Perspectives on Quality of Service in Distributed and Embedded Real-Time Systems

Dissertation

zur Erlangung des Grades eines

D o k t o r s d e r I n g e n i e u r w i s s e n s c h a f t e n

der Technischen Universität Dortmund
an der Fakultät für Informatik

von

Lea Schönberger

Dortmund

2023

Tag der mündlichen Prüfung:	16. August 2023
Dekan:	Prof. Dr.-Ing. Gernot A. Fink
Gutachter:	Prof. Dr. Jens Teubner
	Prof. Dr. Jian-Jia Chen

Abstract

As a consequence of technological advancements, a trend towards the development of smart cities has emerged, i.e., towards urban areas that comprise a multitude of sensors, actuators as well as computation and communication resources. Being integrated into buildings, infrastructure elements, and other objects, these components constitute a large and heterogeneous distributed hardware platform. Traffic participants and other actors of a smart city can use this platform in an on-demand fashion to make use of advanced functionalities such as, for instance, smart means of transportation. In fact, vehicles of different levels of autonomy rely on a smart city's distributed infrastructure when performing sophisticated operations that come with specific quality of service (QoS) requirements, including a multitude of parameters such as timing and reliability constraints. Against the background of a shared, heterogeneous hardware infrastructure, however, guaranteeing the satisfaction of QoS requirements and, thus, ensuring the operations' correctness is an intricate matter.

This dissertation addresses selected challenges arising in the context of smart cities, focusing on the underlying distributed system as well as on individual systems interacting with it. All challenges contemplated are related to the notion of quality of service and aim to either guarantee the satisfaction of applications' QoS requirements or to enable the system(s) to enhance the level of service provided to (specific types of) applications. Concretely, a concept of QoS contracts concluded between the distributed system and each executed application is proposed that allows to provide QoS guarantees and, moreover, to detect contract violations. An extension of this concept including applications with robustness requirements is provided as well. For individual systems, focusing especially on smart vehicles, recovery protocols are proposed that enable the system to safely offload parts of critical applications to a smart city's distributed system, even under unreliable connections, while ensuring the temporal correctness. In addition, an approach for the optimization of hardware message filters in controller area network is proposed by means of which the overhead due to unnecessary message inspection can be reduced, allowing to spend the saved resource capacity on the execution of other applications. All concepts and approaches contributed in this dissertation have been evaluated and shown to be effective.

Acknowledgments

This dissertation is dedicated to my mother Ruth Schönberger, who, by fate, was not given the chance to witness the completion of my PhD. I thank her as well as my brother Jonas Schönberger and my sister Rabea Schönberger for their incredible and unconditional support.

At a prominent position, I would like to express my gratitude to Jian-Jia Chen, not only for hiring me in 2017 and allowing me to pursue a PhD degree, but also for showing me the exciting world of real-time systems and, thus, for averting me from leaving computer science for the humanities. Thereon, I would like to thank my supervisors Jens Teubner and Selma Saidi for taking me under their wing in 2020 and giving me the opportunity to complete my PhD. I appreciate their effort, patience, and openness to sharing their knowledge and letting me learn from them. Being part of the final step of my PhD by serving in the examination committee, I thank Jakob Rehof, Jens Teubner, Jian-Jia Chen, and Klaus-Tycho Förster for investing their time.

Many people have been involved in my PhD – people with whom I collaborated, people who have enriched my everyday work life as colleagues, who have shared the same dedication to common objectives, or who have simply been companions on the path towards the same goal. Since listing and expressing gratitude to all of them would fill pages, I would like to take the liberty of thanking a few selected individuals, without claiming completeness and without implying any particular order. To begin with, I thank all of my collaborators during my PhD for their time and effort invested in conducting research with me, in particular, Susanne Graf and Georg von der Brüggen. I also thank my mentor Petra Wiederkehr for her support and her always very helpful feedback. A person without which completing this dissertation would have been even harder is my colleague Jan Mühlig, whom I thank for our PhD-completion self-help group, for supporting me in solving dissertation-related problems, and for the "don't panic". Very importantly, I thank Claudia Graute for her support, her optimism, her enthusiasm, and her encouragement. Moreover, I thank Helena Kotthaus, with whom I had the pleasure of sharing an office for a period of time, for her positive influence on me and for pushing me into the right direction. I thank Horst Schirmeier for listening to my complaints, problems, and ideas and for always giving good advice. Likewise, I thank Kuan-Hsun Chen for constantly answering my random questions without complaints and for his advice in various situations. I also thank Alexander Lochmann for his advice and for the valuable

exchange (and for providing the best office dog ever). In addition, I express my gratitude to the strong and smart women of our women-in-computer-science network, particularly mentioning Meliha Sezgin, Christin Schumacher, Carina Newen, Clara Scherbaum, and Ines Heining. I appreciate the exchange of ideas, their support, and our collective effort in challenging an (academic) world where gender equality often remains a mere concept on paper. Not to be left unconsidered are Nadine Finke-Micheel, Benjamin Brast, and Anke Kujawski from TU Dortmund University's Graduate Center, whom I thank for their extensive support in many respects that essentially contributed to my PhD. Finally, I would like to thank a few unnamed people related to an unnamed university's humanities department for enriching my life in the last years and for providing a counterbalance to my PhD that allowed me to replenish my strength. If you wonder if you are included, you are included.

Parts of the research included in this thesis have received funding from Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876, projects A2 and A3, as well as from the French Embassy in Germany in the context of a Procope mobility grant.

List of Publications

A number of ideas and results presented in this dissertation have been published in the proceedings of international conferences:

Lea Schönberger, Susanne Graf, Selma Saidi, Dirk Ziegenbein, and Arne Hamann. "Contract-Based Quality-of-Service Assurance in Dynamic Distributed Systems". In: *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2022, pp. 132-135. DOI: 10.23919/DATE54114.2022.9774529 [SGS+22]

Lea Schönberger, Georg von der Brüggen, Kuan-Hsun Chen, Benjamin Sliwa, Hazem Youssef, Aswin Karthik Ramachandran Venkatapathy, Christian Wietfeld, Michael ten Hompel, and Jian-Jia Chen. "Offloading Safety- and Mission-Critical Tasks via Unreliable Connections". In: *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*. Ed. by M. Völp. Vol. 165. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 18:1–18:22. DOI: 10.4230/LIPIcs.ECRTS.2020.18 [SBC+20]

Lea Schönberger, Georg von der Brüggen, Horst Schirmeier and Jian-Jia Chen. "Design Optimization for Hardware-Based Message Filters in Broadcast Buses". In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 606-609. DOI: 10.23919/DATE.2019.8714793 [SBS+19]

The author of this dissertation also contributed to the following works that are out of its scope and have not been included:

Lea Schönberger, Mohammad Hamad, Javier Velasquez Gomez, Sebastian Steinhorst, and Selma Saidi. "Towards an Increased Detection Sensitivity of Time-Delay Attacks on Precision Time Protocol". In: *IEEE Access* 9 (2021), pp. 157398-157410. DOI: 10.1109/ACCESS.2021.3127852 [SHG+21]

Sebastian Schwitalla, **Lea Schönberger**, and Jian-Jia Chen. "Priority-Preserving Optimization of Status Quo ID-Assignments in Controller Area Network". In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2020, pp. 834-839. DOI: 10.23919/ DATE48585.2020.9116565 [SSC20]

Helena Kotthaus, **Lea Schönberger**, Andreas Lang, Jian-Jia Chen, and Peter Marwedel. "Can Flexible Multi-Core Scheduling Help to Execute Machine Learning Algorithms Resource-Efficiently?". In: *Proceedings of the 22nd International Workshop on Software and Compilers for Embedded Systems*. SCOPES '19. Sankt Goar, Germany: Association for Computing Machinery, 2019, pp. 59–62. DOI: 10.1145/3323439.3323986 [KSL+19]

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: *2018 IEEE 23rd Pacific Rim International Symposium on Dependable Computing (PRDC)*. 2018, pp. 1-10. DOI: 10.1109/PRDC.2018.00010 [BSC18]

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: *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. 2018, pp. 157-167. DOI: 10.1109/RTCSA.2018.00027 [SHV+18]

Contents

Part I

Fundamentals

CHAPTER 1

Introduction

Contents

1.1 Selected Challenges Arising in a Connected World

In recent years, a trend towards the development of smart cities has emerged, i.e., towards urban areas that comprise a multitude of sensors, actuators as well as computation and communication resources. Being integrated into buildings, infrastructure elements, and other objects, these components constitute a large and heterogeneous distributed hardware platform [HKS+19]. To improve their inhabitants' and visitors' quality of life, smart cities use this distributed system to provide advanced functionalities [KYT+20] such as, for instance, smart means of transportation. In this context, enhancing vehicles with sensors, computing devices, and communication interfaces, is a promising strategy for making transportation more comfortable, more efficient, and more safe [HKS+19].

Following from the evolution of wireless communication technologies [LXC+19], vehicles of different levels of autonomy [SZ18] rely on a smart city's distributed infrastructure when performing sophisticated operations that come with specific *quality of service (QoS)* requirements, including, among others, timing and reliability constraints. Guaranteeing, however, the satisfaction of QoS requirements and thus

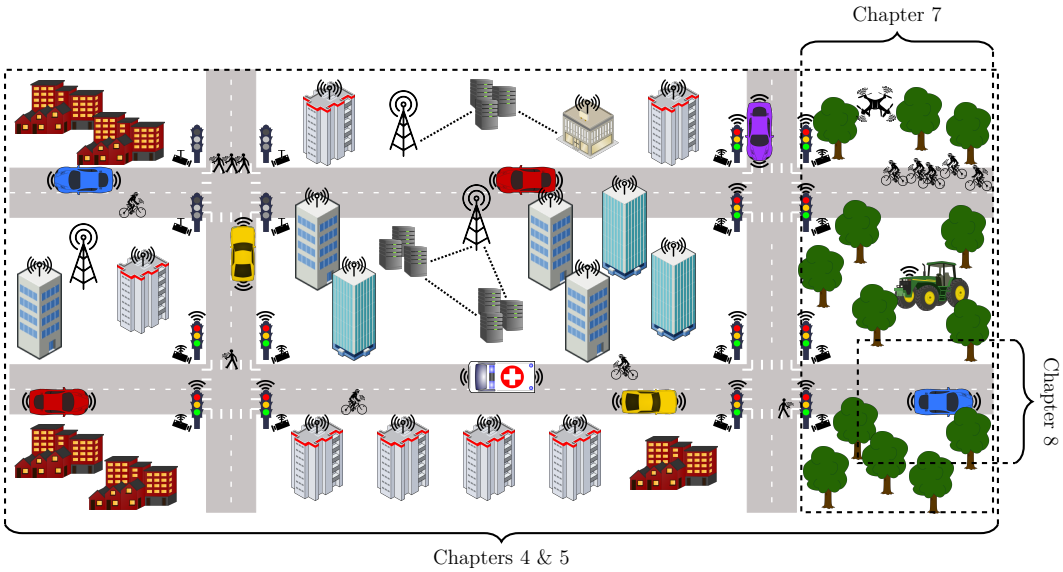


Figure 1.1: Illustration of an exemplary smart city including its components and actors. The areas, to which the challenges addressed in this dissertation are related, are marked and annotated with the chapters, in which further details can be found.

ensuring the operations' correctness and the vehicle's safety against the background of a shared, heterogeneous hardware infrastructure is very challenging [HSG+20], especially in an environment evolving over time [SZD+22]. Consequently, in a world that is becoming increasingly connected and permeated by dependencies between different systems, it is not meaningful to focus on the individual systems only when designing smart vehicles or devices; instead, the interplay between different traffic participants and actors of a smart city must be taken into account [LXC+19].

This dissertation addresses selected challenges arising in the context of smart cities, adopting the perspective of the underlying distributed system as well as of individual systems interacting with it. Specifically, all challenges considered are related to the topic of quality of service. In Fig. 1.1, an exemplary smart city is illustrated that serves as a use case throughout this thesis and indicates the areas to which the challenges explored in the respective chapters are related. In the following, the addressed challenges are summarized and the corresponding contributions are shortly emphasized.

1.2 Contributions of this Dissertation

This dissertation makes four contributions that are divided into two parts, one focusing on the distributed infrastructure of a smart city and one on individual systems

connected to such a distributed system. Subsequently, each addressed challenge is shortly outlined and the related contribution is highlighted.

1.2.1 Distributed Systems

A distributed system serving as the infrastructure of a smart city is assumed to have an admission control that decides which applications are executed on the system. Applications are assumed to have a QoS requirement specifying the maximum amount of time that must elapse between their start and their completion and can be executed on the distributed system on demand. For the considered system, it must be ensured that applications are only admitted if their QoS requirement can be satisfied, and that the satisfaction of an application's QoS requirement is guaranteed whenever it is admitted to the system.

Contract-Based Quality of Service Assurance

QoS contracts are introduced that ensure the satisfaction of the QoS requirement of each application on the system. Moreover, a monitoring approach is proposed that allows to verify the satisfaction of QoS contracts during the execution of an application and to identify the reasons of contract violations.

This contribution corresponds to Chapter 4. Parts of the presented content have previously been published in [SGS+22].

Some applications do not require their QoS requirement to be always satisfied in order to function correctly, as long as it is satisfied for a minimum number of times. This so-called robustness requirement must be taken into consideration by the admission control and must be guaranteed for each application exhibiting such a requirement that is executed on the system.

Robustness-Aware Quality of Service Contracts

Robustness-aware QoS contracts are introduced that guarantee end-to-end latency requirements in combination with robustness requirements for specific applications executed on the system. Robustness-aware QoS contracts can co-exist with regular QoS contracts on the same system.

This contribution corresponds to Chapter 5.

1.2.2 Embedded Systems

To activate enhanced functionalities despite local resource limitations, some individual systems offload parts of applications to the distributed infrastructure of a smart city.

In the case of autonomous vehicles, i.e., vehicles with advanced driver assistance systems, also parts of critical applications may be offloaded. For critical applications of such a system, QoS guarantees regarding the maximum time that must elapse from the release until the completion of an application must be always given, even in the case of connectivity issues.

Safe Offloading under Unreliable Connections

Two recovery protocols are proposed for satisfying the QoS requirements of all critical tasks in the case of unsuccessful offloading operations due to connectivity issues. For each protocol, a schedulability analysis and a schedulability test is provided that can be used for an a-priori verification of the system.

This contribution corresponds to Chapter 7. Parts of the presented content have previously been published in [SBC+20].

In modern vehicles, controller area network (CAN) is typically used as the in-vehicle communication backbone. Since CAN is a broadcast bus, each bus participant receives all messages transmitted on the bus, even if they are irrelevant, which introduces unnecessary overhead due to the further message processing. The amount of irrelevant messages received by a bus participant can be reduced by applying hardware message filters. In order to minimize the overhead following from irrelevant messages, the design of these filters must be optimized.

Hardware Message Filter Optimization

Approaches for the computation of hardware message filter configurations are provided applicable under different constraints with respect to the availability of the hardware and the required effectiveness of the filters.

This contribution corresponds to Chapter 8. Parts of the presented content have previously been published in [SBS+19].

1.3 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", the author's contribution to the material included in this dissertation is indicated in the following:

- **Chapter 3:** The underlying idea for the distributed system model has been provided by Selma Saidi, Dirk Ziegenbein, and Arne Hamann. The system model as considered in this dissertation is a result of discussions involving Selma

Saidi, Dirk Ziegenbein, Arne Hamann, Susanne Graf, and the author of this dissertation.

- **Chapter 4:** Parts of the presented content have been published in [SGS+22], authored by the author of this dissertation, Susanne Graf, Selma Saidi, Dirk Ziegenbein, and Arne Hamann, where the author of this dissertation was the principal author. The proposed system admission process has been developed in discussions involving Susanne Graf, Selma Saidi, Dirk Ziegenbein, Arne Hamann, and the author of this dissertation. The MITL formulations have been developed in cooperation between Susanne Graf and the author of this dissertation. The implementation and the evaluation have been done by the author of this dissertation.
- **Chapter 5:** The underlying idea of a co-existence of different types of contracts in the considered type of systems has been discussed by Selma Saidi and the author of this dissertation. The underlying idea of monitoring the satisfaction of dependability requirements in the form of (m, k)-constraints has been discussed by Susanne Graf, Selma Saidi, and the author of this dissertation. The concept of soft QoS contracts as proposed in this thesis has been developed by the author of this dissertation. The constraint formulation has been developed by the author of this dissertation. The implementation and the evaluation have been done by the author of this dissertation.
- **Chapter 6:** The embedded system model corresponds to the system model considered in [SBC+20], which relies on a typical task model considered in the real-time community and was extended in discussions involving Jian-Jia Chen, Georg von der Brüggen, and the author of this dissertation.
- **Chapter 7:** Parts of the presented content have been published in [SBC+20], authored by the author of this dissertation, Georg von der Brüggen, Kuan-Hsun Chen, Benjamin Sliwa, Hazem Youssef, Aswin Ramachandran Venkatapathy, Christian Wietfeld, Michael ten Hompel, and Jian-Jia Chen, where the author of this thesis was the principle author. The idea for the recovery-protocols has been developed by the author of this dissertation. The schedulability analysis including the proofs has been carried out in cooperation of Jian-Jia Chen, Georg von der Brüggen, and the author of this dissertation. Realistic and evidence-based information about unreliable wireless connections has been provided by Benjamin Sliwa. Robotic data used for the evaluation has been obtained by Hazem Youssef and Aswin Ramachandran Venkatapathy. The simulator used for the evaluation has been implemented by Kuan-Hsun Chen and has been modified by Kuan-Hsun Chen and the author of this dissertation. The evaluation has been carried out by the author of this dissertation.
- **Chapter 8:** Parts of the presented content have been published in [SBS+19], authored by the author of this dissertation, Georg von der Brüggen, Horst Schirmeier, and Jian-Jia Chen, where the author of this dissertation was the principle author. The underlying ideas have been collected in cooperation of Jian-Jia Chen, Georg von der Brüggen, and the author of this thesis. The constraint

formulations and the theorem regarding minimal perfect filter configurations have been developed in cooperation of Jian-Jia Chen and the author of this dissertation. The scenarios considered in the evaluation have been designed by Horst Schirmeier. The implementation and the evaluation have been done by the author of this dissertation.

1.4 Outline

This dissertation is divided into four parts, which are structured as follows:

- **Part I** continues with Chapter 2, dedicated to the background knowledge that is essential for understanding the subsequent content.
- **Part II** addresses distributed systems serving as the backbone of a smart city. Within this part, Chapter 3 introduces the system model used throughout the following chapters. Thereon, Chapter 4 presents an approach for contract-based quality of service assurance, building on which Chapter 5 introduces the concept of robustness-aware quality of service contracts.
- **Part III** focuses on embedded systems, addressing individual systems, e.g., smart vehicles, that are not part of a smart city's distributed system but are connected to it. Chapter 6 introduces the system model used within this part. In Chapter 7, an approach for enabling safe offloading under unreliable connections is proposed. Thereafter, Chapter 8 proposes approaches for hardware message filter optimization, aiming to reduce the computation overhead for resource-constrained systems that is caused by unnecessarily received broadcast messages.
- **Part IV** concludes this dissertation with Chapter 9 by giving a summary and and pointing out open problems and future research directions.

To enhance readability, each chapter concludes with a table of the introduced notation.

CHAPTER 2

Background

Contents

2.1 Typical Task Model

To provide the knowledge necessary for understanding this dissertation, first, a task model is introduced that is commonly used in the real-time research community, either in the presented form or as a basis for more sophisticated task models. In this dissertation, modified variants of this task model are considered, which are introduced at a later point; however, clarifying the standard task model is necessary in order to explain a set of fundamental concepts. Note that this section largely follows [But11], using a notation consistent with the rest of this dissertation.

On a system, a set of n *real-time tasks*, short *task*, $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ with $n \in \mathbb{N}$ is assumed to be executed. A task τ_i is typically characterized by a set of parameters as illustrated in Fig. 2.1. Tasks can be either *aperiodic*, *periodic*, or *sporadic*. Aperiodic tasks are executed only once, while a periodic or sporadic task releases a sequence of task instances, called *jobs*. The pattern according to which jobs are released is defined by the *period* or *minimum inter-arrival time* P_i. More precisely, if a task is periodic, each job is released exactly P_i time units after the release of the previous job. If a task is sporadic, each job is released at minimum P_i time units after the release of the previous job. The point in time when the first job of a task is released, i.e., it becomes ready for execution, is defined by the *phase* Φ_i. If the phase is not included in the

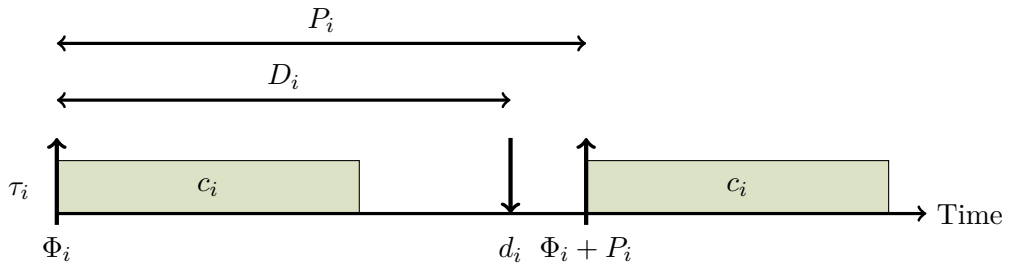


Figure 2.1: Illustration of a sporadic real-time task τ_i.

specification of a task, it is typically assumed to be zero. The *execution time* of a job of a task describes the time required for its uninterrupted execution, i.e., excluding potential waiting times. The *worst-case execution time* of a task is denoted by c_i and refers to the maximum execution time over all jobs. Based on the worst-case execution time, the *task utilization* is defined as $U_i = \frac{c_i}{P_i}$. Consequently, the utilization of a task set is given by $\sum_{\tau_i \in \mathcal{T}} U_i$. As *finishing time* or *completion time* f_i of a job of a task it is referred to the point in time in which the execution of the job is completed. The *response time* of a job of a task τ_i is defined as the difference between its finishing time and its release time. The *worst-case response time* R_i of a task τ_i is given by the maximum response time among all its jobs. The point in time until when the execution of a job of a task must be completed is indicated by the *absolute deadline* d_i and is given by the by point in time in which a job is released plus the *relative deadline* D_i that specifies the maximum amount of time that must elapse between the release of a job and its completion[1]. Relative deadlines can be either *implicit*, i.e., $D_i = P_i$, *constrained*, i.e., $D_i \leq P_i$, or *arbitrary*, i.e., it can be the case that $D_i > P_i$. Note that implicit deadlines are a subset of constrained deadlines and constrained deadlines are a subset of arbitrary deadlines.

If a task's job is not completed before its deadline, this can have very diverse consequences, based on which three types of real-time tasks can be distinguished: If a *hard real-time* task[2] exceeds its deadline, this can lead to catastrophic results and dangerous consequences for the system and its environment, e.g., when considering safety-critical control tasks. The results produced by a *firm real-time* task after missing its deadline, in contrast, are useless, but not harmful. If a *soft real-time* task[3] is completed after its deadline, the outcome can still be useful for the system to a certain degree [BSC18; BCH+16]. A system capable of handling real-time tasks is referred to as a *real-time system*. In this dissertation, tasks of different types are considered, which have so-called *quality of service (QoS) requirements* including the satisfaction of their deadlines and potential further parameters, as discussed more in detail in Sec. 2.4.

[1] Subsequently, the term *deadline* refers to the relative deadline unless specified differently.
[2] Synonymously, the expression *task with hard real-time requirements* is used.
[3] Synonymously, the expression *task with soft real-time requirements* is used.

2.2 Priority-Based Scheduling

To execute a set of tasks, where multiple jobs are ready for execution at the same time, it is necessary to determine an execution order, i.e., a *schedule*. In priority-based scheduling, this decision is made based on priorities assigned to the individual jobs by a *scheduling algorithm* according to algorithm-specific criteria. Priority-based scheduling algorithms can be classified into two distinct categories [But11], namely, *fixed-priority* scheduling algorithms that make scheduling decisions based on offline-assigned priorities, and *dynamic-priority* scheduling algorithms that assign priorities on the fly, i.e., based on parameters changing over time. With respect to fixed-priority scheduling, it is possible to make a further distinction: *Fixed task-level* priority scheduling algorithms assign the same priority to all jobs of a task, whereas *fixed job-level* priority scheduling algorithms may assign different priorities to different jobs of a task. A well-known example of fixed task-level priority scheduling is rate monotonic (RM) scheduling, where priorities are assigned based on the period or minimum inter-arrival time of a task. Consequently, jobs of tasks with a shorter period receive a higher priority, ties are broken arbitrarily. Independent of the category into which a scheduling algorithm belongs, it can be either *preemptive*, i.e., it allows the interruption of a job's execution at any point in time, or *non-preemptive*, i.e., once a job is started, it must be executed until its completion [But11]. In this dissertation, preemptive fixed task-level priority scheduling is considered.

A schedule is identified as *feasible* if all included tasks can be completed such that a given set of constraints, typically involving their deadlines, is satisfied [But11]. A set of tasks is termed *schedulable* if at least one scheduling algorithm exists, by means of which a feasible schedule can be computed [But11]. To determine if a task set is schedulable under a specific scheduling algorithm, a *schedulability test* can be applied. One approach for testing the schedulability of a set of tasks is the *worst-case response time analysis*, by means of which it is verified for each task in the task set if its worst-case response time is less than or equal to its relative deadline[4]. For this purpose, a worst-case scenario, called *critical instant* [LL73], is considered in which for each task under analysis the interference, i.e., the blocking time, by higher-priority tasks is quantified within a specific window of interest, i.e., typically, the time between the release and the absolute deadline of a job of the respective task. For the result of the response-time analysis to be valid, however, a safe worst-case execution time value for each task must be taken into account.

2.3 Worst-Case Execution Time

As introduced in Sec. 2.1, the *execution time* of a job is the time required from its start until its completion considering only its plain execution, i.e., neglecting waiting times. This time, however, depends on various factors such as the task inputs and

[4]The worst-case response time analysis is subsequently also referred to as *schedulability analysis*.

the states of the hardware, on which it is executed. The *worst-case execution time* (WCET) of a task refers to the maximum execution time among all jobs of the task that can be achieved under all possible (and valid) inputs and hardware states. Since the state space is very large and not all states can be anticipated a priori, but partly are revealed only at run-time, it is usually not possible to compute the exact WCET of a task. Against this background, several approaches exist for determining approximations of the WCET, which result in distinct WCET concepts that can be found in the literature. In this thesis, three different WCET concepts are considered.

For hard real-time tasks, safe WCET approximations are needed, i.e., approximations that over-approximate the actual worst case. These can be obtained by static analysis methods that employ abstractions and conservative over-approximations wherever the complete state space cannot be covered. By this means, safe upper bounds on the WCET are obtained, which, however, can lead to an over-reservation and waste of resources, since the worst case occurs only rarely [WEE+08]. This kind of WCET over-approximation is subsequently referred to as *worst-case execution time upper bound* and is considered in Chapter 7.

Approximations of the WCET of a task can also be obtained using measurement- or simulation-based methods, which are commonly used in the industry [WEE+08; DC19]. In the course of these, execution time values are observed for several test cases. The resulting worst observed execution time, however, does not necessarily reflect the worst case and therefore potentially under-approximates the real worst-case execution time. Consequently, values obtained using such approaches cannot be safely used for applications with hard real-time requirements. This type of WCET approximation is subsequently termed *estimated worst-case execution time*, short *time estimate*, and is used in Chapter 4.

Unlike the worst-case execution time upper bound and the estimated worst-case execution time, the *probabilistic worst-case execution time* (pWCET) does not consist of a single WCET value but represents a distribution of execution times over a sequence of jobs of a task [DC19]. More precisely, the pWCET of a task consists of a set of execution time values, each of which is associated with a probability of its occurrence. It can be obtained using several approaches such as, e.g., probabilistic static analysis [DC19]. Although the pWCET of a task does not provide a safe upper bound on the WCET, using this concept in the context of hard real-time systems is anyway possible if the probability that a task's deadline is violated is sufficiently small [DC19]. The pWCET is considered in Chapter 5.

2.4 Quality of Service

Applications as considered in this dissertation come with individual so-called *quality of service (QoS)* requirements. To understand these more in detail, it is necessary to know that applications are represented either as a task or as a set of dependent tasks in this dissertation; details are provided in the respective chapters and are

omitted at this point. Either way, an application can have different types of QoS requirements, with the primary requirement that the application's result is *functionally correct*, i.e., for instance, that a value is computed correctly or a consequent system behavior is exhibited as expected. This type of requirement is known as *functional QoS requirement*. In addition, a number of *non-functional QoS requirements* exist that specify how, i.e., under which constraints, the functional QoS requirement of an application must be satisfied.

Temporal correctness requirements are the most common and essential non-functional QoS requirements in real-time systems [But11] and demand that the deadline of each task or, respectively, the *end-to-end deadline*, also known as *end-to-end latency requirement*, of each set of dependent tasks is always satisfied. The end-to-end deadline of a set of tasks is defined similarly to the deadline of a task, namely, as the maximum amount of time that must elapse from the release of the first task in the set until the completion of the last task in the set. As noted in Sec. 2.1, the consequences of an unsatisfied temporal correctness requirement are not equally severe for all applications; while the continuous satisfaction of temporal correctness constraints is essential for hard real-time tasks, this is not always the case for soft real-time tasks. As case studies show, robotic tasks, for instance, are able to maintain a proper operation despite a limited number of deadline violations [CBC+16; YCC18]. For this reason, the temporal correctness requirement of soft real-time tasks can be relaxed by considering additional QoS parameters, allowing for a certain degree of flexibility regarding the satisfaction of the temporal correctness as long as a certain minimal level of service can be ensured. This can be achieved by formulating *robustness requirements*, which can be expressed in two different ways, namely, in the form of specifying percentages of cases in which the temporal correctness requirement must be satisfied, subsequently termed *confidence levels*, or through (m, k)*-criteria*, as explained below.

A confidence level indicates the minimum percentage of cases in which a temporal correctness requirement must be satisfied. This percentage, however, does not include any statement about the distribution of cases over multiple application instances. For clarification, assume that a confidence level of 0.9 is given, implying that in 90 % of cases the temporal correctness requirement of an application must be satisfied. It may be possible that a reasonable result is obtained despite 10 % of cases within a sequence of 100 application instances, in which the temporal correctness is violated, if these cases are evenly distributed over the 100 application instances. If, in contrast, for 10 % of successive application instances the temporal correctness requirement is not satisfied, this may lead to a degradation of the application functionality. To quantify the amount of temporal correctness violations more accurately that is tolerable without leading to an unacceptable outcome, (m, k)-constraints are typically used. Originally introduced in the context of firm real-time tasks [HR95], an (m, k)-constraint expresses that at least m out of k consecutive jobs of a task (or application instances, respectively) must satisfy their timeliness constraints. These m jobs, however, are not required to be consecutive. In recent years, (m, k)-constraints have been related to the area of fault

tolerance of (control) tasks [Che19], considering deadline misses as consequences of fault compensation mechanisms such as, e.g., (partial) job re-execution. Accordingly, (m, k)-constraints relax temporal correctness requirements, but must be guaranteed at any point in time themselves.

Another way of specifying the strictness and ensuring the satisfaction of temporal correctness requirements is the notion of *criticality* and, resulting from this, the concept of *mixed-criticality systems* [BD18]. These have been formally introduced for the first time by [Ves07] and describe systems integrating tasks with different *criticality levels* on the same platform, which provides distinct system modes, usually one per criticality level. So-called *dual-criticality systems* are frequently considered, comprising high-criticality tasks (typically corresponding to hard real-time tasks) and low-criticality tasks (typically corresponding to soft real-time tasks). In case of special events such as fault-occurrence, mixed-criticality systems perform a mode change, i.e., a switch to another system mode, which permits to maintain the system safety by ensuring that the temporal correctness requirements of all tasks with a criticality level equal to or higher than the current system mode are still met. For all lower-criticality tasks, however, no more QoS guarantees are provided.

In this dissertation, the satisfaction of all above-mentioned non-functional QoS requirements is addressed, with an emphasis on temporal correctness requirements. At the beginning of each chapter, a short recap regarding the currently considered types of QoS requirements will be provided.

Notation	**Meaning**
c_i	Worst-case execution time of τ_i
d_i	Absolute deadline of τ_i
D_i	Local deadline of τ_i
f_i	Finishing time/completion time of τ_i
Φ	Phase of τ_i
P_i	Period or minimum inter-arrival time of τ_i
$\mathcal{T}$	Set of real-time tasks
τ_i	Real-time task
U_i	Utilization of τ_i

Table 2.1: Overview of the notation introduced in Chapter 2.

Part II

Distributed Systems

CHAPTER 3

System and Application Model

Parts of this chapter have previously been published in [SGS+22].

Contents

3.1 System Architecture

The heterogeneous hardware infrastructure of a smart city as considered in this dissertation (cf. Chapter 1.1) can be represented as a centralized, hierarchical distributed system with an admission control that decides about the execution of applications on the system. Contemplating a functional abstraction of the system, two distinct layers can be distinguished, namely, a *management layer* and an *infrastructure layer*, as illustrated in Fig. 3.1. Note that an overview of the notation introduced in this chapter is given in Table 3.1.

3.1.1 Management Layer

The management layer abstracts the underlying hardware infrastructure and provides a functional system view to users such as, for instance, traffic participants demanding to execute applications on the system, as well as to application designers. It comprises

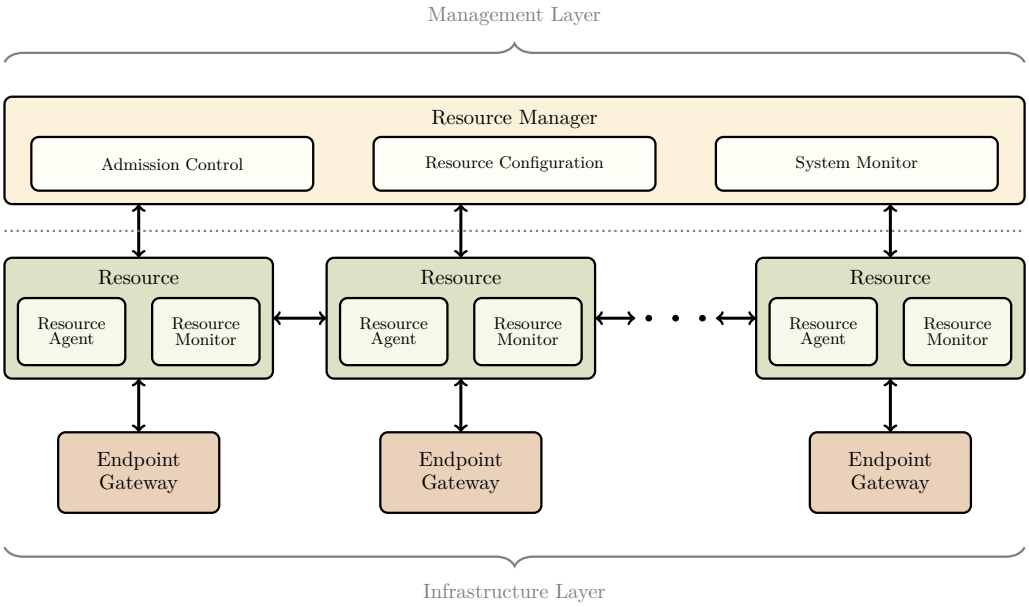


Figure 3.1: A schematic overview of the system architecture.

a *resource manager* that has several duties: It implements the admission control, which evaluates for each new application whether it can be executed on the system such that its QoS requirement can be satisfied, and decides about the application's admission to the system. For this purpose, it transforms the functional requirements of the application into system-specific constraints[1] that are used for the configuration and reservation of hardware resources. Moreover, the resource manager includes a system monitor that collects and evaluates information about the system state. Note that for all components of this system, a global, i.e., system-wide, notion of time is assumed to exist that is maintained by any suitable clock synchronization protocol, for which the resource manager serves as the master clock.

3.1.2 Infrastructure Layer

The infrastructure layer is hidden by the management layer and, thus, is not visible to users and application developers. It comprises the hardware infrastructure of the system that can be abstracted as a set of computation and communication *resources* as well as of *endpoint gateways*.

Resources

Resources are assumed to be heterogeneous, i.e., to have different types such as, e.g., specific computation platforms or communication technologies. In fact, each resource corresponds to an individual, physically existent hardware element, e.g., a CPU, and

[1] Details will be provided in Chapter 4 and Chapter 5.

is administrated by a dedicated *resource agent* implemented on the resource. The resource agent is in charge of scheduling; in the course of this, it needs to ensure that all system-specific constraints computed and imposed by the resource manager are satisfied, e.g., by preempting or aborting jobs of tasks if necessary. Unless specified differently, each resource is assumed to employ a preemptive task-level fixed-priority scheduling policy. Moreover, to decouple communication and computation, so that resources are independently assignable, each resource is assumed to be equipped with adequately dimensioned buffers. Apart from the resource agent, each resource has a resource monitor[2] that provides status and potentially further information to the resource manager.

Endpoint Gateways

Resources can, but do not necessarily have to be connected to an *endpoint gateway*, which allows endpoint systems such as, e.g., smart vehicles with advanced driver assistance systems, to use the system's hardware infrastructure. Although endpoint systems can be temporarily located within a smart city, they are not considered as part of the distributed system in this dissertation, aiming to keep the core infrastructure static. Instead, they can connect to the distributed system dynamically via an endpoint gateway and enhance their local resources by using the distributed system as an execution platform for applications, e.g., for performing autonomous navigation, provided that these pass the admission control. To cover different technologies that allow for connecting to the distributed system, different types of endpoint gateways are assumed to exist.

Infrastructure Model

The hardware on the infrastructure layer is modeled as a directed *resource graph*, illustrated in Fig. 3.2, where each node represents either a resource $r_i \in \mathfrak{R}$ or an endpoint gateway $e_\ell \in \mathfrak{E}$ and every edge (r_i, r_j), (r_i, e_ℓ), (e_ℓ, r_i) with $i, j, \ell \in \mathbb{N}$ and $i \neq j$, indicates that data flow is possible from resource r_i to resource r_j, from resource r_i to endpoint gateway e_ℓ, and from endpoint gateway e_ℓ to resource r_i, respectively. Direct data flow between two endpoint gateways is not possible.

3.2 Application Model

Each application that can be executed on the system is characterized by a tuple $a_i = \left(\mathcal{T}_i, D_i^{E2E}, P_i\right)$. $\mathcal{T}_i$ is a partially ordered set of sporadic tasks that can be modeled by a directed acyclic graph (DAG) with one source task and one sink task[3] referred to as *task graph* (cf. Fig. 3.3 for a schematic illustration of an exemplary task graph).

[2] Details will be provided in Chapter 4.

[3] If more than one source or sink task exist, an additional source or sink task requiring an execution or transmission time of zero time units can be modeled.

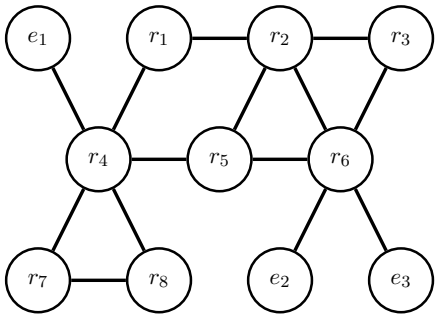


Figure 3.2: Schematic illustration of an exemplary resource graph.

Each node in the task graph represents a task $\tau_{i,j}$, where an edge $(\tau_{i,j}, \tau_{i,\ell})$ with $j, \ell \in \mathbb{N}$ and $j \neq \ell$ indicates a dependency of $\tau_{i,\ell}$ on $\tau_{i,j}$, e.g., due to a computation result that is required as an input. Two tasks $\tau_{i,j}, \tau_{i,\ell}$ are independent if neither a direct nor an indirect connection between $\tau_{i,j}$ and $\tau_{i,\ell}$ exists in the task graph. D_i^{E2E} indicates the end-to-end deadline (or end-to-end latency requirement; cf. Chapter 2.4) of a_i, while P_i is the minimum inter-arrival time according to which application instances are released. The end-to-end deadline of each application is assumed to be constrained, i.e., $D_i^{E2E} \leq P_i$.

Each task $\tau_{i,j} \in \mathcal{T}_i$ is characterized by a tuple that differs depending on the admission status of the application. More precisely, some task parameters are provided by the application developers and remain unchanged, whereas others are computed by the resource manager while the application is being admitted to the system. Accordingly, tasks of unadmitted applications and tasks of admitted applications must be distinguished.

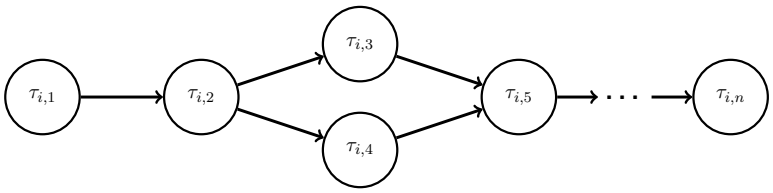


Figure 3.3: Schematic illustration of a task graph with $n \in \mathbb{N}$ tasks.

3.2.1 Tasks of Unadmitted Applications

A task $\tau_{i,j} = (\mathfrak{C}_{i,j}, E_{i,j}, P_{i,j})$ of an unadmitted application is characterized by a nonempty set of time estimates $\mathfrak{C}_{i,j}$, where each time estimate $c_{i,j}$ indicates the time required for the execution or transmission[4] of the task on a resource of a specific type. Recall, however, that a time estimate does not necessarily reflect the worst-case execution time of a task (cf. Chapter 2.3). $P_{i,j}$ refers to the minimum inter-arrival

[4]For the sake of readability, subsequently, only the term *execution* is used.

time the task inherits[5] from the application a_i and $E_{i,j}$ is a set of endpoint gateway type requirements. Each element of $E_{i,j}$ indicates one possible type of endpoint gateways that must be connected to a resource on which the task $\tau_{i,j}$ can be executed. If a resource is not connected to an endpoint gateway of any type included in $E_{i,j}$, it is not suitable for the execution of $\tau_{i,j}$.

3.2.2 Tasks of Admitted Applications

Each task $\tau_{i,j} = (c_{i,j}, P_{i,j}, \omega_{i,j}, D_{i,j}, \Pi_{i,j})$ of an admitted application, as illustrated in Fig. 3.4, is characterized by one time estimate $c_{i,j}$ that is related to the type of the resource allocated for $\tau_{i,j}$. $P_{i,j}$ refers to the minimum inter-arrival time inherited from the application a_i, which has the same value for each $\tau_{i,j}$ of a_i. $\omega_{i,j}$ is the *release offset* of $\tau_{i,j}$ on the allocated resource that is specified relative to the release of the application, i.e., of the first task in the task graph. $D_{i,j}$ specifies the relative *local deadline*, i.e., the execution of $\tau_{i,j}$ must be finished until $\omega_{i,j} + D_{i,j}$.

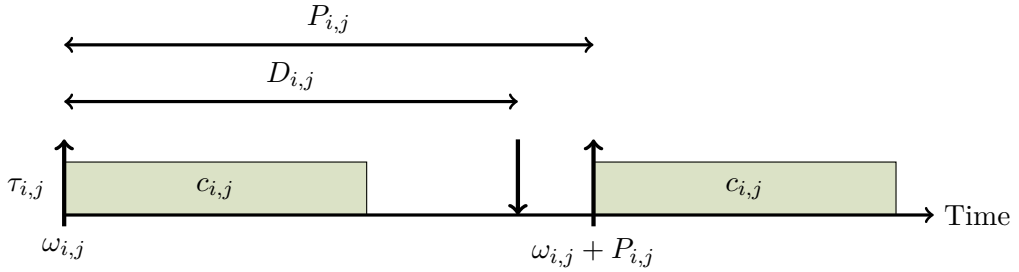


Figure 3.4: Illustration of an exemplary task $\tau_{i,j}$ and of a subset of its parameters.

With each task $\tau_{i,j}$, a priority $\Pi_{i,j}$ is associated that is unique at each point in time on the allocated resource. Considering a task $\tau_{i,j}$ in any set of tasks $\mathcal{T}'$, the set of *higher-priority tasks* is defined as $hp(\tau_{i,j}) := \{\tau_{i,\ell} \in \mathcal{T}' | \Pi_{i,\ell} > \Pi_{i,j}\}$ with $i, \ell \in \mathbb{N}$, and the set of *lower-priority tasks* as $lp(\tau_{i,j}) := \{\tau_{i,\ell} \in \mathcal{T}' | \Pi_{i,\ell} < \Pi_{i,j}\}$ with $i, \ell \in \mathbb{N}$. Note that the priority of a task can be modified only if the set of tasks scheduled on the allocated resource changes.

Each task $\tau_{i,j}$ of an admitted application a_i releases a sequence of *jobs*, i.e., task instances, that inherit the task parameters[6]. The *response time* of a job is defined as the time elapsing between its release and its completion (or abortion). Consequently, the *worst-case response time* $R_{i,j}$ of a task is given by the maximum response time over all of its jobs.

[5] Note that due to the inheritance $P_{i,j}$ has the same value for each task $\tau_{i,j}$ of an application a_i.

[6] Note that the release of an instance of a_i implies that each task $\tau_{i,j}$ releases a job at its respective release offset.

3.2.3 Time and Data Flow Determinism

The computation of a release offset and a local deadline for each task during the admission of an application to the system has the purpose of ensuring time and data flow determinism. More precisely, for an application a_i to behave and function correctly, no successor task of a task $\tau_{i,j}$ must start before $\tau_{i,j}$ has been completed (or aborted). Accordingly, it is assumed that the release offsets and local deadlines are computed such that, if an edge $(\tau_{i,j}, \tau_{i,\ell})$ exists in the task graph, it holds that $\omega_{i,\ell} \geq \omega_{i,j} + D_{i,j}$. This is a realization of the *logical execution time* paradigm [EAG18], which assumes that each task has a fixed read and write instance at the begin and, respectively, at the end of its execution, which is already included in the task's execution time. The computation and enforcement of release offsets and local deadlines is revisited in Chapter 4.

Notation	Meaning
a_i	Application
$\mathfrak{C}_{i,j}$	Set of time estimates of a task $\tau_{i,j}$
$c_{i,j}$	Time estimate of a task $\tau_{i,j}$ for one resource type
D_i^{E2E}	End-to-end deadline of an application a_i
$D_{i,j}$	Relative local deadline of a task $\tau_{i,j}$
$\mathfrak{E}_{i,j}$	Set of endpoint gateway requirements of a task $\tau_{i,j}$
e_ℓ	Endpoint gateway
$hp(\tau_{i,j})$	Set of tasks with higher priority than task $\tau_{i,j}$
$lp(\tau_{i,j})$	Set of tasks with lower priority than task $\tau_{i,j}$
P_i	Minimum inter-arrival time of an application a_i
$P_{i,j}$	Minimum inter-arrival time of a task $\tau_{i,j}$
$\Pi_{i,j}$	Priority of a task $\tau_{i,j}$ on the allocated resource
$R_{i,j}$	Worst-case response time of a task $\tau_{i,j}$
$\mathfrak{R}$	Set of all resources
r_i	Communication or computation resource
(r_i, r_j)	Edge from r_i to r_j in the resource graph
$\mathcal{T}_i$	Task set of an application a_i
$\tau_{i,j}$	Task of an application a_i
$(\tau_{i,j}, \tau_{i,\ell})$	Edge from $\tau_{i,j}$ to $\tau_{i,\ell}$ in the task graph, $\tau_{i,\ell}$ depends on $\tau_{i,j}$
$\omega_{i,j}$	Release offset of a task $\tau_{i,j}$ on the allocated resource

Table 3.1: Overview of the notation introduced in Chapter 3.

CHAPTER 4

Contract-Based Quality of Service Assurance

Parts of this chapter have previously been published in [SGS+22].

Contents

4.1 Introduction

Smart cities involve a variety of different actors, some of which utilize the provided infrastructure on a continuous basis, while others make only temporary use of it. Long-term functions such as, for instance, traffic analyses performed by permanently installed roadside units can already be considered when designing a smart city and dimensioning its underlying system. The workload, however, introduced by variable elements, e.g., by modern vehicles with advanced driver assistance systems that use the smart city's infrastructure on demand to enable autonomous navigation functionalities, cannot be anticipated.

Accordingly, in the use case of this chapter (cf. Fig. 4.1), a centralized, hierarchical distributed system with an admission control, as introduced in Chapter 3, is considered that is shared by a set of applications originating from different actors of a smart city. Since this set of application changes over time and is not known a priori, satisfying the individual QoS requirements of each application is highly challenging.

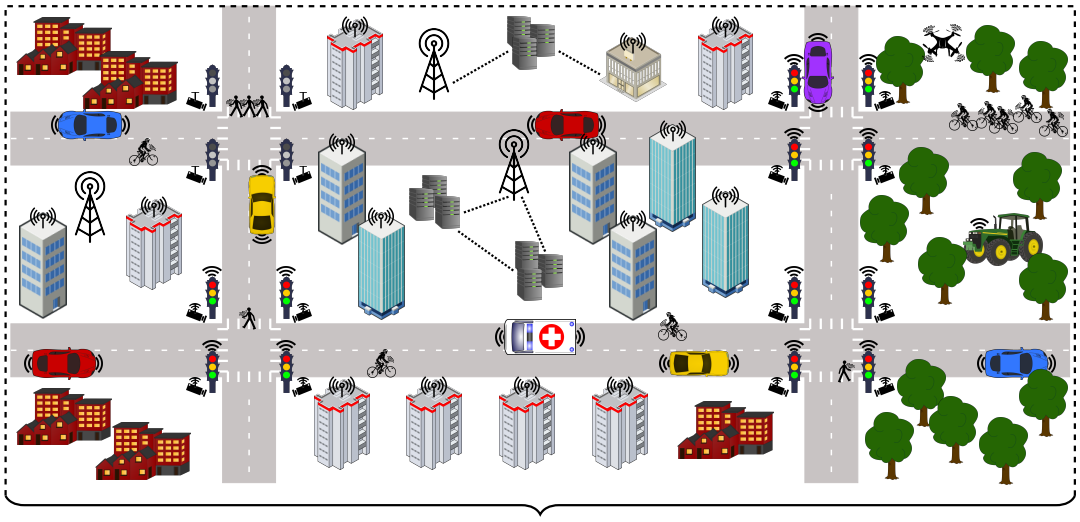


Figure 4.1: A smart city involving a multitude of different participants that use the distributed infrastructure on demand. This chapter introduces the concept of QoS contracts for applications, a system admission process, and a monitoring approach for detecting contract violations.

In the literature, a limited number of works exist that address related challenges. In fact, with respect to QoS guarantees in distributed systems, two distinct perspectives can be encountered, one focusing on centralized and another focusing on decentralized distributed systems; nevertheless, approaches for providing QoS guarantees in the latter type of systems, e.g., [PSE21], are not directly transferable to the system model considered in this chapter, for which reason a further discussion is omitted. With

respect to centralized distributed systems, two related works are highlighted: Not explicitly targeting at distributed systems, but at networks with static and dynamic topology including an admission control, [MEB19] proposes a protocol for managing the admission of applications by computing an optimal admission instant, as of which QoS guarantees are provided. Due to the assumed variability of task parameters, specifically, of local deadlines, however, this approach is not applicable to the system considered in this chapter. In [NSE11], a hierarchical architecture for a centralized distributed system, comparable to the one considered in this chapter, is introduced, on which QoS guarantees are given by means of contracts concluded between an application and different system components.

Adopting the idea of [NSE11], this chapter aims to satisfy the QoS requirements of applications in dynamic systems by introducing the concept of QoS contracts, which deviates from the type of contracts considered in [NSE11], as discussed later on. These contracts allow to provide QoS guarantees for all applications in the system at any point in time and ensure that the admission of a new application to the system does not lead to the violation of another application's QoS requirement. A detailed description of this objective as well as an outline of the chapter are given in the following section.

4.2 Problem Statement

An application a_i can request to be executed on a distributed system as described in Chapter 3. Following an admission process, it is either accepted or rejected. At every time instant, only one application can be in the admission process. Each application a_i is assumed to have a QoS requirement in terms of temporal correctness, i.e., an end-to-end deadline D^{E2E}. If an application's QoS requirement cannot be satisfied by the system, the application must be rejected by the admission control. Accordingly, for each application executed on the system, it is assumed that its QoS requirement can be satisfied.

Objective of this Chapter

For the considered system and type of applications executable on the system, this chapter aims to introduce a concept of QoS contracts that guarantee the satisfaction of the QoS requirement of each application a_i admitted to the system, and to provide the workflow of a system admission process that allows to establish such QoS contracts. Moreover, a monitoring approach shall be proposed for verifying the satisfaction of QoS contracts at run-time that allows to identify the reasons of contract violations.

In the following section, background knowledge about *assume-guarantee contracts* is provided that is necessary for the introduction of QoS contracts in Sec. 4.4. A

monitoring approach for the verification of QoS contracts and the detection of contract violations at run-time is proposed in Sec. 4.5 and evaluated in Sec. 4.6. An overview of the notation introduced in this chapter is provided in Table 4.1.

4.3 Assume-Guarantee Contracts

Assume-guarantee contracts [BCN+18] (or short *contracts*) are a concept that is frequently used in the context of systems development and software engineering. In brief, a contract $\mathscr{C}$ is an abstraction or a model representing an element of a system, e.g., a physical part, a subsystem, or a piece of software, denoted as *component*. For a component κ, a contract $\mathscr{C}$ describes a set of *assumptions* $\mathscr{A}$ under which it can be used and provides a set of *guarantees* $\mathscr{G}$ that are given if the assumptions are satisfied. If a component κ complies with a contract $\mathscr{C} = (\mathscr{A}, \mathscr{G})$, the component is said to *satisfy* the contract, denoted by $\kappa \vdash \mathscr{C}$.

Two contracts $\mathscr{C}_1, \mathscr{C}_2$ can be combined to a new contract $\mathscr{C}_3$ either by *composition* or by *conjunction*. The composition ($\otimes$) of two contracts corresponds to merging the models of two components to a model of a new, more complex component, denoted by $\mathscr{C}_3 = \mathscr{C}_1 \otimes \mathscr{C}_2$. For a composed contract $\mathscr{C}_3$, it holds that $\kappa \vdash \mathscr{C}_3$ if $\kappa \vdash \mathscr{C}_1 \otimes \mathscr{C}_2$. The conjunction ($\wedge$) of two contracts allows to group the models of two independent components. Accordingly, for $\mathscr{C}_3 = \mathscr{C}_1 \wedge \mathscr{C}_2$, it holds that $\kappa \vdash \mathscr{C}_3$ if $\kappa \vdash \mathscr{C}_1$ and $\kappa \vdash \mathscr{C}_2$.

4.4 Quality of Service Contracts

For guaranteeing the QoS requirement of each application a_i executed on the considered system, the concept of assume-guarantee contracts can be adopted. By concluding *quality of service (QoS) contracts* between the system, represented by the resource manager (cf. Chapter 3), and each application a_i, it is possible to ensure that the end-to-end deadlines of all a_i are satisfied. Two types of QoS contracts are considered in the following, namely, *application-level QoS contracts* and *task-level QoS contracts*. Concretely, Sec. 4.4.1 introduces task-level QoS contracts, followed by the proposal of a system admission process in Sec. 4.4.2 that can be used for computing the parameters required by task-level QoS contracts. In Sec. 4.4.3, it is explained how task-level QoS contracts can be combined to application-level QoS contracts. The proposed system admission process is discussed in the context of existing works in Sec. 4.4.4.

4.4.1 Task-Level QoS Contracts

It can be guaranteed that the local deadline $D_{i,j}$ of a task $\tau_{i,j}$ is met by concluding a task-level QoS contract. To clarify this, reflect on the prerequisites necessary for $D_{i,j}$ being always satisfied: Intuitively, it must hold that the response time of each job of $\tau_{i,j}$ is less than or equal to the local deadline. Accordingly, the worst-case response time $R_{i,j}$ of $\tau_{i,j}$ must be considered for the computation of a safe $D_{i,j}$. Since the response time of a job, however, strongly depends on its execution time, it must be

computed based on a realistic time estimate $c_{i,j}$. Moreover, $R_{i,j}$ includes the maximum amount time a job of $\tau_{i,j}$ can be blocked by jobs of interfering tasks, i.e., tasks with higher priority scheduled on the same resource. In order to quantify this interference, it is necessary that a correct minimum inter-arrival time is communicated by each application a_i during the system admission process. Based on these requirements, a task-level QoS contract can be defined as follows:

Definition 1 (Task-Level QoS Contract)**.** For a task $\tau_{i,j}$ of an application a_i scheduled on a resource of the considered system, a *task-level QoS contract* is defined as $\mathscr{C}_{i,j} = (\mathscr{A}_{i,j}, \mathscr{G}_{i,j})$ with assumptions $\mathscr{A}_{i,j} = \{c_{i,j}, P_{i,j}\}$ and guarantees $\mathscr{G}_{i,j} = \{D_{i,j}\}$. If each job of $\tau_{i,j}$ complies with the values of $c_{i,j}$ and $P_{i,j}$ as revealed in the system admission process, then $\tau_{i,j} \vdash \mathscr{C}_{i,j}$.

4.4.2 System Admission Process

To compute the local deadlines $D_{i,j}$ of all tasks $\tau_{i,j}$ of an application a_i that are required for a task-level QoS contract, a_i must be evaluated in the course of a system admission process. An incremental admission workflow is proposed in the following.

Mapping

In a first step, a representation of the task graph, denoted as *mapping* $\mathfrak{M}_i$, on the system's hardware infrastructure must be computed. The task graph of an application a_i is mapped to the system, i.e., each task $\tau_{i,j} \in \mathcal{T}_i$ is associated with a resource, by a mapping algorithm:

Definition 2 (Mapping Algorithm)**.** A mapping algorithm defines one or multiple functions $map : \mathcal{T}_i \to \mathfrak{R}$ for each $\tau_{i,j} \in \mathcal{T}_i$.

If more than one function *map* and, thus, multiple mappings exist for an application, only one of these is chosen. This decision can be based on any criterion imposed by the system designers.

Local Deadline Assignment

To compute local deadlines for all tasks of an application, each $\tau_{i,j}$ must be integrated into the schedule of the resource it is mapped to, so that the schedulability of all other tasks in the schedule is preserved. If this is not possible on one or more resources but further mappings exist for the considered application, the system admission process can be repeated using another mapping; otherwise, it is aborted and the application is rejected by the admission control. If, however, the respective sets of tasks are schedulable for all tasks of the application, for each such $\tau_{i,j}$, an upper bound on the worst-case response time is computed (independently of all other tasks of a_i) by the

responsible resource agent. Based on the computed upper bound on $R_{i,j}$[1], a local deadline $D_{i,j}$ is assigned for each task $\tau_{i,j}$ such that $D_{i,j} \geq R_{i,j}$[2].

Independent timing analyses can be applied as a consequence of the LET paradigm (cf. Chapter 3.2.3) being considered for all mapped tasks. More precisely, the task dependencies are decoupled by the enforcement of release offsets, i.e., no task is released previous to its release offset even if the execution of its predecessor(s) is already completed, and by the buffering of input data, which can be reused in case a predecessor task is aborted before its completion. In fact, the considered scenario corresponds to a partitioned multiprocessor scheduling scenario of independent tasks after the partitioning has been completed, where the original scheduling problem has been transformed into a set of uniprocessor scheduling problems that can be solved individually (for further information refer to [DB11]).

For the computation of an upper bound on $R_{i,j}$, any analysis can be applied that is suitable for the resource type and that remains valid at run-time, such as the one provided by Theorem 1 that makes use of well-known standard analysis techniques.

Theorem 1. The worst-case response time $R_{i,j}$ of a task $\tau_{i,j}$ mapped to a resource $r_z, z \in \mathbb{N}$, on which tasks are feasibly scheduled according to a preemptive task-level fixed-priority policy, is upper-bounded by the minimum positive value of Δ for which

$$\Delta = c_{i,j} + \sum_{\tau_{x,y} \in hp(\tau_{i,j})} \left(\left\lceil \frac{\Delta}{P_{x,y}} \right\rceil + 1 \right) c_{x,y}$$

where $i, j, x, y \in \mathbb{N}$ and $hp(\tau_{i,j})$ is the set of tasks with higher priority than $\tau_{i,j}$ mapped to resource r_z with $D_{x,y} \leq P_{x,y}$.

Proof. Note that all tasks mapped to the resource r_z are independent, sporadic real-time tasks and that the task under analysis $\tau_{i,j}$ is also a sporadic real-time task that is independent from all tasks mapped to r_z. Assume that all tasks mapped to r_z are associated with a fixed priority and that a concrete preemptive fixed-priority schedule exists for r_z. Assume this schedule to be work-conserving, i.e., the resource does not idle as long as a released job of a task exists which is neither completed nor aborted. Based on these assumptions, all jobs of tasks in $lp(\tau_{i,j})$, i.e., with lower priority than $\tau_{i,j}$, can be removed from the schedule, since they do not interfere with the execution of $\tau_{i,j}$ or of any task in $hp(\tau_{i,j})$. To quantify the interference of the tasks in $hp(\tau_{i,j})$ on $\tau_{i,j}$ within an interval $[t, t + \Delta)$ beginning at a point in time t, the maximum amount of time for which a task $\tau_{x,y} \in hp(\tau_{i,j})$ is executed within this interval must be determined. Recall that $D_{x,y} \leq P_{x,y}$ for all $\tau_{x,y} \in hp(\tau_{i,j})$ and that

[1] Note that $R_{i,j}$ is subsequently considered to be an upper bound on the worst-case response time of $\tau_{i,j}$.

[2] Note that $D_{i,j} = R_{i,j}$ is necessary to ensure that the deadline can be met by each job of $\tau_{i,j}$. However, the local deadline assignment can also be extended. For instance, initial deadline values can be assigned and, if the system admission process determines that a shorter end-to-end latency can be guaranteed than required by the application, these can be increased later on. This, however, is not relevant for the objective of this chapter and, therefore, is left to the system designers.

$D_{i,j} \leq P_{i,j}$ is required for $\tau_{i,j}$. Consider two scenarios: *a)* At time t, no job of $\tau_{x,y}$ exists that is neither completed nor aborted. *b)* At time t, a job of $\tau_{x,y}$ exists that is either completed or aborted within $[t, t+\Delta)$. In case *a)*, jobs of $\tau_{x,y}$ are released at most $\left\lceil \frac{\Delta}{P_{x,y}} \right\rceil$ times within $[t, t+\Delta)$. Each released job is executed for up to $c_{x,y}$ time units. Accordingly, $\tau_{x,y}$ is executed within $[t, t+\Delta)$ for at most $\left\lceil \frac{\Delta}{P_{x,y}} \right\rceil c_{x,y}$ time units. In case *b)*, $\tau_{x,y}$ is first executed for up to $c_{x,y}$ time units, i.e., until the job released before t is completed or aborted. Additionally, $\tau_{x,y}$ can be released within $[t, t+\Delta)$ at most $\left\lceil \frac{\Delta}{P_{x,y}} \right\rceil$ times. Thus, in case *b)* $\tau_{x,y}$ can be executed for at most $\left(\left\lceil \frac{\Delta}{P_{x,y}} \right\rceil + 1\right) c_{x,y}$ time units. Therefore, case *b)* dominates case *a)*. The task $\tau_{i,j}$ under analysis itself is executed for at most $c_{i,j}$ time units. Although due to the release offsets of the tasks mapped to resource r_z it is possible that not each $\tau_{x,y} \in hp(\tau_{i,j})$ is executed within $[t, t+\Delta)$ at all, the maximum possible interference of all $\tau_{x,y} \in hp(\tau_{i,j})$ must be considered to cover the worst-case interference. □

Release Offset Assignment

In order to explain how the task release offsets can be computed it is necessary to make several auxiliary definitions.

Definition 3 (Path)**.** A path in a mapping $\mathfrak{M}_i$ of an application a_i is a totally ordered set of mapped tasks $\mathfrak{P}_i^{(j,\ell)} = \{\tau_{i,j}, \ldots, \tau_{i,\ell}\}$ from a source $\tau_{i,j}$ to a sink $\tau_{i,\ell}$, such that an edge $(\tau_{i,v}, \tau_{i,w})$ exists in $\mathfrak{M}_i$ for any two $\tau_{i,v}, \tau_{i,w} \in \mathfrak{M}_i$, $v \neq w$, with $\tau_{i,w}$ being a successor of $\tau_{i,v}$. If $\tau_{i,j} = \tau_{i,\ell}$, then $\mathfrak{P}_i^{(j,\ell)} = \{\tau_{i,j}\}$.

For a path $\mathfrak{P}_i^{(j,\ell)} \in \mathfrak{M}_i$, the end-to-end latency is given by the sum of local deadlines, i.e., by $\sum_{\tau_{i,v} \in \mathfrak{P}_i^{(j,\ell)}} D_{i,v}$.

Definition 4 (Submapping)**.** A submapping $\mathfrak{M}_i^{(j,\ell)}$ of an application a_i is a mapping of a partially ordered set of tasks $\mathcal{T}_i' \subseteq \mathcal{T}_i$, which includes all paths from a source $\tau_{i,j}$ to a sink $\tau_{i,\ell}$ that are included in $\mathfrak{M}_i$.

The end-to-end latency of a submapping $\mathfrak{M}_i^{(j,\ell)}$ of a_i can be computed by a *time composition function*:

Definition 5 (Time Composition Function)**.** For a submapping $\mathfrak{M}_i^{(j,\ell)}$ of an application a_i, a time composition function

$$comp(\mathfrak{M}_i^{(j,\ell)}) := \max_{\mathfrak{P}_i^{(j,\ell)} \in \mathfrak{M}_i^{(j,\ell)}} \left(\sum_{\tau_{i,v} \in \mathfrak{P}_i^{(j,\ell)}} D_{i,v} \right)$$

returns the end-to-end latency from a source $\tau_{i,j}$ to a sink $\tau_{i,\ell}$.

Having defined the above, the release offsets of all tasks $\tau_{i,j}$ of an application a_i can be computed as follows: To the first mapped task $\tau_{i,1}$, assign $\omega_{i,1} = 0$. Starting from $\tau_{i,1}$, consider the remaining tasks following a breadth-first search. For each considered task $\tau_{i,v}$, create a submapping $\mathfrak{M}_i^{(1,v)}$ and assign $\omega_{i,v} = comp(\mathfrak{M}_i^{(1,v)}) - D_{i,v}$.

Latency Composition and Admission Decision

After computing the local deadlines and release offsets of all tasks of an application, it must be evaluated if the end-to-end deadline of a_i can be satisfied, aiming to decide about the admission of the application. Accordingly, for $\mathfrak{M}_i^{(1,n)}$ with $n = |\mathcal{T}_i|$, if $comp\left(\mathfrak{M}_i^{(1,n)}\right) \leq D_i^{E2E}$, the application is accepted and an application-level contract is concluded. Otherwise, the application is rejected[3].

4.4.3 Application-Level QoS Contracts

Being able to compute the guarantees provided by task-level QoS contracts, further explanation is needed on how task-level QoS contracts can be combined to an application-level QoS contract. In fact, a task-level QoS ensures the satisfaction of a task's local deadline, whereas an application-level QoS contract guarantees that an application's end-to-end deadline is met. To understand how this is possible, it is necessary to recapitulate what the satisfaction of the end-to-end deadline actually means. As explained in Chapter 2.4, for an application a_i, the end-to-end deadline D_i^{E2E} specifies the maximum amount of time that must elapse from the release of the first task until the completion of the last task in the task graph. Therefore, the satisfaction of the end-to-end deadline is based on the satisfaction of the local deadlines of all $\tau_{i,j} \in \mathcal{T}_i$.

In order to construct an application-level QoS contract based on the task-level QoS contracts of all $\tau_{i,j} \in \mathcal{T}_i$, however, it must be discussed how these contracts can be combined, i.e., by contract composition or by contract conjunction (cf. Sec. 4.3).

Recall that contract composition is applied when the models of multiple components of a system are combined to a more complex component. At first glance, this seems to be appropriate, since the task dependencies given by the task graph lead to the formation of something more complex, namely, the application itself. However, as already elucidated in Sec. 4.4.2, the task dependencies are dissolved once the application is executed, owing to the release offsets enforced by the resource manager and the buffering of input data. In consequence, it is possible to consider each task and, therefore, the satisfaction of each task-level QoS contract independently, which leads to the following definition of an application-level QoS contract:

Definition 6 (Application-Level QoS Contract)**.** For an application a_i admitted to the considered system, an *application-level QoS contract* is defined as $\mathscr{C}_i = \bigwedge_{\tau_{i,j} \in \mathcal{T}_i} \mathscr{C}_{i,j}$.

Accordingly, an application-level QoS contract requires that all task-level QoS contracts are satisfied, i.e., that each task $\tau_{i,j} \in \mathcal{T}_i$ satisfies its local deadline that has been determined based on an independent worst-case response time analysis. Based on Def. 6, the following theorem can be formulated:

[3]Note that it is in general possible to repeat the admission process for a rejected application if multiple mappings exist, excluding the mapping(s) under which the application could not be accepted.

Theorem 2. For an application-level QoS contract $\mathscr{C}_i$ of an application admitted after undergoing the proposed system admission process, it holds that $a_i \vdash \mathscr{C}_i$ if $\tau_{i,j} \vdash \mathscr{C}_{i,j} \forall \tau_{i,j} \in \mathcal{T}_i$.

Proof. This follows from the definition of an application-level QoS contract. □

Note that it is assumed that all functions and algorithms used in the system admission process are correct, which can be verified a priori, and that the system (specifically, the schedulers) operates correctly, i.e., that the release offsets are enforced and uncompleted tasks are aborted at their local deadlines.

4.4.4 Remarks

The problem of admitting an application as introduced in Chapter 3.2 to a system including multiple resources corresponds to the problem of scheduling DAGs on multiprocessor systems which has been extensively addressed in the literature. Being combined in the context of the proposed system admission process, the individual elements, i.e., the idea of mapping tasks to resources, of assigning release offsets and local deadlines, as well as of performing independent worst-case response time analyses per resource and combining latencies along a maximum (or so-called critical) path, are not new but build on existing knowledge. In the proposed system admission process, they are harnessed in order to establish application-level QoS contracts whose satisfaction can be verified at run-time (for further details on the detection of contract violations refer to Sec. 4.5).

Regarding the scheduling of tasks with precedence constraints on multiprocessor systems, distinct models can be considered such as the fork/join model [LKR10], the synchronous parallel model [SAL+11], and DAG models, i.e., representations of dependent tasks as a directed acyclic graph. When modeling applications[4] as DAGs, three primary types of scheduling approaches are typically applied: global scheduling, federated scheduling, and decomposition-based scheduling.

Under global scheduling, tasks of an application can in general be executed on any suitable resource or processor, respectively, so that no explicit mapping (or partitioning) step is required. As the name suggests, one central scheduler is in charge of all resources. Addressing global scheduling, a generalized model for sporadic DAGs is introduced in [BBM+12], where also two schedulability tests are proposed to determine if an application can be scheduled under the global earliest deadline first (EDF) policy on homogeneous resources. In the schedulability tests, however, the existence of no more than one DAG is assumed. Building on [BBM+12], [BMS+13] considers global EDF and global deadline monotonic (DM) scheduling, providing schedulability tests and speedup bounds, i.e., informally, a quantification of the maximum speedup that can be achieved by executing an application on multiple resources instead of a single one. This result is further improved by [Bar14].

[4]Note that the terminology varies among different works. However, for the sake of consistency, it is adjusted with the one used in this dissertation.

[LAL+13] addresses scheduling applications under global EDF and offers a capacity augmentation bound, i.e., a metric that can be used as a schedulability test. Another capacity augmentation bound for global EDF is proposed by [SGJ+18]. A worst-case response time analysis for sporadic applications under global fixed-priority scheduling is offered in [FNN17]. For computing the worst-case response time of applications with arbitrary deadlines under global EDF, an analysis technique is provided in [WJG+19]. Limited-preemptive global fixed-priority scheduling, i.e., preemption can only occur at specific points, is addressed by [SMB+16], where a worst-case response time analysis for sporadic applications is provided. In [NNB19], release jitter and execution time uncertainty are taken into consideration for applications scheduled under limited-preemptive global job-level fixed-priority policies; a technique for computing the best- and worst-case response times of jobs is provided. A recent approach to computing tight worst-case response times under limited-preemptive global fixed-priority scheduling is presented in [CZG+23]. Considering full preemption, [PVS18] proposes a two-level global preemptive fixed-priority scheduling algorithm, where priorities are assigned to each application and task, so that the highest-priority task of the highest-priority application is always chosen for execution. For this scheduling algorithm, a schedulability test is presented. Server-based global scheduling, where each task is executed in a dedicated reservation server that must be scheduled, is addressed by [AA22] for a special type of applications, namely, for periodic, pseudo-harmonic applications. A simulation-based approach is provided for computing response-time bounds. The scheduling of applications with typed tasks, i.e., tasks that require to be executed on a specific resource type, on heterogeneous resources is considered in [SFS+22], where a worst-case response time analysis is proposed that is based on reconstructing the DAG structure of an application. A model for sporadic applications represented as conditional DAGs, i.e., where only one of certain parallel (alternative) paths within a DAG must be executed, is introduced in [BBM15].

With respect to federated scheduling, the underlying principle is similar to the one of global scheduling except for a subset of tasks that are exclusively mapped to specific resources, which cannot be used for the execution of the remaining tasks. The first federated scheduling algorithm for sporadic applications on homogeneous resources is proposed by [LCA+14], where implicit deadlines are considered, i.e., the end-to-end deadline of an application equals the minimum inter-arrival time, followed by [Bar15b] considering constrained deadlines, i.e., the end-to-end deadline is not larger than the minimum inter-arrival time. An approach to federated scheduling of applications with arbitrary deadlines is presented in [Bar15a] and of applications represented as conditional DAGs in [Bar15c]. For applications with constrained deadlines, [Che16] shows that federated scheduling algorithms do not have constant speedup factors with respect to an optimal scheduling algorithm. In [Bar16], mixed-criticality systems (cf. Chapter 2.4) are considered and an algorithm for scheduling applications in dual-criticality systems is provided. Another federated scheduling algorithm for mixed-criticality systems is offered by [LFA+17], where also capacity

augmentation bounds are given for dual- and multi-criticality systems. For applications with arbitrary deadlines, [UBC+18] proposes a reservation-based scheduling approach, where specific reservation servers are selected for each application, which, in turn, can be scheduled by other multiprocessor scheduling algorithms. A response-time analysis technique for applications with arbitrary deadlines scheduled under standard federated scheduling is presented in [WJG+19]. [HZL+21] distinguish two types of tasks, namely, heavy and light tasks and provide an algorithm for mapping heavy tasks to dedicated resources. Additionally, a worst-case response time analysis for light tasks is given. The federated scheduling of applications with typed tasks on heterogeneous resources is addressed by [LSU+23], where type-aware federated scheduling algorithms are proposed and respective worst-case response time analyses are provided that are based on analysis techniques for self-suspending tasks[5] under preemptive fixed-priority scheduling on uniprocessors. Combining ideas of federated and partitioned scheduling, [JGL+17] introduce an approach to semi-federated scheduling of applications with constrained deadlines, where multiple tasks can be mapped to specific resources while the remaining resources are (globally) shared by other tasks.

Unlike approaches under global and federated scheduling as discussed above, partitioned scheduling is based on dissolving the dependencies of tasks within an application and treating them as independent tasks. The decomposition of an application can be performed in order to enable partitioned scheduling, i.e., tasks are assigned to specific resources which employ individual schedulers independent of other resources' schedulers, but, however, can also be used to obtain independent tasks that are scheduled under a global policy. To allow the independent scheduling of tasks that are part of an application, it is necessary to resolve the task dependencies, i.e., the precedence constraints, given by the DAG structure, as also done during the system admission process proposed in this dissertation.

For achieving task independence, [SAL+11] introduces the initially mentioned synchronous parallel task model, an early and restricted model for the parallel execution of tasks, where synchronization points are inserted before and after each part of an application, in which tasks are executed in parallel. These synchronization points are determined by assigning local deadlines based on the task density which take the execution order of tasks into account. Referring to the model of [SAL+11], [CBN+18] provides a partitioning algorithm as well as a worst-case response time analysis using techniques for segmented self-suspending tasks with non-preemptable segments. [NBG+12] also considers partitioned scheduling under the synchronous parallel task model, aiming to reduce the number of resources required to schedule a set of applications. For assigning local deadlines, an online algorithm is proposed that takes the task density into account. Another offline approach for assigning release offsets and local deadlines is provided in [WGD14], where task dependencies are resolved in order to perform per-resource response-time analyses under EDF using established techniques for uniprocessors. Distinguishing between so-called heavy and

[5]For further information on self-suspending tasks refer to Chapter 6.2

light tasks, another approach for the assignment of release offsets and local deadlines for the scheduling under EDF is proposed by [QGM14]. [JLG+16] provides a global EDF algorithm relying on local deadlines that are obtained by means of a presented decomposition algorithm. This algorithm is based on so-called structure characteristic values, which can also be applied in the context of schedulability tests. In [FNN+16], a worst-case response-time analysis for partitioned fixed-priority scheduling is provided based on analysis methods for self-suspending tasks. More precisely, each application is modeled as a set of self-suspending tasks, which are obtained by means of a proposed algorithm that characterizes the worst-case scheduling scenario of an application.

From the discussed works, it becomes evident that the decomposition of applications and the worst-case response time analysis of the resulting individual tasks is indeed an established practice in the field of DAG scheduling on multiprocessors or multiple resources, respectively. In particular, when assigning deadlines and release offsets, it is common to proceed along the maximum (or critical) path of an application and, thereon, to recursively consider tasks on paths parallel to the maximum (or critical) path. The chosen decomposition approach as well as the criteria according to which local deadlines are dimensioned depend on each work's optimization objective and therefore are not necessarily interchangeable. The strategy adopted in the system admissin process proposed in this dissertation, namely, assigning a task's worst-case response time as its local deadline, has been selected in order to guarantee that tasks can be completed within their local deadlines.

The choice of decomposition strategies, however, is also related to the order of the different steps, in which the mapping of tasks to resources, the assignment of deadlines and release offsets, ad the worst-case response time analysis of tasks as well as of an application are performed. In this dissertation, the order is given by the proposed system admission process; nevertheless, this process is not claimed to be the only valid solution for admitting applications to a system, but rather is one possible workflow. Another approach is presented by [HCC+21], where a model for conditional applications executed on heterogeneous resources is proposed as well as multiple heuristic algorithms for the mapping of tasks to resources. Moreover, a schedulability analysis is provided. Here, unlike in this dissertation, the assignment of local deadlines is not linked to the analysis; instead, local deadlines are assigned based on the worst-case execution time of a task plus a certain share of the application's slack, i.e., of the difference between the end-to-end latency and the sum of WCETs of the tasks on the maximum (or critical) path. Schedulability analyses per resource are performed after the assignment of local deadlines and release offsets is completed. Neither of the two approaches exhibits a clear advantage.

Beyond the field of DAG scheduling on multiprocessors or multiple resources, the topic of compositional timing analysis, i.e., of integrating separate worst-case response time analyses, has also been thoroughly investigated. Noteworthy are the works of [SL03], [SL04], and [SL05], which introduce a compositional framework for real-time systems that aims to independently analyze non-functional properties per system component and to compose these at the system level. In this context, a hierarchy

of schedulers is considered, for which several resource models and corresponding schedulability analyses are proposed. While this framework is indeed powerful, it considers systems with a complexity that is considerably higher than the one of the system described in Chapter 3.1[6]. Nevertheless, this dissertation follows the underlying idea of the respective works, namely, the independent analysis of non-functional properties and their higher-level composition, which is realized by the proposed concept of QoS contracts. Although the QoS contracts discussed in this chapter consider timing properties only, it is intended to cover further parameters in the future (cf. Chapter 5). In addition to the respective extensions proposed in Chapter 5, further enhancements are imaginable and desirable, especially in light of current challenges related to the safety/security and sustainability of systems.

Not to remain unmentioned, [Ern05] proposes an important approach to the worst-case response time analysis of hierarchical systems, based on which the analysis tool PyCPA [JE12] has been developed[7].

4.5 Detection of Contract Violations

Despite an a-priori verification of the correctness of the system admission process and the system behavior, especially of the schedulers maintained by the resource agents, it is possible that a QoS contract is not satisfied. Since different factors can lead to such a *contract violation*, it is not only desirable to detect contract violations online, i.e., at run-time, but also to draw conclusions about their causes.

In Sec. 4.5.1, possible reasons for a contract violation are discussed. Thereon, in Sec. 4.5.2, an introduction into Metric Interval Temporal Logic (MITL) [AFH96] is given which is used to provide a constraint-based description of the system behavior expected under the satisfaction of QoS contracts in Sec. 4.5.3. How these constraints can be used for the detection of contract violations and for identifying their reasons is explained in Sec. 4.5.4.

4.5.1 Reasons for Contract Violations

In this chapter, two causes of contract violations are considered, each of which is assumed to occur rarely, namely, a *specification error* and a *scheduler fault*.

Specification Error

As stated in Chapter 3, the worst-case execution time of each task is characterized using time estimates instead of worst-case execution time upper bounds, which do not necessarily cover the worst case. If one or multiple jobs of a task $\tau_{i,j}$ exhibit an execution time that exceeds the time estimate, the assumptions $\mathscr{A}_{i,j} = \{c_{i,j}, P_{i,j}\}$

[6] Recall that the system considered in this chapter does not comprise hierarchical schedulers. One scheduler exists per resource and each resource is abstracted by a resource agent. The resource manager must not be mistaken for a scheduler.

[7] PyCPA is used in the context of the evaluation of this chapter (cf. Sec. 4.6).

of the task-level QoS contract $\mathscr{C}_{i,j}$ are violated. In consequence, the guarantees $\mathscr{G}_{i,j} = \{D_{i,j}\}$ cannot be given, since they have been computed based on $\mathscr{A}_{i,j}$ and are only valid if $\mathscr{A}_{i,j}$ is satisfied. Note that a deviation from the time estimates may also result from malicious attacks such as code injection attacks, however, this case is beyond the scope of this work.

Scheduler Fault

Although the correctness of the system has been verified a priori, a scheduler fault can occur at some point during the system's run-time, e.g., a transient fault caused by environmental influences or a permanent fault resulting from a malicious attack. In the presence of such a fault, the scheduler does not enforce the task release offsets and/or does not abort uncompleted jobs of tasks at their local deadlines. Consequently, not only the QoS contract $\mathscr{C}_{i,j}$ of an affected task $\tau_{i,j}$ is violated, but also the task-level QoS contracts of potentially all lower-priority tasks scheduled on the same resource, since their de facto response times may be prolonged due to the interference of $\tau_{i,j}$ and thus may exceed the previously computed wort-case response times that have been used for the local deadline assignments.

4.5.2 Metric Interval Temporal Logic

To formally specify the expected behavior of a system over time, Metric Interval Temporal Logic (MITL), introduced by Alur et al. [AFH96], can be used. In this dissertation, a variant of MITL by Maler and Nickovic [MN04] is adopted that, in contrast to the proposal in [AFH96], considers only time intervals of finite length.

The state of a system is represented by a set of state variables $\{x_1, \ldots, x_n\}$ with $n \in \mathbb{N}$. The state space of the system is defined as $\mathbb{S} = \mathbb{S}_1 \times \cdots \times \mathbb{S}_n$, where $\mathbb{S}_i$ is the value domain of variable x_i. A *signal* is a function $\sigma : \mathbb{T} \to \mathbb{S}$ from the discrete time domain $\mathbb{T}$ to the state space and represents a behavior of the system. The set of all signals over $\mathbb{S}$ that a system can exhibit is referred to by $\mathbb{\Sigma}$. $\sigma[t] \in \mathbb{S}$ denotes the value of signal σ at time instant t, i.e., the system state at time t. A Boolean expression over $\{x_1, \ldots, x_n\}$ that evaluates to a Boolean value for each system state $\sigma[t] \in \mathbb{S}$ is denoted a *predicate* π.

The syntax of MITL over a set of predicates is defined by the grammar

$$\varphi := \pi \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \, \mathcal{U}_{[a,b]} \, \varphi_2$$

where φ is a *property*, $[a, b]$ is an interval of finite length over $\mathbb{T}$ with endpoints $t + a, t + b \in \mathbb{T}$ relative to a time instant t, and $\mathcal{U}_{[a,b]}$ is the time-constrained *until* operator. Further derived standard operators can be used such as the time-constrained *eventually* operator $\Diamond_{[a,b]} \, \varphi := \mathit{true} \, \mathcal{U}_{[a,b]} \, \varphi$ and the time-constrained *always* operator $\Box_{[a,b]} \, \varphi := \neg\Diamond_{[a,b]}\neg\varphi$.

The set of signals that satisfy a property φ is denoted by $\mathbb{\Sigma}_\varphi \subseteq \mathbb{\Sigma}$. The *satisfaction relation* $\vDash$ indicates if a signal σ satisfies a property φ at time t.

Definition 7 (Satisfaction Relation)**.** The satisfaction relation is defined as follows:

$$
\begin{array}{rcl}
(\sigma,t) \vDash \pi & \Leftrightarrow & \pi(\sigma[t]) \equiv true \\
(\sigma,t) \vDash \neg\varphi & \Leftrightarrow & (\sigma,t) \nvDash \varphi \\
(\sigma,t) \vDash \varphi_1 \vee \varphi_2 & \Leftrightarrow & (\sigma,t) \vDash \varphi_1 \text{ or } (\sigma,t) \vDash \varphi_2 \\
(\sigma,t) \vDash \varphi_1 \,\mathcal{U}_{[a,b]}\, \varphi_2 & \Leftrightarrow & \exists t' \in [t+a, t+b] \text{ s.t. } (\sigma,t') \vDash \varphi_2 \\
 & & \text{and } \forall t'' \in [t,t') : (\sigma,t'') \vDash \varphi_1 \\
(\sigma,t) \vDash \Diamond_{[a,b]}\, \varphi & \Leftrightarrow & \exists t' \in [t+a, t+b] \text{ s.t. } (\sigma,t') \vDash \varphi \\
(\sigma,t) \vDash \Box_{[a,b]}\, \varphi & \Leftrightarrow & \forall t' \in [t+a, t+b] : (\sigma,t') \vDash \varphi
\end{array}
$$

Note that the satisfaction relation as defined in this dissertation relaxes the constraints of the definition of Maler and Nickovic [MN04], where $(\sigma,t) \vDash \varphi_1 \,\mathcal{U}_{[a,b]}\, \varphi_2 \Leftrightarrow \exists t' \in [t+a, t+b]$ s.t. $(\sigma,t') \vDash \varphi_2$ and $\forall t'' \in [t,t'] : (\sigma,t'') \vDash \varphi_1$.

Let a specific set of signals be denoted as *events*. If for an event $\sigma[t] \equiv true$ at a time instant t, but $\sigma[t-1] \equiv false$, the event is said to *occur* at time t. For each task $\tau_{i,j}$, the following events can be defined that can occur at most once per (minimum) inter-arrival time:

- $\sigma_{i,j}^{rel} := true$ if a job of $\tau_{i,j}$ is released,
- $\sigma_{i,j}^{start} := true$ if a job of $\tau_{i,j}$ has been granted at least one time unit by the scheduler, i.e., has been executed for at least one time unit,
- $\sigma_{i,j}^{comp} := true$ if a job of $\tau_{i,j}$ has been completed,
- $\sigma_{i,j}^{abort} := true$ if a job of $\tau_{i,j}$ has been aborted, and
- $\sigma_{i,j}^{est} := true$ if a job of $\tau_{i,j}$ has been executed for the number of time units specified by its time estimate.

With each event, a predicate is associated that indicates if the respective event has occurred within the current inter-arrival time:

- $\pi_{i,j}^{rel} := true$ if $\sigma_{i,j}^{rel} \equiv true$, and $\pi_{i,j}^{rel} := false$ otherwise,
- $\pi_{i,j}^{start} := true$ if $\sigma_{i,j}^{start} \equiv true$, and $\pi_{i,j}^{start} := false$ otherwise,
- $\pi_{i,j}^{comp} := true$ if $\sigma_{i,j}^{comp} \equiv true$, and $\pi_{i,j}^{comp} := false$ otherwise,
- $\pi_{i,j}^{abort} := true$ if $\sigma_{i,j}^{abort} \equiv true$, and $\pi_{i,j}^{abort} := false$ otherwise,
- $\pi_{i,j}^{est} := true$ if $\sigma_{i,j}^{est} \equiv true$, and $\pi_{i,j}^{est} := false$ otherwise.

4.5.3 Specification of the System Behavior

The satisfaction of the task-level QoS contract of each task $\tau_{i,j}$ implies that the system operates correctly. This expected system behavior is subsequently expressed by means of MITL constraints. Note that all intervals are interpreted relative to the release of a considered job of $\tau_{i,j}$ and that redundancies are required to allow for a more fine-grained reasoning about contract violations.

Start at Least Once per Scheduling Interval

Each job of a task $\tau_{i,j}$ must be started at least once, i.e., must be granted at least one time unit by the scheduler before the next job is released.

$$\pi_{i,j}^{rel} \implies \Box_{[0,P_{i,j}]} \left(\Diamond_{[0,D_{i,j}]} \, \pi_{i,j}^{start} \right) \tag{4.1}$$

No Local Deadline Overrun

Between its release and the local deadline, each job of $\tau_{i,j}$ must be either completed or aborted at some point. Accordingly, no job of $\tau_{i,j}$ overshoots its local deadline.

$$\pi_{i,j}^{rel} \implies \Box_{[0,P_{i,j}]} \left(\Diamond_{[0,D_{i,j}]} \left(\pi_{i,j}^{comp} \vee \pi_{i,j}^{abort} \right) \right) \tag{4.2}$$

No Overexecution

To ensure that the scheduler does not allow any job to be executed longer than indicated by its time estimate, each job of $\tau_{i,j}$ must be either completed or aborted if it has been executed for the amount of time specified by its time estimate, at some point between its release and the local deadline.

$$\pi_{i,j}^{rel} \implies \Box_{[0,P_{i,j}]} \left(\Diamond_{[0,D_{i,j}]} \; \left(\pi_{i,j}^{comp} \vee \left(\pi_{i,j}^{est} \implies \pi_{i,j}^{abort} \right) \right) \right) \tag{4.3}$$

Timely Termination according to Time Estimate

Each job is required to be completed until its local deadline without being executed longer than indicated by its time estimate. This can be expressed by two individual constraints. First, each job of $\tau_{i,j}$ must be completed at some point between its release and the local deadline.

$$\pi_{i,j}^{rel} \implies \Box_{[0,P_{i,j}]} \left(\Diamond_{[0,D_{i,j}]} \, \pi_{i,j}^{comp} \right) \tag{4.4}$$

Second, the schedule must be designed in such a way that each job can be completed at all, i.e., at some point between the release and the local deadline, each job of $\tau_{i,j}$ must have been executed for the amount of time indicated by its time estimate.

$$\pi_{i,j}^{rel} \implies \Box_{[0,P_{i,j}]} \left(\Diamond_{[0,D_{i,j}]} \, \pi_{i,j}^{est} \right) \tag{4.5}$$

4.5.4 Reasoning about Contract Violations

By monitoring the MITL constraints specifying the expected system behavior of a correctly operating system at run-time, it is possible to detect different types of contract violations and to identify their causes.

No Execution

If Eq. (4.1) is violated for a task $\tau_{i,j}$, the considered job of $\tau_{i,j}$ has not been started at all before its deadline. This contract violation indicates a scheduler fault.

Time Estimate too Small

If Eq. (4.4) is violated, but Eq. (4.5) holds for a task $\tau_{i,j}$, it is not possible to complete the considered job of $\tau_{i,j}$ by executing it for the amount of time indicated by its time estimate. This results from a deviation of the job's execution time from the time estimate of $\tau_{i,j}$, which follows from a specification error.

Insufficient Execution Time

If both Eq. (4.4) and Eq. (4.5) are violated, the scheduler did not ensure that the considered job of $\tau_{i,j}$ can be executed for the amount of time specified by its time estimate, which indicates a scheduler fault.

Time Estimate Overrun

If Eq. (4.3) is violated, the scheduler did not abort the considered job at its local deadline, but grants more time for its execution than specified by the time estimate of $\tau_{i,j}$. This contract violation indicates the existence of a scheduler fault and of a specification error at the same time.

Deadline Overrun

If Eq. (4.2) is violated, the scheduler did not abort the considered job at its local deadline. Note that this contract violation differs from a time estimate overrun and indicates a scheduler fault only.

Time Estimate Underrun

Note that the MITL constraints can also be used to detect if a task's time estimate has been dimensioned too conservatively. More precisely, if Eq. (4.4) holds and Eq. (4.5) is violated, the execution time of the considered job of $\tau_{i,j}$ was shorter than the time estimate of $\tau_{i,j}$. This behavior does not cause a contract violation, however, a frequent detection of a time estimate underrun could be communicated to the developers of the respective application in order to determine a more realistic time estimate, which contributes to avoiding resource overreservation.

4.6 Evaluation

To evaluate the proposed system admission approach for concluding QoS contracts and to validate the monitoring-based reasoning about contract violations, simulations have been performed using an event-based simulator. Subsequently, details about the simulator will be provided, followed by a discussion of the two evaluations and their respective results.

4.6.1 Simulator

An event-based simulator representing a system as described in Chapter 3 has been implemented in Python 3.9. The simulation workflow consists of three stages, as illustrated in Fig. 4.2, namely, the data generation, the system admission simulation, and the scheduling simulation. The output of the data generation serves as input for the system admission simulation, and, in turn, the output of the system admission simulation serves as input for the scheduling simulation. As an output of the scheduling simulation, diagrams are generated automatically that display the schedules and the activity of the resource monitor for each resource in the system. In the following, the stages of the simulation workflow are described in detail.

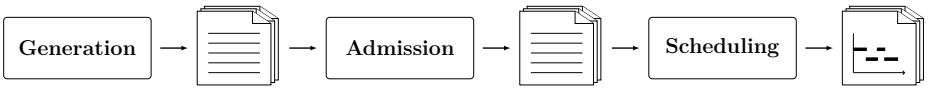


Figure 4.2: Schematic overview of the simulation workflow.

Data Generation

The considered system topology as well as the set of applications are created synthetically. In the course of the data generation, several parameters are chosen randomly that are based on a common, configurable random seed to allow for reproducibility. First, a topology, i.e., a resource graph, consisting of a given number of resources with a given number of resource types is generated. The number of resources and resource types can be configured. With a subset of resources, connected endpoint gateway types are associated. In a second step, a set of directed acyclic graphs is generated, where each DAG corresponds to one application. For the generation of applications, multiple parameters can be configured such as the number of tasks, the period, and the utilization of an application. During the generation of an application, the DAG structure is constructed first, where forks and joins are chosen randomly to create parallel paths. Subsequently, the application parameters are created according to the following strategy: The period[8] and the application's utilization are selected randomly out of a configurable set of periods and utilization values, respectively. Based on the period and the utilization, the overall execution time of the application is computed, which is distributed as the tasks' time estimates. The end-to-end deadline of the application is chosen out of the closed interval between the latency of the maximum path, i.e., the path in the DAG exhibiting the largest sum of time estimates, and the period. Resource type and endpoint gateway type requirements are assigned to the tasks randomly. After the generation, it is verified for each application if it can be mapped to the resource graph; if this is not the case, the application is re-generated.

[8]The simulator considers periodic tasks instead of sporadic tasks (cf. Chapter 2.1), which is not an uncommon practice in the literature [ANN+22]; due to the properties of sporadic and periodic tasks, the results are anyway representative.

System Admission Simulation

The system admission simulation implements the system admission process proposed in Sec. 4.4.2. To compute a mapping of a considered application to the resource graph, a number of constraints is encoded that need to be satisfied if the task graph is represented correctly on the resource graph, and solved by the z3 SMT solver [MB08]. *Satisfiability modulo theories (SMT)* refers to the automatic satisfiability verification of logic constraints restricting the interpretation of all symbols to the same logic background theory [BT18]. If a solution, i.e., a set of variables, exists for which all constraints can be satisfied, one solution is provided by the SMT solver. Accordingly, if a solution is retrieved, it is chosen as the application's mapping; otherwise, the application is rejected. To assign the local deadlines, a worst-case response-time analysis is carried out for each task using pyCPA [DAT+], a Python implementation of the compositional performance analysis approach [HHJ+06; HAE17], whereat a rate-monotonic priority assignment (cf. Chapter 2.2) is considered. Note that the worst-case response-time analysis technique used by pyCPA is a busy-window analysis as proposed by [Leh90], which is applicable for periodic real-time tasks as considered in this evaluation. For details on the internals of the analysis as implemented in pyCPA refer to [JE12]. If a set of tasks is deemed unschedulable on a resource, the application is rejected. Otherwise, the computed worst-case response times are assigned as the local deadlines of the respective tasks. The release offsets of all tasks are computed recursively, beginning with the first task in the task graph. Finally, it is verified if the end-to-end latency satisfies the end-to-end deadline of the application. If this is not the case, the application is rejected. Otherwise, the tasks' local deadlines are extended by the so-called slack, i.e., the difference between the end-to-end deadline of the application and the actual end-to-end latency, proportional to a task's time estimate, i.e., such that a task with a larger time estimate receives a longer prolongation of its local deadline. Note that during the admission stage no task execution is simulated, but only resource reservation for the admitted applications takes place.

Scheduling Simulation

To simulate the execution of applications on the considered system, a set of resource instances is created, each being endowed with a fixed-priority preemptive scheduler (cf. Chapter 2.2). All schedulers operate according to a global notion of discrete time, i.e., each unit of simulation time corresponds to one unit of execution time on each resource. When the simulation is started, for each task, a set of predicates corresponding to those introduced in Sec. 4.5.2 is initialized, which are updated after each elapsed time unit. The resource monitors indicated in the schematic overview of the system abstraction in Fig. 3.1 (cf. Chapter 3) are implemented such that for each task on a resource a monitor is instantiated that checks the set of constraints provided in Sec. 4.5.3 after each simulation time unit. Note that the monitors have been implemented manually. However, for real-world systems, it can be meaningful to include a module that generates monitors directly from MITL formulae, as done, for

instance, in [MN04], in order to allow for easy extensibility with respect to monitoring functionalities without additional implementation effort. To validate the approach proposed in this chapter, however, this would not generate any additional value in terms of gaining further insights. If a monitor detects a contract violation and draws conclusions regarding its origin according to the conditions provided in Sec. 4.5.4, the violation is recorded. Based on the recorded data, diagrams are automatically created for each resource at the end of the simulation that illustrate task-wise when the respective monitor has checked the monitoring constraints and when which type of contract violation has been detected.

4.6.2 Comparison with Related Work

The proposed system admission process has the purpose of computing QoS contracts that guarantee the satisfaction of an application's QoS requirement. To evaluate it, the related approach of [NSE11] can be considered for comparison: In a shared execution platform with multiple CPUs that exhibits a layered architecture, similar to the two layers of the system model described in Chapter 3, a central control instance analyzes the effects that the execution of a new application on the system would have on the overall system behavior and decides about its admission accordingly. Unlike the system admission process proposed in this chapter, the analysis in [NSE11] relies on an end-to-end response-time analysis based on compositional performance analysis [HHJ+06], i.e., a global, centralized analysis that contrasts the distributed analysis of the proposed admission process. Both approaches are compared by means of simulations using the previously introduced event-based simulator, including an extension that implements a global response-time analysis. For the ease of readability, the proposed approach is subsequently referred to as the *distributed approach* and the approach from [NSE11] as the *global approach*. The detailed experiment setup is described subsequently, followed by a discussion of the results.

Experiment Setup

As described in Sec. 4.6.1, the event-based simulator uses pyCPA to compute the tasks' response times. To provide a comparable implementation of the global response-time analysis considered in [NSE11], an additional end-to-end analysis function relying on pyCPA is implemented. In order to compare both approaches, the time required for the analyses is measured, assuming the mapping of the application to the resource graph to be given, i.e., not including it into the measurements[9]. When considering the distributed approach, the time needed for the complete admission process (without the mapping) is measured. This includes the response-time analyses, the composition of the local deadlines, and the comparison of the resulting end-to-end latency against the application's end-to-end deadline (cf. Sec. 4.4.2). Since the simulator used for

[9]Since the run-time of the SMT solver the mapping is based on can vary strongly, the results could otherwise be distorted.

the evaluation simulates a distributed system, but does not perform the analysis operations on distinct resources in parallel, as it is possible on the resources of a real-world system, the maximum analysis time over all resources is considered as the duration of the overall analysis process.

For the evaluation, randomly generated system topologies with 3 resource types, representing computation, wired, and wireless communication, are considered. More precisely, a topology consisting of 5, 10, and 20 resources is contemplated, representing different parts of a smart city. For each topology, the system admission simulation is performed under different scenarios, i.e., for different configurations of the parameters used for the generation of applications. To reduce the potential bias resulting from the randomness of the generated data, the simulation under each scenario is carried out with 10 distinct sets of applications, where each set comprises 50 applications.

For the generation of applications, the following configuration parameters are considered: In all scenarios, the periods are selected out of the set $\{50, 100, 200, 1000\}$ ms, which is based on the set of periods that is typically used in automotive systems [KZH15][10]. To explore the impact of different parameters on the performance of the proposed approach, in different simulation scenarios, different numbers of tasks per application as well as different application utilization values are considered. In one scenario, the number of tasks is fixed while the utilization is altered, whereas in another scenario the utilization is fixed and the number of tasks is varied. More precisely, for a randomly chosen number of tasks per application out of the interval $[1, 20]$, application sets are generated, where the applications' utilization is selected out of the sets $\{0.1, 0.2\}$, $\{0.1, 0.2, 0.3, 0.4\}$, and $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$. Moreover, for utilization values chosen out the set $\{0.1, 0.6\}$, the number of tasks per application is selected from the intervals $[1, 10]$, $[1, 20]$, and $[1, 30]$. Note that simulations have been performed under further configurations, but due to the similarity of the results, these are omitted here.

Results

In fact, the absolute time measured for both approaches has little informative value since it is related to the simulator only, and, therefore, is not representative[11]. Instead, the ratio between the time measurements for the global and the distributed approach is contemplated, which allows to infer the order of magnitude of the difference in time required by both approaches. The resulting ratios of computation time show by which factor the global approach is more time intensive than the distributed approach.

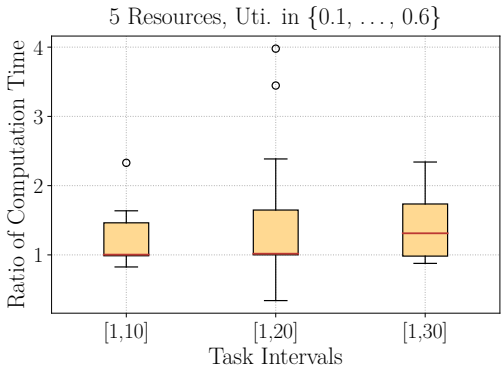
[10] Note that the complete set of periodic tasks in automotive systems is typically activated according to periods out of the set $\{1, 2, 10, 20, 50, 100, 200, 1000\}$. However, since the considered system model described in Chapter 3 comprises constrained deadlines only and the implemented simulator requires each task to have an execution time of at least 1 ms, short periods have been omitted in order to reduce the time required to generate a valid application.

[11] Note that the run-time can vary in a real-world system depending on the hardware and can be optimized by using different programming languages and strategies.

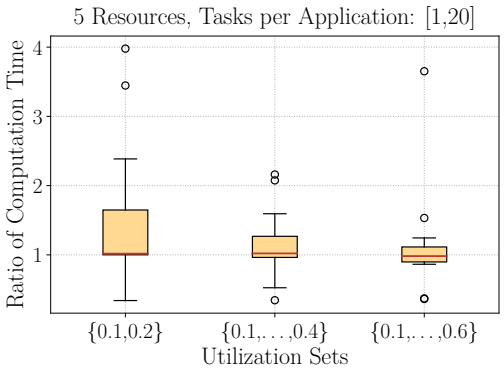
The simulation results are discussed subsequently, illustrated by boxplots that can be understood as follows: The box, termed *interquartile range* IQR, represents the middle 50 % of data points, where the red line marks the median. Accordingly, the lower border of the box indicates the middle value between the smallest data point and the median, denoted by $Q1$, whereas the the upper border of the box indicates the middle value between the largest data point and the median, denoted by $Q3$. The black so-called *whiskers* indicate the distribution of data points outside the interquartile range; the lower whisker marks $Q1 - 1.5 \cdot IQR$ and the upper whisker $Q3 + 1.5 \cdot IQR$. So-called *outliers*, i.e., data points that are located outside of the area limited by the whiskers, are indicated by circles. Note that each plot represents the results of a simulation for one set of applications.

In Fig. 4.3a and Fig. 4.3b, the obtained ratios of computation time are illustrated for a system consisting of 5 resources; analogously, Fig. 4.3c and Fig. 4.3d show the results for a system consisting of 10 resources, and Fig. 4.3e and Fig. 4.3f for a system with 20 resources. From all figures, it becomes apparent that the ratio of computation time is larger than 1 in many, but not in all cases. For instance, as shown in Fig. 4.3d, in a system with 10 resources, where the application utilization is chosen from the set $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$, a median of 1.48, an upper whisker of 6.15, and a lower whisker of 0.41 for the ratio of computation time are obtained in a scenario where the number of tasks per application is in the interval $[1, 10]$, a median of 1.11, an upper whisker of 4.12, and a lower whisker of 0.51 in a scenario where the number of tasks per application is in the interval $[1, 20]$, and a median of 0.98, an upper whisker of 3.03, and a lower whisker of 0.5 in a scenario where the number of tasks per application is in the interval $[1, 30]$. For comparison, as depicted in Fig. 4.3f, in a system with 20 resources, where the application utilization is also chosen from the set $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$, a median of 2.15, an upper whisker of 11.25, and a lower whisker of 0.39 is obtained in a scenario where the number of tasks per application is in the interval $[1, 10]$, a median of 3.31, an upper whisker of 9.08, and a lower whisker of 0.5 in a scenario where the number of tasks per application is in the interval $[1, 20]$, and a median of 2.09, an upper whisker of 9.23, and a lower whisker of 0.51 in a scenario where the number of tasks per application is in the interval $[1, 30]$. Based on these and further results, it can be stated that the distributed approach requires considerably less time than the global approach in the majority of simulated cases, although variations exist from scenario to scenario.

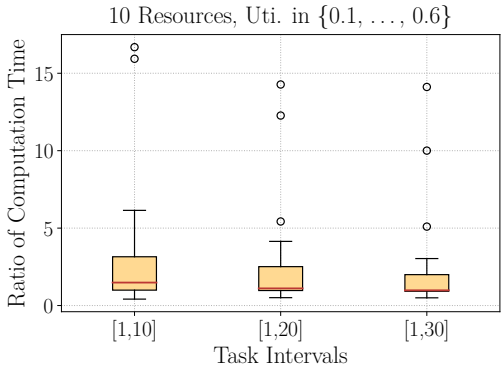
Aiming to analyze, which parameters affect the ratio of computation time, it can be conceived that the number of tasks per application does not have a decisive impact, as visible, e.g., in Fig. 4.3d and Fig. 4.3f, where no clear trend is discernible.



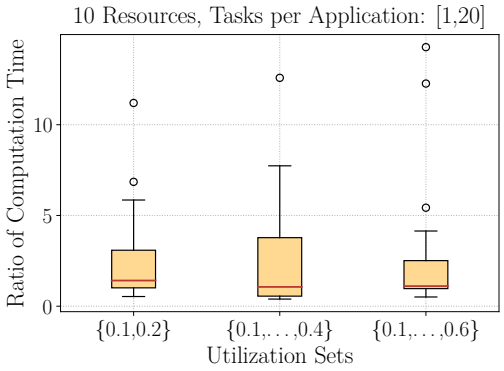
(a) The ratio of computation time depending on the intervals from which the number of tasks per application is chosen for a system with 5 resources.



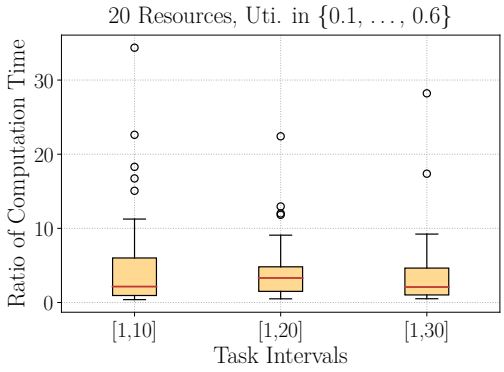
(b) The ratio of computation time depending on the sets out of which the utilization of an application is chosen for a system with 5 resources.



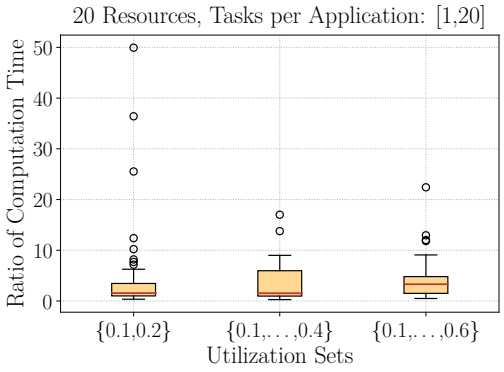
(c) The ratio of computation time depending on the intervals out of which the number of tasks per application is chosen for a system with 10 resources.



(d) The ratio of computation time depending on the sets out of which the utilization of an application is chosen for a system with 10 resources.



(e) The ratio of computation time depending on the intervals out of which the number of tasks per application is chosen for a system with 20 resources.



(f) The ratio of computation time depending on the sets out of which the utilization of an application is chosen for a system with 20 resources.

Figure 4.3: The ratio of the time required by the global and distributed approach depending on different parameters. Higher is better.

Although in Fig. 4.3b, where the application utilization is again chosen from the set $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$, a trend exists with respect to the median – here, a median of 2.14, an upper whisker of 11.25, and a lower whisker of 0.39 is obtained in a scenario where the number of tasks per application is in the interval $[1, 10]$, a median of 3.31, an upper whisker of 9.08, and a lower whisker of 0.5 in a scenario where the number of tasks per application is in the interval $[1, 20]$, as well as a median of 2.09, an upper whisker of 9.23, and a lower whisker of 0.51 in a scenario where the number of tasks per application is in the interval $[1, 30]$ – this result is not sufficiently expressive against the background of further measurements to claim a correlation between the number of tasks per application and the ratio of computation time.

A similar statement can be made with respect to the impact of the set out of which the utilization of an application is chosen on the ratio of computation time. For clarification, compare Fig. 4.3b, Fig. 4.3d, and Fig. 4.3f, where simulation results are illustrated considering applications with a number of tasks out of the interval $[1, 20]$: As shown in Fig. 4.3b, a median of 1.02, an upper whisker of 2.39, and a lower whisker of 0.34 are obtained in a scenario where the application utilization is chosen out of the set $\{0.1, 0.2\}$, a median of 1.02, an upper whisker of 1.59, and a lower whisker of 0.52 in a scenario where the application utilization is chosen out of the set $\{0.1, 0.2, 0.3, 0.4\}$, and a median of 0.98, an upper whisker of 1.24, and a lower whisker of 0.86 for a scenario where the application utilization is chosen out of the set $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$. In Fig. 4.3d, a median of 1.41, an upper whisker of 5.85, and a lower whisker of 0.53 are obtained in a scenario where the application utilization is chosen out of the set $\{0.1, 0.2\}$, a median of 1.06, an upper whisker of 7.74, and a lower whisker of 0.39 in a scenario where the application utilization is chosen out of the set $\{0.1, 0.2, 0.3, 0.4\}$, and a median of 1.12, an upper whisker of 4.12, and a lower whisker of 0.51 in a scenario where the application utilization is chosen out of the set $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$. Lastly, in the results depicted in Fig. 4.3f, a median of 1.55, an upper whisker of 6.26, and a lower whisker of 0.36 is obtained in a scenario where the application utilization is chosen out of the set $\{0.1, 0.2\}$, a median of 1.53, an upper whisker of 9, and a lower whisker of 0.28 in a scenario where the application utilization is chosen out of the set $\{0.1, 0.2, 0.3, 0.4\}$, and a median of 3.31, an upper whisker of 9.08, and a lower whisker of 0.5 in a scenario where the application utilization is chosen out of the set $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$. As regarding the number of tasks per application, no clear impact of the sets out of which the utilization of an application are chosen on the ratio of computation time can be perceived from the simulation results.

However, when comparing the results, it becomes evident that another parameter has a clear impact on the ratio of computation time, namely, the system size, i.e., the number of resources. When examining the medians of a scenario across different system sizes, it is conceivable that they increase with an increasing system size, which indicates that the ratio of computation time and, consequently, the advantage of the distributed approach over the global approach in terms of computation time becomes greater as the system size grows. This can be explained by the fact that

the availability of a larger number of resources allows to map tasks requiring the same resource type to distinct resources, for which reason the time required for the response-time analysis on each resource is reduced under the distributed approach[12].

4.6.3 Validation of the Detection of Contract Violations

To validate the effectiveness of the detection of contract violations by means of the proposed monitoring-based approach, a scheduling simulation is performed using the event-based simulator, into which scheduler faults and specification errors are injected. Subsequently, the experiment setup is described, followed by a discussion of the results.

Experiment Setup

For an example system consisting of 8 resources, to which a set of applications has been admitted, a scheduling simulation is performed. First, the scheduling simulation is carried out without any modifications. Second, a scheduler fault as well as a specification error (cf. Sec. 4.5.1) are injected on one resource and for one task, respectively. More precisely, the scheduler fault is injected into resource r_6, on which the set of tasks $\{\tau_4, \tau_5, \tau_7, \tau_9\}$ is scheduled, where $\tau_7 = \{c_7 = 8, P_7 = 100, \omega_7 = 15, D_7 = 8, \Pi_7 = 1\}$, $\tau_4 = \{c_4 = 15, P_4 = 50, \omega_4 = 18, D_4 = 20, \Pi_4 = 2\}$, $\tau_5 = \{c_5 = 2, P_5 = 50, \omega_5 = 38, \Pi_3 = 3\}$, and $\tau_9 = \{c_9 = 40, P_9 = 100, \omega_9 = 0, D = 100, \Pi_9 = 4\}$. Note that application indices are omitted here to improve readability. The specification error is injected into task τ_7, leading to an increased execution time of its first job.

Results

During the unmodified, i.e., fault- and error-free, scheduling simulation, as expected, no contract violations have been detected. A visualization of the resulting schedule is omitted since it does not provide any knowledge gain. For the scheduling simulation under fault and error injection, the resulting scheduling diagram that is automatically generated by the simulator is shown in Fig. 4.4, where it is contextualized with a schematic representation of the system topology.

[12]Recall that the response-time analysis has not only the purpose to verify the schedulability of the tasks set on the resource, but also to determine a local deadline for each task (cf. Sec. 4.4.2). Accordingly, if multiple tasks of an application are mapped to the same resource, multiple response-time analyses must be performed successively, since it is not possible to verify the schedulability of a task set, i.e., to check for deadline violations, when local deadlines have not yet been assigned.

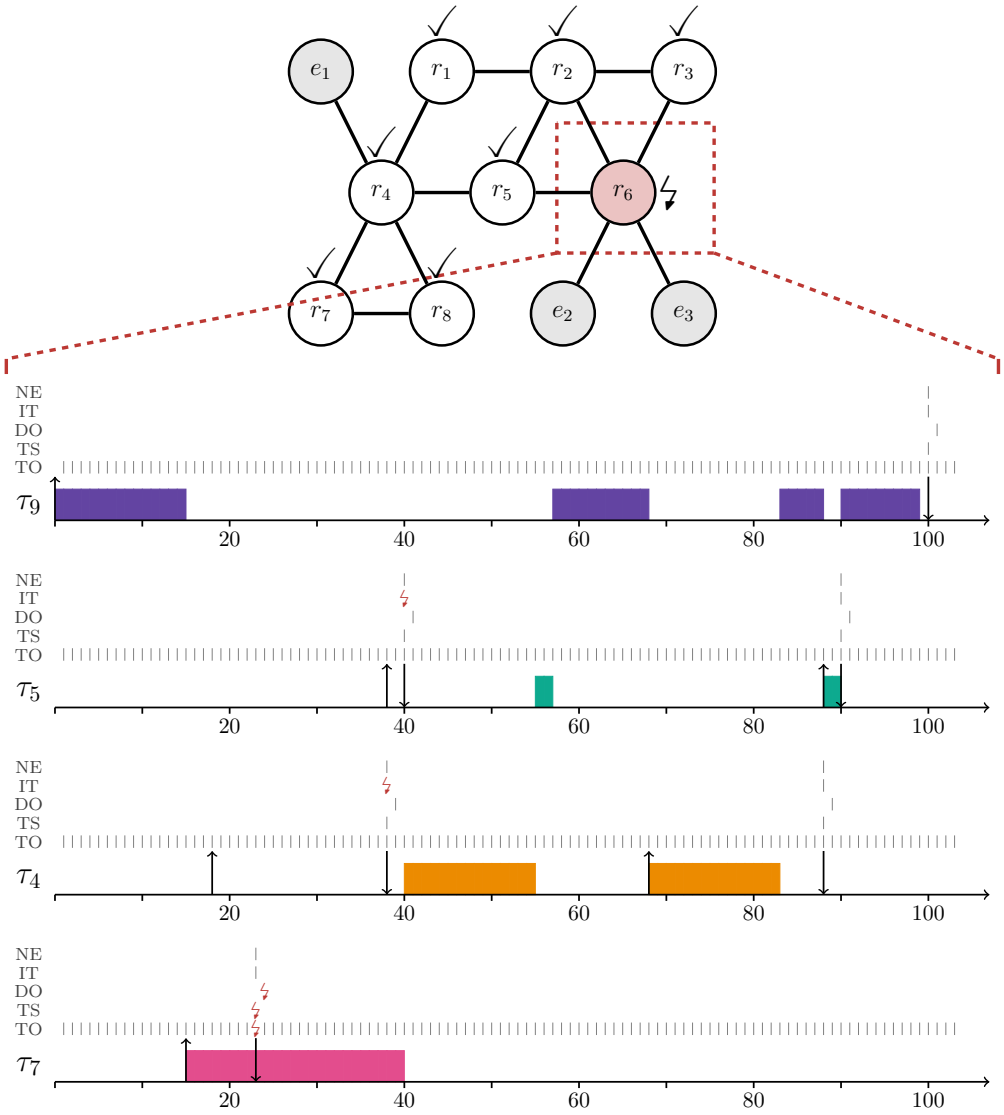


Figure 4.4: Scheduling diagram for one resource of an example system that includes indications of the moments in which the monitors evaluate the conditions for detecting different types of contract violations. Tasks scheduled on the resource: $\tau_7 = \{c_7 = 8, P_7 = 100, \omega_7 = 15, D_7 = 8, \Pi_7 = 1\}$, $\tau_4 = \{c_4 = 15, P_4 = 50, \omega_4 = 18, D_4 = 20, \Pi_4 = 2\}$, $\tau_5 = \{c_5 = 2, P_5 = 50, \omega_5 = 38, \Pi_3 = 3\}$, and $\tau_9 = \{c_9 = 40, P_9 = 100, \omega_9 = 0, D = 100, \Pi_9 = 4\}$.

The diagram can be understood as follows: The release times, local deadlines, and execution intervals of each job are indicated as common in the literature (cf. Chapter 2.1). The points in time when the monitor for each task evaluates the conditions for detecting different types of contract violations, as proposed in Sec. 4.5.4, are indicated by vertical lines above the timeline, on which each task's execution is

depicted, where *NE* abbreviates *no execution*, *IT* abbreviates *insufficient execution time*, *DO* abbreviates *deadline overrun*, *TS* abbreviates *time estimate too small*, and *TO* abbreviates *time estimate overrun*. If a contract violation is detected, this is indicated by a red lightning symbol. Note that although all predicates are updated after each unit of simulation time, not all contract violations become evident immediately, for which reason the related conditions do not need to be evaluated at each point in time. Since no statement can be made about the contract violations of type *time estimate too small*, *insufficient execution time*, and *no execution* before a job's deadline, the monitor checks for these types of violations at each job's deadline. Analogously, it evaluates the conditions for a contract violation of type *deadline overrun* at one time unit after a job's local deadline.

During all simulations carried out, neither false positives nor false negatives have occurred with respect to any of the above contract violation types[13]. In fact, both false positives as well as false negatives are in general possible, although only under specific circumstances. Whether a contract violation is detected or not, depends on the satisfaction of the constraints proposed in Sec. 4.5.3 that, in turn, is based on the values of the included predicates. Consequently, to false-positively detect a contract violation, the value of one or more predicates must be incorrect, which can result, for instance, from bit flips due to electromagnetic interference or from a malicious attack. With respect to false negatives, one specific case must be noticed, namely, the co-existence of a *time estimate too small* and of an *insufficient execution time* contract violation regarding the same task. Under these conditions, only *insufficient execution time* can be detected while *time estimate too small* remains invisible. However, since *insufficient execution time* indicates a scheduler faults that affects not only the considered task, this trade-off appears to be acceptable. Owing to the fact that the proposed monitoring approach is based on the system and application model considered in this chapter, more false negatives can occur if the approach is transferred to a real-world system, where unknown factors potentially come into play. Therefore, it is meaningful to extend the approach according to the characteristics of the respective system and the requirements of its users.

4.7 Summary

To provide QoS guarantees for all applications executed on a distributed system as described in Chapter 3, which serves as the backbone of a smart city and is shared by a set of applications that changes over time, the concept of QoS contracts has been introduced in this chapter. More precisely, an approach for concluding application-level QoS contracts has been proposed that consist of a conjunction of task-level QoS contracts. This contract conjunction, enabled by the considered system design, simplifies the verification of contract satisfaction on the application level,

[13]Note that all injected faults and errors are kept track of during the simulation and can be compared with the simulator's output.

compared to approaches based on contract composition, since the satisfaction of an application-level QoS contract can be verified by verifying the satisfaction of all related task-level QoS contracts. Moreover, the proposed system admission approach relies on distributed response-time analyses, offering an advantage in terms of computation time compared to approaches applying global response-time analyses. In simulations based on synthetic data, it was shown that the proposed approach outperforms a related approach in [NSE11], which makes use of global response-time analyses, in the majority of cases and increases in time-efficiency with a growing system size.

To detect violations of the proposed QoS contracts that can result from distinct causes, a set of MITL constraints describing the expected system behavior has been proposed, based on which run-time monitors can be designed that are able to distinguish different types of contract violations. By means of simulations under fault- and error-injection, this approach has been proven to be effective. In fact, the provided MITL constraints and the resulting monitoring approach offer benefits that extend beyond the objective of this chapter. On the one hand, the information about the reason for a contract violation that is obtained by the monitors can be used to invoke suitable reconfiguration measures to the system in order to re-establish the contract satisfaction for all applications executed on the system. On the other hand, the retrieved monitor data can be analyzed using meta-monitoring strategies. For instance, it is possible to observe how frequently a specific application's contract is violated within a sequence of application instances. The resulting information can be exploited when further QoS parameters such as, e.g., dependability requirements are taken into consideration during the system admission process and shall be guaranteed by means of contracts. This idea is further explored in the following chapter.

Notation	Meaning
$\mathscr{A}$	Assumptions of a contract
$\mathscr{C}$	Contract
$\mathscr{C}_i$	Application-level QoS contract
$\mathscr{C}_{i,j}$	Task-level QoS-contract
$\mathscr{C}_1 \otimes \mathscr{C}_2$	Contract composition
$\mathscr{C}_1 \wedge \mathscr{C}_2$	Contract conjunction
$\mathscr{G}$	Guarantees of a contract
κ	Component
$\kappa \vdash \mathscr{C}$	A component satisfies a contract
φ	Property
$\mathfrak{M}_i$	Mapping of the task graph of application a_i
$\mathfrak{M}_i^{(j,\ell)}$	Submapping of an application a_i
$\mathfrak{P}_i^{(j,\ell)}$	Path in a mapping from $\tau_{i,j}$ to $\tau_{i,\ell}$
π	Predicate
$R_{i,j}$	Worst-case response time of a task $\tau_{i,j}$
$\mathbb{S}$	State space of the system
$\mathbb{S}_i$	State domain of x_i
σ	Signal
$\mathbb{\Sigma}$	Set of all signals the system can exhibit over $\mathbb{S}$
$\mathbb{T}$	Discrete time domain
t	Time instant
$\mathcal{U}_{[a,b]}$	Time-constrained until operator
x_i	State variable
$\Diamond_{[a,b]}$	Time-constrained eventually operator
$\Box_{[a,b]}$	Time-constrained always operator
$\models$	Satisfaction relation

Table 4.1: Overview of the notation introduced in Chapter 4.

CHAPTER 5

Robustness-Aware Quality of Service Contracts

Contents

5.1 Introduction

The time estimate of a task does not necessarily reflect its worst-case execution time, as discussed in Chapter 2.3. In consequence, the actual execution demand of a task can be larger than the time estimate, especially, if the execution time depends on external factors that cannot be fully anticipated in test scenarios by the application developers. Revisiting the use case of this thesis (cf. Fig. 5.1), the number of traffic participants at a smart city's intersection, for instance, can have a strong impact on the time required by object detection and trajectory planning tasks of autonomous vehicles. From Chapter 4 it is known that cases, in which a task's execution time demand

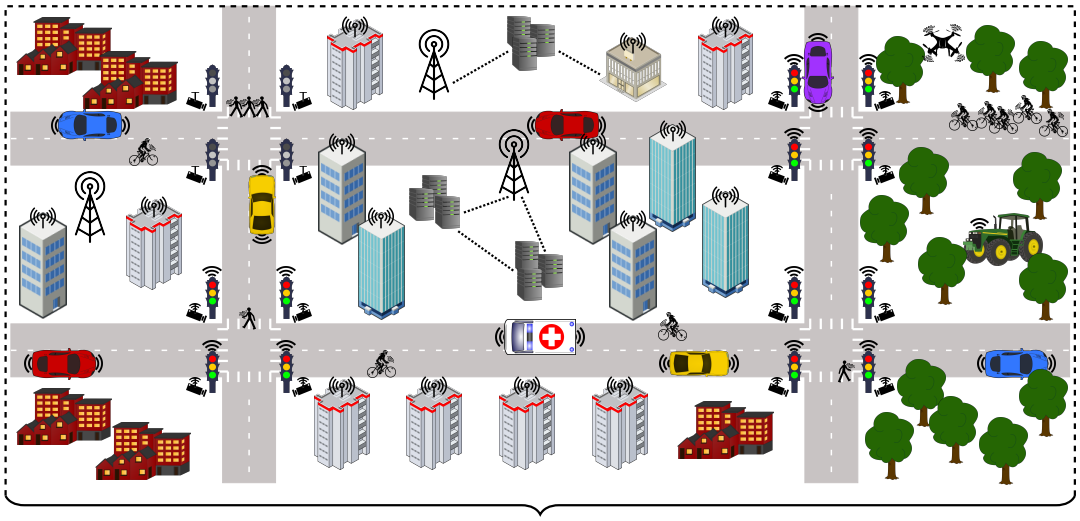


Figure 5.1: On distributed systems underlying a smart city, applications with multifaceted QoS requirements can be executed. This chapter extends the concept of QoS contracts as well as their construction process, such that robustness requirements of applications are taken into account.

exceeds the time estimate, can lead to a violation of the QoS contract concluded between the corresponding application and the system. This follows from the fact that QoS contracts are constructed based on the information, i.e., characteristics and requirements, provided by the applications and can be fulfilled only while these remain valid.

To prevent contract violations, worst-case execution time bounds (cf. Chapter 2.3) can be considered instead of time estimates, leading, however, to two drawbacks: First, the usage of worst-case execution time bounds implies an overreservation of resources, i.e., more time is reserved for a task than needed by the majority of jobs. Second, and following from the first, the chance of applications with high resource demands to be admitted to the system is reduced, since insufficient resource capacity for satisfying the application's QoS requirement can remain available due to other applications' overreservation. As an alternative, probabilistic worst-case execution time (cf. Chapter 2.3) can be used to describe an execution time distribution covering manifold execution scenarios, although considering the pWCET without further modifications of the system does not suffice to avoid contract violations. In fact, the system admission process described in Chapter 4.4.2 takes only one execution time value per task into account. Selecting a single execution time value, however, leads to cases similar to those occurring when worst-case execution time bounds or time estimates are considered. Accordingly, when application characteristics are not described by fixed values, but exhibit a certain degree of variability, it is meaningful to

provide a more flexible kind of QoS guarantees than achievable by the QoS contracts introduced in Chapter 4.

As discussed in Chapter 2.4, not each task necessarily requires a satisfaction of its temporal correctness requirement at any point in time, but may be able to sustain a correct function as long as it is satisfied for a minimum number of jobs. Analogously, the end-to-end latency requirement of an application does not always need to be satisfied by each application instance, but, depending on the application type, can underlie additional *robustness requirements*. In light of this, the idea of *soft contracts* introduced by [RBH+08] is adopted in this chapter. For applications having QoS requirements in terms of temporal correctness along with robustness requirements, a novel type of contracts is proposed that takes the variability of tasks' execution times into account and, in consequence, aims to reduce the risk of excessive resource overreservation as well as of immoderately frequent contract violations. Before this objective is described more in detail, an overview of related works is given that aim to provide QoS guarantees under variable application characteristics.

5.2 Related Work

One option for coping with dynamically changing parameters are probabilistic QoS guarantees, which, instead of specifying only one (typically pessimistic) QoS value, provide a probability distribution of QoS values, i.e., the achievement of each QoS value is guaranteed with the corresponding probability. Probabilistic QoS guarantees in terms of latency are frequently used when pWCET distributions are considered. To compute these, i.e., to derive probabilistic response times (considering individual tasks) and end-to-end latencies (considering DAGs of tasks), probabilistic schedulability analysis techniques can be applied. In [BCM+20], for instance, a probabilistic schedulability analysis approach is proposed considering fixed-priority partitioned scheduling for tasks with precedence constraints. For DAGs including tasks with different periods that are scheduled under EDF on partitioned multi-cores, an analysis method is presented in [LCH+22] and an approach for minimizing the probabilistic end-to-end latencies in [HK23]. Against the background of the system considered in this chapter, however, applying such methods is not meaningful. In fact, even under the usage of pWCET values instead of worst-case execution time upper bounds, standard (i.e, non-probabilistic) analysis techniques can be applied due to the system design: Recall that jobs are aborted at their local deadlines if they have not been completed. Therefore, the worst-case response time of each task scheduled on the system is upper-bounded by its local deadline.

Probabilistic QoS guarantees can also be encountered in the area of cloud computing, where on-demand requests exhibiting a high variability must be handled. In this context, typically not only end-to-end latencies, but also further QoS parameters are considered at the same time. For providing and satisfying these guarantees, load prediction and dynamic resource reservation techniques can be applied [QWS+22]. In

fact, similar approaches, are not applicable to the considered system, since resources are reserved statically during the system admission of an application and extending a job's local deadline afterwards would have an impact on all successor tasks as well as on the end-to-end latency.

Robustness guarantees (cf. Chapter 2.4) are another option for handling factors that exhibit a certain variability and are not necessarily controllable. Originating from the field of tolerance against deadline misses and faults, (m, k)-constraints can be considered. For guaranteeing these, two classes of approaches, i.e., static and dynamic approaches, exist that share the same basic idea: Granting additional time for fault compensation techniques such as (partial) re-execution to a certain number of jobs allows to ensure that at minimum these particular jobs are reliable. Using static approaches, the choice of reliable jobs is made offline, i.e., at design time [HR95; NQ06; QH00; KS95], while dynamic approaches perform a selection at run-time, for instance, based on monitoring [CBC+16] or on error prediction [SUC+23]. However, these approaches cannot be applied to the considered system, since it relies on static resource reservation.

5.3 Problem Statement

In this chapter, a system as described in Chapter 3 is considered. Regarding the applications that can be executed on the system, the following assumptions are made: Each task $\tau_{i,j}$ of an application a_i is characterized by a set $\mathfrak{C}_{i,j}$ of pWCET distributions, where each pWCET distribution $C_{i,j}$ is related to one particular resource type on which the task can be executed. A pWCET distribution consists of a set of execution time values $c_{i,j}^{p_{i,j}}$, where $p_{i,j}$ (with $0 < p_{i,j} \leq 1$ and $p_{i,j} \in \mathbb{Q}$) is the probability that a job of $\tau_{i,j}$ exhibits the respective execution time[1].

An application instance is said to be *completed successfully* if all of its jobs have been completed within their local deadlines. Consequently, the unsuccessful completion of an application instance is equivalent to the case that one or more jobs of the instance do not meet their local deadlines[2]. By design, the violation of a local deadline does not imply its overrun, since uncompleted jobs are by default aborted at their local deadlines (cf. Chapter 3), such that the end-to-end latency requirement D_i^{E2E} of an application a_i is always satisfied once it has been admitted to the system. Additionally, in this chapter, it is assumed that uncompleted jobs that uncompleted jobs are aborted after their maximum execution time (cf. Chapter 5.4) if they are not completed, to ensure that an execution time overshoot does not have an impact on other tasks scheduled on the same resource. However, if a job is aborted, its successor job(s) make use of older or default data, leading to an increased data

[1] To improve readability, the indices i and j of p will be omitted whenever the context is clear, i.e., $c_{i,j}^{p}$ will be written instead of $c_{i,j}^{p_{i,j}}$.

[2] Note that this is always assumed to result from an exceedance of the task's execution time value considered during the computation of the local deadline. The existence of scheduler faults or potential further causes is neglected in this chapter.

age and, therefore, potentially to a service degradation (e.g., a reduced accuracy of planned trajectories due to old sensor data) of the application.

Unlike in Chapter 4, it is assumed that not each instance of an application a_i needs to be completed successfully. Adopting and combining different parameters presented in Chapter 2.4, this property is expressed by an additional QoS requirement, denoted as *robustness requirement*, that is described by the (m, k)-criterion $\rho_i = (m_i, k_i)$ and the confidence level ζ_i. While ρ_i specifies the minimum number m_i out of k_i successive application instances (with $m_i, k_i > 0$) that must be completed successfully, ζ_i (with $0 < \zeta_i \leq 1$) indicates the minimum probability required for the satisfaction of ρ_i. Note that this way of using (m, k)-criteria deviates from the one that is most common in the literature (cf. Chapter 2.4). Considering a confidence level ζ_i in addition to the (m, k)-criterion has the purpose of compensating for the variability of the task execution time without overreserving resources, which would be necessary if ζ_i is omitted. In fact, the considered type of robustness requirement has the purpose of enabling *probabilistic robustness guarantees* that allow for a certain degree of flexibility.

Objective of this Chapter

For the considered system and the considered type of applications executable on the system, this chapter aims to propose a concept of robustness-aware QoS contracts that provide probabilistic robustness guarantees, i.e., that guarantee the satisfaction of QoS requirements consisting of an end-to-end latency requirement combined with a robustness requirement, and to provide an approach to constructing these.

In the following section, the notion of robustness-aware contracts, briefly referred to as *soft QoS contracts*, is introduced, which is evaluated later on in Sec. 5.5. An overview about the notation introduced in this chapter is provided in Table 5.1.

5.4 Soft Quality of Service Contracts

Soft QoS contracts can be concluded between the system and an application a_i to provide probabilistic robustness guarantees for a_i. More precisely, if a_i has been admitted to the system under a soft QoS contract, it must be guaranteed that its end-to-end latency requirement D_i^{E2E} is satisfied and that ρ_i holds, i.e., that at least m_i out of k_i successive application instances are completed successfully, with a minimum probability of ζ_i. Analogously to the QoS contracts proposed in Chapter 4, soft QoS contracts are assume-guarantee contracts, which can be classified into two types, namely, into task-level soft QoS contracts (introduced in Sec. 5.4.1) and into application-level soft QoS contracts (addressed in Sec. 5.4.2). A set of constraints that need to be satisfied in the course of the system admission process for concluding soft QoS contracts is proposed in Sec. 5.4.3.

5.4.1 Task-Level Soft QoS Contracts

A task-level soft QoS contract guarantees that a task $\tau_{i,j}$ of an application a_i is completed successfully until its local deadline $D_{i,j}$. This guarantee, however, is not given absolutely, i.e., not in the sense that each job is completed successfully, but with a certain probability that depends on the pWCET distribution of the respective task. In fact, the local deadline $D_{i,j}$ is assumed to be assigned based on a worst-case response time analysis for sporadic real-time tasks scheduled under a preemptive fixed-priority policy, analogously to Chapter 4.4.2, in which one particular execution time value $c^p_{i,j} \in C_{i,j}$ of $\tau_{i,j}$ is considered that is chosen during the mapping. To compute the worst-case response time of a task $\tau_{i,j}$, any suitable analysis can be applied, such as the one given by Theorem 3 that makes use of well-known standard analysis techniques.

Theorem 3. The worst-case response time $R_{i,j}$ of a task $\tau_{i,j}$ mapped to a resource $r_z, z \in \mathbb{N}$, on which tasks are feasibly scheduled according to a preemptive task-level fixed-priority policy, is upper-bounded by the minimum positive value of Δ for which

$$\Delta = c^p_{i,j} + \sum_{\tau_{x,y} \in hp(\tau_{i,j})} \left(\left\lceil \frac{\Delta}{P_{x,y}} \right\rceil + 1 \right) c^p_{x,y}$$

where $i, j, x, y \in \mathbb{N}$ and $hp(\tau_{i,j})$ is the set of tasks with higher priority than $\tau_{i,j}$ mapped to resource r_z with $D_{x,y} \leq P_{x,y}$.

Proof. This theorem can be proven analogously to Theorem 1 in Chapter 4.4.2. Note that all tasks mapped to the resource r_z are independent, sporadic real-time tasks and that the task under analysis $\tau_{i,j}$ is also a sporadic real-time task that is independent from all tasks mapped to r_z. Assume that all tasks mapped to r_z are associated with a fixed priority and that a concrete preemptive fixed-priority schedule exists for r_z. Assume this schedule to be work-conserving, i.e., the resource does not idle as long as a released job of a task exists which is neither completed nor aborted. Based on these assumptions, all jobs of tasks in $lp(\tau_{i,j})$, i.e., with lower priority than $\tau_{i,j}$, can be removed from the schedule, since they do not interfere with the execution of $\tau_{i,j}$ or of any task in $hp(\tau_{i,j})$. To quantify the interference of the tasks in $hp(\tau_{i,j})$ on $\tau_{i,j}$ within an interval $[t, t + \Delta)$ beginning at a point in time t, the maximum amount of time for which a task $\tau_{x,y} \in hp(\tau_{i,j})$ is executed within this interval must be determined. Recall that $D_{x,y} \leq P_{x,y}$ for all $\tau_{x,y} \in hp(\tau_{i,j})$ and that $D_{i,j} \leq P_{i,j}$ is required for $\tau_{i,j}$. Consider two scenarios: *a)* At time t, no job of $\tau_{x,y}$ exists that is neither completed nor aborted. *b)* At time t, a job of $\tau_{x,y}$ exists that is either completed or aborted within $[t, t + \Delta)$. In case *a)*, jobs of $\tau_{x,y}$ are released at most $\left\lceil \frac{\Delta}{P_{x,y}} \right\rceil$ times within $[t, t + \Delta)$. Since only one specific $c^p_{x,y}$ is considered after $\tau_{x,y}$ is mapped to a r_z and since any released job of $\tau_{x,y}$ is aborted after $c^p_{x,y}$ time units if it is not completed, each released job of $\tau_{x,y}$ is executed for for at most $c^p_{x,y}$ time units. Accordingly, $\tau_{x,y}$ is executed within $[t, t + \Delta)$ for at most $\left\lceil \frac{\Delta}{P_{x,y}} \right\rceil c^p_{x,y}$ time units. In case *b)*, $\tau_{x,y}$ is first executed for

up to $c^p_{x,y}$ time units, i.e., until the job released before t is completed or aborted. The value of $c^p_{x,y}$ upper-bounds the execution time of a job of $\tau_{x,y}$ for the same reason as explained with respect to case *a)*. Additionally, $\tau_{x,y}$ can be released within $[t, t+\Delta)$ at most $\left\lceil \frac{\Delta}{P_{x,y}} \right\rceil$ times. Thus, in case *b)* $\tau_{x,y}$ can be executed for at most $\left(\left\lceil \frac{\Delta}{P_{x,y}} \right\rceil + 1\right) c^p_{x,y}$ time units, where the execution time of a job of $\tau_{x,y}$ is upper-bounded by $c^p_{x,y}$ for the same reason as explained with respect to case *a)*. Therefore, case *b)* dominates case *a)*. The task $\tau_{i,j}$ under analysis itself is executed for at most $c^p_{i,j}$ time units, since, as explained above, only one $c^p_{i,j}$ is considered after $\tau_{i,j}$ is mapped to a r_z and since any released job of $\tau_{i,j}$ is aborted after $c^p_{i,j}$ time units if it is not completed. Although due to the release offsets of the tasks mapped to resource r_z it is possible that not each $\tau_{x,y} \in hp(\tau_{i,j})$ is executed within $[t, t+\Delta)$ at all, the maximum possible interference of all $\tau_{x,y} \in hp(\tau_{i,j})$ must be considered to cover the worst-case interference. □

A job of task $\tau_{i,j}$ is completed successfully before $D_{i,j}$ if the exhibited execution time does not overshoot the chosen $c^p_{i,j}$, i.e., if $\tau_{i,j}$ is not aborted after being executed for $c^p_{i,j}$ time units. For an arbitrary but fixed p, intuitively, the probability that a job of $\tau_{i,j}$ is completed successfully before $D_{i,j}$ corresponds to the probability that the job's execution time is at most $c^p_{i,j}$, denoted by $\mathbb{P}\left(\leq c^p_{i,j}\right)$. Since this probability relies on the value of p indicated by the pWCET distribution $C_{i,j}$ for the occurrence of the considered $c^p_{i,j}$, it is necessary that a correct pWCET distribution is communicated by an application during the system admission process. Moreover, to ensure a correct response-time analysis, the task's minimum inter-arrival time must be provided correctly, as explained in Chapter 4.4.1. Accordingly, a *task-level soft QoS contract* can be defined as follows:

Definition 8 (Task-Level Soft QoS Contract)**.** For a task $\tau_{i,j}$ of an application a_i scheduled on a resource of the considered system, a *task-level soft QoS contract* is defined as $\mathscr{C}_{i,j} = (\mathscr{A}_{i,j}, \mathscr{G}_{i,j})$ with assumptions $\mathscr{A}_{i,j} = \{C_{i,j}, P_{i,j}\}$ and guarantees $\mathscr{G}_{i,j} = \left\{D_{i,j}, \mathbb{P}\left(\leq c^p_{i,j}\right)\right\}$. If each job of $\tau_{i,j}$ complies with $C_{i,j}$ and $P_{i,j}$ as communicated in the system admission process, $\tau_{i,j} \vdash \mathscr{C}_{i,j}$.

5.4.2 Application-Level Soft QoS Contracts

Task-level soft QoS contracts can be combined to an application-level soft QoS contract by means of contract conjunction (cf. Chapter 4.3), since the task dependencies are dissolved once an application is executed, as explained in Chapter 4.4.2. Correspondingly, an *application-level soft QoS contract* can be defined as follows:

Definition 9 (Application-Level Soft QoS Contract)**.** For an application a_i admitted to the considered system, an *application-level soft QoS contract* is defined as $\mathscr{C}_i = \bigwedge_{\tau_{i,j} \in \mathcal{T}_i} \mathscr{C}_{i,j}$, where each $\mathscr{C}_{i,j}$ is a task-level soft QoS contract. Accordingly, it holds that $a_i \vdash \mathscr{C}_i$ if $\tau_{i,j} \vdash \mathscr{C}_{i,j} \forall \tau_{i,j} \in \mathcal{T}_i$.

5.4.3 System Admission Constraints

To understand the link between the probabilities $\mathbb{P}\left(\leq c_{i,j}^{p}\right)$ that a job of a task $\tau_{i,j}$ is completed successfully and the robustness requirement of an application a_i, recall that the confidence level ζ_i indicates the minimum probability required for the satisfaction of the (m,k)-criterion ρ_i. Accordingly, for the robustness requirement of a_i to be satisfied, it must hold that

$$\zeta_i \leq \mathbb{P}_i^{k_i} + \sum_{x=m_i}^{k_i-1} \frac{k_i!}{(k_i-x)!x!} \cdot \mathbb{P}_i^{x} \cdot (1-\mathbb{P}_i)^{(k_i-x)} \tag{5.1}$$

where $\mathbb{P}_i$ is the probability that the jobs of all tasks of the application are completed successfully, that is

$$\mathbb{P}_i = \prod_{j=0}^{n} \mathbb{P}\left(\leq c_{i,j}^{p_j}\right) \tag{5.2}$$

with $n = |\mathcal{T}_i|$. Note that the probability $\mathbb{P}\left(\leq c_{i,j}^{p_j}\right)$ is independent for all $\tau_{i,j}$ due to the system design.

For clarification, consider the following example: For an application a_1 with $\rho_1 = (2,3)$ and the probability $\mathbb{P}_i = 0.6$ that all jobs of an application instance are completed successfully (short: that *the application instance is successful*), all possible execution scenarios of a sequence of $k_1 = 3$ successive application instances are depicted in Fig. 5.2. The (m,k)-criterion ρ_1 is satisfied if either two out of three or all three successive application instances are successful. While in general only one execution scenario exists, where all application instances in an observed sequence are successful, multiple execution scenarios exist for the case that two out of three successive application instances are successful. To compute the probability that ρ_1 is satisfied, following well-known rules of stochastics, it is necessary to sum up the occurrence probabilities of all execution scenarios satisfying ρ_1, i.e., $\mathbb{P}_1^3+\mathbb{P}_1^2\cdot(1-\mathbb{P}_1)+\mathbb{P}_1^2\cdot(1-\mathbb{P}_1)+\mathbb{P}_1^2\cdot(1-\mathbb{P}_1) = 0.6^3+0.6^2\cdot0.4+0.6^2\cdot0.4+0.6^2\cdot0.4 = 0.684$. The computation of this probability is generalized by Eq. (5.1).

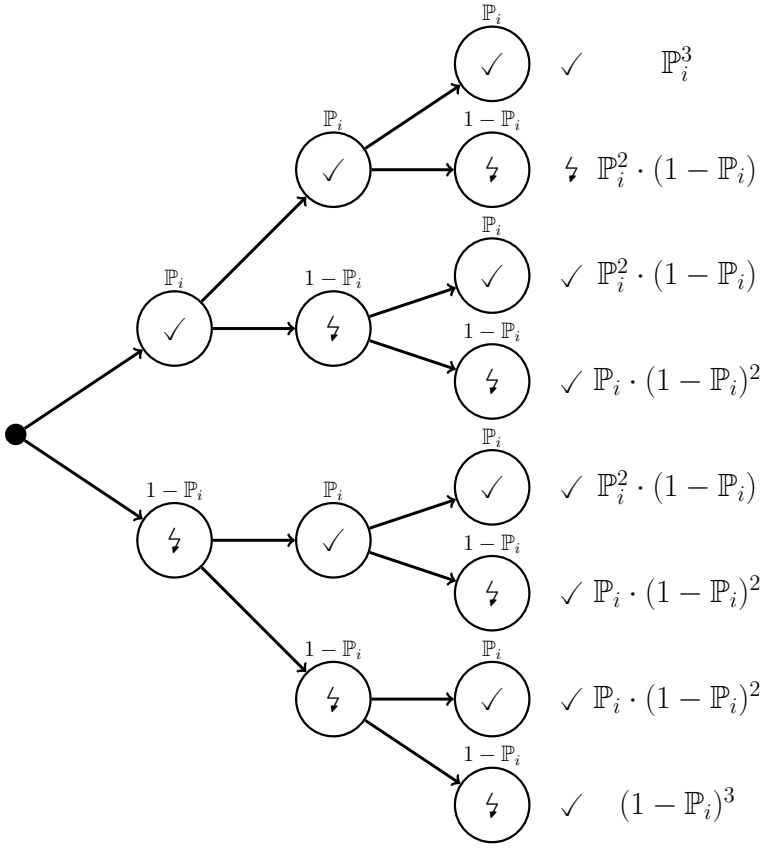


Figure 5.2: Possible execution scenarios of a sequence of $k_1 = 3$ successive application instances for an application a_1 with $\rho_1 = (2, 3)$ and $\mathbb{P}_i = 0.6$.

When an application undergoes the system admission process, the same approaches for computing local deadlines, release offsets, and the resulting end-to-end latency can be applied as during the system admission process introduced in Chapter 4. For this purpose, it is necessary to first choose an execution time value for each task $\tau_{i,j}$ out of the respective pWCET distribution $C_{i,j}$. In theory, $\mathbb{P}\left(\leq c_{i,j}^p\right)$, according to which the related $c_{i,j}^p$ can be selected, can be computed based on $\mathbb{P}_i$ by deriving a combination of the probabilities $\mathbb{P}\left(\leq c_{i,j}^p\right)$ for all $\tau_{i,j}$; however, $\mathbb{P}_i$ is unknown at this point. It is, in general, possible to compute possible values of $\mathbb{P}_i$ based on the known parameters, nevertheless, this is not trivial, since, depending on m_i and k_i, a polynomial with high degree must be solved.

Against this background, one possibility to design a system admission process is to formulate a set of constraints expressing the properties expected from an admitted application, for which a solution is computed by means of an SMT solver. Recall that as satisfiability modulo theories (SMT), the automatic satisfiability verification of logic constraints is denoted, where the interpretation of all symbols is restricted to the same logic background theory [BT18]. If a set of variables, termed *solution*, exists

under which all constraints are satisfied, it can be obtained using an SMT solver. However, if multiple solutions exist, an SMT solver does not necessarily provide the optimal solution. Accordingly, if an execution time value exists in the pWCET distribution $C_{i,j}$ of each task $\tau_{i,j}$ based on which values of $D_{i,j}$ and $\mathbb{P}\left(\leq c_{i,j}^{p}\right)$ can be computed that lead to a satisfaction of the application's QoS requirements, the respective execution time value can be determined and, moreover, the local deadlines can be retrieved for each $\tau_{i,j}$ by solving the following constraints:

$$\zeta_i \leq \mathbb{P}_i^{k_i} + \sum_{x=m_i}^{k_i-1} \frac{k_i!}{(k_i-x)!x!} \cdot \mathbb{P}_i^{x} \cdot (1-\mathbb{P}_i)^{(k_i-x)} \tag{5.3}$$

$$\mathbb{P}_i \quad = \quad \prod_{j=1}^{n} \mathbb{P}\left(\leq c_{i,j}^{p}\right) \quad \text{with } n = |\mathcal{T}_i| \tag{5.4}$$

$$\forall j, \quad \mathbb{P}\left(\leq c_{i,j}^{p}\right) \quad = \sum_{\{c_{i',j'}{}^{p} \in C_{i,j} \ : \ c_{i',j'}{}^{p_{i',j'}} \leq c_{i,j}^{p_{i',j'}}\}} p_{i',j'} \tag{5.5}$$

$$\forall j, \quad c_{i,j}^{p} \quad \in \quad C_{i,j} \tag{5.6}$$

$$D_{i,j} \quad \geq \quad R_{i,j} \tag{5.7}$$

$$comp(\mathfrak{M}_i^{(1,n)}) \quad \leq \quad D_i^{E2E} \quad \text{with } n = |\mathcal{T}_i| \tag{5.8}$$

The constraints can be understood as follows: Eq. (5.3) corresponds to Eq. (5.1) and requires that the application's robustness requirement is satisfied, as explained above. Eq. (5.4) specifies how $\mathbb{P}_i$ is computed, while Eq. (5.5) defines the computation of $\mathbb{P}\left(\leq \mathbb{P}_{i,j}^{p}\right)$. Eq. (5.6) enforces that only execution time values are chosen for a task $\tau_{i,j}$ that are contained in the corresponding pWCET distribution $C_{i,j}$. The remaining equations reflect operations known from the system admission process in Chapter 4. In fact, Eq. (5.7) requires that all local deadlines are dimensioned large enough to cover the response time of the respective task under the chosen execution time value, whereas Eq. (5.8) targets at the satisfaction of the application's end-to-end deadline, where $\mathfrak{M}_i^{(1,n)}$ is a submapping of the application from $\tau_{i,1}$ to $\tau_{i,n}$, as defined in Chapter 4.4.2. If a solution is found for which the constraints are satisfied, the application is accepted and the release offsets $\omega_{i,j}$ can be assigned for each task $\tau_{i,j}$ as explained in Chapter 4.4.2; otherwise, the application is rejected. Note that it is assumed that a mapping of the considered application has already been computed when the above constraint formulation is solved, for which reason no mapping-related constraints are included.

5.5 Evaluation

The proposed concept of soft QoS contracts has the purpose of providing QoS guarantees without overreserving resources, taking the variability of tasks' execution times as well as the applications' robustness requirements into account. To evaluate if and to which extent soft QoS contracts contribute to a reduction of resource reservations, the approach is compared against the concept of QoS contracts proposed in Chapter 4, subsequently referred to as hard QoS contracts. To this end, the event-based simulator described in Chapter 4.6.1 is extended. In the following, the experiment setup, including the modifications of the simulator, is described, followed by a discussion of the results.

5.5.1 Experiment Setup

To carry out simulations in which soft QoS contracts are concluded, two extensions to the simulator presented in Chapter 4.6.1 have been made, namely, to the application generation and to the system admission simulation. In order to generate applications for which soft QoS contracts can be concluded, it is necessary to create a pWCET distribution for each task. For this purpose, first, an application is created as explained in Chapter 4.6.1, where one time estimate is associated with each task. On this basis, a pWCET distribution is built: Out of a configurable interval, a value is randomly[3] chosen describing the probability that the execution time of a job of the respective task is no more than the related time estimate. The counterprobability is associated with a second generated execution time value that can overshoot the initial time estimate by a configurable factor. The probability that the execution time of a job does not exceed the initial time estimate is split up and each resulting probability is associated with generated execution time values that are shorter than the initial time estimate. The resulting pWCET distribution of each task includes at minimum 2 and at maximum 10 execution time values with associated probabilities. For each task, a confidence as well as an (m, k)-constraint are chosen from configurable sets.

A system admission process according to Sec. 5.4.3 has been implemented, relying on the z3 SMT solver [MB08] for solving the encoded constraints. The worst-case response time of each task for each individually considered execution time value is computed using pyCPA [DAT+], analogously to Chapter 4.6.1, since each task corresponds to a standard periodic[4] real-time task under fixed-priority preemptive scheduling if only one of its execution time values is contemplated. To reduce the simulation time, only one mapping is considered for each application, i.e., if multiple mappings exist for an application and the application's QoS requirements cannot

[3] Note that the same random seed as for the application generation is used for all other random operations to ensure the reproducibility of results.

[4] Recall that periodic instead of sporadic real-time tasks are considered in the simulations, as stated in Chapter 4.6.1.

be satisfied under the contemplated mapping, the admission process is not repeated under another mapping, but the application is rejected.

For the experiments, a topology consisting of 10 resources is generated as described in Chapter 4.6.1, on which different scenarios are simulated. Each scenario is simulated with 10 sets of applications to reduce the potential bias resulting from the random nature of the input data. For the generation of applications, the following parameters known from Chapter 4.6.2 are considered: The number of tasks per application is determined out of the interval $[1, 20]$. The application period is chosen out of the set $\{50, 100, 200, 1000\}$. The applications' utilization is selected out of the set $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$.

To evaluate the concept of QoS contracts, the number of accepted applications under soft QoS contracts is compared to the one under hard QoS contracts (cf. Chapter 4), where the impact of different factors on the number of accepted applications is investigated. Concretely the following scenarios are examined: Different intervals out of which the probability associated with a task's initial time estimate is chosen are considered, since it has a strong impact on the design of the pWCET distribution, namely, the intervals $[0.6, 0.7]$, $[0.7, 0.8]$, and $[0.8, 0.9]$. Moreover, different intervals out of which an application's confidence level is chosen are contemplated, i.e., $[0.1, 0.3]$, $[0.4, 0.6]$, $[0.7, 0.9]$. Also, sets of (m, k)-criteria with distinct strictness are taken into account, for which the strictness is defined based on the ratio $\frac{m}{k}$. More precisely, (m, k)-criteria from $(1, 3)$ to $(9, 10)$ are considered, where the (m, k)-criteria are referred to as *low* if $\frac{m}{k} \in (0, 0.3]$, as *medium* (short: *med*) if $\frac{m}{k} \in (0.3, 0.6]$, and as *high* if $\frac{m}{k} \in (0.6, 1)$.

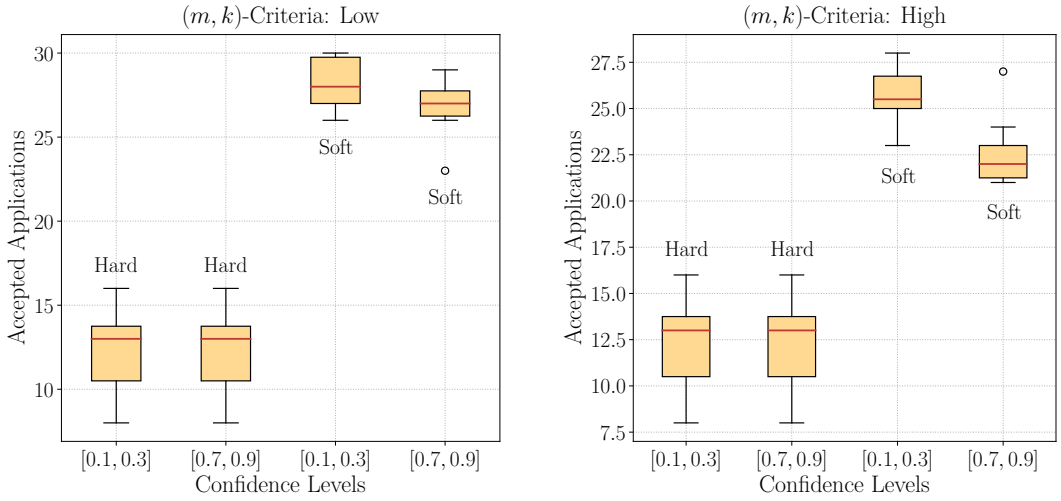
5.5.2 Results

First, it must be noted that, when conducting simulations considering different intervals out of which the probability that the execution time of a job is at most the initial time estimate of a task is chosen, no impact on the number of accepted applications can be observed, for which reason a visualization of the results is omitted. This is indeed plausible, since the respective probability influences the design of the pWCET distribution of a task, but is not directly involved in the system admission process. Instead, it is likely to rather have an impact on the number of contract violations when the application is scheduled; however, investigating this is beyond the scope of this chapter. Due to this lack of impact, in the following, only results for a respective probability out of the interval $[0.8, 0.9]$ are shown.

Recall that the boxplots visualizing the results must be understood as follows: The box, termed *interquartile range* IQR, represents the middle 50 % of data points, within which the red line marks the median. Accordingly, the lower border of the box indicates the middle value between the smallest data point and the median, denoted by $Q1$, whereas the the upper border of the box indicates the middle value between the largest data point and the median, denoted by $Q3$. The black so-called *whiskers* indicate the distribution of data points outside the interquartile range; the

lower whisker marks $Q1 - 1.5 \cdot IQR$ and the upper whisker $Q3 + 1.5 \cdot IQR$. So-called *outliers*, i.e., data points that are located outside of the area limited by the whiskers, are indicated by circles.

From Fig. 5.3 as well as from Fig. 5.4 it becomes evident that concluding soft QoS contracts leads to a higher number of accepted applications than concluding hard QoS contracts. For instance, in Fig. 5.3a, i.e., under low (m, k)-criteria, the median is 13, the upper whisker is 16, and the lower whisker is 8 under hard QoS contracts for both intervals of confidence levels, in contrast to a median of 28, an upper whisker of 30, and a lower whisker of 26 for confidence levels out of $[0.1, 0.3]$ under soft QoS contracts and to a median of 27, an upper whisker of 29, and a lower whisker of 26 for confidence levels out of $[0.7, 0.9]$ for soft QoS contracts.



(a) The number of accepted applications for different intervals out of which the confidence level of an application is chosen compared for low (m, k)-criteria.

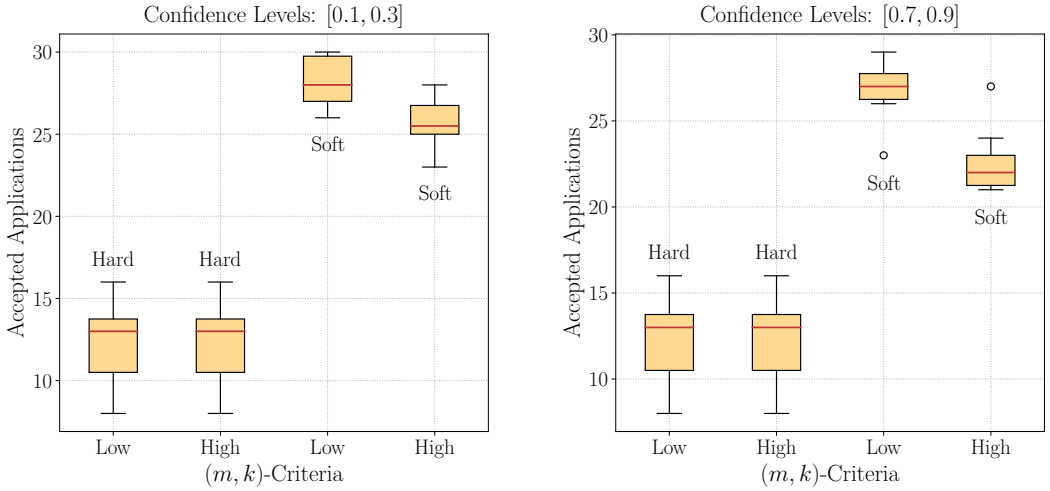
(b) The number of accepted applications for different intervals out of which the confidence level of an application is chosen compared for high (m, k)-criteria.

Figure 5.3: The impact of the interval out of which the confidence level of an application is chosen on the number of accepted applications is compared under the system admission process for QoS contracts (Hard) and the system admission process for soft QoS contracts (Soft) for low and high (m, k)-criteria. Higher is better.

A similar order of magnitude with respect to the difference of accepted applications under hard and soft QoS contracts can be found in other scenarios; in fact, the number of accepted applications is nearly doubled under soft QoS contracts compared to the results under hard QoS contracts. This can be explained as a consequence of the usage of execution time values shorter than the (initial) time estimates for the computation of local deadlines, which anyway suffice for satisfying the end-to-end deadline under the given robustness requirement. Thus, a reduction of resource overreservations

is achieved, which allows more applications to be executed on the system and thus satisfies the objective formulated in Sec. 5.3.

Investigating the impact of the type of (m, k)-criteria and of the interval out of which an application's confidence level is chosen, it is conceivable that both parameters do not influence the number of accepted applications under hard QoS contracts, since they are neither involved in the respective system admission process nor affect other relevant parameters. Under soft QoS contracts, however, a clear impact can be seen. When comparing Fig. 5.3a and Fig. 5.3b, it can be noticed that under higher confidence levels a lower number of applications is accepted (median of 28, upper whisker of 30, lower whisker of 26 for a confidence level in $[0.1, 0.3]$ and median of 27, upper whisker of 29, lower whisker of 26 for a confidence level in $[0.7, 0.9]$ in Fig. 5.3a, median of 25.5, upper whisker of 28, lower whisker of 23 for a confidence level in $[0.1, 0.3]$ and median of 22, upper whisker of 24, lower whisker of 21 for a confidence level in $[0.7, 0.9]$ in Fig. 5.3b).



(a) The number of accepted applications for different (m, k)-criteria compared for confidence levels out of $[0.1, 0.3]$.

(b) The number of accepted applications for different (m, k)-criteria compared for confidence levels out of $[0.7, 0.9]$.

Figure 5.4: The impact of different (m, k)-criteria on the number of accepted applications is compared under the system admission process for QoS contracts (Hard) and for soft QoS contracts (Soft) for applications' confidence levels chosen out of $[0.1, 0.3]$ and $[0.7, 0.9]$. Higher is better.

In Fig. 5.4a and Fig. 5.4b it can be perceived that higher (m, k)-criteria lead to a lower number of accepted applications compared to lower (m, k)-criteria. Concretely, a median of 28, an upper whisker of 30, and a lower whisker of 26 is encountered for low (m, k)-criteria compared to median of 25.5, an upper whisker of 28, and a lower whisker of 23 under confidence levels in $[0.1, 0.3]$ (cf. Fig. 5.4a). Analogously, a median of 27, an upper whisker of 29, and a lower whisker of 26 is encountered

for low (m, k)-criteria compared to a median of 22, an upper whisker of 24, and a lower whisker of 21 for high (m, k)-criteria under confidence levels in $[0.7, 0.9]$ (cf. Fig. 5.4b).

5.6 Summary

Against the background of static resource reservation in the considered system and the end-to-end latency requirement of each application, resources are reserved with respect to a time estimate describing each task's worst-case execution time if QoS contracts as proposed in Chapter 4 are concluded. However, job execution times typically exhibit a certain degree of variability, i.e., the considered time estimate is frequently undershot, such that the reserved resource capacity is not fully used. In a distributed system serving as the backbone of a smart city, on which a potentially large number of individual systems, e.g., smart vehicles, require the execution of applications at the same time, such a waste of resources is a limiting factor. To reduce resource overreservation and, thus, to enable more applications to make use of the system's infrastructure, this chapter considered applications that do not require the satisfaction of their end-to-end deadline at any point in time, but only according to an additional robustness requirement. Exploiting the pWCET distribution of each task, the concept of soft QoS contracts has been introduced that allows to take all QoS parameters, i.e., end-to-end deadline as well as robustness requirement, into account when making resource reservations. The proposed approach has been proven to be effective in simulations with synthetic data, where up to 50 % more applications could be admitted to the system under soft QoS contracts than under the type of QoS contracts presented in Chapter 4. In the course of the evaluation, it was noticed that applications' robustness criteria, more precisely, the strictness of (m, k)-criteria as well as the intervals out of which confidence levels are chosen, have an impact on the number of accepted applications. Actually, the higher the confidence levels and the stricter the (m, k)-criteria, the less applications can be admitted to the system.

The evaluation's findings can be fruitfully used when designing distributed systems as described in Chapter 3. In fact, the considered system architecture allows for the coexistence of QoS contracts and soft QoS contracts[5]. Although QoS contracts lead to a higher resource overreservation, it is not possible for each application to be admitted under a soft QoS contract, since certain safety-critical applications may indeed require that their end-to-end deadline is always met. Consequently, it seems to be convenient to conclude different types of QoS contracts on the same system, establishing quota not only for each type of contract, but also for different categories of applications. To define such categories, the applications' confidence levels and (m, k)-criteria can be meaningful indicators; however, for each system to be designed, individual objectives and requirements must be taken into consideration.

[5]Recall that uncompleted jobs are aborted at their local deadlines, so that one application violating its (soft) QoS contract cannot have any impact on another application's (soft) QoS contract if the scheduler operates correctly.

Notation	Meaning
$\mathfrak{C}_{i,j}$	Set of pWCET distributions of $\tau_{i,j}$
$C_{i,j}$	pWCET distribution of $\tau_{i,j}$
$c_{i,j}^{p_{i,j}}$	Execution time value
$c_{i,j}^{p}$	$c_{i,j}^{p_{i,j}}$
ζ_i	Confidence level of a_i
(m_i, k_i)	(m, k)-criterion of application a_i
$p_{i,j}$	Occurrence probability of an execution time value
p	$p_{i,j}$
ρ_i	(m_i, k_i)
$\mathbb{P}\left(\leq c_{i,j}^{p}\right)$	Probability that the execution time of a job of $\tau_{i,j}$ is at most $c_{i,j}^{p}$
$\mathbb{P}_i$	Probability of successful completion of all jobs of all tasks of a_i

Table 5.1: Overview of the notation introduced in Chapter 5.

Part III

Embedded Systems

CHAPTER 6

System and Application Model

Parts of this chapter have previously been published in [SBC+20].

Contents

6.1 Endpoint and Local System

The infrastructure of a smart city, represented by a centralized, hierarchical distributed system, as introduced in Chapter 3, can be used in an on-demand fashion by individual endpoint systems. For instance, a vehicle with advanced driver assistance system can temporarily connect to the distributed system in order to enable autonomous navigation in specific areas of the city. Similar to the distributed system, an endpoint system comprises multiple communication and computation resources, although to a lesser extent. In fact, its computation resources are assumed to have limited capacity, so that the activation of additional autonomy functionalities is not possible either without making use of the distributed system's resources or without degrading or deactivating other functionalities of the endpoint system. A computation resource of an endpoint system, e.g., an electronic control unit (ECU), denoted as *local system*, is subsequently considered as a uniprocessor.

6.2 Application Model

On an endpoint system, applications of different criticality (cf. Chapter 2.4) are executed. More precisely, *critical* applications with hard real-time requirements and *non-critical* applications with soft real-time requirements can be distinguished. Independent of their criticality, applications executed on an endpoint system are comparable to those executed on the distributed system of a smart city (cf. Chapter 3), i.e., they consist of a partially ordered set of dependent, sporadic tasks sharing the same minimum inter-arrival time that can be modeled as a directed acyclic task graph. Due to the above mentioned resource limitations, it is assumed that the execution of each application begins and ends on one particular local system and that the intermediate part of the task graph is offloaded to the distributed system. To model such applications, however, the view of a local system is adopted. For this purpose, the part of an application's task graph that is executed on the distributed system can be summarized and hidden as an *offloading operation*; the resulting simplified task graph is assumed to not contain any independent tasks, i.e., its tasks exhibit a total instead of a partial order. This perspective of the local system corresponds to the concept of *self-suspension* [CBH+17]. More precisely, instead of modeling the application as a task graph, it can be considered as a single task executed on one local system that temporarily interrupts its execution, while it waits for the completion of an offloading operation, and proceeds as soon as the latter is finished. The duration of an offloading operation, i.e., the time elapsing between the moment a message is sent to the remote system and the latest safe moment in which a response may be received, can be upper-bounded, assuming that a QoS contract as proposed in Chapter 4 exists for the considered (part of the) application.

Since modeling the time required for an offloading operation as additional execution time rather than as so-called *suspension time*, i.e., waiting time, would lead to a pessimistic resource under-utilization [SHV+18; BHC+16], one of the state-of-the-art models can be applied[1] such as the dynamic self-suspension model (cf., e.g., [HCL15], [LC14]), the segmented self-suspension model (cf., e.g., [SHV+18]), or a hybrid model (cf. e.g. [BHC17]). Subsequently, the segmented self-suspension model is considered, which allows to model a specific, known suspension pattern, i.e., to specify the exact point in time in which an offloading operation begins as well as an upper bound on its duration.

6.3 Task Model

The set of applications executed on a local system is represented by a set of tasks[2] $\mathcal{T}$ that is divided into a set of critical tasks $\mathcal{T}_{crit}$ and a set of non-critical tasks $\mathcal{T}_{non}$, such that $\mathcal{T} = \mathcal{T}_{crit} \cup \mathcal{T}_{non}$ and $\mathcal{T}_{crit} \cap \mathcal{T}_{non} = \emptyset$. Each recurrent task $\tau_i \in \mathcal{T}$, as illustrated

[1] For a detailed overview of self-suspension and how it can be modeled refer to [CNH+18; CBH+17].
[2] Recall that each application is modeled as one self-suspending task on the local system.

in Fig. 6.1, is assumed to have a sporadic release pattern and is characterized by a tuple $\left(c_{i,1}, c_{i,s}, c_{i,2}, S_i, c_i^{pre}, c_i^{post}, D_i, P_i\right)$.

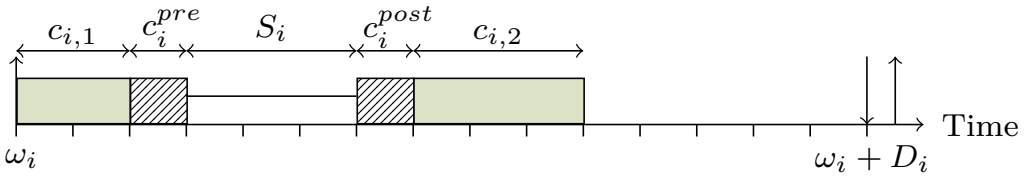


Figure 6.1: An offloading operation of a job of task τ_i is performed successfully.

Each taskτ_i releases an infinite number of task instances denoted as jobs. P_i indicates the minimum inter-arrival time of τ_i, i.e., the release times of any two consecutive jobs of τ_i must be separated by at least P_i time units. D_i describes the QoS requirement of task τ_i in terms of temporal correctness (cf. Chapter 2.4), i.e., its relative deadline. The absolute deadline of a job of task τ_i released at time ω_i is given by $\omega_i + D_i$. Deadlines are assumed to be constrained for all tasks, i.e., $D_i \leq P_i$ for each task τ_i. The completion time of a job is indicated by f_i. Note that, unlike in a system as described in Chapter 3, no uncompleted job of a task τ_i is aborted at its deadline unless specified differently.

$c_{i,1}$ and $c_{i,2}$ denote the worst-case execution time (WCET) of the first and second *computation segments* of τ_i, i.e., parts of the task. $c_{i,S}$ is the worst-case execution time of the typically offloaded share of the task that is required for its execution on the local system. c_i^{pre} and c_i^{post} are the worst-case execution times of the computation segments that perform pre- and post-processing routines, which are executed before and after the offloading operation of a job of task τ_i, respectively. S_i is the offloading or *suspension* time of τ_i. Note that worst-case execution time upper bounds (cf. Chapter 2.3) are considered for each computation segment of τ_i.

In the following, it is assumed that $P_i \geq D_i > 0$ and $c_{i,1}, c_{i,S}, c_{i,2}, S_i, c_i^{pre}, c_i^{post} \geq 0$. Moreover, the natural assumption is made that $c_i^{pre} + c_i^{post} \leq c_{i,S}$, since offloading is not meaningful otherwise. Furthermore, the worst-case execution time of a job of task τ_i under any possible execution scenario is greater than 0, i.e., $c_{i,1} + c_{i,S} + c_{i,2} > 0$ and $c_{i,1} + c_i^{pre} + c_i^{post} + c_{i,2} > 0$. For notational brevity, let $c_i^\sharp = c_{i,1} + c_{i,S} + c_{i,2}$ and $c_i^\flat = c_{i,1} + c_i^{pre} + c_i^{post} + c_{i,2}$.

The set of tasks $\mathcal{T}$ is scheduled according to a preemptive task-level fixed-priority policy, under which each task is assigned a unique priority. If at any point in time multiple jobs are ready, i.e., eligible for being executed on the local system, the job with the highest priority is executed. For each task τ_i, the unique set of the higher-priority tasks is denoted as $hp(\tau_i)$. Note that a summary of the notation used in this chapter is given in Table 6.1.

6.4 Execution Behavior and Execution Scenarios

According to the classification of tasks into two subsets, two different execution behaviors of the system are specified, i.e., *normal* and *local* execution behavior. This corresponds to the characteristics of mixed-criticality systems (cf. Chapter 2.4), although the considered system is assumed *not* to be a classical mixed-criticality system[3]. When the system exhibits normal execution behavior, offloading operations are performed and all QoS requirements, i.e., all deadlines, of all tasks are satisfied at any point in time, whereas, if the system exhibits local execution behavior, tasks are executed fully on the local system and QoS guarantees can only be given for all critical tasks $\tau_i \in \mathcal{T}_{crit}$. Depending on the execution behavior of the system, for a job of task τ_i released at time ω_i, different execution scenarios are possible:

- The job is *executed locally* (cf. Fig. 6.2). In this case, the worst-case execution time of the job released at time ω_i is $c_{i,1} + c_{i,S} + c_{i,2}$, i.e., $c_i^\sharp$.
- The job is *offloaded* (cf. Fig. 6.1). In this case, the job is first executed locally for up to $c_{i,1}$ execution time units and, thereon, enters the pre-processing routine for offloading for up to c_i^{pre} execution time units. Suppose that the first computation segment as well as the pre-processing routine are finished at time ρ. Then, the considered job is offloaded to the distributed system at time ρ.

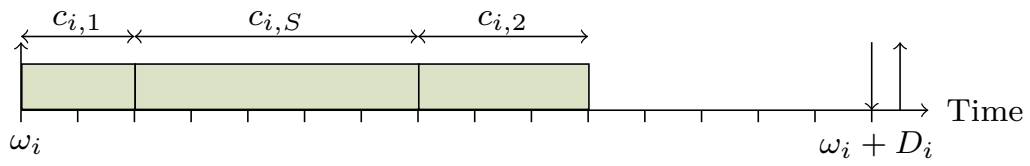


Figure 6.2: A job of task τ_i is executed locally.

When a job of task τ_i is offloaded, the offloading operation can be either *successful* or *unsuccessful*:

- *Offloading is successful* if the computation result or *offloading response* is returned to the local system until time $\rho + S_i$. In this case, the offloading response is post-processed for up to c_i^{post} time units and the second computation segment is executed for up to $c_{i,2}$ time units (cf. Fig. 6.1). Accordingly, the execution time of the job of τ_i on the local system is at most $c_i^\flat$.
- *Offloading is unsuccessful* otherwise. In this case, at time $\rho + S_i$, a local re-execution of the offloaded task share is performed for up to $c_{i,S}$ time units followed by the execution of the second computation segment for up to $c_{i,2}$ time units. Hence, the execution time of the job of τ_i on the local system is at most $c_i^\sharp + c_i^{pre}$.

[3]The considered system does not provide explicit mode changes, but rather transitions between different execution behaviors, as discussed later.

An overview of the notation introduced in this section is given in Table 6.1.

Notation	Meaning
$c_{i,1}$	WCET of a task's first computation segment
$c_{i,2}$	WCET of a task's second computation segment
$c_{i,S}$	Local WCET of a typically offloaded task share
c_i^{pre}	WCET of pre-processing before offloading
c_i^{post}	WCET of post-processing after offloading
$c_i^\sharp$	$c_{i,1} + c_{i,S} + c_{i,2}$
$c_i^\flat$	$c_{i,1} + c_i^{pre} + c_i^{post} + c_{i,2}$
D_i	Relative deadline of a task
$hp(\tau_i)$	Set of tasks with higher priority than a task τ_i
P_i	Minimum inter-arrival time of a task
ρ	Moment in which a considered job of a task is offloaded
S_i	Suspension time of a task
$\mathcal{T}$	Set of tasks on the local system
$\mathcal{T}_{crit}$	Set of critical tasks, $\mathcal{T}_{crit} \subseteq \mathcal{T}$
$\mathcal{T}_{non}$	Set of non-critical tasks, $\mathcal{T}_{non} \subseteq \mathcal{T}$
τ_i	Task
ω_i	Release time of a task

Table 6.1: Overview of the notation introduced in Chapter 6.

CHAPTER 7

Safe Offloading under Unreliable Connections

Parts of this chapter have previously been published in [SBC+20].

Contents

7.1 Introduction

Processing large amounts of sensor data within short, pre-defined intervals of time is crucial for many endpoint systems, e.g., smart vehicles with advanced driver assistance systems, in order to accomplish their mission or even to maintain their operability. With respect to the execution of applications on such systems, three options exist: It is possible to perform all computations locally on the endpoint system

without any external acceleration, to perform all computations remotely, e.g., on the infrastructure of a smart city (cf. Chapter 3), or to offload only certain parts of an application that are expensive in terms of time or energy, while the remainder is executed locally. In order to activate enhanced functionalities, e.g., autonomous driving, it is assumed that endpoint systems implement the latter option, as described in Chapter 6. Accordingly, after a part of an application has been offloaded and the remote execution is completed, the computation result is transmitted back to the endpoint system, where it is further processed.

To ensure the satisfaction of an application's temporal correctness requirement, reliable data transmission between the endpoint system and the distributed system is required. However, especially wireless connections, e.g., over 4G/5G or IEEE802.11p [EE17], exhibit a certain level of unreliability, which must be taken into account. For instance, the end-to-end latency achieved within cellular vehicular communication systems is severely influenced by highly dynamic channel conditions related to shadowing effects, multipath fading, handover situations, and technology switches [SFL+19]. Wireless data transmissions can also be strongly impacted by electromagnetic interference resulting from natural or human-made sources or by incomplete network coverage, e.g., in rather rural areas of a smart city, as considered as the use case of this chapter (cf. Fig. 7.1).

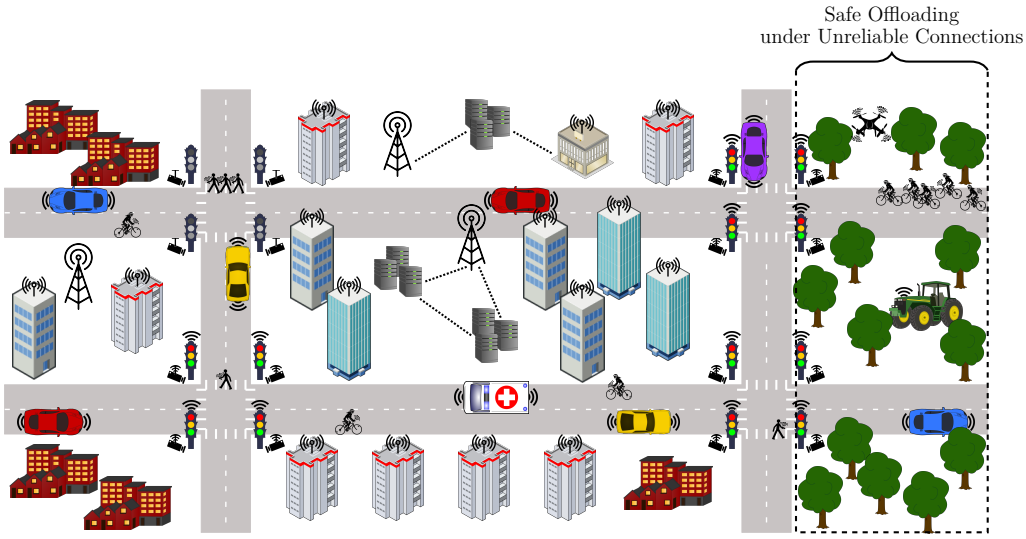


Figure 7.1: Wireless connections, especially in rural areas of a smart city, are not always reliable. This chapter introduces an approach for offloading critical applications from an endpoint system to a distributed system while providing QoS guarantees.

Even if connectivity issues are considered to be rather rare cases in a smart city, it is necessary to ensure that the QoS requirements, especially of critical applications, are always satisfied. Consequently, this chapter addresses the challenge of providing

QoS guarantees for applications that are offloaded from an endpoint system to a distributed system, as considered in Chapter 3, over unreliable connections.

7.2 Problem Statement

For a local system, i.e., a computation resource, of an endpoint system, as described in Chapter 6, which is connected to a distributed system, as described in Chapter 3, an approach for offloading critical and non-critical tasks from the local system to the distributed system is required that provides QoS guarantees for all critical tasks. To this end, it is necessary to anticipate all events that may occur during an offloading operation, e.g., a missing response due to an unreliable wireless connection, and to specify a deterministic mechanism for handling these.

Objective of this Chapter

For the considered system and the considered type of applications executed on the system, this chapter aims to propose two alternative recovery protocols allowing the system to satisfy the QoS requirements of all critical tasks under local execution behavior if a task is offloaded unsuccessfully, to return to normal execution behavior and to re-establish QoS guarantees for both critical and non-critical tasks as soon as possible. For each of these recovery protocols, a schedulability analysis and a schedulability test shall be provided that allow for an a-priori verification of a system implementing the respective protocol.

Subsequently, an overview about existing works addressing the offloading of tasks over unreliable connections as well as of critical tasks is given. In Sec. 7.4, the recovery protocols are introduced. The workload of a task exhibited in different execution scenarios is examined in Sec. 7.5. On this basis, a schedulability analysis for both recovery protocols is provided in Sec. 7.6. An evaluation of the protocols is presented in Sec. 7.7. Table 7.1 provides an overview of the notation introduced in this chapter.

7.3 Related Work

With respect to the offloading of tasks over unreliable connections, existing research focuses on making offloading decisions, i.e., on deciding if and when a task is offloaded. For instance, considering unmanned aerial vehicles (UAVs), [HAA+19] takes the UAVs' position into account, which impacts the quality of wireless connections, when formulating an optimization problem to make offloading decisions. Focusing on the minimization of a system's energy consumption, [LCP+20] also integrates latency and reliability requirements of applications into the offloading decision problem. [MA21] propose an adaptive offloading decision scheme for drones that starts and stops offloading depending on the system's workload and on the quality of the wireless

connection. An adaptive offloading controller taking into consideration the network conditions as well as the requirements of an application is presented in [ARS18]. In the context of autonomous driving, [ZZL+19] propose an approach based on deep learning to compute optimal offloading schemes including the choice of a wireless transmission mode. To handle transmission failures, offloading operations are repeated under a combination of two such transmission modes in order to increase the connection's reliability.

Critical tasks are addressed by [OHC+23], however, these are not considered for offloading. Instead, the proposed approach aims to maximize the number of offloaded resource-intensive soft real-time tasks in order to maximize the local system's capacity available for the execution of time- and safety-critical applications. [XYY+19] considers the offloading of mission critical tasks and provides an algorithm for making offloading decisions. However, reliability aspects and connectivity issues are neglected.

7.4 Recovery Protocols

As known from Chapter 6, the point in time when a task is offloaded to the distributed system under normal execution behavior is already known, for which reason no offloading decision needs to be made for any task, in contrast to the tasks considered in all works outlined in Sec. 7.3. Instead, the case must be handled that a task's offloading operation is unsuccessful.

Recall that the set of tasks executed on the local system is divided into a set of critical tasks with hard real-time requirements and a set of non-critical tasks with soft real-time requirements. Although a task with soft real-time requirements does not require its deadline to be always satisfied, it is in general desirable[1] to provide a QoS to a non-critical task that is as high as possible, i.e., to satisfy its deadline as frequently as possible and, if a deadline cannot be met, to minimize the tardiness[2]. Against this background, two recovery protocols with distinct objectives are proposed in this chapter:

- The *service protocol* aims to provide a QoS as high as possible to non-critical tasks, even under local execution behavior.
- The *return protocol* aims to minimize the amount of time, in which the system exhibits local execution behavior after an unsuccessful offloading operation.

Independent of the implemented protocol, assume that the local system exhibits normal execution behavior at time 0, such that offloading is enabled for all tasks in $\mathcal{T}$. The schedule considers the execution of all tasks until the first moment $\gamma_{1,\searrow}$, in which the offloading operation of a certain task τ_i is unsuccessful, i.e., a job of task τ_i that has offloaded its computation at time $\gamma_{1,\searrow} - S_i$ does not receive a response

[1] Note that *non-critical* is not equivalent to *unimportant*. For further discussion on the relation between criticality and importance refer to [ENN+15].

[2] The tardiness of a job of a task is 0 if its deadline is met, and the difference between its completion time and its absolute deadline otherwise.

from the distributed system until time $\gamma_{1,\searrow}$ (cf. Fig. 7.2). Immediately after $\gamma_{1,\searrow}$, the local system exhibits local execution behavior. Until time $\gamma_{1,\searrow}$, three scenarios are possible for each uncompleted job a critical task $\tau_i \in \mathcal{T}_{crit}$:

- *The job of τ_i has not been offloaded*: In this case, no offloading operation will be performed for this job. Instead, it is executed on the local system. Since it is possible that the pre-processing routine for offloading is already active at time $\gamma_{1,\searrow}$, the worst-case execution time of this job is upper-bounded by $c_{i,1} + c_i^{pre} + c_{i,S} + c_{i,2}$, i.e., $c_i^\sharp + c_i^{pre}$.
- *The job of τ_i is already offloaded, but no response from the distributed system was received until time $\gamma_{1,\searrow}$*: In this case, the offloading process is aborted and the job is executed on the local system as of time $\gamma_{1,\searrow}$. Therefore, the worst-case execution time of this job is upper-bounded by $c_{i,1} + c_i^{pre} + c_{i,S} + c_{i,2}$, i.e., $c_i^\sharp + c_i^{pre}$.
- *The job of τ_i is already offloaded and the response from the distributed system has been received prior to time $\gamma_{1,\searrow}$*: In this case, the job continues its final processing. Therefore, the worst-case execution time of this job is upper-bounded by $c_{i,1} + c_i^{pre} + c_i^{post} + c_{i,2}$, i.e., $c_i^\flat$.

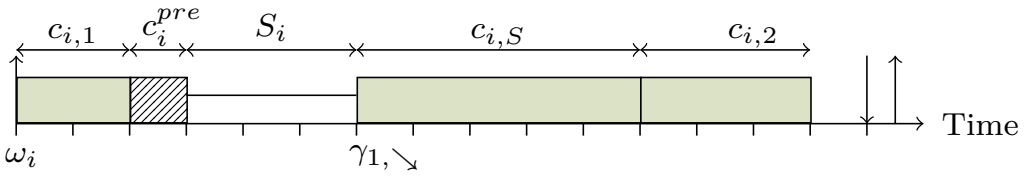


Figure 7.2: An unsuccessful offloading operation of τ_i results in the transition to the local system behavior at time $\gamma_{1,\searrow}$.

After $\gamma_{1,\searrow}$, QoS guarantees are only given for $\mathcal{T}_{crit}$. Moreover, offloading is inhibited for all critical tasks in the near future of $\gamma_{1,\searrow}$, owing to the currently unreliable connection to the distributed system that was indicated by the missing response. The offloading decision for non-critical tasks, however, depends on the applied recovery protocol:

- **Service Protocol**: Under the *service protocol*, offloading is inhibited for all jobs of all tasks that are active as long as the system exhibits local execution behavior. The task share of each $\tau_i \in \mathcal{T}$ that is offloaded under normal execution behavior is executed on the local system within $c_{i,S}$ units of execution time. Since this leads to a higher workload on the local system, QoS guarantees cannot be given for any non-critical task. Nevertheless, no non-critical task is aborted.
- **Return Protocol**: The *return protocol* does not inhibit offloading for *all* tasks, but only for critical ones under local execution behavior. Non-critical tasks, in contrast, are offloaded regardless, but neither a re-execution nor a

re-transmission is performed if a response from the distributed system is not received in time. More precisely, the second computation segment of τ_i is only executed if a response is received, and aborted otherwise. Moreover, a job of $\tau_i \in \mathcal{T}_{non}$ is aborted whenever it misses its deadline.

As of time $\gamma_{1,\searrow}$, the local system exhibits local execution behavior until the point in time $\gamma_{1,\nearrow}$, in which QoS guarantees can be given again for all tasks in $\mathcal{T}$. In the proposed protocols, two options exist for the transit from local to normal execution behavior, which should be chosen depending on the actual system requirements:

- **Abort-Transit**: This option aims to re-establish the normal execution behavior as quickly as possible. Suppose that $\gamma_{1,\nearrow}$ is the earliest moment (after $\gamma_{1,\searrow}$) in which no uncompleted job from $\mathcal{T}_{crit}$ exists. All released but not yet finished jobs of non-critical tasks are discarded.
- **Idle-Transit**: This option re-establishes the normal execution behavior in the earliest moment $\gamma_{1,\nearrow}$ (after $\gamma_{1,\searrow}$) in which no uncompleted job from $\mathcal{T}$ exists.

Note that the above transitions are well-defined and the local system exhibits normal and local execution behavior in an interleaving manner.

7.5 Workload Characteristics

When the system exhibits normal execution behavior, task execution patters are the same under both proposed protocols, i.e., an offloading operation is performed for each task. Hence, each $\tau_i \in \mathcal{T}$ is a (segmented) self-suspending task consisting of two computation segments as well as of one *suspension interval* of length S_i, and can therefore be analyzed applying any suitable technique.

Definition 10. Suppose that the system always exhibits normal execution behavior. Then, for each task $\tau_i \in \mathcal{T}$, the worst-case response time R_i^{normal} is the worst-case response time of task τ_i and R_i^1 is the worst-case response time of the first computation segment of task τ_i. By definition, $R_i^1 \leq R_i^{normal}$. It is assumed that $R_i^{normal} \leq D_i \leq P_i$, $\forall \tau_i \in \mathcal{T}$.

Lemma 1. If $\tau_i \in \mathcal{T}_{non}$, the worst-case response time of τ_i under normal execution behavior is upper-bounded by R_i^{normal} regardless of the adopted protocol.

Proof. This is based on the definition. □

Assume that an existing and correct[3] analysis, e.g., the analysis proposed in [SHV+18], has been used for verifying that each task in $\mathcal{T}$ meets its deadline if the system always exhibits normal execution behavior. The following lemma characterizes the maximum workload of a task τ_i under analysis, i.e., the maximum amount of

[3]Note that several misconceptions exist in the literature regarding the analysis of self-suspending tasks. Further information can be found in [CNH+18].

execution time, within in a time interval $[t, t+\Delta)$ under the assumption that the local system resumes from idling at time t, i.e., it idles at $t-\varepsilon$ for an infinitesimal ε, under normal execution behavior and that it does not switch from the local execution behavior to the normal execution behavior before $t+\Delta$ for $\Delta > 0$.

Lemma 2. Suppose that the local system resumes from idling at time t, i.e., it idles at $t-\varepsilon$ for an infinitesimal ε and executes a certain job at time t, under normal execution behavior and it does not switch from the local execution behavior to the normal execution behavior before $t+\Delta$ for $\Delta > 0$. For a task τ_i, in which

- $\tau_i \in \mathcal{T}$ under the service protocol or
- $\tau_i \in \mathcal{T}_{crit}$ under the return protocol,

the amount of execution time for which task τ_i is executed in time interval $[t, t+\Delta)$ on the local system is upper-bounded by $\max\{f_1(\tau_i, \Delta), f_2(\tau_i, \Delta)\}$, where

$$f_1(\tau_i, \Delta) = c_i^{pre} + \left\lceil \frac{\Delta}{P_i} \right\rceil c_i^\sharp \tag{7.1}$$

and

$$f_2(\tau_i, \Delta) = c_{i,S} + c_{i,2} + \left\lceil \frac{\Delta - (P_i - (R_i^1 + S_i))}{P_i} \right\rceil c_i^\sharp \tag{7.2}$$

Recall that $c_i^\sharp$ is defined as $c_{i,1} + c_{i,S} + c_{i,2}$.

Proof. First, consider the simpler case, in which the local system stays in the normal execution behavior from t to $t+\Delta$. In this case, there are two scenarios:

- Case 1a: if no uncompleted job of τ_i exists before t (cf. Fig. 7.3), then, the workload of task τ_i executed in time interval $[t, t+\Delta)$ is at most $\left\lceil \frac{\Delta}{P_i} \right\rceil c_i^\flat \leq f_1(\tau_i, \Delta)$.
- Case 1b: If an uncompleted job of τ_i exists before t, then, by the definition that the local system returns from idling at time t, task τ_i has been suspended (cf. Fig. 7.4). Since the system still exhibits normal execution behavior prior to time t, it is known that there is at most one such suspended job of τ_i and its release time ω_i cannot be earlier than $t - (R_i^1 + S_i)$. Therefore, the first job of task τ_i released after t is released no earlier than $t - (R_i^1 + S_i) + P_i$. Since the system exhibits normal execution behavior from t to $t+\Delta$, the workload of task τ_i within the time interval $[t, t+\Delta)$ is at most $c_i^{post} + c_{i,2} + \left\lceil \frac{\Delta - (P_i - (R_i^1 + S_i))}{P_i} \right\rceil c_i^\flat \leq f_2(\tau_i, \Delta)$.

For the case, in which the local system switches to local execution behavior at time γ, where $t \leq \gamma < t+\Delta$, there are also two scenarios:

- Case 2a: There is no job of τ_i arriving before t that has not yet been completed by time t (cf. Fig. 7.5). From t to γ, at most $\left\lceil \frac{\gamma - t}{P_i} \right\rceil$ jobs of task τ_i are released. Specifically, among them, the last job of τ_i offloaded prior to γ may be offloaded unsuccessfully. The worst-case execution time of this particular

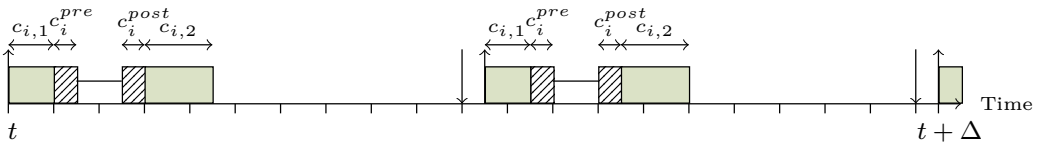


Figure 7.3: Execution of a task τ_i under analysis in $[t, t+\Delta)$ in case 1a of Lemma 2.

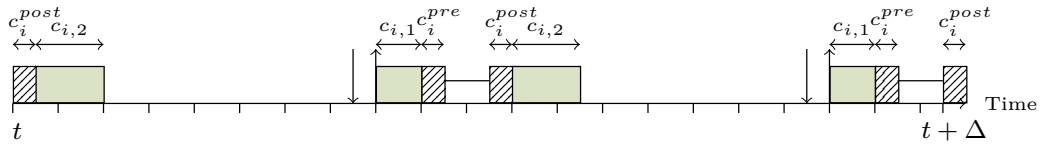


Figure 7.4: Execution of a task τ_i under analysis in $[t, t+\Delta)$ in case 1b of Lemma 2.

job is $c_{i,1} + c_i^{pre} + c_{i,S} + c_{i,2} = c_i^\sharp + c_i^{pre}$, while the worst-case execution time of the other $\left\lceil \frac{\gamma - t}{P_i} \right\rceil - 1$ jobs is at most $c_i^\flat \leq c_i^\sharp$. Moreover, the worst-case execution time of any job of τ_i released after γ, under the service protocol or the return protocol when $\tau_i \in \mathcal{T}_{crit}$, is at most $c_i^\sharp$. Therefore, the workload of τ_i from t to $t+\Delta$ is

$$\left(\left(\left\lceil \frac{\Delta}{P_i} \right\rceil - 1\right) c_i^\sharp\right) + (c_i^\sharp + c_i^{pre}) \leq \; c_i^{pre} + \left\lceil \frac{\Delta}{P_i} \right\rceil c_i^\sharp \;\; \stackrel{\text{def}}{=} \;\; f_1(\tau_i, \Delta)$$

- Case 2b: A job of τ_i suspended prior to t and its second computation segment is released at or after t (cf. Fig. 7.6). Identically to the discussion in case 1b, the next job of task τ_i is released no earlier than $t - (R_i^1 + S_i) + P_i$. If the job suspended prior to t is offloaded unsuccessfully, i.e., its execution time after t is at most $c_{i,S} + c_{i,2}$, the other subsequent jobs of τ_i will be executed on the local system, until the moment in which the local system switches to the normal execution behavior again, under the service protocol or under the return protocol when $\tau_i \in \mathcal{T}_{crit}$. Therefore, the workload of τ_i from t to $t+\Delta$ is as defined in $f_2(\tau_i, \Delta)$. If the job suspended prior to t is offloaded successfully, at most of one of the jobs released after $t - (R_i^1 + S_i) + P_i$ is offloaded unsuccessfully. In this case, the workload of τ_i from t to $t+\Delta$ is at most

$$c_i^{post} + c_{i,2} + c_i^{pre} + \left\lceil \frac{\Delta - (P_i - (R_i^1 + S_i))}{P_i} \right\rceil c_i^\sharp \leq f_2(\tau_i, \Delta),$$

since $c_i^{pre} + c_i^{post} \leq c_{i,S}$.

□

So far, the maximum workload that can be contributed to a time interval $[t, t+\Delta)$ by a task $\tau_i \in \mathcal{T}$ under the service protocol and by a task $\tau_i \in \mathcal{T}_{crit}$ under the return

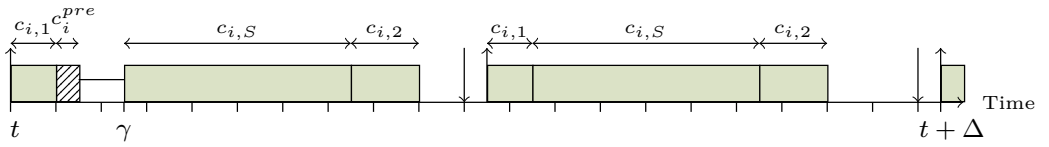


Figure 7.5: Execution of a task τ_i under analysis in $[t, t + \Delta)$ in case 2a of Lemma 2.

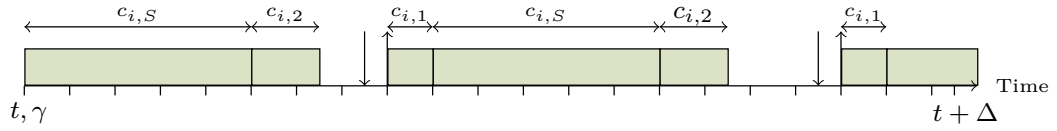


Figure 7.6: Execution of a task τ_i under analysis in $[t, t + \Delta)$ in case 2b of Lemma 2.

protocol has been analyzed. For the missing case that $\tau_i \in \mathcal{T}_{non}$ under the return protocol, the workload in the time interval $[t, t + \Delta)$ can be reduced based on the definition of the protocol (cf. Sec. 7.4), as given in the following lemma:

Lemma 3. For a task τ_i in $\mathcal{T}_{non}$ under the return protocol, under the same condition for t and $t + \Delta$ as specified in Lemma 2, the amount of execution time that task τ_i is executed in the time interval $[t, t + \Delta)$ is upper-bounded by $\left(\left\lceil \frac{\Delta}{T_i} \right\rceil + 1\right) C_i^\flat$.

Proof. Under the return protocol, a job of task $\tau_i \in \mathcal{T}_{non}$ is aborted whenever it misses its deadline. Therefore, the number of jobs of τ_i that have not yet missed their deadlines in a time interval $[t, t + \Delta)$ is at most $\left(\left\lceil \frac{\Delta}{P_i} \right\rceil + 1\right)$ since $D_i \leq P_i$. Under the return protocol, a task $\tau_i \in \mathcal{T}_{non}$ is always offloaded if the first computation segment is completed before the job's deadline. Any execution of such a job on the local system requires up to $c_i^\flat$ execution time units by definition. Therefore, $\left(\left\lceil \frac{\Delta}{P_i} \right\rceil + 1\right) c_i^\flat$ is the upper bound of the workload. □

7.6 System Behavior and Response Time Analysis

To analyze the worst-case timing behavior of the local system, the timing behavior of each task τ_i must be analyzed, beginning with the highest-priority task, considering the following two scenarios:

- If $\tau_i \in \mathcal{T}_{crit}$, the worst-case response time under local and normal execution behavior must be analyzed.
- If $\tau_i \in \mathcal{T}_{non}$, only the worst-case response time under normal execution behavior must be analyzed.

Note that the worst-case response time of $\tau_i \in \mathcal{T}_{non}$ under local execution behavior is not of interest, since its jobs may be aborted (cf. Sec. 7.4).

In the following analysis, a concrete fixed-priority preemptive schedule σ for the task set $\mathcal{T}$ is assumed to exist from time 0 onward. For the concrete schedule σ, let $\gamma_{h,\searrow}$ be the h-th moment in which σ switches from normal to local execution behavior. Moreover, let $\gamma_{h,\nearrow}$ be the h-th moment in which σ switches from local behavior to normal behavior. By the definition of the recovery protocols (cf. Sec. 7.4), $\gamma_{h,\searrow} < \gamma_{h,\nearrow} < \gamma_{h+1,\searrow}$.

Consider the ℓ-th job of task τ_i, denoted as $\tau_{i,\ell}$, in schedule σ and assume that at the release time of job $\tau_{i,\ell}$ no uncompleted job of task τ_i exists in the schedule σ. Now, remove all lower-priority jobs from the schedule σ. Since σ is a fixed-priority preemptive schedule and all tasks in $\mathcal{T}$ are independent of each other, the removal of these jobs does not have any impact on the execution of any remaining job in the schedule σ. Thereon, remove all jobs of task τ_i released before the release of $\tau_{i,\ell}$ at time $\omega_{i,\ell}$ from the schedule σ. Due to the assumption that the jobs of τ_i released before $\omega_{i,\ell}$ have been completed before $\omega_{i,\ell}$, the removal of these jobs of τ_i does not have any impact on the execution of any remaining job in the schedule σ.

To improve readability, the index ℓ will be removed when the context is clear. Accordingly, ω_i denotes the release time of the job of τ_i under analysis and f_i its completion time. By definition, the next job of τ_i cannot be released before $\omega_i + P_i$. The remainder of this section will focus on the analysis of the case that $\tau_i \in \mathcal{T}_{crit}$.

Lemma 4. For any $\tau_i \in \mathcal{T}_{crit}$ under both service and return protocol as well as under both abort- and idle-transit, the local system switches at most once from normal to local behavior in the interval $[\omega_i, f_i)$. That is, at most one $\gamma_{h,\searrow}$ exists in $[\omega_i, f_i)$.

Proof. This property results from the definition of the protocols and abort- and idle-transits. That is, the local system only switches from the local to normal execution behavior when no uncompleted job of any task in $\mathcal{T}_{crit}$ exists. □

Based on Lemma 4, only four cases must be considered:

- σ is executed under local execution behavior at time ω_i and under normal execution behavior at time f_i, denoted as *L2N*.
- σ is executed under normal execution behavior at time ω_i and under normal execution behavior at time f_i, denoted as *N2N*.
- σ is executed under normal execution behavior at time ω_i and under local execution behavior at time f_i, denoted as *N2L*.
- σ is executed under local execution behavior at time ω_i and under local execution behavior at time f_i, denoted as *L2L*.

In the following, each of these cases is considered individually, beginning with those that do not depend on the implemented recovery protocol, i.e., L2N and N2N. The remaining cases are inspected under each protocol separately, since the timing behavior of a task $\tau_i \in \mathcal{T}_{crit}$ under analysis differs depending on the specification of the execution behavior exhibited by the system.

Lemma 5. The case L2N is not possible under Abort-Transit and Idle-Transit.

Proof. In both transitions from local to normal execution behavior, no uncompleted job exists in the local system at time $\gamma_{h,\nearrow}$ for any $h \geq 0$. □

Lemma 6. The response time $f_i - \omega_i$ in the case N2N is at most R_i^{normal}, as defined in Definition 10.

Proof. This case is identical to a system with self-suspending tasks. Suppose that $\gamma_{h,\nearrow} \leq \omega_i < \gamma_{h,\searrow}$. All jobs in the schedule σ before $\gamma_{h,\nearrow}$ can be removed without changing any execution in σ after $\gamma_{h,\nearrow}$. Therefore, the jobs released in $[\gamma_{h,\nearrow}, f_i)$ are exactly the same as in the system analyzed in Definition 10. □

7.6.1 Analysis of the Service Protocol

For case N2L, the worst-case response time of task τ_i is given by the following lemma:

Lemma 7. Under the service protocol, the response time $f_i - \omega_i$ in the case N2L is upper-bounded by the minimum positive value of Δ, for which

$$\Delta = c_i^{pre} + c_i^{\sharp} + \sum_{\tau_j \in hp(\tau_i)} \max\{f_1(\tau_j, \Delta), f_2(\tau_j, \Delta)\} \tag{7.3}$$

if $\Delta \leq P_i$.

Proof. By definition of case N2L, the execution behavior of the local system changes at time $\gamma_{h,\searrow}$, in which $\omega_i \leq \gamma_{h,\searrow} < f_i$.

There are two cases to be considered:

- **Case 1**: In the interval $[\omega_i, \gamma_{h,\searrow})$, the schedule σ does not idle at all.
- **Case 2**: In the interval $[\omega_i, \gamma_{h,\searrow})$, the schedule σ idles at some time previous to $\gamma_{h,\searrow}$.

Note that the schedule σ is busy from $\gamma_{h,\searrow}$ to f_i, since σ is a work-conserving schedule[4] and there is no suspending behavior between $\gamma_{h,\searrow}$ and f_i under the service protocol.

Proof of Case 1: Let t be the earliest moment such that the schedule σ is busy from t to ω_i. Note that such a t exists. Under the above construction, the schedule σ is busy from t to f_i and idles right prior to t. If the release time of the job of τ_i is altered from ω_i to t, its response time becomes $f_i - t$, which is no less than $f_i - \omega_i$.

If the job of τ_i is not offloaded, its execution time is at most $c_{i,1}+c_{i,S}+c_{i,2}$. If the job of τ_i is offloaded successfully, its execution time is at most $c_{i,1}+c_i^{pre}+c_i^{post}+c_{i,2}$. If the job of τ_i is offloaded unsuccessfully, its execution time is at most $c_{i,1}+c_i^{pre}+c_{i,S}+c_{i,2}$. Due to the assumption that $c_i^{pre} + c_i^{post} \leq c_{i,S}$ (cf. Chapter 6), its execution time is upper-bounded by the maximum of the above three scenarios, which is at most $c_{i,1} + c_i^{pre} + c_{i,S} + c_{i,2} = c_i^{pre} + c_i^{\sharp}$.

[4] A schedule is referred to as *work-conserving* if it never idles while a job is available for being executed [LKA04].

Since the local system idles prior to t under normal execution behavior, the interference of the higher-priority tasks can be quantified using Lemma 2. Therefore, the worst-case response time of τ_i in this case is the minimum positive value of Δ such that

$$\Delta = c_i^{pre} + c_i^{\sharp} + \sum_{\tau_j \in hp(\tau_i)} \max\{f_1(\tau_j, \Delta), f_2(\tau_j, \Delta)\} \tag{7.4}$$

Proof of Case 2: Let t' be the latest moment such that the schedule σ is busy from t' to f_i. Note that such a t' exists since the schedule idles at some moment in $[\omega_i, \gamma_{h,\searrow})$. By definition, $t' - \omega_i \leq R_i^1 + S_i$.

Now, an upper bound of $f_i - t'$ will be analyzed. Since the schedule σ idles prior to time t', the job of τ_i must have been offloaded. If the offloading operation is successful, the execution time of task τ_i in the interval $[t', f_i)$ is at most $c_i^{post} + c_{i,2}$. If the job of τ_i is offloaded unsuccessfully, its execution time in the interval $[t', f_i)$ is at most $c_{i,S} + f_{i,2}$. Due to the assumption that $c_i^{pre} + c_i^{post} \leq c_{i,S}$ (cf. Chapter 6) its execution time in the interval $[t', f_i)$ is at most $c_{i,S} + c_{i,2}$.

The interference of the jobs of higher-priority tasks in the time interval $[t', f_i)$ can be obtained using Lemma 2. Since $t' - \omega_i \leq R_i^1 + S_i$ and the interference of a higher-priority task τ_j from t' to $t' + \Delta$ is at most $\max\{f_1(\tau_j, \Delta), f_2(\tau_j, \Delta)\}$, the worst-case response time of τ_i in case 2 is $R_i^1 + S_i + \Delta$, where Δ is the minimum positive value with

$$\Delta = c_{i,S} + c_{i,2} + \sum_{\tau_j \in hp(\tau_i)} \max\{f_1(\tau_j, \Delta), f_2(\tau_j, \Delta)\} \tag{7.5}$$

Because $c_{i,S} + c_{i,2} \leq c_i^{\sharp}$, the worst-case response time obtained by Eq. (7.4) dominates the one from Eq. (7.5). □

Having examined the case N2L under the service protocol, subsequently the case L2L is considered for a task $\tau_i \in \mathcal{T}_{crit}$ under analysis. The worst-case response time of task τ_i in this case can be determined by the following lemma:

Lemma 8. Under the service protocol, the response time $f_i - \omega_i$ in the case L2L is upper-bounded by the worst-case response time derived in Lemma 7 if $\Delta \leq P_i$.

Proof. Note that the schedule σ is busy from ω_i to f_i, since σ is a work-conserving schedule and there is no suspending behavior between ω_i and f_i under the service protocol. Based on the schedule σ, the following two moments are examined[5]:

- Let t be the earliest moment such that schedule σ is busy from t to ω_i.
- Let t' be the latest moment such that local system switches from normal to local execution behavior before or at ω_i.

[5]Note that the schedule σ is already modified by removing lower-priority jobs. Therefore, it is possible that the reduced schedule idles but the local system exhibits local execution behavior.

Note that both t and t' exist. There are two scenarios to be analyzed:

- $t \leq t'$: The local system exhibits normal execution behavior prior to time t'. The release time of a the considered job of τ_i can be changed to t' without decreasing its response time. Consequently, the analysis of case 1 in Lemma 7 can be applied directly, since the worst-case execution time of τ_i is at most $R_i^\sharp$ in this scenario.
- $t > t'$: The schedule σ idles at $t - \varepsilon$ for an infinitesimal ε and the local system exhibits local execution behavior from t' to t. Therefore, the idle time in schedule σ results from the removal of the lower-priority tasks from the original schedule. In this case, there exists no uncompleted job of any task in $hp(\tau_i)$ at time t. All jobs released by τ_i and all tasks in $hp(\tau_i)$ are executed on the local system. Therefore, the classical critical instant theorem by Liu and Layland [LL73] can be applied. The worst-case response time in this case is at most the minimum positive value of Δ, for which

$$\Delta = c_i^\sharp + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{\Delta}{P_j} \right\rceil c_j^\sharp \tag{7.6}$$

  if $\Delta \leq P_i$. This case is dominated by Eq. (7.4).

□

Resulting from the above analyses, the schedulability of a task $\tau_i \in \mathcal{T}_{crit}$ under the service protocol can be verified by the following theorem:

Theorem 4. Consider the service protocol. Suppose that Definition 10 holds, i.e., every task $\tau_i \in \mathcal{T}$ meets its deadline under normal execution behavior. Every task $\tau_i \in \mathcal{T}_{crit}$ meets its deadline under local execution behavior if there exists a Δ with $0 < \Delta \leq D_i$ such that the condition in Eq. (7.3) holds.

Proof. According to Lemma 8, the scenario L2L is dominated by N2L. By Lemma 5, L2N is not possible under both recovery protocols. By Lemma 6, the worst-case response time due to N2N is at most R_i^{normal}. By Definition 10, $R_i^{normal} \leq D_i$. Therefore, the only condition to check the schedulability is to verify the scenario N2L based on Eq. (7.3) in Lemma 7. □

7.6.2 Analysis of the Return Protocol

The return protocol is designed to reduce the workload on the local system so that a faster transit from local to normal execution behavior is possible. Under the return protocol, the execution time of a job of $\tau_i \in \mathcal{T}_{non}$ on the local system is always no more than $c_i^\flat$, independent of the system execution behavior. The analysis of the service protocol in Lemma 7 and Lemma 8 can be slightly changed to accommodate such a workload reduction under the return protocol, as stated in the following lemmata:

Lemma 9. Under the return protocol, the response time $f_i - \omega_i$ in case N2L is upper bounded by the minimum positive value of Δ, for which

$$\Delta = c_i^{pre} + c_i^{\sharp} + \sum_{\tau_j \in hp(\tau_i) \cup \mathcal{T}_{crit}} \max\{f_1(\tau_j, \Delta), f_2(\tau_j, \Delta)\} + \sum_{\tau_j \in hp(\tau_i) \cup \mathcal{T}_{non}} \left(\left\lceil \frac{\Delta}{P_j} \right\rceil + 1 \right) c_j^{\flat} \tag{7.7}$$

if $\Delta \leq P_i$.

Proof. The proof is almost identical to the proof of Lemma 7 by considering different interferences of the higher-priority task τ_j using Lemma 2 when $\tau_j \in \mathcal{T}_{crit}$ or Lemma 3 when $\tau_j \in \mathcal{T}_{non}$. Note that the main argument in the proof of Lemma 7, that the local system is busy from $\gamma_{h,\searrow}$ to f_i, remains valid, since task $\tau_i \in \mathcal{T}_{crit}$ cannot offload from $\gamma_{h,\searrow}$ to f_i. □

Lemma 10. Under the return protocol, the response time $f_i - \omega_i$ in the case L2L is upper bounded by the worst-case response time derived in Lemma 9 if $\Delta \leq P_i$.

Proof. The proof is identical as a patch of Lemma 8 by considering different interferences of the higher-priority task τ_j using Lemma 2 when $\tau_j \in \mathcal{T}_{crit}$ or Lemma 3 when $\tau_j \in \mathcal{T}_{non}$. □

Resulting from the above analyses, the schedulability of a task $\tau_i \in \mathcal{T}_{crit}$ under the return protocol can be verified by the following theorem:

Theorem 5. Consider the return protocol. Suppose that Definition 10 holds, i.e., every task $\tau_i \in \mathcal{T}$ meets its deadline under normal execution behavior. Every task $\tau_i \in \mathcal{T}_{crit}$ meets its deadline under local execution behavior if there exists a Δ with $0 < \Delta \leq D_i$ such that the condition in Eq. (7.7) holds.

Proof. According to Lemma 10, the case L2L is dominated by N2L. By Lemma 5, L2N is not possible under both recovery protocols. By Lemma 6, the worst-case response time due to N2N is at most R_i^{normal}. By Definition 10, $R_i^{normal} \leq D_i$. Therefore, the only condition to check the schedulability is to verify the case N2L under the return protocol based on Eq. (7.7) in Lemma 9. □

7.7 Evaluation

To evaluate the proposed recovery protocols, comprehensive simulations are performed considering two metrics, namely, the *percentage of run-time*, in which the system exhibits local execution behavior, and the *acceptance ratio* of the schedulability tests in Theorem 4 and Theorem 5, i.e., the number of synthesized task sets that are deemed schedulable. The percentage of run-time under local execution behavior indicates to what extent the recovery protocols are beneficial for maintaining the system's full operability. If, for instance, a few very rarely occurring unsuccessful offloading

operations would lead to permanent local execution behavior, the recovery protocols would fail to satisfy their objectives (cf. Sec. 7.4). The acceptance ratio, in turn, quantifies the price payed for the QoS guarantees given to critical tasks, i.e., the reduced maximum utilization of the local system under normal execution behavior. Subsequently, the experiment setup including a description of the simulation data is given, before the results are discussed.

7.7.1 Experiment Setup

The local system is simulated by an event-based miss rate simulator based on [CVC18], that receives synthesized tasks as well as data obtained from an industrial robot as an input. Depending on the considered input data, different experiments are conducted.

Experiment 1: Synthetic Data

Simulations under the usage of synthetic data are referred to as experiment 1. In these simulations, the system is examined under variations of 1a) the system utilization, 1b) the probability that an offloading operation is performed unsuccessfully, 1c) the percentage of critical tasks in the task set (CT), and 1d) the interval out of which the task periods are generated. Accordingly, for a variation of these parameters, the percentage of time the system exhibits local execution behavior is investigated in the experiments 1a-I) and 1b)-1d), whereas the acceptance ratios are contemplated in experiment 1a-II). To generate the input data for each experiment 1a)-1d)[6], synthetic sets of 10 non-suspending sporadic tasks (per experiment) are created:

- Based on a specific system utilization value, task utilization values are created according to the well-known UUniFast method [BB05].
- The task periods in experiments 1a)-1c) are specified according to a log-uniform distribution over the interval $[1, 100]$ ms, which is a common practice based on [DZB08], and according to a uniform distribution over the intervals $[1, 2]$ ms, $[1, 10]$ ms, $[1, 20]$ ms, $[1, 50]$ ms, and $[1, 100]$ ms in experiment 1d).
- The worst-case execution time under normal execution behavior is given by $c_i = P_i \cdot U_i$.
- The deadlines of all tasks τ_i are set to implicit deadlines, i.e., $D_i = P_i$.

After creating the sets of non-suspending tasks, these are transformed into sets of self-suspending tasks with two computation segments as considered in Chapter 6:

- The worst-case execution time value c_i is divided into $c_{i,1}$ and $c_{i,2}$.
- The length of each task's suspension interval is chosen according to a uniform random distribution out of the interval $[0.01 \cdot (P_i - c_i), 0.1 \cdot (P_i - c_i)]$.

[6]Note that other configurations have been tested as well. Since the results were similar, the configurations are not discussed here.

- To generate the corresponding local computation segments $c_{i,S}$, a scaling factor $\alpha = 2$ is applied such that $C_{k,s} = S_i \cdot \alpha$, since it is assumed that $S_i < c_{i,S}$ according to Chapter 6.
- Priorities are assigned on the task-level according to the well-known rate-monotonic (RM) approach. The percentage of critical tasks in the system (CT) is set to 10 %, 20 %, 30 %, 40 %, 50 %, and 60 % in experiment 1c) and to 20 % otherwise.

The probability that a task τ_i offloads unsuccessfully is assumed to follow a Poisson distribution, i.e., $1 - exp(-\lambda \cdot S_i)$, which models that for a task with longer suspension time the probability of experiencing an unsuccessful offloading operation is higher. λ is set to $0.01 \cdot \frac{1}{ms}, 0.05 \cdot \frac{1}{ms}, 0.1 \cdot \frac{1}{ms}, 0.5 \cdot \frac{1}{ms}$, and $1 \cdot \frac{1}{ms}$ in experiment 1a) and 1b), and to $0.1 \cdot \frac{1}{ms}$ otherwise. Each experiment is repeated 100 times, except experiment 1a-I), which is repeated 10 times. For experiments 1a-I) and 1b)-1d), only task sets are considered that passed the schedulability tests in Theorem 4 and Theorem 5.

Experiment 2: Measured Robot Data

Simulations under the usage of measured data from a Robotnik RB-1 robot platform [Robb] are referred to as experiment 2. The Robotnik RB-1 is capable of performing loading operations of logistics objects in a highly unstructured environment, using the Robot Operating System (ROS) [Roba]. To obtain the input data for the experiment, first, the navigation of the robot in a virtual map was simulated in a Gazebo-based environment [Gaz] and measurements were performed during a time frame of 60 seconds using the Real-Time Scheduling Framework for ROS (ROSCH) [SSA+18] and RESCH [KRI09]. More precisely, the execution times of the so-called *ROS topics* the `move_base` node subscribed to were measured, i.e., `/rb1_base/front_laser/scan`, which is the laser scanner data topic, `/rb1_base/robotnik_base_control/odom`, which is the odometry topic that contains motor encoder readings and is used for navigation, and `/tf`, which contains the transformations between different ROS 3D coordinate frames. Resulting from this, three periodic, implicit-deadline tasks have been obtained:

- $\tau_{laser} = (C_{laser}, T_{laser})$ with $C_{laser} = 6.732$ ms, $T_{laser} = 64.516$ ms
- $\tau_{odom} = (C_{odom}, T_{odom})$ with $C_{odom} = 1.046$ ms, $T_{odom} = 60.0$ ms
- $\tau_{tf} = (C_{tf}, T_{tf})$ with $C_{tf} = 0.333$ ms, $T_{tf} = 60.0$ ms

Considering the cases that 20 %, 40 %, and 60 % of the task workload are offloaded, these tasks are transformed into self-suspending tasks analogously to the tasks in experiment 1). Note that it is assumed that $\mathcal{T}_{crit} = \{\tau_{odom}\}$ and $\mathcal{T}_{non} = \{\tau_{laser}, \tau_{tf}\}$. The local system is simulated with a probability that an offloading operation is unsuccessful of $\lambda = 0.1 \cdot \frac{1}{ms}$. For each scenario, the simulation is repeated 100 times.

7.7.2 Results

Experiment 1: Synthetic Data

1a-I): In Fig. 7.7, the mean values of the percentage of simulation time, in which the system exhibits local execution behavior, over all simulations is depicted as a function of the system utilization under normal execution behavior[7]. While the percentage of time in which the system exhibits local execution behavior under the return protocol is always either close to 0 % or 0 %, the values under the service protocol vary strongly between 0 % and approximately 30 %. From these results, it can be concluded that the percentage of time the system exhibits local execution behavior is not only dependent on the system utilization, but very likely also on other factors discussed later on.

1a-II): The outcome of experiment 1a-II) is shown in Fig. 7.8, namely, the acceptance ratios for the schedulability tests in Theorem 4 and Theorem 5. The service protocol achieves an acceptance ratio of nearly 100 % until a system utilization of approximately 20 % system utilization; an acceptance ratio of 0 % is approached at approximately 70 % system utilization. The return protocol achieves an acceptance ratio of nearly 100 % until a system utilization of approximately 40 %; 0 % are reached under a system utilization of approximately 95 %. Note that similar results were obtained under different configurations.

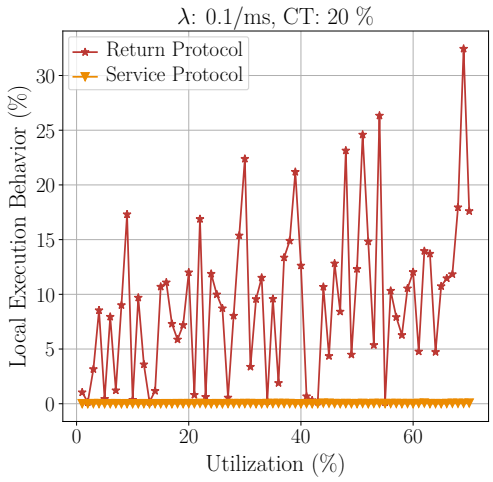


Figure 7.7: The percentage of time in which the system exhibits local execution behavior depending on the system utilization (experiment 1a-I).

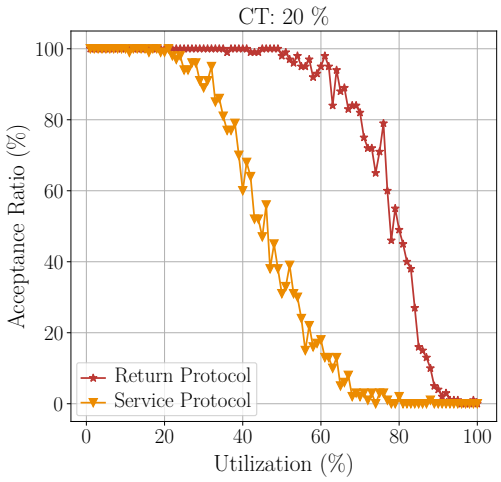
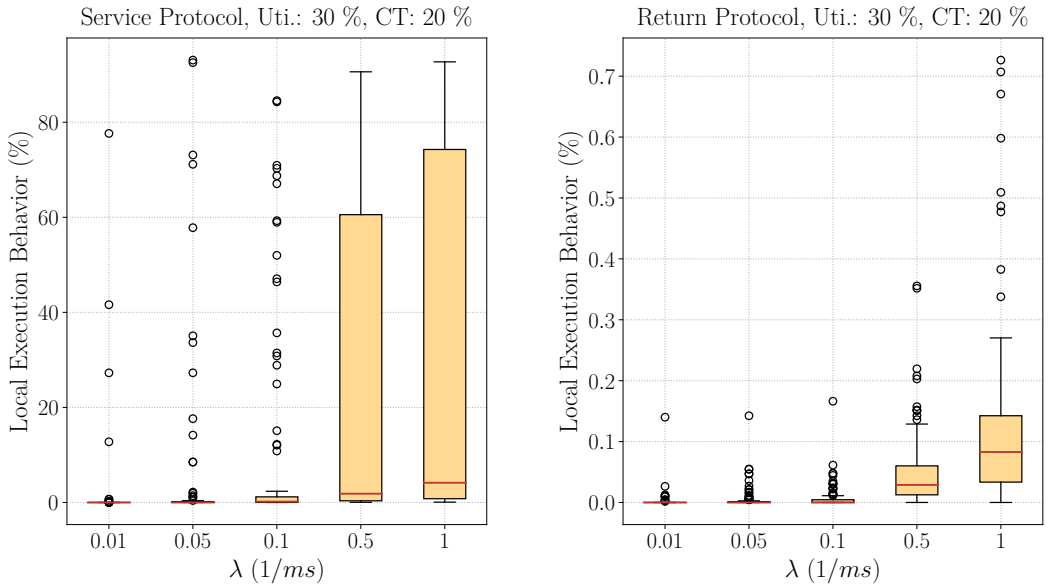


Figure 7.8: The acceptance ratios obtained for the schedulability tests of the service and the return protocol (experiment 1a-II).

[7]Note that simulations have only been carried out for integer utilization percentages; the remaining data points are interpolated.

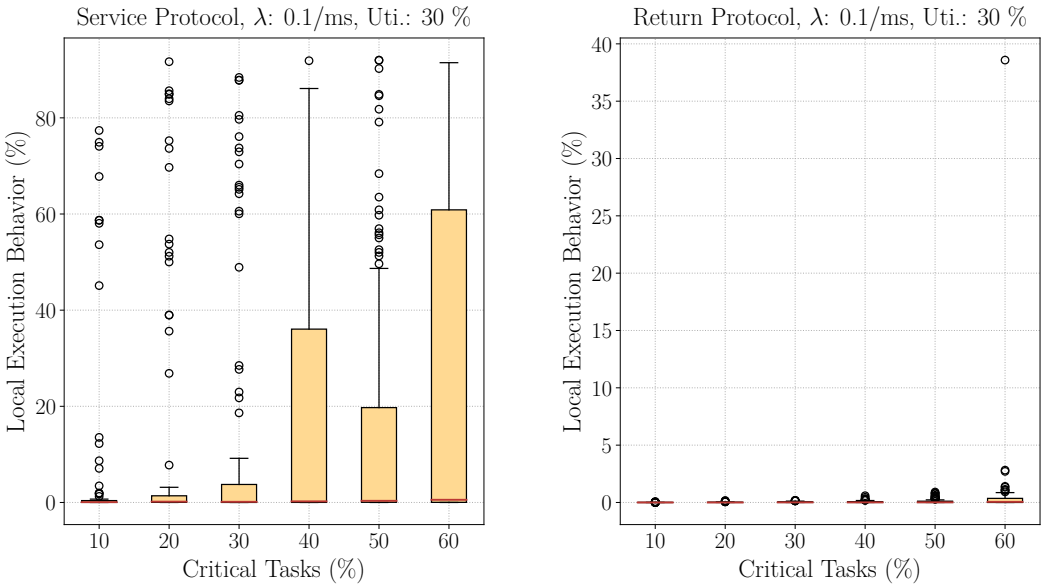


(a) The percentage of local execution behavior for different probabilities of unsuccessful offloading operations under the service protocol.

(b) The percentage of local execution behavior for different probabilities of unsuccessful offloading operations under the return protocol.

Figure 7.9: Experiment 1b): The percentage of time the system exhibits local execution behavior during the simulation for different probabilities of unsuccessful offloading operations under the service and the return protocol with a system utilization of 30 % and 20 % critical tasks. Lower is better.

1b): The percentage of time in which the system exhibits local execution behavior under different probabilities of unsuccessful offloading operations is illustrated in Fig. 7.9a for the service protocol and Fig. 7.9b for the return protocol. Recall that the diagrams can be understood as follows: The box, termed *interquartile range* IQR, represents the middle 50 % of data points, within which the red line marks the median. Accordingly, the lower border of the box indicates the middle value between the smallest data point and the median, denoted by $Q1$, whereas the the upper border of the box indicates the middle value between the largest data point and the median, denoted by $Q3$. The black so-called *whiskers* indicate the distribution of data points outside the interquartile range; the lower whisker marks $Q1 - 1.5 \cdot IQR$ and the upper whisker $Q3 + 1.5 \cdot IQR$. So-called *outliers*, i.e., data points that are located outside of the area limited by the whiskers, are indicated by circles. Under both protocols, the time the system exhibits local execution behavior increases with an increasing probability of unsuccessful offloading operations, despite a small number of outliers. If λ is low, i.e., $0.01 \cdot \frac{1}{ms}$ and $0.05 \cdot \frac{1}{ms}$, in the majority of cases, the



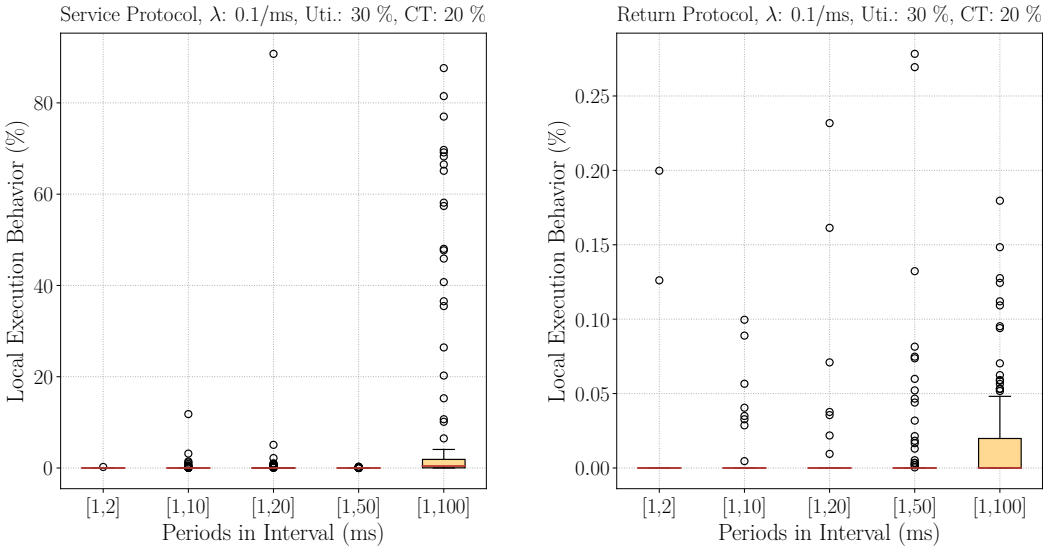
(a) The percentage of time the system exhibits local execution behavior under the service protocol for different percentages of critical tasks in the system.

(b) The percentage of time the system exhibits local execution behavior under the return protocol for different percentages of critical tasks in the system.

Figure 7.10: Experiment 1c): The percentage of time the system exhibits local execution behavior during the simulation under the service and the return protocol for different percentages of critical tasks in the system with a system utilization of 30 % and $\lambda = 0.1 \cdot \frac{1}{ms}$. Lower is better.

system exhibits local execution behavior in a quite low percentage of time under both protocols. If, however, λ is high, i.e., $1 \cdot \frac{1}{ms}$, the service protocol leads to a significantly higher percentage of time under local execution behavior (median at approximately 5 %, $Q3$ at approximately 75 %, upper whisker at approximately 95 %) than the return protocol (median approximately at 0.08 %, $Q3$ approximately at 0.15 %, upper whisker approximately at 0.28 %). This follows from the different handling of non-critical tasks by both protocols under local execution behavior. The outliers can, in general, be explained by the fact that different tasks can experience unsuccessful offloading operations, leading to different consequences (consider, e.g., a task with a short period in contrast to a task with a long period).

1c): The percentage of time in which the system exhibits local execution behavior under different percentages of critical tasks in the system is illustrated in Fig. 7.10a for the service protocol and in Fig. 7.10b for the return protocol. It can be perceived that the percentage of time the system exhibits local execution behavior increases with an increasing percentage of critical tasks in the system, although the increase is larger under the service protocol than under the return protocol (except one outlier



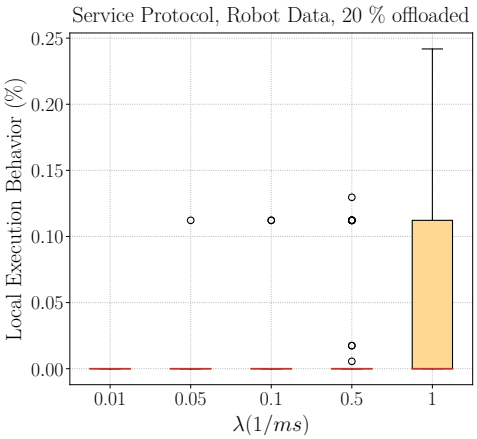
(a) The percentage of time in which the system exhibits local execution behavior under the service protocol for different period intervals.

(b) The percentage of time in which the system exhibits local execution behavior under the service protocol for different period intervals.

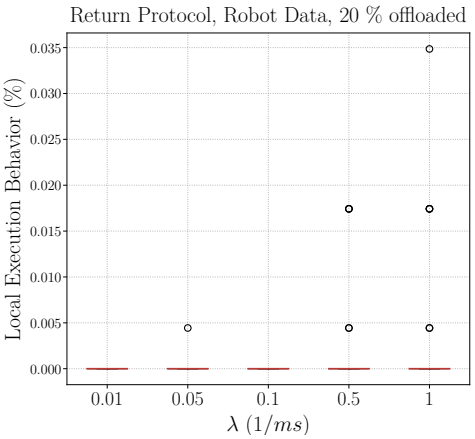
Figure 7.11: Experiment 1d): The percentage of time the system exhibits local execution behavior during the simulation under the service and the return protocol for different intervals for the period generation with UUnifast with a system utilization of 30 %, 20 % critical tasks, and $\lambda = 0.1 \cdot \frac{1}{ms}$. Lower is better.

under a percentage of critical tasks of 60 % in Fig. 7.10b). However, comparing the medians in Fig. 7.10a to those in Fig. 7.9a, it can be stated that the effect of the percentage of critical tasks on the percentage of local execution behavior is weaker than the impact of the probability of unsuccessful offloading operations.

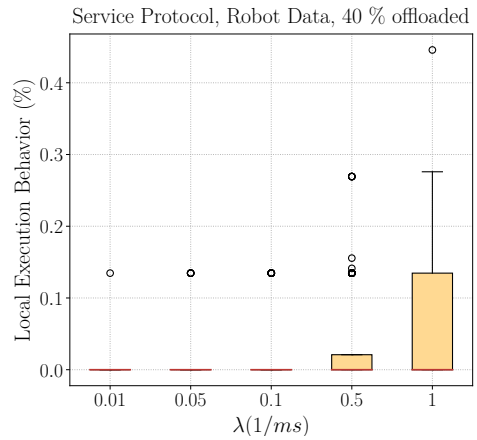
1d): Fig. 7.11a and Fig. 7.11b show the percentage of time the system exhibits local execution behavior for different intervals, with respect to which the task periods are generated with UUnifast, considering a system utilization of 30 %, 20 % critical tasks, and a probability that an offloading operation is unsuccessful of $\lambda = 0.1 \cdot \frac{1}{ms}$, for the service protocol and the return protocol, respectively. Under both protocols, no clear correlation is discernible between the percentage of time the system exhibits local execution behavior and the intervals out of which the task periods are chosen, except for the interval $[1, 100]$. Although the medians are close to 0 % under both protocols, a slight increase of the percentage of time the system exhibits local execution behavior is visible. As stated with respect to the results of experiment 1b), this is likely to result from widely differing task periods leading to an increased amount of time under local execution behavior if a task with a long period offloads unsuccessfully.



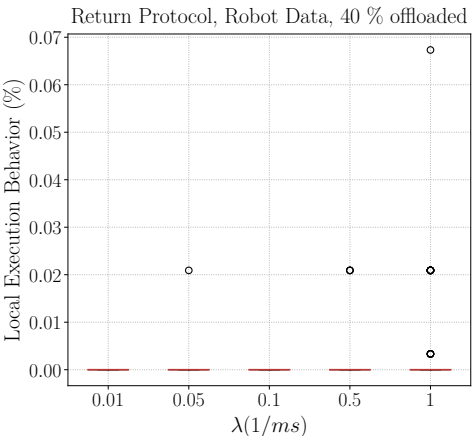
(a) Local execution behavior, 20 % task workload offloaded, *service* protocol.



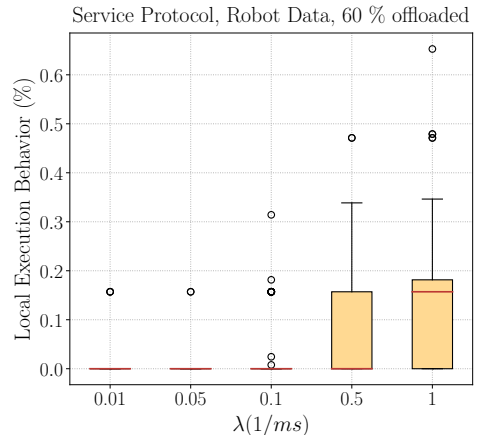
(b) Local execution behavior, 20 % task workload offloaded, *return* protocol.



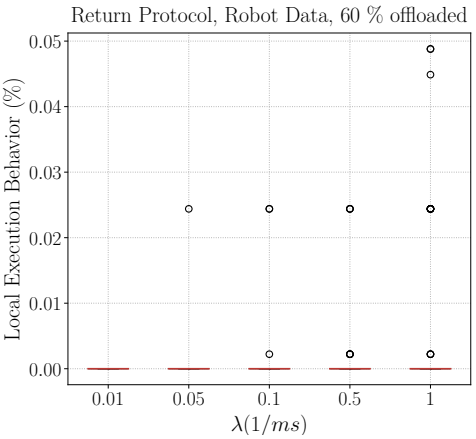
(c) Local execution behavior, 40 % task workload offloaded, *service* protocol.



(d) Local execution behavior, 40 % task workload offloaded, *return* protocol.



(e) Local execution behavior, 60 % task workload offloaded, *service* protocol.



(f) Local execution behavior, 60 % task workload offloaded, *return* protocol.

Figure 7.12: Experiment 2): The percentage of time the robot exhibits local execution behavior during the simulation under the service and the return protocol for different probabilities of unsuccessful offloading operations and percentages of offloaded workload per task. Lower is better.

Experiment 2: Measured Robot Data

In Fig. 7.12, the percentage of time the robot exhibits local execution behavior under the service protocol and the return protocol is illustrated for different probabilities of unsuccessful offloading operations and percentages of workload offloaded per task. From Fig. 7.12b, Fig. 7.12d, and Fig. 7.12f, it is discernible that the amount of offloaded workload per task has no crucial impact on the percentage of time in which the system exhibits local execution behavior.

Under the service protocol, for higher probabilities of unsuccessful offloading operations, i.e. for $\lambda \in \left\{0.5 \cdot \frac{1}{ms}, 1 \cdot \frac{1}{ms}\right\}$, the time the system exhibits local execution behavior clearly increases when the amount of offloaded workload per task is increased. In consequence, it can be concluded that the amount of offloaded workload per task has a strong impact on the system execution behavior under the service protocol and thus should be taken into account when making decisions regarding the system design.

Discussion

By means of the experiments, it has been shown that both proposed protocols do not introduce an intolerable overhead into the system, which makes them attractive for being implemented in real-world systems, and that both protocols fulfill the purposes they have been designed for. The return protocol has been developed with the intention to speed up the system's return from local to normal execution behavior. This effect has been proven throughout the experiments, where the system exhibits less local execution behavior under the return protocol than under the service protocol when other experimental conditions remain unchanged. The purpose of the service protocol is to provide as much service as possible to non-critical tasks, even if the system exhibits local execution behavior. This benefit comes with a certain cost that is reflected in the acceptance ratios: The service protocol is applicable to less task sets, i.e., systems, than the return protocol.

It has been proven empirically that the probability of an unsuccessful offloading operation has a strong impact on the time a system exhibits local execution behavior; however, this parameter is outside system designers' sphere of influence. In consequence, it is meaningful to control a number of other parameters that have been shown to have an impact on the time a system exhibits local execution behavior: For both protocols, the time the system exhibits local execution behavior can be reduced by reducing the amount of critical tasks in the system, by carefully and smartly designing the task periods, and by executing as much workload as possible locally instead of offloading it under normal execution behavior. As seen in the experimental results, the effect of these measures is stronger under the service protocol than under the return protocol, following from the anyway smaller amount of time the system exhibits local execution behavior under the return protocol.

7.8 Summary

In order to ensure the correct and safe operation of a system, e.g., a smart vehicle, that offloads parts of its critical applications to a smart city's distributed infrastructure, two recovery protocols with distinct objectives have been introduced in this chapter. By implementing the proposed protocols in a system, it can be guaranteed that the QoS requirements of all critical applications are satisfied at any point in time, even in situations where the connections used for offloading are unreliable, and that the QoS requirements of all applications executed on the system are satisfied whenever offloading operations are performed successfully. This can be verified at system design time by applying the schedulability tests that have been provided together with the recovery protocols. By performing simulations based on synthetic data as well as on data measured on a real-world robotic system, it has been shown that both recovery protocols come with reasonable acceptance ratios, i.e., that the reduction of the maximum system utilization payed as the price for an increased safety is acceptable. Moreover, the protocols have been proven to not cause the system to exhibit local execution behavior excessively, during which no QoS guarantees are given for non-critical tasks. In fact, it was detected that the system behavior under the proposed protocols depends on different factors, namely, on the probability of unsuccessful offloading operations, the percentage of critical tasks in the system, and the amount of offloaded workload. Evidence has also been found that the percentage of time the system exhibits local execution behavior depends on the actual task experiencing an unsuccessful offloading operation and, in consequence, also on the interval out of which task periods are chosen. Consequently, these aspects must be taken into account when designing a system that is intended to use one of the proposed protocols.

Notation	Meaning
α	Scaling factor
Δ	Time interval
$\gamma_{1,\searrow}$	First moment, when the offloading operation of τ_i is unsuccessful
$\gamma_{1,\nearrow}$	Moment, when QoS guarantees can be given again for all tasks
$\gamma_{h,\searrow}$	h-th moment σ switches from normal to local execution behavior
$\gamma_{h,\nearrow}$	h-th moment σ switches from local behavior to normal behavior
λ	Probability that a task offloads unsuccessfully
R_i^{normal}	Worst-case response time of τ_i under normal execution behavior
R_i^1	Worst-case response time of the first computation segment of τ_i under normal execution behavior
σ	Concrete fixed-priority preemptive schedule for $\mathcal{T}$

Table 7.1: Overview of the notation introduced in Chapter 7.

CHAPTER 8

Hardware Message Filter Optimization

Parts of this chapter have previously been published in [SBS+19].

Contents

8.1 Introduction

A smart city's infrastructure can be used for the execution of computationally intensive applications that cannot be executed on an endpoint system without leading to a degradation of other applications' QoS. Especially resource-constrained systems such as smart vehicles with advanced driver assistance systems (cf. Fig. 8.1) benefit from this opportunity, since offloading computation saves local resource capacity and thus

allows to satisfy the QoS requirements of critical and non-critical tasks likewise, as explained in Chapter 7. There exist, however, applications for which offloading is not possible, but that can introduce a non-negligible workload to a smart vehicle's electronic control unit (ECU). One such application is the inspection of messages received via communication resources, which determines if a message is relevant to the ECU and must be further handled, or if it can be discarded. While the inspection of a single message does not create a relevant computational overhead, the accumulation of such small workloads can become problematic if messages are received with a short minimum inter-arrival time or period[1], particularly for ECUs with low clock speeds.

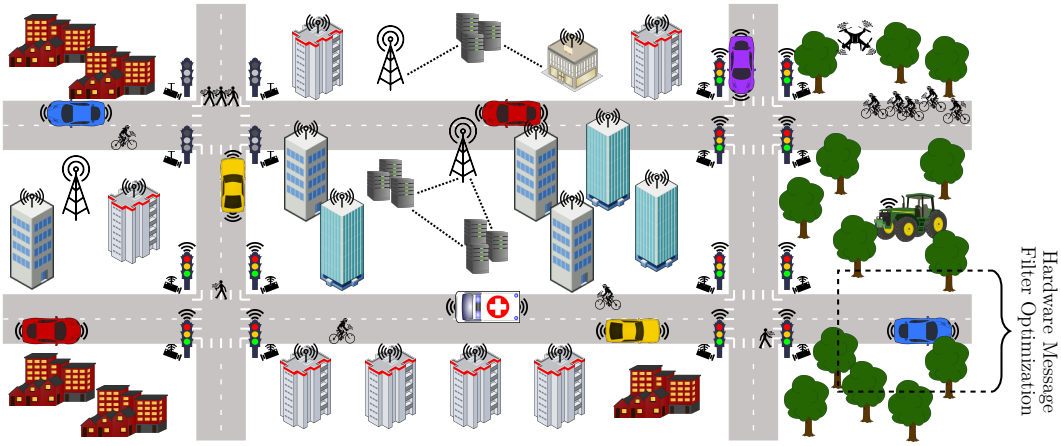


Figure 8.1: Undesired CAN messages introduce computational overhead to a smart vehicle's ECUs, but can be blocked by configurable hardware message filters. This chapter proposes approaches for optimizing hardware message filter configurations.

Modern vehicles still rely on Controller Area Network (CAN) [Rob91] as their communication backbone, a broadcast bus that does not provide point-to-point communication. Accordingly, messages written to the bus are not transmitted to a specific destination, but are received by all bus participants, i.e., connected ECUs. In consequence, each ECU is required to inspect all incoming messages, even though only a subset of these is deemed relevant. To reduce this overhead, *hardware message filters* can be used by an ECU that allow to match the bit-vector of a message's identifier with predefined bit-patterns in order to block *undesired messages*. These filters are provided by the majority of commercial off-the-shelf CAN controllers and can be configured according to the system's needs. This chapter addresses the optimization of hardware message filters, aiming to minimize the number of undesired messages received by each ECU.

[1]Note that subsequently the term *period* is used instead of *minimum inter-arrival time*, complying with the majority of related works.

8.2 Problem Statement

In an endpoint system as described in Chapter 6, where a set of computation resources is connected to a CAN bus, each resource, subsequently referred to as a *node*, broadcasts messages that can be received by all other nodes connected to the bus. In CAN, message are transmitted in the standardized form of data frames that consist of different fields, such as, among others[2], an *identifier* (ID). The ID is a vector of 11 bits (or 29 bits in the extended data frame format) length that is used for bus arbitration, i.e., to decide which message is transmitted if multiple messages are written to the bus at the same time. Consequently, the ID encodes the message priority and must be unique to ensure the correct operation of the bus arbitration. To cover both data frame formats, a message is subsequently characterized by a tuple $m = (b_m, P_m)$, where b_m indicates the message ID of length H and P_m the period. Each node connected to the bus can send messages with IDs that are prespecified at system design-time. The overall set of messages with prespecified IDs that can be used by one or multiple nodes is indicated by M; its subset of messages that is relevant to a considered node is indicated by M^D and termed the set of *desired messages*. The set of *undesired messages* of a node is denoted by M^U. Note that $M = M^D \cup M^U$ and $M^D \cap M^U = \emptyset$.

Each node has a set of hardware *filters* $F = f_1, \ldots, f_n$ with $n \in \mathbb{N}$, also referred to as *filter configuration*. Each filter f is characterized by a tuple x_i, Y_i, where x_i is a *mask* with $i \in \mathbb{N}$ and Y_i is a set of *tags* $y_{i,k} \in Y_i$ with $k \in \mathbb{N}$. Both masks and tags are bit vectors of length H that are matched with the message ID b_m of an incoming message m. Since b_m is unique, the patterns represented by a mask x_i and its tags Y_i can be modified so that they correspond only to a selected set of message IDs. A message m is *accepted* by a filter f consisting of a mask x_i with tag $y_{i,k}$ if and only if the statement $b\_and(x_i, y_{i,k}) = b\_and(x_i, b_m)$ is true, where b_and is the bit-wise *and*-operation. This filtering process is illustrated by an example in Table 8.1, where a filter consisting of one mask with one tag is given as well as a set of message IDs.

The filter pattern resulting from the given mask and tag can be understood as follows: Whenever a bit of an ID is not considered by the filter, it is marked as irrelevant by setting the corresponding bit of the mask to 0. An irrelevant bit is represented by a *don't care*, denoted by $*$, in the resulting filter pattern. All other bits indicate the values that a message ID must exhibit for the message to be accepted.

Depending on the set of accepted messages, a filter configuration can be classified as *correct*, *perfect*, or *imperfect*. A correct filter configuration accepts all desired messages, whereas a perfect filter configuration accepts all desired messages and, additionally, blocks all undesired messages. An imperfect filter configuration accepts all desired messages, but does *not* block all undesired messages and is therefore not preferable. However, due to the limited number of hardware filters in commercial off-the-shelf components, imperfect filters are not always avoidable. A quality metric

[2]Since only the *identifier* field is relevant for this chapter, the remaining ones are not further discussed. A complete overview can be found in [Rob91].

	Bit Pattern	Accepted?
Mask	011 1111 1110	
Tag	000 1100 1001	
Filter	∗00 1100 100∗	
ID	000 1100 0001	no
ID	000 1100 1000	yes
ID	000 1100 1001	yes
ID	100 1000 1000	no
ID	100 1100 1000	yes
ID	100 1100 1001	yes

Table 8.1: A filter (one mask, one tag per mask) and a set of exemplary message IDs.

for hardware filter configurations that quantifies the penalty due to undesired but accepted messages, the so-called *Quality of Filter* (QoF), is introduced in [PDB17] and given as $QoF = \sum_{m \in M^U} \frac{[m \text{ is accepted}]}{P_m}$, where the numerator uses the Iverson bracket. The QoF is optimal if the given expression evaluates to zero, i.e., if the filter is perfect.

In this chapter, two distinct scenarios are contemplated regarding the optimization of hardware filter configurations. First, the design of new systems is addressed where the number of undesired messages accepted by the filter(s) may be reduced to zero if a suitable number of hardware filters can be installed. Accordingly, an approach is required for designing perfect filter configurations under no hardware limitations. Second, the optimization of filters in existing systems is considered, where hardware components cannot be easily exchanged, so that minimizing the QoF is highly important in order to reduce the unnecessary overhead for handling the undesired, but accepted, messages.

Objective of this Chapter

Under the considered system model, this chapter aims to provide approaches for the computation of hardware filter configurations, namely, of perfect filter configurations under no hardware limitations, of imperfect filter configurations with minimized QoF, and of imperfect filter configurations with minimized hardware cost for a given maximum QoF threshold.

Subsequently, approaches for the computation of perfect filters are introduced in Sec. 8.3 and of imperfect filter configurations in Sec. 8.4. An evaluation of the proposed approaches is presented in Sec. 8.5. Related works addressing message IDs and hardware message filtering in CAN are discussed in Sec. 8.6. An overview of the notation introduced in this chapter is given in Table 8.3.

8.3 Perfect Filters

Recall that each ID b_m for $m \in M$ is a unique sequence of H bits, i.e., a bit-vector of length H. Considering each bit $b_{m,\ell}$ with $\ell \in \mathbb{N}$ as a Boolean variable, it is possible to express an ID as a Boolean function $I : \{0,1\}^H \rightarrow \{0,1\}$ with $I(b_{m,1}, b_{m,2}, \ldots, b_{m,H}) := 1$ if $m \in M^D$ and $I(b_{m,1}, b_{m,2}, \ldots, b_{m,H}) := 0$ otherwise. Accordingly, if m is a desired message, the conjunction $b_{m,1} \wedge b_{m,2} \wedge \cdots \wedge b_{m,H}$ must evaluate to 1. Such a conjunction of all function variables that evaluates to 1 is known as a *minterm* of a Boolean function.

In order to obtain a perfect filter configuration for one node connected to a CAN bus, the fact that each message and each filter can be expressed by Boolean algebra can be exploited to formulate a number of constraints specifying the actual requirements a filter must satisfy. To compute such a filter configuration, an SMT solver can be applied. SMT refers to the automatic satisfiability verification of logic constraints restricting the interpretation of all symbols to the same logic background theory [BT18]. If a solution, i.e., a set of variables, exists for which all constraints can be satisfied, one such solution is provided by the SMT solver. This solution, however, is not necessarily optimal. In the following, two constraint formulations expressing the characteristics expected from a perfect filter configuration are proposed.

8.3.1 Accepting Desired and Nonexistent Messages

As a consequence of the metric's definition, the acceptance or rejection of messages with unspecified, i.e., nonexistent, IDs has no impact on the QoF of a filter[3]. Therefore, each filter can be permitted to accept any $m \in M^{ok}$, where M^{ok} is defined as $M^{ok} = M^{all} \backslash M^U$ and M^{all} is the set of all messages whose IDs that can be expressed by an H-bit vector, to reduce the number of filters required for a perfect filter configuration. For a given number of masks x_i with tags $y_{i,k} \in Y_i$, a perfect filter configuration of such kind can be obtained by solving the following constraint formulation, where *b_and* indicates the bit-wise *and*-operation and $z_{m,i,k}$ is a Boolean variable:

$$\forall m, i, k, \quad z_{m,i,k} = (b\_and(x_i, y_{i,k}) = b\_and(x_i, b_m)) \tag{8.1}$$

$$\forall m \in M^D, \quad \vee_i \vee_k \; z_{m,i,k} = true \tag{8.2}$$

$$\forall m \in M^U, \quad \vee_i \vee_k \; z_{m,i,k} = false \tag{8.3}$$

Eq. (8.1) expresses the filtering process during which a message m is either accepted or blocked by a filter with mask x_i and tag $y_{i,k}$ (cf. Sec. 8.2). Eq. (8.2) and Eq. (8.3) ensure the correctness and the perfectness of the filter configuration. More precisely, Eq. (8.2) enforces that only desired messages are accepted and Eq. (8.3) that only undesired messages are rejected. Consequently, every filter configuration satisfying Eq. (8.1)-(8.3) that is retrieved by an SMT solver, is a perfect filter configuration.

[3]Note that this is not the case if messages with unspecified IDs are sent in the context of a malicious attack. This, however, is beyond the scope of this chapter.

8.3.2 Filtering with Prime Implicants

If a further reduction of the number of filters is possible, a *minimal* perfect filter configuration can be computed by using *prime implicants*. An *implicant* of a Boolean function is an expression over its function variables for which must hold that it evaluates to 1 whenever the Boolean function evaluates to 1. A *prime implicant* is an implicant that cannot be further reduced, i.e., from which no variables can be removed, without loosing the property of being an implicant. Prime implicants can be generated by merging two or more minterms, e.g., by the well-known algorithm of Quine & McCluskey [Nel53].

Enforcing that each filter $f \in F$ is a prime implicant of M^{ok}, i.e., $f \in PI(M^{ok})$, for the case that each mask x_i has only one tag $y_{i,k}$, i.e., $k = 1$, the constraint formulation proposed in Sec. 8.3.1 can be extended as follows, where b_xor is the bit-wise xor-operation:

$$\forall m, i, k, \quad z_{m,i,k} = (b\_and(x_i, y_{i,k}) = b\_and(x_i, b_m)) \tag{8.4}$$

$$\forall m \in M^D, \quad \vee_i \vee_k \; z_{m,i,k} = true \tag{8.5}$$

$$\forall m \in M^U, \quad \vee_i \vee_k \; z_{m,i,k} = false \tag{8.6}$$

$$\forall i, k, \quad b\_xor(x_i, y_{i,k}) \in PI(M^{ok}) \tag{8.7}$$

By solving this extended constraint formulation, a minimal perfect filter configuration can be computed, as stated by the subsequent theorem.

Theorem 6 (Minimal Perfect Filter Configuration)**.** Any filter configuration F obtained applying Eq. (8.4)–(8.7), where each mask x_i has one tag $y_{i,k}$, i.e., $k = 1$, is perfect and minimal.

Proof. F being perfect results directly from Eq. (8.6). To show that F is minimal, consider a non-minimal, perfect filter configuration $F' = f_1, \ldots, f_n$. Since F' is not minimal, at least two filters $f'_a, f'_b \in F'$ can be reduced to a filter f_c by substituting at least one filter bit by a *don't care*, so that f'_c accepts all messages accepted by f_a and f_b but blocks all undesired messages $m \in M^U$. Assume that the reduction of F' is continued as far as possible while maintaining its optimal QoF, i.e., F' is transformed from a non-minimal perfect filter configuration to a minimal perfect filter configuration. Since F' cannot be further reduced without accepting any $m \in M^U$, it is clear that $F' \subseteq PI(M^{ok})$ with $M^{ok} = M^{all} \backslash M^U$. □

Note that the generation of prime implicants may become computationally intractable depending on the size of the message space, which depends on the considered length of the message ID H. As an alternative, the SMT formulation proposed in Sec. 8.3.1 can be applied if the availability of hardware filters is not limited. Otherwise, an *imperfect* filter can be used.

8.4 Imperfect Filters

Perfect filter configurations cannot always be realized for several reasons. When considering existing systems, hardware components can typically be modified only up to a certain extent or not at all, such that the achievable QoF is limited by the given hardware. When designing new systems, hardware limitations may be given as well, for instance, due to cost or space constraints. In consequence, it is often unavoidable that a filter configuration accepts a number of undesired messages. However, not all undesired messages introduce the same overhead: Consider two messages $m_a, m_b \in M^U$. If m_a has a short period and m_b a long one, m_a must be further processed by the ECU more frequently than m_b and thus is more costly. Accordingly, when designing imperfect filter configurations, the set of undesired but accepted messages should be chosen such that the introduced overhead, quantified by the QoF, is minimized.

8.4.1 Minimizing the QoF

Aiming to minimize the QoF for a given filter hardware, i.e., a given number of masks x_i with a given number of tags $y_{i,k}$, the constraint formulation proposed in Sec. 8.3.1 can be modified and extended as follows, where c_m is the cost, i.e., the overhead, introduced by a message m, c_g is an upper bound on the overall cost, i.e., the QoF, and $\oplus$ is the logic *xor* operator:

$$\forall m, i, k, \quad z_{m,i,k} = (b\_and(x_i, y_{i,k}) = b\_and(x_i, b_m)) \tag{8.8}$$

$$\forall m \in M^D, \quad \vee_i \vee_k \; z_{m,i,k} = true \tag{8.9}$$

$$\forall m \in M^U, \quad \left(c_m = \frac{1}{P_m}\right) \oplus (\vee_i \vee_k \;\; z_{m,i,k} = false) \tag{8.10}$$

$$\forall m \in M^U, \quad (c_m = 0) \oplus (\vee_i \vee_k \;\; z_{m,i,k} = true) \tag{8.11}$$

$$\sum_{m \in M^U} c_m \leq c_g \tag{8.12}$$

The constraints Eq. (8.10)-(8.12) can be understood as follows: Eq. (8.10) requires that each $m \in M^U$ must be either rejected by all filters or introduces a cost greater than zero. Correspondingly, Eq. (8.11) enforces each $m \in M^U$ introduces either zero cost or is accepted by at least one filter. In Eq. (8.12), a QoF threshold c_g is set that must not be exceeded. Recall that an SMT solver does not provide optimal solutions. For this reason, c_g is initialized with an upper bound on the QoF that is stepwise reduced during an optimization process, for which, e.g., simple heuristics such as binary search, random search etc., can be applied.

8.4.2 Minimizing the Hardware Cost

When designing new systems, it can be meaningful to reduce the hardware cost resulting from a filter configuration in order to lower the overall system cost on the one hand and the system size on the other hand. Especially with respect to low-budget embedded devices, both factors may be non-negligible. To compute filter configurations for such system, the constraint formulation proposed in Sec. 8.4.1 can be used, for which a fixed upper-bound on the QoF must be given that quantifies the tolerable overhead introduced by undesired but accepted messages. In contrast to the previous scenario, only the number k of tags per mask but not the number i of masks is fixed, since the considered hardware component type is assumed to be given. To determine the minimum required number of hardware components, i is decreased in each iteration until the cost threshold c_g is overshot.

8.5 Evaluation

To validate the approaches proposed in this chapter, distinct experiments based on multiple benchmarks are performed. Subsequently, the benchmarks and the experiment setup are described, before the results are discussed.

8.5.1 Benchmarks

To evaluate the proposed approaches, different benchmarks are considered. The *epsilon* benchmark refers to the the network specification of the battery electric vehicle *epsilon*'s [Eps] heating, ventilation, and air conditioning controllers provided by [PDB17]. It comprises 55 11-bit message IDs with periods, out of which 11 are desired and 44 undesired. The *SAE* benchmark consists of 55 messages that were originally presented in [Aut94] and provided in a modified version in [LJP17]. From the modified benchmark, 18 desired messages are selected. Since no message IDs are provided, 11-bit message IDs in a range from 0 to 2047 are created and assigned randomly. The resulting set of message IDs is given in Table 8.2.

The *NDA* benchmark is an industrial benchmark with 17 desired and 12 undesired messages. Due to a non-disclosure agreement, further details about the benchmark are omitted. As the *Racing* benchmark it is referred to a benchmark of a Formula Student Germany [For] racing car provided by GET racing Dortmund e.V. [GET]. For reasons of competitiveness, further details about the benchmark are omitted. Moreover, synthetic benchmarks are created by generating sets of message IDs to which periods from the set $\{50ms, 100ms, 500ms, 1000ms\}$ are assigned randomly.

8.5.2 Experiment Setup

For the evaluation, different experiments are performed. In order to provide comparable time measurements, all experiments are conducted on a machine with Intel® Xeon® i7-9800X processor (3.80GHz, 16 CPUs, 8 cores per socket, 2 logical threads

ID [hex]	T [ms]	Des.	ID [hex]	T [ms]	Des.	ID [hex]	T [ms]	Des.
0x058	20	yes	0x28A	20		0x5B4	5	yes
0x06B	20		0x2AD	20	yes	0x5CE	10	
0x085	5	yes	0x2C6	1000		0x601	1000	
0x0C6	20	yes	0x2EF	10		0x624	20	
0x0DC	20		0x309	20	yes	0x644	10	
0x11E	5		0x316	10		0x667	20	
0x138	1000		0x336	20		0x66B	20	yes
0x13F	20	yes	0x351	20	yes	0x6AB	20	yes
0x148	20	yes	0x364	20	yes	0x6D2	20	
0x15E	20		0x38A	20		0x6D3	20	yes
0x1BA	5		0x3A1	20		0x6F4	20	yes
0x1F8	20		0x3F7	20		0x6F8	100	
0x209	20	yes	0x40A	5		0x725	10	
0x212	1000		0x479	1000		0x755	20	
0x235	10		0x4B4	5		0x772	20	
0x236	20		0x4B5	20	yes	0x77C	5	yes
0x23F	20	yes	0x564	20		0x7D8	1000	

Table 8.2: The modified SAE benchmark by Lesi et al. [LJP17] with randomly chosen desired messages and synthetic message IDs generated according to a uniform distribution.

per core) and 31 GiB RAM. The proposed constraints are solved using the Z3 solver by Microsoft [MB08].

Experiment 1: Minimum and Varying Number of Filters

A minimum perfect filter configuration for the epsilon benchmark is computed considering a filter component with one tag per mask. Beginning with the retrieved minimum number of filters required for a minimum perfect filter configuration, the number of filters is stepwise reduced to 1. For each considered number of filters, an imperfect filter configuration is computed. The QoF of the filter configurations as well as the time required for the computation are documented and compared to the results reported in [PDB17]. The SAE benchmark is examined under the same settings. For comparison, the experiment is performed for the NDA benchmark and for the Racing benchmark as well.

Experiment 2: Varying Number of Tags

The impact of the number of tags per mask on the time required to find an imperfect filter configuration is analyzed based on the epsilon benchmark and the SAE bench-

mark. For this purpose, a filter component with one mask and different numbers of tags is considered.

Experiment 3: Hardware Cost Optimization

With respect to the optimization of the hardware cost, the number of required hardware filter components for a given QoF threshold as well as the time required for computing a filter configuration is examined using synthetic benchmarks. For this purpose, a component with 1 tag per mask and a set of 100 randomly created message IDs are considered out of which 20 IDs are marked as desired. A maximum QoF threshold of 5 %, 10 %, and 20 % of the maximum possible QoF (also referred to as the allowed QoF overhead) is considered. For each configuration, 10 sets of message IDs are generated.

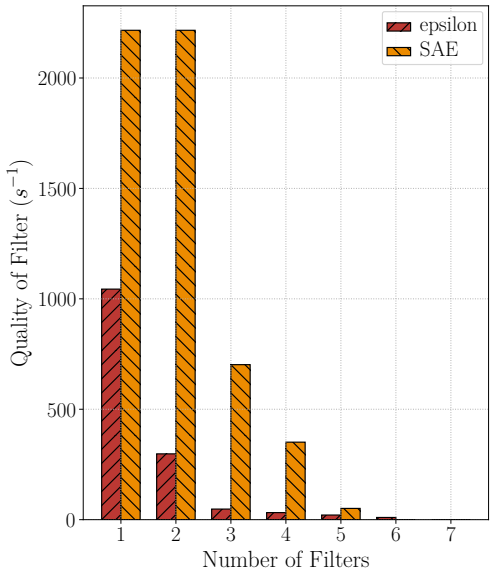
8.5.3 Results

Experiment 1: Minimum and Varying Number of Filters

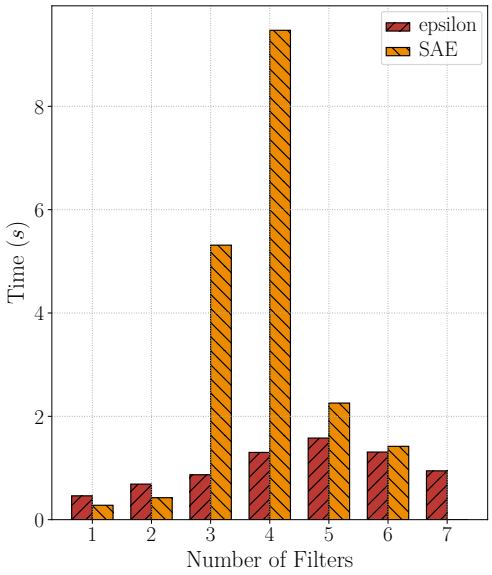
For the epsilon benchmark and an unlimited number of available hardware components with 1 mask and 1 associated tag, a perfect filter configuration with 7 filters is obtained within 0.94 seconds. Identical results are reported by [PDB17], however, no information about the run-time is provided. For 3 filters, a QoF of 48 s^{-1} is obtained within 0.87 seconds, as illustrated in Fig. 8.2a and Fig. 8.2b. The QoF achieved for this scenario by [PDB17] is 48 s^{-1} as well, but, analogously to the previous case, no run-time is documented. Note that the maximum, i.e., the worst, QoF that can be achieved for the epsilon benchmark is 1180 s^{-1}. As evident from Fig. 8.2a, the QoF improves with an increasing number of masks, i.e., filters. This observation can be made as well with respect to the SAE benchmark. Note, that a QoF of $2216 s^{-1}$ is achieved for the SAE benchmark, i.e., its maximum possible QoF, under 1 as well as under 2 masks, since the number of filters is not sufficient to block any undesired message.

In general, the time required for computing a filter configuration for the SAE benchmark is higher than for the epsilon benchmark, as shown in Fig. 8.2b. One important aspect affecting the run-time that can explain this observation is the ID design. In fact, the message IDs of the SAE benchmark are created randomly. Hence, it is possible that some IDs differ by an unnecessarily high number of bits, resulting in an increase of computation time for the SMT solver. With respect to the epsilon benchmark, in contrast, it can be assumed that the ID assignment has been done very carefully, since these message IDs are actively used in a real-world system. The validity of this explanation is supported by the results obtained for the NDA benchmark and the Racing benchmark[4].

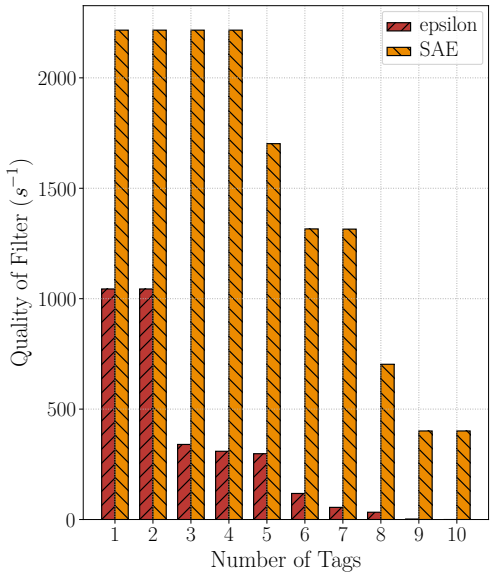
[4]Due to a non-disclosure agreement concerning the NDA benchmark and for reasons of competitiveness with respect to the Racing benchmark, details about the results are omitted.



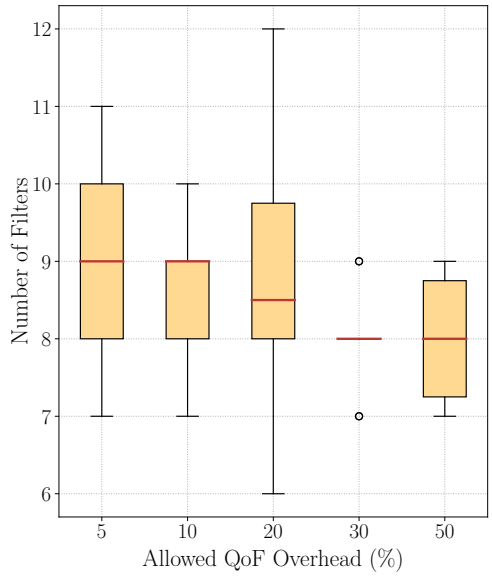
(a) QoF for different number of filters considering two benchmarks.



(b) Computation time for different number of filters considering two benchmarks.



(c) QoF for one mask with different number of tags considering two benchmarks.



(d) Minimum number of filters for allowed QoF (synthetic benchmarks).

Figure 8.2: Results of different experimental scenarios investigating the QoF for different numbers of filters, the time required for computing filter configurations, and the minimum number of filters ensuring that a given QoF threshold is not overshot. Lower is better.

Experiment 2: Varying Number of Tags

When considering hardware filter components consisting of only one mask with multiple tags, a perfect filter configuration for the epsilon benchmark can be obtained when 10 tags are available, as shown in Fig. 8.2c. The time required to compute the solutions was measured for both the epsilon and the SAE benchmark, however, due to strong variations without a discernible trend it cannot be generally stated if solutions can be computed faster when considering multiple filter components with 1 tag per mask or for one filter component with multiple tags. Consequently, the results are not illustrated here. When deciding about the type of filter components to be used in a system, however, it should be taken into account which hardware components are available as commercial off-the-shelf hardware, since filters with very specific numbers may need to be expensively custom-made.

Experiment 3: Hardware Cost Optimization

Regarding the optimization of the number of required hardware components under a given upper bound on the QoF, the computed minimum number of filters is illustrated in Fig. 8.2d. The diagram can be understood as follows: The box, termed *interquartile range* IRQ, represents the middle 50 % of data points, within which the red line marks the median. Accordingly, the lower border of the box indicates the middle value between the smallest data point and the median, denoted by $Q1$, whereas the the upper border of the box indicates the middle value between the largest data point and the median, denoted by $Q3$. The black so-called *whiskers* indicate the distribution of data points outside the interquartile range; the lower whisker marks $Q1 - 1.5 \cdot IQR$ and the upper whisker $Q3 + 1.5 \cdot IQR$. So-called *outliers*, i.e., data points that are located outside of the area limited by the whiskers, are indicated by circles.

It can be observed that with increasing allowed QoF overhead less filters are required, except for an overhead of 30 %. However, the number of filters strongly varies for each considered scenario, i.e., for each QoF overhead value. The computation times required for obtaining a solution varies as well: For an overhead of 5 % of the maximum possible QoF, a solution is computed within an average time of 11.84 seconds with a standard deviation of 18.85 seconds. For 10 % overhead, an average computation time of 9.21 seconds with a standard deviation of 8.87 seconds is required, for 20 % overhead, 60.88 seconds with a standard deviation of 234.69 seconds, for 30 % overhead, 6.35 seconds with a standard deviation of 1.99 seconds, and for 50 % overhead 6.62 seconds with a standard deviation of 4.1 seconds.

Based on these results, it is not possible to make a clear statement regarding the dependence of the computation time on the of permitted QoF overhead. However, the lack of a clear trend as well as the relatively high standard deviations suggest a relation to the random nature of the synthetic message IDs considered in the experiment and thus support the previous statement that the number of filters as well as the time consumed to compute a solution strongly depend on the ID design. In consequence, it can be concluded that it is necessary to optimize the ID assignment

of message sets in CAN for multiple reasons: An optimization does not only allow to design better hardware message filters and, in the course of this, to reduce the computational overhead for inspecting incoming messages in terms of their relevance, but also contributes to reducing the time required for configuring hardware filters as well as the actual (hardware) cost of a system.

8.6 Related Work

The topic of hardware message filtering in CAN has been addressed quite scarcely in the literature. In fact, the majority of works taking CAN message IDs into consideration focuses on schedulability analysis, e.g., [DBB+07], or priority assignment. For instance, [PDB16] proposes an approach for designing extensible ID assignments, such that further messages can be integrated into the system at a later point in time without violating the schedulability. An optimal priority assignment for the scheduling of a mixture of CAN and CAN FD [Rob12] frames is proposed in [PS19]. As part of this approach, IDs are split into distinct priority and filter sections.

The first work addressing the the optimization of hardware message filter configurations in CAN has been published in 2017 [PDB17], followed by [SBS+19], which covers the approaches proposed in this chapter. Building on the findings emphasized in Sec. 8.5, strategies for the optimization of already existing ID assignments are proposed in [SSC20].

Message IDs are also considered in the context of security. For instance, [SLJ+19] proposes a so-called dynamic ID virtualization technique. By assigning ID ranges instead of IDs to each message and by randomly changing each ID within its range, attackers shall be prevented from injecting messages with predefined IDs into the system. This approach, however, could be detrimental to the QoF of hardware message filters in the system. In fact, since no fixed IDs are assigned, filters cannot be configured in a fine-granular manner, but may need to exhibit a high number of *don't care* bits. It may, nevertheless, be possible to choose sets of IDs instead of ranges, which are composed of IDs that can be merged easily, in order to anyway design filter configurations with a reasonable QoF. For this purpose, the constraint formulations proposed in this chapter can be supportive.

8.7 Summary

The objective of this chapter was to propose an approach to reduce the workload imposed on a smart vehicle's ECU when inspecting received messages in terms of their relevance. To achieve this, methods for computing perfect and imperfect hardware message filter configurations have been proposed and their effectiveness has been confirmed by evaluations involving real-world benchmarks. It has been observed that the design of message IDs has a strong impact on the minimal number of filters needed for perfect filtering when the number of available filters is not limited, on the

attainable QoF under a limitation of filter components, as well as on the time required for computing filter configurations. Against the background of these findings, the necessity of optimizing the ID assignments of sets of messages has been emphasized in order to compute filter configurations with lower, i.e., better, QoF within shorter time and to save hardware components. Based on the contributions and insights presented in this chapter, an optimization approach has already been published in [SSC20].

Notation	Meaning
m	CAN message
b_m	Message ID
P_m	Period of m
H	Message ID length
M^D	Set of desired messages
M^U	Set of undesired messages
F	Filter configuration
f	Filter
x_i	Mask of a filter
Y_i	Set of tags of a mask
$y_{i,k}$	Tag in a set of tags
b_and	Bit-wise *and* operation
b_xor	Bit-wise *xor* operation
$\oplus$	Logic *xor* operation
$*$	*don't care* bit
QoF	Quality of Filter; the lower, the better
$b_{m,\ell}$	Bit in a bit-vector
M^{all}	Set of messages whose IDs can be expressed by an H-bit vector
M^{ok}	$M^{ok} = M^{all} \backslash M^U$
$PI(M^{ok})$	Prime implicants of M^{ok}
c_g	QoF upper bound
c_m	Cost introduced by message m

Table 8.3: Overview of the notation introduced in Chapter 8.

Part IV

Conclusion and Outlook

CHAPTER 9

Conclusion

Contents

9.1 Summary

In this dissertation, the use case of a smart city has been considered, which is becoming increasingly relevant due to technological advancements, especially with respect to wireless technologies [LXC+19]. Against this background, a set of arising challenges has been addressed, concerning distributed systems that serve as the technological backbone of a smart city, as introduced in Chapter 3, and embedded systems, as introduced in Chapter 6, that are not considered as part of a distributed system but can connect to it in order to make use of the hardware infrastructure. All contemplated challenges are related to the notion of quality of service: Mechanisms have been proposed that either allow to guarantee the satisfaction of applications' QoS requirements or to enable the system(s) to generally provide more service to (specific types of) executed applications. The areas of an exemplary smart city to which the contemplated challenges correspond as well as the respective contributions made by this dissertation are illustrated in Fig. 9.1.

Concretely, in Part II, the perspective of a distributed system has been adopted, which can be used as an on-demand execution platform for applications originating from distinct actors and traffic participants of a smart city. In this context, Chapter 4 addressed the challenge of providing QoS guarantees in terms of temporal correctness to all applications executed on the system. For this purpose, the concept of QoS contracts has been introduced as well as a system admission process that allows to conclude these between an application and the system. QoS contracts come with

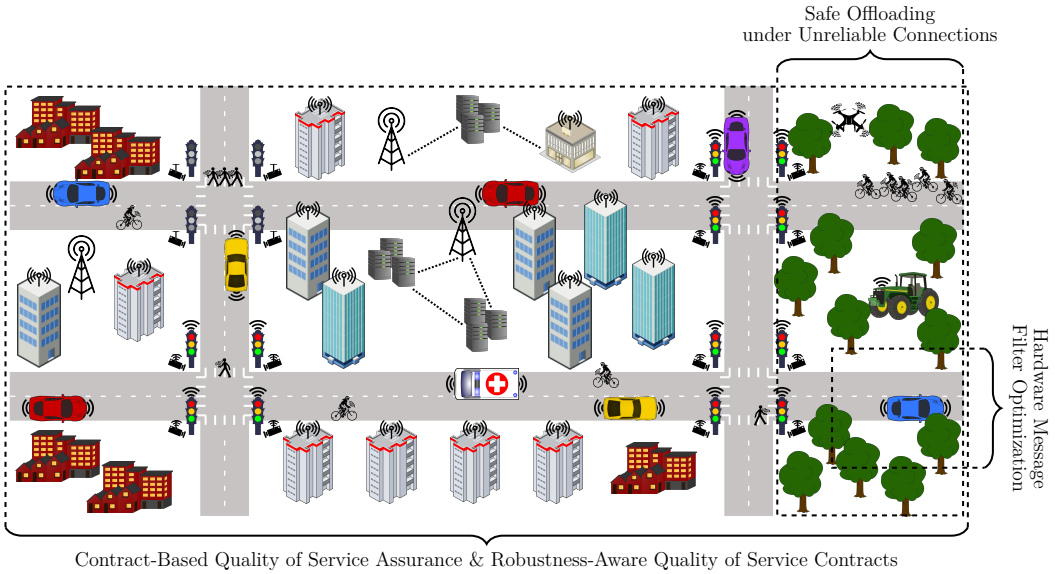


Figure 9.1: Against the background of technological advancements, several challenges arise in the context of a smart city. This dissertation addressed a subset of these related to several aspects of quality of service.

multiple benefits: In the course of the system admission process, distributed response time analyses are used, which are less intensive in terms of computation time than global response-time analyses. By means of simulations, it has been shown that the system admission according to the proposed approach outperforms the related approach of [NSE11] relying on global response-time analyses and that this advantage in computation time increases with a growing system size. Moreover, due to the considered system design, application-level QoS contracts can be constructed as a conjunction instead of a composition of task-level QoS contracts which facilitates the verification of their satisfaction. To detect contract violations, a monitoring-based approach has been introduced, which is based on a set of MITL constraints specifying the expected system behavior. Depending on which constraints are violated, different types of contract violations, including their causes, can be distinguished. This approach has been proven to be effective by means of simulations under error and fault injection.

On this basis, Chapter 5 aimed to reduce the resource waste resulting from resource overreservations made for applications admitted to the system. Since not each application requires its temporal correctness requirement to be satisfied at any point in time, but reasonable results maintaining the system's functionality and safety can also be achieved under a certain amount of temporal correctness violations for some types of applications, applications with additional robustness requirements relaxing the temporal correctness requirement have been considered. For such applications,

the concept of robustness-aware QoS contracts (short: soft QoS contracts) has been proposed that allows to provide temporal correctness guarantees while taking the robustness requirements into consideration. For establishing soft QoS contracts, a set of constraints have been proposed describing the expected characteristics of an admitted application, based on which the system admission process can be performed, for instance, using an SMT solver. By simulations comparing the number of accepted applications under QoS contracts and soft QoS contracts, it has been found that, depending on the simulated scenario, under soft QoS contracts up to 50 % more applications can be accepted than under normal QoS contracts.

In Part III, the focus was shifted towards individual systems, e.g, smart vehicles, facing limitations due to constrained local resources. To activate advanced functionalities such as, e.g., autonomous navigation, regardless, a system can connect to a smart city's distributed system and offload parts of applications for remote execution. Aiming to satisfy the temporal correctness requirements of critical applications and, thus, to ensure the system safety despite the potential unreliability of wireless connections, two recovery protocols have been proposed in Chapter 7. For each protocol, related schedulability tests have been provided that allow to verify offline, i.e., at design time if a system implementing the respective protocol can provide QoS guarantees for critical tasks at any point in time. By means of simulations involving data measured on a real-world robotic system, it has been shown that the reduction of the maximum system utilization payed as the price for an increased safety is acceptable. Furthermore, it has been observed that the duration of the system exhibiting a local fallback execution behavior is not excessive when compared to the overall simulation time. Additionally, in the course of the evaluation, different factors have been identified that have an impact on the amount of time the system exhibits local execution behavior, which can be taken into account for optimization purposes during the system design. Building on these contributions, an extension of the recovery protocols for systems with multiple criticality levels, e.g., for drones monitoring sporting events, has been developed in the context of a bachelor thesis co-supervised by the author of this dissertation.

To reduce the computational overhead introduced to a resource-constrained system by the inspection of received broadcast messages in terms of their relevance, Chapter 8 addressed the optimization of hardware message filters in controller area network. Since a participant of a broadcast bus, e.g., an ECU, receives all messages written to the bus, it must be checked for each incoming message if further message handling is necessary. To reduce this overhead resulting from irrelevant received messages, hardware message filters can be applied that prevent the reception of undesired messages based on their IDs, provided that they are configured accordingly. To optimize such hardware message filter configurations, methods for computing perfect and imperfect hardware message filter configurations have been proposed and their effectiveness has been confirmed by evaluations involving real-world benchmarks. The evaluation results have indicated that the ID assignment of message sets is crucial in order to compute more effective filter configurations within shorter time

and, depending on the considered scenario, to save hardware components at system design time. Based on the findings presented in this dissertation, an approach for the optimization of ID assignments in existing systems has been published in [SSC20] to which the author of this dissertation has contributed.

9.2 Open Problems and Future Research Directions

Beyond the challenges addressed in this dissertation, manifold other open problems exist that will continue to arise in consequence of technological advances. With respect to distributed systems as considered in Part II, further challenges emerge directly from the concepts of QoS contracts. Although soft QoS contracts already refine the notion of QoS as required by an application and guaranteed by a system, it is desirable to establish a more comprehensive type of QoS guarantees by including further parameters. In addition, it is advisable to extend the concept of QoS contracts also to applications with arbitrary deadlines, i.e., applications having an end-to-end deadline longer than their period. This can, for instance, be the case for applications processing the input of sensors that sample with a high frequency.

Taking up the proposal made in Chapter 5, it can be meaningful to introduce quota for different classes of applications executed on a system. To this end, it is necessary to define such classes of applications in a fine-grained way, supported by case studies carried out on real-world systems. One possible criterion for defining such classes is the amount of time for which an application is executed on the distributed system. In fact, after the admission of an application to the system, it is not necessarily executed in perpetuity, for instance, if originating from a smart vehicle that only passes through the smart city. Information about how long an application is likely to utilize the system infrastructure can also be taken into account during the system admission process, i.e., with respect to the reservation of resources.

To obtain such as well as other information, it is necessary to perform more extensive monitoring, i.e., beyond the monitors based on the MITL constraints provided in Chapter 4, and to make predictions based on the retrieved data. In general, the information obtained through monitoring can be exploited to a larger extent; for instance, it is meaningful to develop system reconfiguration strategies that can be directly applied as countermeasures to the events and system states detected by the monitors.

When offloading computation from an individual system, as considered in Part III, to a distributed system making use of recovery protocols as proposed in Chapter 7, it is recommendable to take the advancements of new cellular networking technologies into account, such as the fifth generation of mobile communication networking (5G) [Eur19], where transmission times can be guaranteed depending, among others, on the channel conditions [NSO+22]. By modifying the protocols according to such advancements, it may be possible to reduce the waste of resources and to provide less pessimistic schedulability tests. Furthermore, with respect to individual systems, a primary need

for action exists in the field of security, especially regarding the interplay of security and safety. Although TSN is becoming the new de facto technological standard for in-vehicle communication [SZ18], CAN is still the prevailing communication resource encountered in the majority of vehicles. Therefore, it should be investigated how the contributions made in Chapter 8 with respect to hardware message filtering can be beneficial also in the context of security, e.g., in combination with related works.

List of Figures

List of Tables

Bibliography

[AA22] S. Ahmed and J. H. Anderson. "Exact Response-Time Bounds of Periodic DAG Tasks under Server-Based Global Scheduling". In: *2022 IEEE Real-Time Systems Symposium (RTSS)*. 2022, pp. 447–459. DOI: 10.1109/RTSS55097.2022.00045 (Cited on page 32).

[AFH96] R. Alur, T. Feder, and T. A. Henzinger. "The Benefits of Relaxing Punctuality". In: *J. ACM* 43.1 (01/1996), pp. 116–146. ISSN: 0004-5411. DOI: 10.1145/227595.227602 (Cited on pages 35 sq.).

[ANN+22] B. Akesson, M. Nasri, G. Nelissen, S. Altmeyer, and R. I. Davis. "A Comprehensive Survey of Industry Practice in Real-Time Systems". In: *Real-Time Syst.* 58.3 (09/2022), pp. 358–398. ISSN: 0922-6443. DOI: 10.1007/s11241-021-09376-1 (Cited on page 40).

[ARS18] A. Adiththan, S. Ramesh, and S. Samii. "Cloud-assisted Control of Ground Vehicles Using Adaptive Computation Offloading Techniques". In: *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2018, pp. 589–592. DOI: 10.23919/DATE.2018.8342076 (Cited on page 80).

[Aut94] S. of Automotive Engineers. *Class C Application Requirement Considerations*. Standard. SAE J2056/1, 1994 (Cited on page 108).

[Bar14] S. Baruah. "Improved Multiprocessor Global Schedulability Analysis of Sporadic DAG Task Systems". In: *2014 26th Euromicro Conference on Real-Time Systems*. 2014, pp. 97–105. DOI: 10.1109/ECRTS.2014.22 (Cited on page 31).

[Bar15a] S. Baruah. "Federated Scheduling of Sporadic DAG Task Systems". In: *2015 IEEE International Parallel and Distributed Processing Symposium*. 2015, pp. 179–186. DOI: 10.1109/IPDPS.2015.33 (Cited on page 32).

[Bar15b] S. Baruah. "The Federated Scheduling of Constrained-Deadline Sporadic DAG task systems". In: *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2015, pp. 1323–1328. DOI: 10.7873/DATE.2015.0200 (Cited on page 32).

[Bar15c] S. Baruah. "The Federated Scheduling of Systems of Conditional Sporadic DAG Tasks". In: *Proceedings of the 12th International Conference on Embedded Software*. EMSOFT '15. Amsterdam, The Netherlands: IEEE Press, 2015, pp. 1–10. ISBN: 9781467380799 (Cited on page 32).

[Bar16] S. Baruah. "The Federated Scheduling of Systems of Mixed-Criticality Sporadic DAG Tasks". In: *2016 IEEE Real-Time Systems Symposium (RTSS)*. 2016, pp. 227–236. DOI: 10.1109/RTSS.2016.030 (Cited on page 32).

[BB05] E. Bini and G. C. Buttazzo. "Measuring the Performance of Schedulability Tests". In: *Real-Time Systems* 30.1-2 (2005), pp. 129–154 (Cited on page 91).

[BBM+12] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese. "A Generalized Parallel Task Model for Recurrent Real-Time Processes". In: *2012 IEEE 33rd Real-Time Systems Symposium*. 2012, pp. 63–72. DOI: 10.1109/RTSS.2012.59 (Cited on page 31).

[BBM15] S. Baruah, V. Bonifaci, and A. Marchetti-Spaccamela. "The Global EDF Scheduling of Systems of Conditional Sporadic DAG Tasks". In: *Proceedings of the 2015 27th Euromicro Conference on Real-Time Systems*. ECRTS '15. USA: IEEE Computer Society, 2015, pp. 222–231. ISBN: 9781467375702. DOI: 10.1109/ECRTS.2015.27 (Cited on page 32).

[BCH+16] G. v. d. Brüggen, K.-H. Chen, W.-H. Huang, and J.-J. Chen. "Systems with Dynamic Real-Time Guarantees in Uncertain and Faulty Execution Environments". In: *Real-Time Systems Symposium (RTSS)*. Porto, Portugal, 11/2016 (Cited on page 10).

[BCM+20] S. Ben-Amor, L. Cucu-Grosjean, M. Mezouak, and Y. Sorel. "Probabilistic Schedulability Analysis for Precedence Constrained Tasks on Partitioned Multicore". In: *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. Vol. 1. 2020, pp. 345–352. DOI: 10.1109/ETFA46521.2020.9211973 (Cited on page 55).

[BCN+18] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. A. Henzinger, and K. G. Larsen. "Contracts for System Design". In: *Foundations and Trends® in Electronic Design Automation* 12.2-3 (2018), pp. 124–400. ISSN: 1551-3939. DOI: 10.1561/1000000053 (Cited on page 26).

[BD18] A. Burns and R. I. Davis. "A Survey of Research into Mixed Criticality Systems". In: vol. 50. 6. 2018, 82:1–82:37. DOI: 10.1145/3131347 (Cited on page 14).

[BHC+16] G. von der Brüggen, W.-H. Huang, J.-J. Chen, and C. Liu. "Uniprocessor Scheduling Strategies for Self-Suspending Task Systems". In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems (RTNS)*. 2016, pp. 119–128 (Cited on page 72).

[BHC17] G. von der Brüggen, W.-H. Huang, and J.-J. 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, pp. 1–9 (Cited on page 72).

[BMS+13] V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, and A. Wiese. "Feasibility Analysis in the Sporadic DAG Task Model". In: *2013 25th Euromicro Conference on Real-Time Systems*. 2013, pp. 225–233. DOI: 10.1109/ECRTS.2013.32 (Cited on page 31).

[BSC18] G. von der Brüggen, L. Schönberger, and J.-J. Chen. "Do Nothing, But Carefully: Fault Tolerance with Timing Guarantees for Multiprocessor Systems Devoid of Online Adaptation". In: *2018 IEEE 23rd Pacific Rim International Symposium on Dependable Computing (PRDC)*. 2018, pp. 1–10. DOI: 10.1109/PRDC.2018.00010 (Cited on pages vi, 10).

[BT18] C. Barrett and C. Tinelli. "Satisfiability Modulo Theories". In: *Handbook of Model Checking*. Ed. by E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem. Cham: Springer International Publishing, 2018, pp. 305–343. ISBN: 978-3-319-10575-8. DOI: 10.1007/978-3-319-10575-8_11 (Cited on pages 41, 61, 105).

[But11] G. C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications, Third Edition*. Vol. 24. Real-Time Systems Series. Springer, 2011. ISBN: 978-1-4614-0675-4. DOI: 10.1007/978-1-4614-0676-1 (Cited on pages 9, 11, 13).

[CBC+16] K. Chen, B. Bönninghoff, J. Chen, and P. Marwedel. "Compensate or Ignore? Meeting Control Robustness Requirements through Adaptive Soft-error Handling". In: *Proceedings of the 17th ACM SIGPLAN/SIGBED Conference on Languages, Compilers, Tools, and Theory for Embedded Systems, LCTES 2016, Santa Barbara, CA, USA, June 13 - 14, 2016*. Ed. by T. Kuo and D. B. Whalley. ACM, 2016, pp. 82–91. DOI: 10.1145/2907950.2907952 (Cited on pages 13, 56).

[CBH+17] J.-J. Chen, G. von der Brüggen, W.-H. Huang, and C. 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, pp. 1–10. DOI: 10.1109/RTCSA.2017.8046321 (Cited on page 72).

[CBN+18] D. Casini, A. Biondi, G. Nelissen, and G. Buttazzo. "Partitioned Fixed-Priority Scheduling of Parallel Tasks Without Preemptions". In: *2018 IEEE Real-Time Systems Symposium (RTSS)*. 2018, pp. 421–433. DOI: 10.1109/RTSS.2018.00056 (Cited on page 33).

[Che16] J.-J. Chen. "Computational Complexity and Speedup Factors Analyses for Self-Suspending Tasks". In: *Real-Time Systems Symposium (RTSS)*. 2016, pp. 327–338. DOI: 10.1109/RTSS.2016.039 (Cited on page 32).

[Che19] K. Chen. "Optimization and Analysis for Dependable Application Software on Unreliable Hardware Platforms". PhD thesis. Technical University of Dortmund, Germany, 2019. URL: https://hdl.handle.net/2003/38110 (Cited on page 14).

[CNH+18] J.-J. Chen, G. Nelissen, W.-H. Huang, M. Yang, B. Brandenburg, K. Bletsas, C. Liu, P. Richard, F. Ridouard, N. Audsley, R. Rajkumar, D. de Niz, and G. von der Brüggen. "Many Suspensions, Many Problems: A Review of Self-Suspending Tasks in Real-Time Systems". In: *Real-Time Systems* (09/2018). ISSN: 1573-1383. DOI: 10.1007/s11241-018-9316-9 (Cited on pages 72, 82).

[CVC18] K.-H. Chen, G. Von Der Brüggen, and J.-J. Chen. "Analysis of Deadline Miss Rates for Uniprocessor Fixed-Priority Scheduling". In: *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. 2018, pp. 168–178. DOI: 10.1109/RTCSA.2018.00028 (Cited on page 91).

[CZG+23] N. Chen, S. Zhao, I. Gray, A. Burns, S. Ji, and W. Chang. "Precise Response Time Analysis for Multiple DAG Tasks with Intra-task Priority Assignment". In: *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2023, pp. 174–184. DOI: 10.1109/RTAS58335.2023.00021 (Cited on page 32).

[DAT+] J. Diemer, P. Axer, D. Thiele, and J. Schlatow. *pyCPA*. https://pycpa.readthedocs.io/en/latest (Cited on pages 41, 63).

[DB11] R. I. Davis and A. Burns. "A Survey of Hard Real-Time Scheduling for Multiprocessor Systems". In: *ACM Comput. Surv.* 43.4 (10/2011). ISSN: 0360-0300. DOI: 10.1145/1978802.1978814 (Cited on page 28).

[DBB+07] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien. "Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised". In: *Real-Time Systems* 35.3 (04/2007), pp. 239–272. ISSN: 1573-1383. DOI: 10.1007/s11241-007-9012-7 (Cited on page 113).

[DC19] R. I. Davis and L. Cucu-Grosjean. "A Survey of Probabilistic Timing Analysis Techniques for Real-Time Systems". In: *Leibniz Trans. Embed. Syst.* 6.1 (2019), 03:1–03:60. DOI: 10.4230/LITES-v006-i001-a003 (Cited on page 12).

[DZB08] R. I. Davis, A. Zabos, and A. Burns. "Efficient Exact Schedulability Tests for Fixed Priority Real-Time Systems". In: *IEEE Transactions on Computers* 57.9 (2008), pp. 1261–1276 (Cited on page 91).

[EAG18] R. Ernst, L. Ahrendts, and K. B. Gemlau. "System Level LET: Mastering Cause-Effect Chains in Distributed Systems". In: *IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society, Washington, DC, USA, October 21-23, 2018*. IEEE, 2018, pp. 4084–4089. DOI: 10.1109/IECON.2018.8591550 (Cited on page 22).

[EE17] I. of Electrical and E. Engineers. *802.11p-2010 - IEEE Standard for Information technology – Local and Metropolitan Area Networks – Specific Requirements– Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 6: Wireless Access in Vehicular Environments*. https://standards.ieee.org/standard/802_11p-2010.html. 2017 (Cited on page 78).

[ENN+15] A. Esper, G. Nelissen, V. Nélis, and E. Tovar. "How Realistic is the Mixed-Criticality Real-Time System Model?" In: *Proceedings of the 23rd International Conference on Real Time and Networks Systems*. RTNS '15. Lille, France: Association for Computing Machinery, 2015, pp. 139–148. ISBN: 9781450335911. DOI: 10.1145/2834848.2834869 (Cited on page 80).

[Eps] Epsilon. *Epsilon. Small Electric Passenger Vehicle with Maximized Safety and Integrating a Lightweight Oriented Novel Body Architecture*. https://www.2zeroemission.eu/research-project/epsilon/ (Cited on page 108).

[Ern05] R. Ernst. "System Level Performance Analysis – the SymTA/S Approach". English. In: *IEE Proceedings - Computers and Digital Techniques* 152 (2 03/2005), 148–166(18). ISSN: 1350-2387 (Cited on page 35).

[Eur19] European Telecommunications Standards Institute. *ETSI TR 121 915 V15.0.0 (2019-10). Digital Cellular Telecommunications System (Phase 2+) (GSM); Universal Mobile Telecommunications System (UMTS); LTE; 5G; Release Description; Release 15 (3GPP TR 21.915 Version 15.0.0 Release 15)*. 1019 (Cited on page 120).

[FNN+16] J. Fonseca, G. Nelissen, V. Nelis, and L. M. Pinho. "Response Time Analysis of Sporadic DAG Tasks under Partitioned Scheduling". In: *2016 11th IEEE Symposium on Industrial Embedded Systems (SIES)*. 2016, pp. 1–10. DOI: 10.1109/SIES.2016.7509443 (Cited on page 34).

[FNN17] J. Fonseca, G. Nelissen, and V. Nélis. "Improved Response Time Analysis of Sporadic DAG Tasks for Global FP Scheduling". In: *Proceedings of the 25th International Conference on Real-Time Networks and Systems*. RTNS '17. Grenoble, France: Association for Computing Machinery, 2017, pp. 28–37. ISBN: 9781450352864. DOI: 10.1145/3139258.3139288 (Cited on page 32).

[For] Formula Student Germany. *Formula Student Germany. International Design Competition.* https://www.formulastudent.de/fsg/ (Cited on page 108).

[Gaz] Gazebo. *Gazebo. Robot Simulation Made Easy.* http://gazebosim.org/ (Cited on page 92).

[GET] GET racing Dortmund e. V. *GET racing Dortmund e. V.* https://www.get-racing.de/ (Cited on page 108).

[HAA+19] E. E. Haber, H. A. Alameddine, C. Assi, and S. Sharafeddine. "A Reliability-aware Computation Offloading Solution via UAV-mounted Cloudlets". In: *2019 IEEE 8th International Conference on Cloud Networking (CloudNet)*. 2019, pp. 1–6. DOI: 10.1109/CloudNet47604.2019.9064038 (Cited on page 79).

[HAE17] R. Hofmann, L. Ahrendts, and R. Ernst. "CPA: Compositional Performance Analysis". In: *Handbook of Hardware/Software Codesign*. Ed. by S. Ha and J. Teich. Dordrecht: Springer Netherlands, 2017, pp. 721–751. ISBN: 978-94-017-7267-9. DOI: 10.1007/978-94-017-7267-9_24 (Cited on page 41).

[HCC+21] Z. Houssam-Eddine, N. Capodieci, R. Cavicchioli, G. Lipari, and M. Bertogna. "The HPC-DAG Task Model for Heterogeneous Real-Time Systems". In: *IEEE Transactions on Computers* 70.10 (2021), pp. 1747–1761. DOI: 10.1109/TC.2020.3023169 (Cited on page 34).

[HCL15] W.-H. Huang, J. Chen, and C. Liu. "PASS: Priority Assignment of Real-time Tasks with Dynamic Suspending Behavior under Fixed-priority Scheduling". In: *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. 06/2015, pp. 1–6. DOI: 10.1145/2744769.2744891 (Cited on page 72).

[HHJ+06] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. "System-level Performance Analysis - The SymTA/S Approach". In: 01/2006, pp. 29–74. ISBN: 9780863415524. DOI: 10.1049/PBCS018E_ch2 (Cited on pages 41 sq.).

[HK23] T. Han and K. Kim. "Minimizing Probabilistic End-to-end Latencies of Autonomous Driving Systems". In: *29th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2023, pp. 1–13 (Cited on page 55).

[HKS+19] H. Habibzadeh, C. Kaptan, T. Soyata, B. Kantarci, and A. Boukerche. "Smart City System Design: A Comprehensive Study of the Application and Data Planes". In: *ACM Comput. Surv.* 52.2 (05/2019). ISSN: 0360-0300. DOI: 10.1145/3309545 (Cited on page 3).

[HR95] M. Hamdaoui and P. Ramanathan. "A Dynamic Priority Assignment Technique for Streams with (m,k)-firm Deadlines". In: *IEEE Transactions on Computers* 44.12 (1995), pp. 1443–1451. DOI: 10.1109/12.477249 (Cited on pages 13, 56).

[HSG+20] A. Hamann, S. Saidi, D. Ginthoer, C. Wietfeld, and D. Ziegenbein. "Building End-to-End IoT Applications with QoS Guarantees". In: *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 2020, pp. 1–6. DOI: 10.1109/DAC18072.2020.9218564 (Cited on page 4).

[HZL+21] M. Han, T. Zhang, Y. Lin, and Q. Deng. "Federated Scheduling for Typed DAG Tasks Scheduling Analysis on Heterogeneous Multi-Cores". In: *Journal of Systems Architecture* 112 (2021), p. 101870. ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2020.101870 (Cited on page 33).

[JE12] P. A. Jonas Diemer and R. Ernst. "Compositional Performance Analysis in Python with pyCPA". In: *3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*. 2012 (Cited on pages 35, 41).

[JGL+17] X. Jiang, N. Guan, X. Long, and W. Yi. "Semi-Federated Scheduling of Parallel Real-Time Tasks on Multiprocessors". In: *2017 IEEE Real-Time Systems Symposium (RTSS)*. 2017, pp. 80–91. DOI: 10.1109/RTSS.2017.00015 (Cited on page 33).

[JLG+16] X. Jiang, X. Long, N. Guan, and H. Wan. "On the Decomposition-Based Global EDF Scheduling of Parallel Real-Time Tasks". In: *2016 IEEE Real-Time Systems Symposium (RTSS)*. 2016, pp. 237–246. DOI: 10.1109/RTSS.2016.031 (Cited on page 34).

[KRI09] S. Kato, R. Rajkumar, and Y. Ishikawa. "A Loadable Real-time Scheduler Suite for Multicore Platforms". In: *Technical Report CMU-ECE-TR09-12* (2009) (Cited on page 92).

[KS95] G. Koren and D. Shasha. "Skip-Over: Algorithms and Complexity for Overloaded Systems that Allow Skips". In: *Proceedings 16th IEEE Real-Time Systems Symposium*. 1995, pp. 110–117. DOI: 10.1109/REAL.1995.495201 (Cited on page 56).

[KSL+19] H. Kotthaus, L. Schönberger, A. Lang, J.-J. Chen, and P. Marwedel. "Can Flexible Multi-Core Scheduling Help to Execute Machine Learning Algorithms Resource-Efficiently?" In: *Proceedings of the 22nd International Workshop on Software and Compilers for Embedded Systems*. SCOPES '19. Sankt Goar, Germany: Association for Computing Machinery, 2019, pp. 59–62. ISBN: 9781450367622. DOI: 10.1145/3323439.3323986 (Cited on page vi).

[KYT+20] L. U. Khan, I. Yaqoob, N. H. Tran, S. M. A. Kazmi, T. N. Dang, and C. S. Hong. "Edge-Computing-Enabled Smart Cities: A Comprehensive Survey". In: *IEEE Internet of Things Journal* 7.10 (2020), pp. 10200–10232. DOI: 10.1109/JIOT.2020.2987070 (Cited on page 3).

[KZH15] S. Kramer, D. Ziegenbein, and A. Hamann. "Real World Automotive Benchmarks for Free". In: *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*. Vol. 130. 2015 (Cited on page 43).

[LAL+13] J. Li, K. Agrawal, C. Lu, and C. Gill. "Outstanding Paper Award: Analysis of Global EDF for Parallel Tasks". In: *2013 25th Euromicro Conference on Real-Time Systems*. 2013, pp. 3–13. DOI: 10.1109/ECRTS.2013.12 (Cited on page 32).

[LC14] C. Liu and J. Chen. "Bursty-Interference Analysis Techniques for Analyzing Complex Real-Time Task Models". In: *Real-Time Systems Symposium (RTSS)*. 2014, pp. 173–183 (Cited on page 72).

[LCA+14] J. Li, J. J. Chen, K. Agrawal, C. Lu, C. Gill, and A. Saifullah. "Analysis of Federated and Global Scheduling for Parallel Real-Time Tasks". In: *2014 26th Euromicro Conference on Real-Time Systems*. 2014, pp. 85–96. DOI: 10.1109/ECRTS.2014.23 (Cited on page 32).

[LCH+22] H. Lee, Y. Choi, T. Han, and K. Kim. "Probabilistically Guaranteeing End-to-End Latencies in Autonomous Vehicle Computing Systems". In: *IEEE Transactions on Computers* 71.12 (2022), pp. 3361–3374. DOI: 10.1109/TC.2022.3152105 (Cited on page 55).

[LCP+20] H. Liu, L. Cao, T. Pei, Q. Deng, and J. Zhu. "A Fast Algorithm for Energy-Saving Offloading With Reliability and Latency Requirements in Multi-Access Edge Computing". In: *IEEE Access* 8 (2020), pp. 151–161. DOI: 10.1109/ACCESS.2019.2961453 (Cited on page 79).

[Leh90] J. Lehoczky. "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines". In: *[1990] Proceedings 11th Real-Time Systems Symposium*. 1990, pp. 201–209. DOI: 10.1109/REAL.1990.128748 (Cited on page 41).

[LFA+17] J. Li, D. Ferry, S. Ahuja, K. Agrawal, C. Gill, and C. Lu. "Mixed-Criticality Federated Scheduling for Parallel Real-Time Tasks". In: *Real-Time Syst.* 53.5 (09/2017), pp. 760–811. ISSN: 0922-6443. DOI: 10.1007/s11241-017-9281-8 (Cited on page 32).

[LJP17] V. Lesi, I. Jovanov, and M. Pajic. "Network Scheduling for Secure Cyber-Physical Systems". In: *IEEE Real-Time Systems Symposium (RTSS)*. 2017 (Cited on pages 108 sq.).

[LKA04] J. Leung, L. Kelly, and J. H. Anderson. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. USA: CRC Press, Inc., 2004. ISBN: 1584883979 (Cited on page 87).

[LKR10] K. Lakshmanan, S. Kato, and R. Rajkumar. "Scheduling Parallel Real-Time Tasks on Multi-Core Processors". In: *2010 31st IEEE Real-Time Systems Symposium*. 2010, pp. 259–268. DOI: 10.1109/RTSS.2010.42 (Cited on page 31).

[LL73] C. L. Liu and J. W. Layland. "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment". In: *J. ACM* 20.1 (01/1973), pp. 46–61. ISSN: 0004-5411. DOI: 10.1145/321738.321743 (Cited on pages 11, 89).

[LSU+23] C.-C. Lin, J. Shi, N. Ueter, M. Günzel, J. Reineke, and J.-J. Chen. "Type-Aware Federated Scheduling for Typed DAG Tasks on Heterogeneous Multicore Platforms". In: *IEEE Transactions on Computers* 72.5 (2023), pp. 1286–1300. DOI: 10.1109/TC.2022.3202748 (Cited on page 33).

[LXC+19] K. Liu, X. Xu, M. Chen, B. Liu, L. Wu, and V. C. S. Lee. "A Hierarchical Architecture for the Future Internet of Vehicles". In: *IEEE Communications Magazine* 57.7 (2019), pp. 41–47. DOI: 10.1109/MCOM.2019.1800772 (Cited on pages 3 sq., 117).

[MA21] F. Machida and E. Andrade. "PA-Offload: Performability-Aware Adaptive Fog Offloading for Drone Image Processing". In: *2021 IEEE 5th International Conference on Fog and Edge Computing (ICFEC)*. 2021, pp. 66–73. DOI: 10.1109/ICFEC51620.2021.00017 (Cited on page 79).

[MB08] L. de Moura and N. Bjørner. "Z3: An Efficient SMT Solver". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by C. R. Ramakrishnan and J. Rehof. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–340. ISBN: 978-3-540-78800-3 (Cited on pages 41, 63, 109).

[MEB19] V. Millnert, J. Eker, and E. Bini. "End-To-End Deadlines over Dynamic Topologies". In: *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*. Ed. by S. Quinton. Vol. 133. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 10:1–10:22. ISBN: 978-3-95977-110-8. DOI: 10.4230/LIPIcs.ECRTS.2019.10 (Cited on page 25).

[MN04] O. Maler and D. Nickovic. "Monitoring Temporal Properties of Continuous Signals". In: *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Ed. by Y. Lakhnech and S. Yovine. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 152–166. ISBN: 978-3-540-30206-3 (Cited on pages 36 sq., 42).

[NBG+12] G. Nelissen, V. Berten, J. Goossens, and D. Milojevic. "Techniques Optimizing the Number of Processors to Schedule Multi-threaded Tasks". In: *2012 24th Euromicro Conference on Real-Time Systems*. 2012, pp. 321–330. DOI: 10.1109/ECRTS.2012.37 (Cited on page 33).

[Nel53] R. J. Nelson. "W. V. Quine. The Problem of Simplifying Truth Functions. The American mathematical monthly, vol. 59 (1952), pp. 521–531. (Offprint 1952, on sale by the Mathematical Association of America.)" In: *Journal of Symbolic Logic* 18.3 (09/1953), pp. 280–282. DOI: 10.2307/2267441 (Cited on page 106).

[NNB19] M. Nasri, G. Nelissen, and B. B. Brandenburg. "Response-Time Analysis of Limited-Preemptive Parallel DAG Tasks under Global Scheduling". In: *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*. Ed. by S. Quinton. Vol. 133. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019, 21:1–21:23. ISBN: 978-3-95977-110-8. DOI: 10.4230/LIPIcs.ECRTS.2019.21 (Cited on page 32).

[NQ06] L. Niu and G. Quan. "Energy Minimization for Real-time Systems with (m,k)-guarantee". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 14.7 (2006), pp. 717–729. DOI: 10.1109/TVLSI.2006.878337 (Cited on page 56).

[NSE11] M. Neukirchner, S. Stein, and R. Ernst. "The EPOC Architecture—Enabling Evolution Under Hard Constraints". In: *Organic Computing — A Paradigm Shift for Complex Systems*. Ed. by C. Müller-Schloer, H. Schmeck, and T. Ungerer. Basel: Springer Basel, 2011, pp. 399–412. ISBN: 978-3-0348-0130-0. DOI: 10.1007/978-3-0348-0130-0_26 (Cited on pages 25, 42, 50, 118).

[NSO+22] A. Nota, S. Saidi, D. Overbeck, F. Kurtz, and C. Wietfeld. "Context-based Latency Guarantees Considering Channel Degradation in 5G Network Slicing". In: *2022 IEEE Real-Time Systems Symposium (RTSS)*. 2022, pp. 253–265. DOI: 10.1109/RTSS55097.2022.00030 (Cited on page 120).

[OHC+23] P. Oza, N. Hudson, T. Chantem, and H. Khamfroush. "Deadline-Aware Task Offloading for Vehicular Edge Computing Networks Using Traffic Lights Data". In: *ACM Trans. Embed. Comput. Syst.* (04/2023). Just Accepted. ISSN: 1539-9087. DOI: 10.1145/3594541 (Cited on page 80).

[PDB16] F. Pölzlbauer, R. I. Davis, and I. Bate. "A Practical Message ID Assignment Policy for Controller Area Network That Maximizes Extensibility". In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. RTNS '16. Brest, France: Association for Computing Machinery, 2016, pp. 45–54. ISBN: 9781450347877. DOI: 10.1145/2997465.2997484 (Cited on page 113).

[PDB17] F. Pölzlbauer, R. I. Davis, and I. Bate. "Analysis and Optimization of Message Acceptance Filter Configurations for Controller Area Network (CAN)". In: *Int. Conf. on Real-Time Networks and Systems (RTNS)*. 2017 (Cited on pages 104, 108 sqq., 113).

[PS19] T. Park and K. G. Shin. "Optimal Priority Assignment for Scheduling Mixed CAN and CAN-FD Frames". In: *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2019, pp. 192–203. DOI: 10.1109/RTAS.2019.00024 (Cited on page 113).

[PSE21] J. Peeck, J. Schlatow, and R. Ernst. "Online Latency Monitoring of Time-sensitive Event Chains in Safety-critical Applications". In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2021, pp. 539–542. DOI: 10.23919/DATE51398.2021.9474109 (Cited on page 24).

[PVS18] R. Pathan, P. Voudouris, and P. Stenström. "Scheduling Parallel Real-Time Recurrent Tasks on Multicore Platforms". In: *IEEE Transactions on Parallel and Distributed Systems* 29.4 (2018), pp. 915–928. DOI: 10.1109/TPDS.2017.2777449 (Cited on page 32).

[QGM14] M. Qamhieh, L. George, and S. Midonnet. "A Stretching Algorithm for Parallel Real-time DAG Tasks on Multiprocessor Systems". In: *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*. RTNS '14. Versaille, France: Association for Computing Machinery, 2014, pp. 13–22. ISBN: 9781450327275. DOI: 10.1145/2659787.2659818 (Cited on page 34).

[QH00] G. Quan and X. Hu. "Enhanced Fixed-priority Scheduling with (m,k)-firm Guarantee". In: *Proceedings 21st IEEE Real-Time Systems Symposium*. 2000, pp. 79–88. DOI: 10.1109/REAL.2000.895998 (Cited on page 56).

[QWS+22] H. Qian, Q. Wen, L. Sun, J. Gu, Q. Niu, and Z. Tang. "RobustScaler: QoS-Aware Autoscaling for Complex Workloads". In: *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. Los Alamitos, CA, USA: IEEE Computer Society, 05/2022, pp. 2762–2775. DOI: 10.1109/ICDE53745.2022.00252 (Cited on page 55).

[RBH+08] S. Rosario, A. Benveniste, S. Haar, and C. Jard. "Probabilistic QoS and Soft Contracts for Transaction-Based Web Services Orchestrations". In: *IEEE Transactions on Services Computing* 1.4 (2008), pp. 187–200. DOI: 10.1109/TSC.2008.17 (Cited on page 55).

[Roba] Robot Operating System. *Robot Operating System (ROS)*. https://www.ros.org/ (Cited on page 92).

[Robb] Robotnik Automation S.L. *Mobile Robot RB-1 Base*. https://www.robotnik.eu/mobile-robots/rb-1-base-2/ (Cited on page 92).

[Rob12] Robert Bosch GmbH. *CAN with Flexible Data-Rate Specification Version 1.0*. 2012 (Cited on page 113).

[Rob91] Robert Bosch GmbH. *Controller Area Network Specification 2.0*. 1991 (Cited on pages 102 sq.).

[SAL+11] A. Saifullah, K. Agrawal, C. Lu, and C. Gill. "Multi-Core Real-Time Scheduling for Generalized Parallel Task Models". In: *2011 IEEE 32nd Real-Time Systems Symposium*. 2011, pp. 217–226. DOI: 10.1109/RTSS.2011.27 (Cited on pages 31, 33).

[SBC+20] L. Schönberger, G. von der Brüggen, K.-H. Chen, B. Sliwa, H. Youssef, A. K. R. Venkatapathy, C. Wietfeld, M. ten Hompel, and J.-J. Chen. "Offloading Safety- and Mission-Critical Tasks via Unreliable Connections". In: *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*. Ed. by M. Völp. Vol. 165. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 18:1–18:22. ISBN: 978-3-95977-152-8. DOI: 10.4230/LIPIcs.ECRTS.2020.18 (Cited on pages v, 6 sq., 71, 77).

[SBS+19] L. Schönberger, G. von der Brüggen, H. Schirmeier, and J.-J. Chen. "Design Optimization for Hardware-Based Message Filters in Broadcast Buses". In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2019, pp. 606–609. DOI: 10.23919/DATE.2019.8714793 (Cited on pages v, 6 sq., 101, 113).

[SFL+19] B. Sliwa, R. Falkenberg, T. Liebig, N. Piatkowski, and C. Wietfeld. "Boosting Vehicle-to-cloud Communication by Machine Learning-enabled Context Prediction". In: *IEEE Transactions on Intelligent Transportation Systems* (07/2019) (Cited on page 78).

[SFS+22] C. Shushan, X. Feng, H. Shujuan, Z. Wenjuan, H. Xingxing, and L. Tiansen. "Worst-Case Response Time Analysis of Multitype DAG Tasks Based on Reconstruction". In: *IEEE Access* 10 (2022), pp. 93140–93154. DOI: 10.1109/ACCESS.2022.3203590 (Cited on page 32).

[SGJ+18] J. Sun, N. Guan, X. Jiang, S. Chang, Z. Guo, Q. Deng, and W. Yi. "A Capacity Augmentation Bound for Real-Time Constrained-Deadline Parallel Tasks under GEDF". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.11 (2018), pp. 2200–2211. DOI: 10.1109/TCAD.2018.2857362 (Cited on page 32).

[SGS+22] L. Schönberger, S. Graf, S. Saidi, D. Ziegenbein, and A. Hamann. "Contract-Based Quality-of-Service Assurance in Dynamic Distributed Systems". In: *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2022, pp. 132–135. DOI: 10.23919/DATE54114.2022.9774529 (Cited on pages v, 5, 7, 17, 23).

[SHG+21] L. Schönberger, M. Hamad, J. V. Gomez, S. Steinhorst, and S. Saidi. "Towards an Increased Detection Sensitivity of Time-Delay Attacks on Precision Time Protocol". In: *IEEE Access* 9 (2021), pp. 157398–157410. DOI: 10.1109/ACCESS.2021.3127852 (Cited on page v).

[SHV+18] L. Schönberger, W.-H. Huang, G. Von Der Brüggen, K.-H. Chen, and J.-J. Chen. "Schedulability Analysis and Priority Assignment for Segmented Self-Suspending Tasks". In: *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. 2018, pp. 157–167. DOI: 10.1109/RTCSA.2018.00027 (Cited on pages vi, 72, 82).

[SL03] I. Shin and I. Lee. "Periodic Resource Model for Compositional Real-Time Guarantees". In: *Proceedings of the 24th IEEE International Real-Time Systems Symposium*. RTSS '03. USA: IEEE Computer Society, 2003, p. 2. ISBN: 0769520448 (Cited on page 34).

[SL04] I. Shin and I. Lee. "Compositional Real-Time Scheduling Framework". In: *25th IEEE International Real-Time Systems Symposium*. 2004, pp. 57–67. DOI: 10.1109/REAL.2004.15 (Cited on page 34).

[SL05] I. Shin and I. Lee. "A Compositional Framework for Real-Time Embedded Systems". In: *Service Availability, Second International Service Availability Symposium, ISAS 2005, Berlin, Germany, April 25-26, 2005, Revised Selected Papers*. Ed. by M. Malek, E. Nett, and N. Suri. Vol. 3694. Lecture Notes in Computer Science. Springer, 2005, pp. 137–148. DOI: 10.1007/11560333_12 (Cited on page 34).

[SLJ+19] H. Sun, S. Y. Lee, K. Joo, H. Jin, and D. H. Lee. "Catch ID if You CAN: Dynamic ID Virtualization Mechanism for the Controller Area Network". In: *IEEE Access* 7 (2019), pp. 158237–158249. DOI: 10.1109/ACCESS.2019.2950373 (Cited on page 113).

[SMB+16] M. A. Serrano, A. Melani, M. Bertogna, and E. Quinones. "Response-Time Analysis of DAG Tasks under Fixed Priority Scheduling with Limited Preemptions". In: *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2016, pp. 1066–1071 (Cited on page 32).

[SSA+18] Y. Saito, F. Sato, T. Azumi, S. Kato, and N. Nishio. "ROSCH: Real-Time Scheduling Framework for ROS". In: *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. 08/2018, pp. 52–58. DOI: 10.1109/RTCSA.2018.00015 (Cited on page 92).

[SSC20] S. Schwitalla, L. Schönberger, and J.-J. Chen. "Priority-Preserving Optimization of Status Quo ID-Assignments in Controller Area Network". In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2020, pp. 834–839. DOI: 10.23919/DATE48585.2020.9116565 (Cited on pages v, 113 sq., 120).

[SUC+23] J. Shi, N. Ueter, J.-J. Chen, and K.-H. Chen. "Average Task Execution Time Minimization under (m, k) Soft Error Constraint". In: *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2023, pp. 1–13. DOI: 10.1109/RTAS58335.2023.00008 (Cited on page 56).

[SZ18] S. Samii and H. Zinner. "Level 5 by Layer 2: Time-Sensitive Networking for Autonomous Vehicles". In: *IEEE Communications Standards Magazine* 2.2 (2018), pp. 62–68. DOI: 10.1109/MCOMSTD.2018.1700079 (Cited on pages 3, 121).

[SZD+22] S. Saidi, D. Ziegenbein, J. V. Deshmukh, and R. Ernst. "Autonomous Systems Design: Charting a New Discipline". In: *IEEE Design & Test* 39.1 (2022), pp. 8–23. DOI: 10.1109/MDAT.2021.3128434 (Cited on page 4).

[UBC+18] N. Ueter, G. von der Brüggen, J.-J. Chen, J. Li, and K. Agrawal. "Reservation-Based Federated Scheduling for Parallel Real-Time Tasks". In: *2018 IEEE Real-Time Systems Symposium (RTSS)*. 2018, pp. 482–494. DOI: 10.1109/RTSS.2018.00061 (Cited on page 33).

[Ves07] S. Vestal. "Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance". In: *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*. 12/2007, pp. 239–243. DOI: 10.1109/RTSS.2007.47 (Cited on page 14).

[WEE+08] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. "The Worst-Case Execution-Time Problem—Overview of Methods and Survey of Tools". In: *ACM Trans. Embed. Comput. Syst.* 7.3 (05/2008). ISSN: 1539-9087. DOI: 10.1145/1347375.1347389 (Cited on page 12).

[WGD14] Y. Wu, Z. Gao, and G. Dai. "Deadline and Activation Time Assignment for Partitioned Real-Time Application on Multiprocessor Reservations". In: *J. Syst. Archit.* 60.3 (03/2014), pp. 247–257. ISSN: 1383-7621. DOI: 10.1016/j.sysarc.2013.11.011 (Cited on page 33).

[WJG+19] K. Wang, X. Jiang, N. Guan, D. Liu, W. Liu, and Q. Deng. "Real-Time Scheduling of DAG Tasks with Arbitrary Deadlines". In: *ACM Trans. Des. Autom. Electron. Syst.* 24.6 (10/2019). ISSN: 1084-4309. DOI: 10.1145/3358603 (Cited on pages 32 sq.).

[XYY+19] F. Xu, H. Ye, F. Yang, and C. Zhao. "Software Defined Mission-Critical Wireless Sensor Network: Architecture and Edge Offloading Strategy". In: *IEEE Access* 7 (2019), pp. 10383–10391. DOI: 10.1109/ACCESS.2019.2890854 (Cited on page 80).

[YCC18] M. Yayla, K.-H. Chen, and J.-J. Chen. "Fault Tolerance on Control Applications: Empirical Investigations of Impacts from Incorrect Calculations". In: *2018 4th International Workshop on Emerging Ideas and Trends in the Engineering of Cyber-Physical Systems (EITEC)*. 2018, pp. 17–24. DOI: 10.1109/EITEC.2018.00008 (Cited on page 13).

[ZZL+19] K. Zhang, Y. Zhu, S. Leng, Y. He, S. Maharjan, and Y. Zhang. "Deep Learning Empowered Task Offloading for Mobile Edge Computing in Urban Informatics". In: *IEEE Internet of Things Journal* 6.5 (2019), pp. 7635–7647. DOI: 10.1109/JIOT.2019.2903191 (Cited on page 80).