Singapore, January 20-23, 2014*. IEEE, 2014, pp. 125–130.

[HC15a] Wen-Hung Huang and Jian-Jia Chen. "Response Time Bounds for Sporadic Arbitrary-Deadline Tasks under Global Fixed-Priority Scheduling on Multiprocessors." In: *International Conference on Real Time Networks and Systems, RTNS*. 2015, pp. 215–224.

[HC16] Wen-Hung Huang and Jian-Jia Chen. "Self-suspension real-time tasks under fixed-relative-deadline fixed-priority scheduling." In: *2016 Design, Automation & Test in Europe Conference & Exhibition, DATE 2016, Dresden, Germany, March 14-18, 2016*. 2016, pp. 1078–1083.

[HC15b] Wen-Hung Huang and Jian-Jia Chen. "Techniques for Schedulability Analysis in Mode Change Systems under Fixed-Priority Scheduling." In: *21st IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2015, Hong Kong, China, August 19-21, 2015*. IEEE Computer Society, 2015, pp. 176–186.

[HCZ+15] Wen-Hung Huang, Jian-Jia Chen, Husheng Zhou, and Cong Liu. "PASS: priority assignment of real-time tasks with dynamic suspending behavior under fixed-priority scheduling." In: *DAC*. ACM, 2015, 154:1–154:6.

[HYC16] Wen-Hung Huang, Maolin Yang, and Jian-Jia Chen. "Resource-Oriented Partitioned Scheduling in Multiprocessor Systems: How to Partition and How to Share?" In: *2016 IEEE Real-Time Systems Symposium, RTSS 2016, Porto, Portugal, November 29 - December 2, 2016*. IEEE Computer Society, 2016, pp. 111–122.

[IEC10] IEC-61508. *Functional safety of electrical / electronic / programmable electronic safety-related systems edition 2.0*. Tech. rep. International Electrotechnical Commission (IEC), 2010. URL: http://www.iec.ch/functionalsafety/standards/page2.htm.

[ISO00] ISO-26262-1:2011. *Iso/fdis26262: Road vehicles - functional safety*. Tech. rep. International Organization for Standardization (ISO), 2000. URL: https://www.iso.org/standard/43464.html.

[Jac55] J. R. Jackson. *Scheduling a production line to minimize maximum tardiness*. Tech. rep. University of California, Los Angeles, 1955.

[JZP03] Mathieu Jan, Lilia Zaourar, and Maurice Pitel. "Maximizing the execution rate of low criticality tasks in mixed criticality system." In: *WMC*. 2103, pp. 43–48.

[JSM91] Kevin Jeffay, Donald F. Stanat, and Charles U. Martel. "On non-preemptive scheduling of period and sporadic tasks." In: *Proceedings of the Real-Time Systems Symposium - 1991, San Antonio, Texas, USA*. 1991, pp. 129–139.

[JLG+16] Xu Jiang, Xiang Long, Nan Guan, and Han Wan. "On the Decomposition-Based Global EDF Scheduling of Parallel Real-Time Tasks." In: *Real-Time Systems Symposium (RTSS)*. 2016, pp. 237–246.

[JP86] Mathai Joseph and Paritosh K. Pandya. "Finding Response Times in a Real-Time System." In: *Comput. J.* 29.5 (1986), pp. 390–395.

[KP00] Bala Kalyanasundaram and Kirk Pruhs. "Speed is as powerful as clairvoyance." In: *J. ACM* 47.4 (2000), pp. 617–643.

[KSS+07] Woochul Kang, Sang Hyuk Son, John A. Stankovic, and Mehdi Amirijoo. "I/O-Aware Deadline Miss Ratio Management in Real-Time Embedded Databases." In: *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS 2007), 3-6 December 2007, Tucson, Arizona, USA*. IEEE Computer Society, 2007, pp. 277–287.

[KY08] Shinpei Kato and Nobuyuki Yamasaki. "Portioned EDF-based scheduling on multiprocessors." In: *Proceedings of the 8th ACM & IEEE International conference on Embedded software, EMSOFT 2008, Atlanta, GA, USA, October 19-24, 2008*. ACM, 2008, pp. 139–148.

[KY07] Shinpei Kato and Nobuyuki Yamasaki. "Real-Time Scheduling with Task Splitting on Multiprocessors." In: *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007), 21-24 August 2007, Daegu, Korea*. IEEE Computer Society, 2007, pp. 441–450.

[KY09] Shinpei Kato and Nobuyuki Yamasaki. "Semi-partitioned Fixed-Priority Scheduling on Multiprocessors." In: *15th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2009, San Francisco, CA, USA, 13-16 April 2009*. IEEE Computer Society, 2009, pp. 23–32.

[KLR12] Junsung Kim, Karthik Lakshmanan, and Ragunathan Rajkumar. "Rhythmic Tasks: A New Task Model with Continually Varying Periods for Cyber-Physical Systems." In: *2012 IEEE/ACM Third International Conference on Cyber-Physical Systems, ICCPS 2012, Beijing, China, April 17-19, 2012*. IEEE Computer Society, 2012, pp. 55–64.

[KK07] Israel Koren and C. Mani Krishna. *Fault-Tolerant Systems*. 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007. ISBN: 0120885255, 9780120885251.

[KZH15] Simon Kramer, Dirk Ziegenbein, and Arne Hamann. "Real world automotive benchmarks for free." In: *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*. 2015.

[KT12] Pratyush Kumar and Lothar Thiele. "Quantifying the Effect of Rare Timing Events with Settling-Time and Overshoot." In: *Proceedings of the 33rd IEEE Real-Time Systems Symposium, RTSS 2012, San Juan, PR, USA, December 4-7, 2012*. IEEE Computer Society, 2012, pp. 149–160.

[KCL+03] Tei-Wei Kuo, Li-Pin Chang, Yu-Hua Liu, and Kwei-Jay Lin. "Efficient Online Schedulability Tests for Real-Time Systems." In: *IEEE Trans. Software Eng.* 29.8 (2003), pp. 734–751.

[KM91] Tei-Wei Kuo and Aloysius K. Mok. "Load Adjustment in Adaptive Real-Time Systems." In: *Proceedings of the Real-Time Systems Symposium - 1991, San Antonio, Texas, USA, December 1991*. IEEE Computer Society, 1991, pp. 160–170.

[LNR09] Karthik Lakshmanan, Dionisio de Niz, and Ragunathan Rajkumar. "Coordinated Task Scheduling, Allocation and Synchronization on Multiprocessors." In: *Real-Time Systems Symposium, (RTSS)*. 2009, pp. 469–478.

[LRL09] Karthik Lakshmanan, Ragunathan Rajkumar, and John P. Lehoczky. "Partitioned Fixed-Priority Preemptive Scheduling for Multi-core Processors." In: *21st Euromicro Conference on Real-Time Systems, ECRTS 2009, Dublin, Ireland, July 1-3, 2009*. IEEE Computer Society, 2009, pp. 239–248.

[LMM98a] Sylvain Lauzac, Rami G. Melhem, and Daniel Mossé. "An Efficient RMS Admission Control and Its Application to Multiprocessor Scheduling." In: *IPPS/SPDP*. 1998, pp. 511–518.

[LMM98b] Sylvain Lauzac, Rami G. Melhem, and Daniel Mossé. "Comparison of global and partitioning schemes for scheduling rate monotonic tasks on a multiprocessor." In: *10th Euromicro Conference on Real-Time Systems (ECRTS 1998), 17-19 June 1998, berlin, Germany, Proceedings*. IEEE Computer Society, 1998, pp. 188–195.

[LSP04] Chang-Gun Lee, Lui Sha, and Avinash Peddi. "Enhanced Utilization Bounds for QoS Management." In: *IEEE Trans. Computers* 53.2 (2004), pp. 187–200.

[Leh90] John P. Lehoczky. "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines." In: *Proceedings of the Real-Time Systems Symposium - 1990, Lake Buena Vista, Florida, USA, December 1990*. IEEE Computer Society, 1990, pp. 201–209.

[LSD89] John P. Lehoczky, Lui Sha, and Ye Ding. "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior." In: *Proceedings of the Real-Time Systems Symposium - 1989, Santa Monica, California, USA, December 1989*. 1989, pp. 166–171.

[LRKB77] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. "Complexity of Machine Scheduling Problems." In: *Annals of Discrete Mathematics* 1 (1977), pp. 343–362.

[LW82] Joseph Leung and Jennifer Whitehead. "On the complexity of fixed-priority scheduling of periodic real-time tasks." In: *Performance Evaluation* 2 (1982), pp. 237–250.

[LCA+14] Jing Li, Jian-Jia Chen, Kunal Agrawal, Chenyang Lu, Christopher D. Gill, and Abusayeed Saifullah. "Analysis of Federated and Global Scheduling for Parallel Real-Time Tasks." In: *26th Euromicro Conference on Real-Time Systems, ECRTS 2014, Madrid, Spain, July 8-11, 2014*. IEEE Computer Society, 2014, pp. 85–96.

[LL73] C. L. Liu and James W. Layland. "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment." In: *Journal of the ACM* 20.1 (1973), pp. 46–61.

[LSG+16] Di Liu, Jelena Spasic, Nan Guan, Gang Chen, Songran Liu, Todor Stefanov, and Wang Yi. "EDF-VD Scheduling of Mixed-Criticality Systems with Degraded Quality Guarantees." In: *IEEE Real-Time Systems Symposium, RTSS*. 2016, pp. 35–46.

[LCT+14] Wei Liu, Jian-Jia Chen, Anas Toma, Tei-Wei Kuo, and Qingxu Deng. "Computation Offloading by Using Timing Unreliable Components in Real-Time Systems." In: *Design Automation Conference (DAC)*. Vol. 39:1 – 39:6. 2014.

[LLZ+15] Yuchuan Liu, Cong Liu, Xia Zhang, Wei Gao, Liang He, and Yu Gu. "A Computation Offloading Framework for Soft Real-Time Embedded Systems." In: *Euromicro Conference on Real-Time Systems, (ECRTS)*. 2015, pp. 129–138.

[LGD+00] José María López, Manuel García, José Luis Díaz, and Daniel F. García. "Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems." In: *12th Euromicro Conference on Real-Time Systems (ECRTS 2000), 19-21 June 2000, Stockholm, Sweden, Proceedings*. IEEE Computer Society, 2000, pp. 25–33.

[LY15] David G. Luenberger and Yinyu Ye. *Linear and Nonlinear Programming*. Springer Publishing Company, Incorporated, 2015. ISBN: 3319188410, 9783319188416.

[Mar11] Peter Marwedel. *Embedded System Design - Embedded Systems Foundations of Cyber-Physical Systems, Second Edition*. Embedded Systems. Springer, 2011. ISBN: 978-94-007-0256-1.

[MC13] Dorin Maxim and Liliana Cucu-Grosjean. "Response Time Analysis for Fixed-Priority Tasks with Multiple Probabilistic Parameters." In: *Proceedings of the IEEE 34th Real-Time Systems Symposium, RTSS 2013, Vancouver, BC, Canada, December 3-6, 2013*. 2013, pp. 224–235.

[MHS+12] Dorin Maxim, Mike Houston, Luca Santinelli, Guillem Bernat, Robert I. Davis, and Liliana Cucu-Grosjean. "Re-sampling for statistical timing analysis of real-time systems." In: *20th International Conference on Real-Time and Network Systems, RTNS '12, Pont a Mousson, France - November 08 - 09, 2012*. ACM, 2012, pp. 111–120.

[MEY16] Morteza Mohaqeqi, Pontus Ekberg, and Wang Yi. "On Fixed-Priority Schedulability Analysis of Sporadic Tasks with Self-Suspension." In: *RTNS*. ACM, 2016, pp. 109–118.

[Mok83] Aloysius K. Mok. *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment*. Tech. rep. Cambridge, MA, USA, 1983.

[MC96] Aloysius K. Mok and Deji Chen. "A multiframe model for real-time tasks." In: *Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS '96), December 4-6, 1996, Washington, DC, USA*. 1996, pp. 22–29.

[MNL+10] Noel Tchidjo Moyo, Eric Nicollet, Frederic Lafaye, and Christophe Moy. "On Schedulability Analysis of Non-cyclic Generalized Multiframe Tasks." In: *22nd Euromicro Conference on Real-Time Systems, ECRTS 2010, Brussels, Belgium, July 6-9, 2010*. IEEE Computer Society, 2010, pp. 271–278.

[MWE+03] Shubhendu S. Mukherjee, Christopher T. Weaver, Joel S. Emer, Steven K. Reinhardt, and Todd M. Austin. "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor." In: *Proceedings of the 36th Annual International Symposium on Microarchitecture, San Diego, CA, USA, December 3-5, 2003*. IEEE Computer Society, 2003, pp. 29–42.

[NBF+14] Mitra Nasri, Sanjoy K. Baruah, Gerhard Fohler, and Mehdi Kargahi. "On the Optimality of RM and EDF for Non-Preemptive Real-Time Harmonic Tasks." In: *22nd International Conference on Real-Time Networks and Systems, RTNS '14, Versaille, France, October 8-10, 2014*. ACM, 2014, p. 331.

[NF16] Mitra Nasri and Gerhard Fohler. "Non-work-conserving Non-preemptive Scheduling: Motivations, Challenges, and Potential Solutions." In: *28th Euromicro Conference on Real-Time Systems, ECRTS*. 2016, pp. 165–175.

[NK14] Mitra Nasri and Mehdi Kargahi. "Precautious-RM: a predictable non-preemptive scheduling algorithm for harmonic tasks." In: *Real-Time Systems* 50.4 (2014), pp. 548–584.

[NMF16] Mitra Nasri, Morteza Mohaqeqi, and Gerhard Fohler. "Quantifying the Effect of Period Ratios on Schedulability of Rate Monotonic." In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems, RTNS 2016, Brest, France, October 19-21, 2016*. 2016, pp. 161–170.

[Nat13] National Science Foundation. "Cyber-Physical Systems (CPS)." In: http://www.nsf.gov/pubs/2013/nsf13502/nsf13502.htm (2013).

[NFR+15] Geoffrey Nelissen, José Carlos Fonseca, Gurulingesh Raravi, and Vincent Nélis. "Timing Analysis of Fixed Priority Self-Suspending Sporadic Tasks." In: *ECRTS*. IEEE Computer Society, 2015, pp. 80–89.

[NFR+17] Geoffrey Nelissen, José Fonseca, Gurulingesh Raravi, and Vincent Nélis. *Errata: Timing Analysis of Fixed Priority Self-Suspending Sporadic Tasks*. Tech. rep. CISTER-TR-170205. CISTER, ISEP, INESC-TEC, 2017.

[NNB10] Farhang Nemati, Thomas Nolte, and Moris Behnam. "Partitioning Real-Time Systems on Multiprocessors with Shared Resources." In: *Principles of Distributed Systems - International Conference, OPODIS*. 2010, pp. 253–269.

[Pat17] Risat Mahmud Pathan. "Improving the Quality-of-Service for Scheduling Mixed-Criticality Systems on Multiprocessors." In: *29th Euromicro Conference on Real-Time Systems, ECRTS 2017, June 27-30, 2017, Dubrovnik, Croatia*. Vol. 76. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017, 19:1–19:22.

[PF16] Bo Peng and Nathan Fisher. "Parameter Adaption for Generalized Multiframe Tasks and Applications to Self-Suspending Tasks." In: *22nd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2016, Daegu, South Korea, August 17-19, 2016*. 2016, pp. 49–58.

[PST+97] Cynthia A. Phillips, Clifford Stein, Eric Torng, and Joel Wein. "Optimal Time-Critical Scheduling via Resource Augmentation (Extended Abstract)." In: *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*. ACM, 1997, pp. 140–149.

[PST+02] Cynthia A. Phillips, Clifford Stein, Eric Torng, and Joel Wein. "Optimal Time-Critical Scheduling via Resource Augmentation." In: *Algorithmica* 32.2 (2002), pp. 163–200.

[PFS+13] Victor Pollex, Timo Feld, Frank Slomka, Ulrich Margull, Ralph Mader, and Gerhard Wirrer. "Sufficient real-time analysis for an engine control unit with constant angular velocities." In: *Design, Automation and Test in Europe, DATE 13, Grenoble, France, March 18-22, 2013*. EDA Consortium San Jose, CA, USA / ACM DL, 2013, pp. 1335–1338.

[Pot80] C. N. Potts. "Analysis of a Heuristic for One Machine Sequencing with Release Dates and Delivery Times." In: *Operations Research* 28.6 (1980), pp. 1436–1441.

[QHE12] Sophie Quinton, Matthias Hanke, and Rolf Ernst. "Formal analysis of sporadic overload in real-time systems." In: *2012 Design, Automation & Test in Europe Conference & Exhibition, DATE 2012, Dresden, Germany, March 12-16, 2012*. IEEE, 2012, pp. 515–520.

[QNE13] Sophie Quinton, Mircea Negrean, and Rolf Ernst. "Formal analysis of sporadic bursts in real-time systems." In: *Design, Automation and Test in Europe, DATE 13, Grenoble, France, March 18-22, 2013*. EDA Consortium San Jose, CA, USA / ACM DL, 2013, pp. 767–772.

[Rte] *RTEMS: Real-Time executive for multiprocessor systems*. http://www.rtems.com/. 2013. URL: http://www.rtems.com/.

[Raj91] Ragunathan Rajkumar. "Dealing with suspending periodic tasks." In: *IBM Thomas J. Watson Research Center* (1991).

[Raj90] Ragunathan Rajkumar. "Real-Time Synchronization Protocols for Shared Memory Multiprocessors." In: *10th International Conference on Distributed Computing Systems (ICDCS 1990), May 28 - June 1, 1990, Paris, France*. IEEE Computer Society, 1990, pp. 116–123.

[RSL88] Ragunathan Rajkumar, Lui Sha, and John P. Lehoczky. "Real-Time Synchronization Protocols for Multiprocessors." In: *Real-Time Systems Symposium (RTSS)*. 1988, pp. 259–269.

[RH10] Khaled S. Refaat and Pierre-Emmanuel Hladik. "Efficient Stochastic Analysis of Real-Time Systems via Random Sampling." In: *ECRTS*. IEEE Computer Society, 2010, pp. 175–183.

[RRC04] Frédéric Ridouard, Pascal Richard, and Francis Cottet. "Negative Results for Scheduling Independent Hard Real-Time Tasks with Self-Suspensions." In: *RTSS*. 2004, pp. 47–56.

[SSD+13] A. Sailer, S. Schmidhuber, M. Deubzer, M. Alfranseder, M. Mucha, and J. Mottok. "Optimizing the task allocation step for multi-core processors within AUTOSAR." In: *2013 International Conference on Applied Electronics*. 2013, pp. 1–6.

[SBS+19] Lea Schönberger, Georg von der Bruggen, Horst Schirmeier, and Jian-Jia Chen. "Design Optimization for Hardware-Based Message Filters in Broadcast Buses." In: *Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy, March 25-29, 2019*. IEEE, 2019, pp. 606–609.

[SHB+18] Lea Schönberger, Wen-Hung Huang, Georg von der Brüggen, Kuan-Hsun Chen, and Jian-Jia Chen. "Schedulability Analysis and Priority Assignment for Segmented Self-Suspending Tasks." In: *24th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2018, Hakodate, Japan, August 28-31, 2018*. IEEE Computer Society, 2018, pp. 157–167.

[SRL90] Lui Sha, Ragunathan Rajkumar, and John P. Lehoczky. "Priority Inheritance Protocols: An Approach to Real-Time Synchronization." In: *IEEE Trans. Computers* 39.9 (1990), pp. 1175–1185.

[SUB+19] Junjie Shi, Niklas Ueter, Georg von der Brüggen, and Jian-Jia Chen. "Multiprocessor Synchronization of Periodic Real-Time Tasks Using Dependency Graphs." In: *25th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2019, Montreal, QC, Canada, April 16-18, 2019*. Ed. by Björn B. Brandenburg. IEEE, 2019, pp. 279–292.

[SKK+02] Premkishore Shivakumar, Michael Kistler, Stephen W. Keckler, Doug Burger, and Lorenzo Alvisi. "Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic." In: *2002 International Conference on Dependable Systems and Networks (DSN 2002), 23-26 June 2002, Bethesda, MD, USA, Proceedings*. IEEE Computer Society, 2002, pp. 389–398.

[SB02] Anand Srinivasan and Sanjoy K. Baruah. "Deadline-based scheduling of periodic task systems on multiprocessors." In: *Inf. Process. Lett.* 84.2 (2002), pp. 93–98.

[SR88] J. A. Stankovic and K. Ramamritham. *Tutorial on Hard Real-Time Systems*. IEEE Computer Society Press, 1988.

[SEG+11] Martin Stigge, Pontus Ekberg, Nan Guan, and Wang Yi. "The Digraph Real-Time Task Model." In: *17th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2011, Chicago, Illinois, USA, 11-14 April 2011*. IEEE Computer Society, 2011, pp. 71–80.

[SY12] Martin Stigge and Wang Yi. "Hardness Results for Static Priority Real-Time Scheduling." In: *24th Euromicro Conference on Real-Time Systems, ECRTS 2012, Pisa, Italy, July 11-13, 2012*. IEEE Computer Society, 2012, pp. 189–198.

[SZ13] Hang Su and Dakai Zhu. "An elastic mixed-criticality task model and its scheduling algorithm." In: *Design, Automation and Test in Europe, DATE 13, Grenoble, France, March 18-22, 2013*. Ed. by Enrico Macii. EDA Consortium San Jose, CA, USA / ACM DL, 2013, pp. 147–152.

[SGW+17] Jinghao Sun, Nan Guan, Yang Wang, Qingqiang He, and Wang Yi. "Real-Time Scheduling and Analysis of OpenMP Task Systems with Tied Tasks." In: *2017 IEEE Real-Time Systems Symposium, RTSS 2017, Paris, France, December 5-8, 2017*. IEEE Computer Society, 2017, pp. 92–103.

[SL96] Jun Sun and Jane W.-S. Liu. "Synchronization Protocols in Distributed Real-Time Systems." In: *Proceedings of the 16th International Conference on Distributed Computing Systems*. 1996, pp. 38–45.

[SL14] Youcheng Sun and Giuseppe Lipari. "A Weak Simulation Relation for Real-Time Schedulability Analysis of Global Fixed Priority Scheduling Using Linear Hybrid Automata." In: *22nd International Conference on Real-Time Networks and Systems, RTNS '14, Versaille, France, October 8-10, 2014*. ACM, 2014, p. 35.

[SL16] Youcheng Sun and Giuseppe Lipari. "A pre-order relation for exact schedulability test of sporadic tasks on multiprocessor Global Fixed-Priority scheduling." In: *Real-Time Systems* 52.3 (2016), pp. 323–355.

[SLG+14] Youcheng Sun, Giuseppe Lipari, Nan Guan, and Wang Yi. "Improving the response time analysis of global fixed-priority multiprocessor scheduling." In: *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications, Chongqing, China, August 20-22, 2014*. IEEE Computer Society, 2014, pp. 1–9.

[SN18] Youcheng Sun and Marco Di Natale. "Assessing the pessimism of current multicore global fixed-priority schedulability analysis." In: *Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC 2018, Pau, France, April 09-13, 2018*. ACM, 2018, pp. 575–583.

[TS97] Hiroaki Takada and Ken Sakamura. "Schedulability of generalized multiframe task sets under static priority assignment." In: *4th International Workshop on Real-Time Computing Systems and Applications (RTCSA '97), 27-29 October 1997, Taipei, Taiwan*. IEEE Computer Society, 1997, pp. 80–86.

[TBE+15] Bogdan Tanasa, Unmesh D. Bordoloi, Petru Eles, and Zebo Peng. "Probabilistic Response Time and Joint Analysis of Periodic Tasks." In: *27th Euromicro Conference on Real-Time Systems, ECRTS 2015, Lund, Sweden, July 8-10, 2015*. 2015, pp. 235–246.

[TB94] Ken Tindell and Alan Burns. "Guaranteeing message latencies on control area network (CAN)." In: *Proceedings of the 1st International CAN Conference*. Citeseer. 1994.

[TEH+16] Sebastian Tobuschat, Rolf Ernst, Arne Hamann, and Dirk Ziegenbein. "System-level timing feasibility test for cyber-physical automotive systems." In: *11th IEEE Symposium on Industrial Embedded Systems, SIES 2016, Krakow, Poland, May 23-25, 2016*. IEEE, 2016, pp. 121–130.

[UBC+18] Niklas Ueter, Georg von der Brüggen, Jian-Jia Chen, Jing Li, and Kunal Agrawal. "Reservation-Based Federated Scheduling for Parallel Real-Time Tasks." In: *2018 IEEE Real-Time Systems Symposium, RTSS 2018, Nashville, TN, USA, December 11-14, 2018*. IEEE Computer Society, 2018, pp. 482–494.

[Ves07] Steve Vestal. "Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance." In: *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS 2007), 3-6 December 2007, Tucson, Arizona, USA*. 2007, pp. 239–243.

[WS99] Yun Wang and Manas Saksena. "Scheduling Fixed-Priority Tasks with Preemption Threshold." In: *6th International Workshop on Real-Time Computing and Applications Symposium (RTCSA '99), 13-16 December 1999, Hong Kong, China*. IEEE Computer Society, 1999, p. 328.

[WB13a] Alexander Wieder and Björn B. Brandenburg. "Efficient partitioning of sporadic real-time tasks with shared resources and spin locks." In: *International Symposium on Industrial Embedded Systems, (SIES)*. 2013, pp. 49–58.

[WB13b] Alexander Wieder and Björn B. Brandenburg. "On Spin Locks in AUTOSAR: Blocking Analysis of FIFO, Unordered, and Priority-Ordered Spin Locks." In: *Proceedings of the IEEE 34th Real-Time Systems Symposium, RTSS 2013, Vancouver, BC, Canada, December 3-6, 2013*. IEEE Computer Society, 2013, pp. 45–56.

[WEE+08] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David B. Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter P. Puschner, Jan Staschulat, and Per Stenström. "The worst-case execution-time problem - overview of methods

and survey of tools." In: *ACM Trans. Embedded Comput. Syst.* 7.3 (2008), 36:1–36:53.

[XHK+15] Wenbo Xu, Zain Alabedin Haj Hammadeh, Alexander Kröller, Rolf Ernst, and Sophie Quinton. "Improved Deadline Miss Models for Real-Time Systems Using Typical Worst-Case Analysis." In: *27th Euromicro Conference on Real-Time Systems, ECRTS 2015, Lund, Sweden, July 8-10, 2015*. IEEE Computer Society, 2015, pp. 247–256.

[YCH17] Maolin Yang, Jian-Jia Chen, and Wen-Hung Huang. "A misconception in blocking time analyses under multiprocessor synchronization protocols." In: *Real-Time Systems* 53.2 (2017), pp. 187–195.

[YWB15] Maolin Yang, Alexander Wieder, and Björn B. Brandenburg. "Global Real-Time Semaphore Protocols: A Survey, Unified Analysis, and Comparison." In: *Real-Time Systems Symposium (RTSS)*. 2015, pp. 1–12.

[YBB09] Gang Yao, Giorgio C. Buttazzo, and Marko Bertogna. "Bounding the Maximum Length of Non-preemptive Regions under Fixed Priority Scheduling." In: *15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2009, Beijing, China, 24-26 August 2009*. IEEE Computer Society, 2009, pp. 351–360.

[YBB10] Gang Yao, Giorgio C. Buttazzo, and Marko Bertogna. "Feasibility Analysis under Fixed Priority Scheduling with Fixed Preemption Points." In: *16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2010, Macau, SAR, China, 23-25 August 2010*. IEEE Computer Society, 2010, pp. 71–80.

[YHL04] Wenci Yu, Han Hoogeveen, and Jan Karel Lenstra. "Minimizing Makespan in a Two-Machine Flow Shop with Delays and Unit-Time Operations is NP-Hard." In: *J. Scheduling* 7.5 (2004), pp. 333–348.

INDEX

NOTATION

General Task and Task Set Parameters		
$\mathbf{T} = \{\tau_1, \ldots, \tau_n\}$	a set of n sporadic (or periodic) tasks	Ch. 2.1
n	number of tasks	Ch. 2.1
$\tau_i = (C_i, D_i, T_i)$	a sporadic or periodic real-time task τ_i	Ch. 2.1
C_i	worst-case execution time (WCET) of τ_i	Ch. 2.1
D_i	relative deadline of τ_i	Ch. 2.1
T_i	inter-arrival time / period of τ_i	Ch. 2.1
ϕ_i	phase of τ_i (release time of the first job)	Ch. 2.1
$U_i = \frac{C_i}{T_i}$	utilization of τ_i	Ch. 2.1
$U_{sum} = \sum_{\tau_i \in \mathbf{T}} U_i$	utilization of $\mathbf{T}$	Ch. 2.1
$E_i = \max_j \{E_{i,j}\}$	tardiness of τ_i (maximum over all jobs)	Ch. 2.2
$E_{\mathbf{T}} = \max_{\tau_i \in \mathbf{T}} \{E_i\}$	tardiness of $\mathbf{T}$ (maximum over all tasks)	Ch. 2.2
R_i	worst-case response time of τ_i	Ch. 2.1
$H = LCM_{\tau_i \in \mathbf{T}}(T_i)$	hyperperiod of $\mathbf{T}$	Ch. 2.1
$\mathbf{T}_x = \{\tau_i \in \mathbf{T} \mid T_i = x\}$	tasks in $\mathbf{T}$ with period x	Ch. 2.1
Task Instances (Jobs)		
$\tau_{i,j}$	j^{th} task instance (or job) of task τ_i	Ch. 2.1
$a_{i,j}$	release time of $\tau_{i,j}$	Ch. 2.1
$f_{i,j}$	finishing time of $\tau_{i,j}$	Ch. 2.1
$d_{i,j}$	absolute deadline of $\tau_{i,j}$	Ch. 2.1
$L_{i,j} = f_{i,j} - d_{i,j}$	lateness of $\tau_{i,j}$	Ch. 2.2
$E_{i,j} = max\{0, L_{i,j}\}$	tardiness of $\tau_{i,j}$	Ch. 2.2
Static-Priority Scheduling		
P	the priority assignment	Ch. 2.4
$P(\tau_i)$	the priority of task τ_i	Ch. 2.4
$hp(\tau_k)$	tasks with a priority higher than τ_k	Ch. 2.4
$lp(\tau_k)$	tasks with a priority lower than τ_k	Ch. 2.4
$hep(\tau_k)$	$hep(\tau_k) = hp(\tau_k) \cup \tau_k$	Ch. 2.4
Non-Preemptive Scheduling		
B_i	blocking time of τ_i	Ch. 4.1.1
$\gamma_i = \frac{B_i}{C_i}$	blocking factor of τ_i	Ch. 4.1.1
$\gamma = max_{\tau_i \in \mathbf{T}} \{\gamma_i\}$	maximum blocking factor of $\mathbf{T}$	Ch. 4.1.1
Speedup Factors		
ρ	speedup factor of a algorithm / test	Ch. 2.3
$\Omega \approx 0.56714$	transcendental equation $\Omega = ln\left(\frac{1}{\Omega}\right)$	Ch. 4.4.1
Multiprocessor Environments		
m	number of processors	Ch. 2.5
Resource Sharing		
$\mathcal{R} = \{\mathcal{R}_1, \mathcal{R}_2, ..., \mathcal{R}_r\}$	set of shared resources	Ch. 2.8
r	number of shared resources	Ch. 2.8
$\tau_i = (C_i, A_i, T_i, D_i)$	resource sharing task	Ch. 2.8
C_i	non-critical section WCET of τ_i	Ch. 2.8
A_i	critical section WCET of τ_i	Ch. 2.8
$C_{i,1}$	first non-critical section WCET of τ_i	Ch. 2.8
$C_{i,2}$	second non-critical section WCET of τ_i	Ch. 2.8
$U_i^C = C_i / T_i$	non-critical section utilization of τ_i	Ch. 2.8
$U_i^A = A_i / T_i$	critical section utilization of τ_i	Ch. 2.8
$U^{\mathcal{R}} = \sum_{\mathcal{R}_q \in \mathcal{R}} U^{\mathcal{R}_q}$	utilization	Ch. 2.8

Table 7.1: The notation used in this work (part 1 - general).

	Uncertain Execution Behaviour	
$(C_{i,1}, ..., C_{i,h})$	WCETs of h distinct execution modes	Ch. 2.6
h	number of distinct execution modes	Ch. 2.6
$\mathcal{M}$	set of $\lvert\mathcal{M}\rvert = h$ possible execution modes	Ch. 2.6
$\mathbb{P}_i(j)$	probability of mode j for τ_i	Ch. 2.6
C_i^N	WCET in normal mode	Ch. 2.6
C_i^A	WCET in abnormal mode	Ch. 2.6
$\mathbb{P}_i(N)$	probability of the normal mode	Ch. 2.6
$\mathbb{P}_i(A)$	probability of the abnormal mode	Ch. 2.6
U_i^N	task utilization of τ_i in normal mode	Ch. 2.6
U_i^A	task utilization of τ_i in abnormal mode	Ch. 2.6
$U_{sum}^N = \sum_{\{\tau_i \in \mathbf{T}\}} U_i^N$	task set utilization in normal mode	Ch. 2.6
$U_{sum}^A = \sum_{\{\tau_i \in \mathbf{T}\}} U_i^A$	task set utilization in abnormal mode	Ch. 2.6
E_i^N	tardiness if all jobs are executed normally	Ch. 2.6
E_i^A	tardiness if all jobs are executed abnormally	Ch. 2.6
	Systems with Dynamic Real-Time Guarantees	
$\mathbf{T}_{hard}$	timing strict tasks	Ch. 5.2
$\mathbf{T}_{soft}$	timing tolerable tasks	Ch. 5.2
$hp(\tau_k)^H := hp(\tau_k) \cap \mathbf{T}_{hard}$	higher priority tasks in $\mathbf{T}_{hard}$	Ch. 5.2
$hp(\tau_k)^S := hp(\tau_k) \cap \mathbf{T}_{hard}$	higher priority tasks in $\mathbf{T}_{soft}$	Ch. 5.2
$\mathbf{T}_p$	set of tasks assigned to processor p	Ch. 5.3.1
P_p	static-priority task order on processor p	Ch. 5.3.1
$\mathbf{T}_{p,hard}$	timing strict tasks on processor p	Ch. 5.3.1
$\mathbf{T}_{p,soft}$	timing tolerable tasks on processor p	Ch. 5.3.1
τ_s	shared task	Ch. 5.3.4
	Probability of Deadline Misses	
$\rho_{i,t} = \lceil t/T_i \rceil$	maximum number of jobs of τ_i in $[0, t)$	Ch. 5.4
$J(t) = \sum_{\tau_i \in hep(\tau_k)} \lceil t/T_i \rceil$	total number of jobs released in $[0, t)$	Ch. 5.3
S_t	maximum accumulated workload over $[0, t)$	Ch. 5.4
$\Phi_k = \max_j \{\mathbb{P}(R_{k,j} > D_k)\}$	deadline miss probability (DMP) of τ_k	Ch. 5.4
$\Phi_k = \min_{0<t\leq D_k} \mathbb{P}(S_t > t)$	DMP of τ_k under the critical instant	Ch. 5.4
$\mathbb{P}(S_t > t)$	overload probability at time t	Ch. 5.4
$\boldsymbol{X}(t)$	random variable representing the possible execution modes of all jobs in $[0, t)$	Ch. 5.3
$\mathcal{X}(t)$	state space of $\boldsymbol{X}(t)$ with $\mathcal{X}(t) = \mathcal{M}^{J(t)}$	Ch. 5.3
$\boldsymbol{x} \in \mathcal{X}(t)$	concrete variable assignment for $\boldsymbol{X}(t)$ over $[0, t)$	Ch. 5.3
$\mathbb{P}(\boldsymbol{X}(t) = \boldsymbol{x})$	probability of $\boldsymbol{X}(t)$ to have assignment $\boldsymbol{x}$	Ch. 5.3
$\boldsymbol{X}_i(t)$	subset of $\boldsymbol{X}(t)$ related to τ_i	Ch. 5.3
$C_i(\boldsymbol{X}_{i,j}(t))$	WCET of j^{th} job of τ_i based on $\boldsymbol{X}_{i,j}(t)$	Ch. 5.3
$\mathbb{1}_{\{\text{expression}\}}$	indicator function, i.e., 1 if and only if expression is true, and 0 otherwise	Ch. 5.3
$\sigma(\boldsymbol{x})$	permutation of $\boldsymbol{x}$	Ch. 5.3
$\mathbb{S}_n$	set of all permutations of length n	Ch. 5.3
$[\![\boldsymbol{x}]\!]$	equivalence class of $\boldsymbol{x}$, i.e., all $\boldsymbol{x}' \in \mathcal{X}(t)$ that can be permuted into $\boldsymbol{x}$	Ch. 5.3

Table 7.2: The notation used in this work (part 2 - uncertain execution behaviour).

ABBREVIATIONS

List of Abbreviations	
DM	Deadline Monotonic scheduling
EDA	Equal Deadline Assignment
EDF	Earliest Deadline-First scheduling
FRD	Fixed-Relative-Deadline
GMF	generalized multiframe
LCM	least common multiple
-NP	non-preemptive scheduling algorithm, e.g., RM-NP
OPA	optimal priority assignment
-P	preemptive scheduling algorithm, e.g., RM-P
RM	Rate Monotonic scheduling
RTOS	real-time operation system
SEIFDA	Shortest Execution Interval First Deadline Assignment
TDA	Time Demand Analysis
WCET	worst-case execution time
WCRT	worst-case response time

Table 7.3: The abbreviations used in this work.

LIST OF FIGURES

LIST OF TABLES

8

APPENDIX

8.1 APPENDIX FOR CHAPTER 4

SECTION 4.1

Proof. Theorem 4.3 Similar to Theorem 4.2, the contrapositive is used in this proof as well, showing that if Eq. (4.10) is not satisfied, Eq. (4.22) is also not satisfied. A *linear programming* is constructed to find the minimum of $C_k^* + \sum_{i=1}^{k-1} t_i^* U_i$ to ensure Eq. (4.10) is not satisfied:

linear programming

$$\inf \quad C_k^* + \sum_{i=1}^{k-1} t_i^* U_i \tag{8.1a}$$

$$\text{s.t} \quad C_k^* + \sum_{i=1}^{k-1} t_i^* U_i + \sum_{i=1}^{j-1} t_i^* U_i > t_j^* \qquad \forall\ 1 \le j \le k \tag{8.1b}$$

$$t_j^* \ \ge \ 0 \qquad \forall\ 1 \le j \le k \tag{8.1c}$$

where $t_1^*, \ \ldots, \ t_{k-1}^*$ and C_k^* are variables and t_k^* is defined as t_k for notational brevity. Again, we replace $>$ with $\ge$.

We get $C_k^* + \sum_{i=1}^{k-1} t_i^* U_i \ \ge \ t_k^* - \sum_{i=1}^{k-1} t_i^* U_i$ when considering Eq. (8.1b) with $j \ = \ k$, thus switching to the maximization problem with $\sum_{i=1}^{k-1} t_i^* U_i$ as the objective function. Replacing $C_k^* + \sum_{i=1}^{k-1} t_i^* U_i$ with $t_k^* - \sum_{i=1}^{k-1} t_i^* U_i$ in Eq. (8.1b) leads to

$$\begin{aligned} & t_k^* \ - \ \sum_{i=1}^{k-1} t_i^* U_i \ + \ \sum_{i=1}^{j-1} t_i^* U_i \\ = \ & t_k^* \ - \ \sum_{i=j}^{k-1} t_i^* U_i \ \ge \ t_j^*, \ \forall\ 1 \le j \le k-1 \end{aligned} \tag{8.2}$$

The result is identical to the linear programming in Eq. (4.15), therefore resulting in the same optimal solution as in Theorem 4.2 with the same properties. From Eq. (8.1b) for $j = k$ we get

$$\begin{aligned} & C_k^* + \sum_{i=1}^{k-1} t_i^* U_i \geq t_k^* - \sum_{i=1}^{k-1} t_i^* U_i \\ \overset{(4.16)}{=} & t_k^* - (t_k^* - t_1^*) = t_1^* \overset{(4.18)}{=} \frac{t_k^*}{\prod_{i=1}^{k-1}(U_i + 1)} \\ \Rightarrow & \frac{C_k^* + \sum_{i=1}^{k-1} t_i^* U_i}{D_k} = \frac{1}{\prod_{i=1}^{k-1}(U_i + 1)} \end{aligned} \tag{8.3}$$

We replace C_k^* with $\widehat{C_k}$ as C_k^* is constructed as the minimum value to ensure Eq. (4.10) is not satisfied under the worst case setting of the t_i^* determined by the linear programming. Therefore, if Eq. (4.22) holds, Eq. (4.10) holds as well and the task set is schedulable. □

Proof. Theorem 4.8 For RM-NP, we only have to consider the case when $hp_2(\tau_k)$ is empty, since a task τ_l can only be in $hp_2(\tau_k)$ if $T_l = T_k$ for RM. In this case, the value for $\left(\frac{\widehat{C_k}}{D_k} + 1\right) \prod_{\tau_j \in hp_1(\tau_k)} (U_j + 1)$ only gets smaller, since $C_i > 0 \; \forall i$ and $1 + x + y < 1 + x + y + xy = (1 + x)(1 + y)$ if $x > 0$, $y > 0$. The utilization bound is proved using the Lagrange Multiplier to find the infimum $U_k + \sum_{i=1}^{k-1} U_i$ such that $((1 + \gamma) \cdot U_k + 1) \prod_{j=1}^{k-1} (U_j + 1) > 2$. We know that the infimum for $\frac{C_k}{T_k} + \sum_{i=1}^{k-1} U_i$ results from $U_1 = U_2 = \cdots = U_{k-1}$ since the arithmetic mean is larger than or equal to the geometric mean. Thus, there are only two variables U_k and U_1 to minimize $U_k + (k-1)U_1$ such that $((1 + \gamma) \cdot U_k + 1)(U_1 + 1)^{k-1} \geq 2$.

Let G be $U_k + (k-1)U_1 - \lambda \left(((1 + \gamma) \cdot U_k + 1)(U_1 + 1)^{k-1} - 2\right)$, where λ is the Lagrange Multiplier. To get the minimum $U_k + (k-1)U_1$, we consider the first derivative:

$$\frac{\partial G}{\partial U_1} = (k-1) - \lambda(k-1)\left((1 + \gamma) \cdot U_k + 1\right)(U_1 + 1)^{k-2} = 0$$

$$\frac{\partial G}{\partial U_k} = 1 - \lambda(1 + \gamma)(U_1 + 1)^{k-1} = 0$$

Therefore, $\lambda = \frac{1}{(1+\gamma)(U_1+1)^{k-1}}$. Hence, as a result of the Lagrange Multiplier, the above non-linear programming is minimized when U_1 is $U_k + \frac{1}{1+\gamma} - 1$ and

$$2 = \left((\gamma + 1)U_k + 1\right)(U_1 + 1)^{k-1} = (1 + \gamma)\left(U_k + \frac{1}{1 + \gamma}\right)^k$$

We get $U_k = (\frac{2}{1+\gamma})^{\frac{1}{k}} - \frac{1}{1+\gamma}$ and $U_1 = (\frac{2}{1+\gamma})^{\frac{1}{k}} - 1$.

From a mathematical point of view, U_1 can be negative when $\gamma > 1$. Since $U_1 \geq 0$ by definition, we should set U_1 to 0 and U_k to $\frac{1}{1+\gamma}$ when $\gamma > 1$. By combining these two cases, we reach the conclusion of the proof. □

Proof. Theorem 4.10 The infimum $\frac{C_k}{T_k} + \sum_{i=1}^{k-1} U_i$ happens when all U_i values are the same, i.e., $U_1 = U_2 = \cdots = U_{k-1}$. Thus, there are only two variables U_k and U_1 to minimize $G = U_k + (k-1)U_1$ such that $(\frac{\gamma U_k}{1-U_k} + 1)(U_1 + 1)^{k-1} \geq 2$.

For the minimum total utilization this equation holds with equality. We denote $\ell = k - 1$ and get

$$U_k = \frac{(\frac{2}{1+U_1})^\ell - 1}{\gamma + (\frac{2}{1+U_1})^\ell - 1} = 1 - \frac{\gamma}{\gamma + (\frac{2}{1+U_1})^\ell - 1}$$

We replace U_k in G this value, hence the function has only one variable U_1. To determine the minimum value for $G = \ell U_1 + 1 - \frac{\gamma}{\gamma + (\frac{2}{1+U_1})^\ell - 1}$ we use the first derivative:

$$\begin{aligned}\frac{\partial G}{\partial U_1} &= \ell - \frac{\ell\gamma 2(1+U_1)^{-\ell-1}}{(\gamma - 1 + 2(1+U_1)^{-\ell})^2} \\ &= \ell\left[1 - \frac{\gamma 2(1+U_1)^{-\ell-1}}{(\gamma + 2(1+U_1)^{-\ell} - 1)^2}\right]\end{aligned}$$

We know that $U_1 \in [0;\ 2^{\frac{1}{\ell}} - 1]$, since if $U_1 > 2^{\frac{1}{\ell}} - 1$ then $U_k < 0$. We now show that the minimal value happens for one of the boundaries of U_1. For $U_1 = 0$ we get $\frac{\partial G}{\partial U_1}(0) = \ell(1 - \frac{2\gamma}{(\gamma+1)^2}) > 0$ as $\gamma > 0$. We further analyze the second derivative:

$$\begin{aligned}\frac{\partial^2 G}{\partial U_1} &= \frac{-\ell(-\ell-1)\gamma 2(1+U_1)^{-\ell-2}2(1+U_1)^{-\ell}}{(\gamma - 1 + 2(1+U_1)^{-\ell})^4} \\ &+ \frac{\ell\gamma 2(1+U_1)^{-\ell-1}2(\gamma - 1 + 2(1+U_1)^{-\ell})(-2\ell)(1+U_1)^{-\ell-1}}{(\gamma - 1 + 2(1+U_1)^{-\ell})^4} \\ &= \frac{2\gamma\ell(1+U_1)^{-\ell-2}[(\ell+1)(\gamma-1) + 2(\ell-1)(1+U_1)^{-\ell}]}{(\gamma - 1 + 2(1+U_1)^{-\ell})^3}\end{aligned}$$

The demoninator is always positive as $U_1 \geq 0$ and $\gamma > 0$. In the numerator the same argument holds for the multiplied term outside the bracket. The first term in the bracket is a constant and $2(\ell-1)(1+U_1)^{-\ell}$ is a decreasing function with respect to U_1. Therefore, we conclude that the second order derivative of G with respect to U_1 in the range of $[0;\ 2^{\frac{1}{\ell}} - 1]$ is either (1) always positive $\forall U_1 \in [0;\ 2^{\frac{1}{\ell}} - 1]$, (2) always negative $\forall U_1 \in [0;\ 2^{\frac{1}{\ell}} - 1]$, or (3) changing from positive to negative at a certain value U_1^* for $U_1 \in [0;\ 2^{\frac{1}{\ell}} - 1]$. For the first case the minimum happens when $U_1 = 0$. In the second case and the third case the minimum is one of the boundary conditions, since $\frac{\partial G}{\partial U_1} = 0$ happens when $\frac{\partial^2 G}{\partial U_1} < 0$, which results in a local maximum. □

SECTION 4.2

A task set with a given target utilization U_t cannot be created based on the information in Table III, Table IV, and Table V in [KZH15] directly. Empirical tested showed that a task set containing unscaled tasks with a total utilization of $\approx$ 100% contained around 1500 individually drawn random tasks. These individual random draws were very time consuming. Hence, we used a way to generate tasks withs with a given utilization more efficiently:

1. Drawing the periods of 3000 tasks according to the percentage distribution for period, i.e., the distribution in Table 4.1.
2. According to the number of tasks with this period, drawing the execution time for tasks randomly based on the related Weibull distribution..
3. Drawing the scaling factors and calculating WCETs if selected.
4. Combining the tasks to $\mathbf{T}_{base}$.
5. Shuffling $\mathbf{T}_{base}$.
6. Taking tasks from $\mathbf{T}_{base}$ until the total utilization U_{sum} of the set is larger than the target utilization U_t.
7. Finish, if U_{sum} is in $[U_t, U_t + \gamma]$ for a threshold value γ.
8. If not, discard the last task, take the next task from $\mathbf{T}_{base}$, and check if U_{sum} is in $[U_t, U_t + \gamma]$, etc.

The threshold γ depended on the utilization steps in our experiments, i.e., it was always smaller than the utilization steps, normally 0.1.

SECTION 4.5

Considering the non-critical sections of τ_i, a workload of $C_{i,1}^N + C_{i,2}^N$ must be finished with a relative deadline $\frac{T_i}{2}$ at speed $\frac{1}{2}$. The periodic activation every T_i time units results in the following necessary condition:

$$\forall 0 < t \leq \frac{T_{max}}{2}, \qquad \sum_{\tau_i} \left(\left\lfloor \frac{t + \frac{T_i}{2}}{T_i} \right\rfloor \times 2(C_{i,1}^N + C_{i,2}^N) \right) \leq m \times t \tag{8.4}$$

where T_{max} is the maximum among the periods in the task set and the factor 2 in the summation is due to the speed $\frac{1}{2}$ of the virtual processors. For the critical sections, each of the r virtual processor must be analyzed individually. For the virtual processor executing task set τ^s, i.e., the tasks that access resource $s \in [1, ..., r]$, the demand bound function of the task set is $DBF^s(t) = \sum_{\tau_i \in \tau^s} \left(\left\lfloor \frac{t + \frac{T_i}{2}}{T_i} \right\rfloor \times 2C_i^{crit} \right)$ for any $t > 0$, where the factor 2 again results from the speed of $\frac{1}{2}$ of the virtual processor. The blocking time $B^s(t)$, which is due to the shared resource s, is $B^s(t) = \max_{\forall \tau_i \in \tau^s, D_i > t}(2 \cdot C_i - \Delta)$, where $\Delta > 0$ but infinitesimally small. The exact schedulability test for uniprocessor EDP-NP by George et al. [GRS96] tests whether (i) $\sum_{\tau_i \in \tau^s} \frac{2C_i^{Crit}}{T_i} \leq 1$ and (ii) $DBF^s(t) + B^s(t) \leq t \; \forall t > 0$. For the necessary test, we only evaluated $t = \frac{T_i}{2}$ of the tasks $\tau_i \in \tau^s$ for each resource s.

8.2 APPENDIX FOR CHAPTER 6

SECTION 6.1

Here, we provide some more detailed proofs of Theorem 6.11 and Theorem 6.12 together with some additional observations.

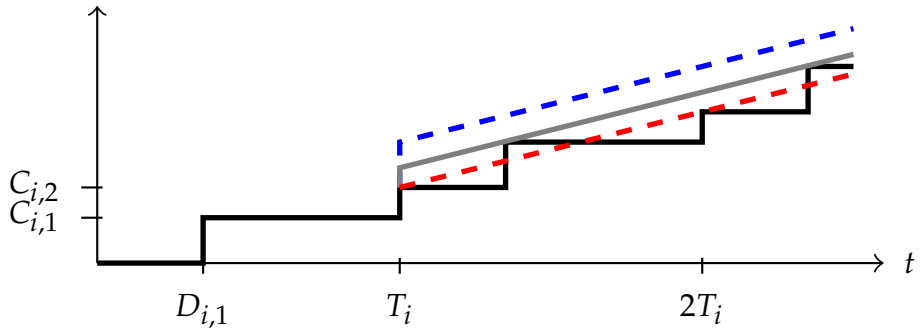


Figure 8.1: The linearized DBF for Eq. (6.22a) with $g = 1$. We use exact steps identical to Eq. (6.1) up to gT_i and linearization after gT_i. We show the linearization of the original, exact curve (black) without adjustment (red dashed, does not work), linearization after jumping by $C_{i,1}$ at gT_i (blue dashed, over approximation), and after adjusting by $D_{i,1}U_{i,1}$ at gT_i (gray, tight).
(Redraw of Figure 6.4 for readers convenience.)

Theorem 6.11: *The function* $\widehat{\text{DBF}^{frd}}_i(t, D_{i,1})$ *in Eq. (6.23) is a safe upper bound of* $\text{DBF}^{frd}(t, D_{i,1})$ *for any* $t \geq 0$ *and a specified* $D_{i,1} \leq (T_i - S_i)/2$. *Therefore, if* $\sum_{\tau_i \in \boldsymbol{T}} U_i \leq 1$ *and*

$$\sum_{\tau_i \in \boldsymbol{T}} \widehat{\text{DBF}^{frd}_i}(t, D_{i,1}) \leq t \qquad \forall t \geq 0$$

then the resulting FRD schedule is feasible. Moreover, this schedulability test can be done in $O(g|\boldsymbol{T}|^2)$ *time complexity.*

Proof. We show that Eq. (6.22a) is an over approximation of Eq. (6.1) and that Eq. (6.22b) is an over approximation of Eq. (6.2). This directly leads to the conclusion that Eq. (6.23) is an over approximation of Eq. (6.3). We only have to consider the parts that are not identical, i.e., $t \geq gT_i$ in Eq. (6.22a) and $t \geq gT_i - S_i$ in Eq. (6.22b).

In Eq. (6.22a) we consider the case where $C_{i,1}$ is released at the beginning of the period as displayed in Fig 6.4 for $g = 1$. This means $\widehat{dbf}^1_i(t, D_{i,1})$ is identical with $dbf^1_i(t, D_{i,1})$ for the first g releases of τ_i, i.e., the functions jumps by $C_{i,1}$ at $\ell T_i + D_{i,1}$ and by $C_{i,2}$ at $(\ell + 1)T_i$ for $\ell = 0, 1, 2, \ldots, g - 1$. We examine the g-th release of τ_i. The total workload created by τ_i in an interval of length T_i is C_i and thus a linear and save upper bound can be achieved by using a straight line with gradient U_i as the task is strictly periodic. Without any adjustment $U_i t$ is a save upper bound for the jump at $(g + 1)T_i + D_{i,2}$ as it happens at the end of the period (red). However, if $\frac{C_{i,1}}{D_{i,1}} > U_i$ this is not sufficient to cover the jump at $gT_i + D_{i,1}$. A simple but not tight solution is to add $C_{i,1}$ as the resulting linear function covers the case that the jump happens at gT_i instead of $gT_i + D_{i,1}$ (blue). Since the utilization created by $C_{i,1}$ in $[gT_i; gT_i + D_{i,1}]$ is $\frac{C_{i,1}}{D_{i,1}}$ we can make the linear approximation tighter by subtracting $D_{i,1}U_{i,1}$ (gray). As the tasks are released with a fixed inter arrival time T_i we know that $\widehat{dbf}^1_i(t, D_{i,1})$ is an over approximation of $dbf^1_i(t, D_{i,1})$.

In Eq. (6.22b) we upper bound the workload for the case that $C_{i,2}$ is released at time $t = 0$, i.e., the functions jumps by $C_{i,2}$ at $\ell T_i + D_{i,2}$ and by $C_{i,1}$ at $\ell T_i + T_i - S_i$ for $\ell = 0, 1, 2, \ldots, g-1$. $U_i(t + S_i)$ is the related linear approximation by a straight line starting at $-S_i$. This is covers the jumps at $\ell T_i + T_i - S_i$ as the workload in $[-S_i; T_i - S_i]$ is C_i. We have to ensure that the jump at $\ell T_i + D_{i,2}$ is covered as well. An easy and save upper bound is to use $U_i(t + S_i + D_{i,1})$, i.e., letting the straight line start $-S_i - D_{i,1}$. We tighten this approach by only adding the amount of utilization that $C_{i,2}$ contributes in an interval of length $D_{i,1}$, i.e. $C_{i,2}\frac{D_{i,1}}{T_i}$, leading to a save upper bound on Eq. (6.2) for $t \geq gT_i - S_i$.

As both Eq. (6.22a) and Eq. (6.22b) are over approximations of Eq. (6.1) and Eq. (6.2), respectively, Eq. (6.23) is an over approximation of Eq. (6.3).

We know that we only have to test the schedulability at the points in time where $\sum_{\tau_i \in \mathbf{T}} \widehat{\mathrm{DBF}_i^{\mathrm{frd}}}(t, D_{i,1})$ changes. Each task τ_i has exactly 3 jump points in each of the g periods when $\widehat{\mathrm{DBF}_i^{\mathrm{frd}}}(t, D_{i,1})$ (Eq. (6.23)) is used which leads to $3g$ discrete jump points at $\ell T_i + D_{i,1}$, $\ell T_i + T_i - S_i - D_{i,1}$, and $t = \ell T_i + T_i - S_i$ with $\ell = 0, 1, 2, \ldots, g-1$ for each $\tau_i \in \mathbf{T}$.[1] Let P be the set of all these $3g|\mathbf{T}|$ jump points of all $\tau_i \in \mathbf{T}$ and let t^* be the maximum of the points in P. It is easy to see that $\sum_{\tau_i \in \mathbf{T}} \widehat{\mathrm{DBF}_i^{\mathrm{frd}}}(t, D_{i,1})$ is a linear function for $t > t^*$. This means if $\sum_{\tau_i \in \mathbf{T}} \widehat{\mathrm{DBF}_i^{\mathrm{frd}}}(t, D_{i,1}) \leq t$ for some $t > t^*$ it holds $\forall t > t^*$, i.e., the schedulability after t^* according to the linearly approximated test can be tested by testing one $t > t^*$. In addition we must evaluate all the time points where $\sum_{\tau_i \in \mathbf{T}} \widehat{\mathrm{DBF}_i^{\mathrm{frd}}}(t, D_{i,1})$ is not linear, i.e., all points in P which are $3g\mathbf{T}$ points in total. As each test hast to calculate the workload up to the tested point for each of the $|\mathbf{T}|$ tasks this leads to $O(g|\mathbf{T}|^2)$ time complexity. □

Theorem 6.12: *For a given integer $g \geq 1$*

$$\widehat{\mathrm{DBF}_i^{frd}}(t, D_{i,1}) \leq \left(1 + \frac{1}{g}\right) \mathrm{DBF}_i^{frd}(t, D_{i,1}) \qquad \forall t \geq 0$$

Proof. We check this condition for both cases individually, i.e., for Eq. (6.22a) compared to Eq. (6.1) and for Eq. (6.22b) compared to Eq. (6.2). We only have to check directly before the jumps of Eq. (6.1) (Eq. (6.2), respectively) happen, as between jump points the exact demand bound function is not changing while the approximated BDF is constantly increasing. It is sufficient to check the first period after the beginning of the linearization as with each completed period the values of all equations are increase by C_i which only leads to a lower ratio.

When we check the ratio of Eq. (6.22a) compared to Eq. (6.1) we can use $\widehat{dbf}_i^1(gT_i + D_{i,1}, D_{i,1})$ and $\widehat{dbf}_i^1((g+1)T_i, D_{i,1})$ as upper bounds for each value $t \in [gT_i; gT_i + D_{i,1})$ and $t \in [gT_i + D_{i,1}; (g+1)T_i)$, respectively, while $dbf_i^1(gT_i, D_{i,1})$

1 The jump of Eq. (6.22a) at $(l+1)T_i$ is covered by the jump at $t = \ell T_i + T_i - S_i$ already as at both points the total workload of the DBF is $(l+1)C_i$.

and $dbf_i^1(gT_i + D_{i,1}, D_{i,1})$ are lower bounds for the values of dbf_i^1 in those intervals. We know $dbf_i^1(gT_i, D_{i,1}) = g(C_{i,1} + C_{i,2})$ while

$$\begin{aligned}\widehat{dbf}_i^1(gT_i + D_{i,1}, D_{i,1}) &= U_i(gT_i + D_{i,1}) - D_{i,1}U_{i,1} + C_{i,1} \\ &= g(C_{i,1} + C_{i,2}) + D_{i,1}U_i - D_{i,1}U_{i,1} + C_{i,1} \\ &= g(C_{i,1} + C_{i,2}) + D_{i,1}U_{i,2} + C_{i,1} < (g+1)(C_{i,1} + C_{i,2})\end{aligned}$$

as $D_{i,1}U_{i,2} < C_{i,2}$. In the other case, $dbf_i^1(gT_i + D_{i,1}, D_{i,1}) = g(C_{i,1} + C_{i,2}) + C_{i,1}$ while furthermore $\widehat{dbf}_i^1((g+1)T_i, D_{i,1}) = (g+1)(C_{i,1} + C_{i,2}) - D_{i,1}U_{i,1} + C_{i,1} < (g+1)(C_{i,1} + C_{i,2}) + C_{i,1}$. By dividing by $(g+1)$ in both cases we reach the conclusion for Eq. (6.22a).

For Eq. (6.22b) compared to Eq. (6.2) we use $\widehat{dbf}_i^2(gT_i - S_i + D_{i,2}, D_{i,1})$ and $\widehat{dbf}_i^2(gT_i + T_i - S_i, D_{i,1})$ as upper bounds for the values of $\widehat{dbf}_i^2$ in the analyzed interval compared to $dbf_i^2(gT_i - S_i, D_{i,1})$ and $dbf_i^2(gT_i - S_i + D_{i,2}, D_{i,1})$ as lower bounds for the values of dbf_i^2 in those intervals. We know that $dbf_i^2(gT_i - S_i, D_{i,1}) = g(C_{i,1} + C_{i,2})$ while

$$\begin{aligned}\widehat{dbf}_i^2(gT_i - S_i + D_{i,2}, D_{i,1}) &= U_i(gT_i + D_{i,2}) + C_{i,2}\frac{D_{i,1}}{T_i} \\ &= g(C_{i,1} + C_{i,2}) + U_i D_{i,2} + C_{i,2}\frac{D_{i,1}}{T_i} \\ &= g(C_{i,1} + C_{i,2}) + D_{i,2}U_i + D_{i,1}U_2 \\ &= g(C_{i,1} + C_{i,2}) + C_{i,2} + D_{i,1}U_1 < (g+1)(C_{i,1} + C_{i,2})\end{aligned}$$

as $D_{i,1}U_1 < C_{i,1}$. In the second case, $dbf_i^2(gT_i - S_i + D_{i,2}, D_{i,1}) = g(C_{i,1} + C_{i,2}) + C_{i,2}$ and in addition $\widehat{dbf}_i^2(gT_i + T_i - S_i, D_{i,1}) = U_i(gT_i + T_i - S_i + S_i) + C_{i,2}\frac{D_{i,1}}{T_i} < (g+1)(C_{i,1} + C_{i,2}) + C_{i,2}$ as $C_{i,2}\frac{D_{i,1}}{T_i} < C_{i,2}$. Dividing by $(g+1)$ in both cases reaches the conclusion for Eq. (6.22b). □

As a special case when g is 1, we approximate $\text{DBF}_i^{\text{frd}}(t, D_{i,1})$ by using the following function with only three dis-continuous points before $T_i - S_i$, which are at $D_{i,1}$, $D_{i,2} = T_i - S_i - D_{i,1}$, and $T_i - S_i$:

$$\text{DBF}_i^{\text{lin}}(t, D_{i,1}) = \begin{cases} 0 & \text{if } t < D_{i,1} \\ C_{i,1} & \text{if } D_{i,1} \le t < D_{i,2} \\ C_{i,1} + C_{i,2} & \text{if } D_{i,2} \le t < T_i - S_i \\ U_i(t + S_i) + C_{i,2}\frac{D_{i,1}}{T_i} & \text{if } T_i - S_i \le t, \end{cases} \tag{8.5}$$

where $D_{i,2}$ is defined as $T_i - S_i - D_{i,1}$.

Lemma 8.1. *The function $\text{DBF}_i^{lin}(t, D_{i,1})$ defined in Eq. (8.5) is a safe upper bound of $\text{DBF}^{frd}(t, D_{i,1})$ for any $t \ge 0$ and a specified $D_{i,1} \le (T_i - S_i)/2$.*

Proof. This comes from Lemma 6.11 when g is 1. □

Lemma 8.2. *Let $D_{i,1}$ for every task τ_i in $\boldsymbol{T}$ be given, where $D_{i,1} \le (T_i - S_i)/2$ and $C_{i,1} \le C_{i,2}$. Let $\boldsymbol{TD}$ be $\{D_{i,1} \mid \tau_i \in \boldsymbol{T}\} \cup \{T_i - S_i - D_{i,1} \mid \tau_i \in \boldsymbol{T}\} \cup \{T_i - S_i \mid \tau_i \in \boldsymbol{T}\}$,*

*i.e., **TD** consists of all the relative deadlines (for both computation segments) and $T_i - S_i$ of the tasks τ_i's in **T**. The resulting FRD schedule is feasible if $\sum_{i=1}^{n} U_i \leq 1$, and*

$$\sum_{i=1}^{n} \text{DBF}_i^{lin}(t, D_{i,1}) \leq t, \forall t \in \boldsymbol{TD}.$$

Proof. The proof is rather straight forward since $\text{DBF}_i^{\text{lin}}(t, D_{i,1})$ can be presented by a linear function if $t \notin \mathbf{TD}$ and $t \geq T_i - S_i$. Therefore, we only have to check whether $\sum_{i=1}^{n} \text{DBF}_i^{\text{lin}}(t, D_{i,1}) \leq t, \forall t \in \mathbf{TD}$. If this holds, the condition $\sum_{i=1}^{n} U_i \leq 1$ implies that $\sum_{i=1}^{n} \text{DBF}_i^{\text{lin}}(t, D_{i,1}) \leq t, \forall t > 0$ and $t \notin \mathbf{TD}$. Therefore, by Lemma 8.1, we reach the schedulability condition in Theorem 6.1. □