Priority Assignment Algorithms for Real-Time Systems

Xuanliang Deng

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Engineering

Haibo Zeng, Chair
Binoy Ravindran
Na Meng
Ryan K. Williams
Chang Woo Min

December 17, 2024
Blacksburg, Virginia

Keywords: Real-Time systems, Priority Assignment, Scheduling, Optimization

Copyright 2025, Xuanliang Deng

Priority Assignment Algorithms for Real-Time Systems

Xuanliang Deng

(ABSTRACT)

Priority Assignment is one of the crucial problems in the scheduling of fixed-priority real-time systems, which requires both timely response and correctness of output under specified timing constraints. As the models and hardware platforms of real-time systems become increasingly more complex, it is necessary to consider various design metrics in addition to the system schedulability when we assign priorities to tasks. There has been a rich set of research studies on the priority assignment algorithms. However, there exist several limitations in the state-of-the-art: 1) the current research focuses on improving existing schedulability analyses but fails to integrate them with priority assignment algorithms which has better solution quality than heuristics; 2) the stochastic nature of task execution time in real operation and the properties of heterogeneous hardware platforms are not fully studied in the priority assignment process; and 3) design metrics other than response time are omitted in the priority assignment. In this dissertation, we seek to address the issues in the following aspects. First, instead of using existing schedulability analysis directly, we leverage the proposed concept of response time estimation range to build a novel priority assignment framework. It can quickly rule out infeasible priority assignments that share common attributes, and be coupled with a more accurate schedulability analysis than Audsley's Optimal Priority Assignment (OPA) with much weaker compatibility conditions. The framework judiciously takes advantage of optimization techniques and heuristics under different task utilization to achieve better overall acceptance ratio of task system. Second, we consider the attributes of tasks, hardware platform (e.g., heterogeneous platform) and design metrics (e.g., reaction time and data age)

of real applications in the process of priority assignment. Current studies assume worst case or certain distribution of tasks' execution times in their response time analysis. However, analysis based on these assumptions are pessimistic and overestimate the response times in some cases. We propose a more general model which does not assume any specific distribution of task execution time and analyze the response time by the convolution method. In addition, we propose stochastic heterogeneous Direct Acyclic Graph (DAG) model to take into account the randomness of execution and heterogeneous hardware platform in real operation. Third, we establish an optimization framework which takes other design metrics (e.g., reaction time, data age), in addition to response time in most studies, as objective and constraints in the optimization process. These design metrics can be important in real applications to guarantee satisfying performance of task system. We demonstrate the effectiveness of our proposed frameworks with several case studies, which have shown that our methods can achieve better system schedulability and/or better run-time.

Priority Assignment Algorithms for Real-Time Systems

Xuanliang Deng

(GENERAL AUDIENCE ABSTRACT)

Real-time systems are crucial in control-centric applications, which require both the correctness and timely response of the output. Such systems have specified timing constraints which the outputs of the systems should meet. Failure to meet timing constraints will result in performance degradation or safety-related issues in extreme cases. Priority assignment is the process to assign priorities to tasks. In general case, tasks with higher priorities will execute ahead of tasks with lower priorities when computing resources are available. Priority assignment has an important role in scheduling, which determines the execution orders of the tasks. Current priority assignment algorithms come across several challenges: 1) existing analyses are directly integrated with heuristic method for priority assignment, which cannot guarantee the optimality of the solution; 2) the attributes of the hardware platform or the stochastic nature of task execution in real applications are not fully considered; 3) more design metrics need to be considered in addition to timely response of the system. In this dissertation, we aim to propose better solutions for the above issues. Experimental evaluation has shown that the proposed frameworks outperform existing priority assignment algorithms in terms of run-time and number of tasks that meet the timing constraints of the real-time systems.

Dedication

This dissertation is dedicated to engineers and designers that work on real-time applications and those interested in the research of priority assignment algorithms.

Acknowledgments

I would like to express my gratitude to my advisor, Dr. Haibo Zeng, for his insightful guidance and expertise. Without his help, many of the work in this dissertation would not have been completed. I would like to thank Dr. Na Meng, Dr. Binoy Ranvindran, Dr. Ryan Williams, Dr. Changwoo Min and Dr. Scot Ransbottom in my PhD committee. They provide valuable feedback on my dissertation and support me when I come across difficulties. I would like to thank all co-authors that I have worked with for projects. Finally, I would like to thank my family and friends for their support, encouragement and caring. I also acknowledge Advanced Research Computing at Virginia Tech for providing the necessary computational resources to conduct the experiments.

Contents

List of Figures xi

List of Tables xiii

1 Introduction 1

1.1 Concepts . . . 2

1.1.1 Task Model . . . 2

1.1.2 Schedulability and analysis . . . 3

1.1.3 Priority Assignment . . . 4

1.2 Existing approaches, Issues and Challenges . . . 4

1.2.1 Global Fixed-Priority Assignment Algorithms . . . 5

1.2.2 Priority Assignment with Constraints Variations . . . 6

1.3 Organization . . . 7

2 Priority Assignment for Global Fixed Priority Scheduling on Multiprocessors 11

2.1 Introduction . . . 11

2.2 Related Work . . . 13

2.3 System Model and Preliminaries . . . 16

2.3.1 Schedulability Analysis . . . 17
2.4 MITER-guided Optimization Algorithm . . . 21
2.4.1 Response Time Dependency . . . 22
2.4.2 Response Time Estimation Range . . . 25
2.4.3 MITER-Guided Framework . . . 32
2.5 Hybrid Algorithm with MITER . . . 37
2.6 Experimental Evaluation . . . 40
2.6.1 First Experiment . . . 41
2.6.2 Second Experiment . . . 50
2.7 Conclusion . . . 54

3 Partitioned Scheduling with Safety-Performance Trade-Offs in Stochastic Conditional DAG Models 56

3.1 Introduction . . . 56
3.2 Related Work . . . 59
3.3 System Model . . . 60
3.3.1 Motivating Applications . . . 60
3.3.2 Assumptions . . . 62
3.3.3 Stochastic Heterogeneous Parallel Conditional DAGs . . . 63
3.3.4 Stochastic Conditional Nodes in DAG Models . . . 66
3.4 Safety-Performance Metric . . . 69

3.4.1 Response time probability distributions . . . 70
3.4.2 Safety-Performance Metric . . . 72
3.5 The LLP Scheduler: Subtask Allocation . . . 75
3.6 The LLP Scheduler: Scheduling . . . 79
3.6.1 Scheduling Algorithm . . . 79
3.6.2 Schedulability analysis . . . 81
3.6.3 Time complexity analysis . . . 82
3.7 Experimental Evaluation . . . 83
3.7.1 Taskset Generation . . . 84
3.7.2 Results Analysis . . . 87
3.8 Conclusion . . . 94

4 Latency-Aware Priority Assignment for Fixed Priority Scheduling in Multi-rate Simulink Models 95

4.1 Introduction . . . 95
4.2 Related Work. . . . 96
4.3 Simulink Model Semantics . . . 98
4.4 Reaction time and Data age of task chain . . . 100
4.4.1 Calculate reaction time and data age for one edge . . . 100
4.4.2 Calculate the upper bound of RXN/DA of task chain . . . 105
4.4.3 Bound of overestimation . . . 107

4.5 The concept of MUDPA 108
4.6 MUDPA-Guided Optimization Framework 111
4.7 Experimental Evaluation 112
4.8 Conclusion 115

5 Conclusions **117**

5.1 Summary 117
5.2 Future Work 119

Appendices **120**

Appendix A RT dependency of schedulability tests **121**

A.1 Proof of RTA Test's compatibility with RT dependency 121
A.2 Proof of RTA-LC Test's compatibility with RT dependency 124
A.3 Proof of EPE-ZLL's incompatibility with RT dependency 125

Bibliography **130**

List of Figures

2.1 MITER-guided priority assignment algorithm 34

2.2 $m = 4, n = 16$, implicit deadline 43

2.3 $m = 8, n = 32$, implicit deadline 44

2.4 $m = 16, n = 64$, implicit deadline 45

2.5 $m = 4, n = 16$, constrained deadline 46

2.6 $m = 8, n = 32$, constrained deadline 47

2.7 $m = 16, n = 64$, constrained deadline 48

2.8 $m = 2$ with $6 \leq n \leq 15$ 52

2.9 $m = 4$ with $11 \leq n \leq 20$ 53

2.10 $m = 6$ with $16 \leq n \leq 25$ 54

3.1 An illustrative example of our *SHPC-DAG* model for subsystems seen in autonomous driving. 66

3.2 An illustrative example of the structure of our proposed stochastic conditional node. 69

3.3 Acceptance ratio comparison with varied probability tolerance λ. 90

3.4 Average SP value comparison with varied probability tolerance λ. 91

4.1 High priority block τ_1 driving low prority block τ_2, $T_1 < T_2$. 101

4.2 High priority block τ_1 driving low prority block τ_2, $T_1 > T_2$. 102

4.3 Low priority block τ_1 driving high priority block τ_2, $T_1 < T_2$. 103

4.4 Low priority block τ_1 driving high priority block τ_2, $T_1 > T_2$. 104

4.5 The MUDPA-guided optimization framework. 113

List of Tables

2.1 Average Runtime for each Priority Assignment (+ Schedulability Test) in seconds 50

3.1 Exemplary computational kernels for realistic robotic systems (time in seconds) 84

3.2 Average Execution Time for Different Methods 92

3.3 Average run-time of varied scheduling algorithms (time in seconds) 93

3.4 Average SP metric with varied probability tolerance λ 93

4.1 Results with varying RXN upper bounds (in milliseconds) 115

4.2 Results with varying DA upper bounds (in milliseconds) 115

List of Abbreviations

CTX Context

DA Data age

DA test Deadline Analysis test

DA-LC Improved Deadline Analysis with Limited Carry-in

DCMPO Deadline Minus Computation Monotonic Priority Ordering

DMPO Deadline Monotonic Priority Ordering

DPA Deadline and Priority Assignment

G-FP Global Fixed-Priority Scheduling

ILP Integer Linear Programming

LLP Least Latency Priority Assignment

MITER Maximal Infeasible Response Time Estimation Range

MUDPA Maximal Unschedulable Deadline and Priority Assignment

OPA Audsley's Optimal Priority Assignment

PDF Probability Distribution Function

PMF Probability Mass Function

RT Rate Transition

RTA Response Time Analysis

RTA-LC Improved Response Time Analysis with Limitted Carry-in

RTE Response Time Estimation

RXN Reaction Time

SA Simulated Annealing

SAR Search and Rescue

SHPC-DAG Stochastic Heterogeneous Parallel Conditional - Direct Acyclic Graph

SP Safety-Performance metric

SR Synchronous Reactive

TSL Time Sensitivty Level

VD Virtual Deadline

WCET Worst Case Execution Time

Chapter 1

Introduction

Real-time systems are systems that require correct output be obtained in the correct time, that is, within the specified time constraints [13]. Real-time systems are widely applied in various control-centric applications, such as aerospace, robotics, telecommunication, etc. In building hard real-time applications, it must be guaranteed that the deadlines of all tasks are met by a scheduling algorithm, which determines the execution orders of the tasks [13].

Priority assignment plays an important role in the real-time scheduling algorithm. It assigns a priority ordering to a task set, and the scheduler allocates the available computing resources (e.g. processors) to tasks with the highest priority during run-time [5]. There exists a rich set of research results on the priority assignment algorithms. Based on policies of assigning priorities, priority assignment algorithms can be classified into several categories, which we detail in section 1.2.1.

Among the existing categories of priority assignment algorithms, fixed priority assignment is one of the best choices for industrial applications. The fixed priority scheduler applies to the vast majority of commercial real-time operating systems, including OSEK and AUTOSAR [34]. The tasks and the jobs they generate will be assigned fixed priorities by the scheduler. During run-time, the jobs with highest priorities will be selected for execution when computing resources are available. Determining the optimal, or close-to-optimal priority assignment is crucial for both performance and commercial purposes. A poor priority assignment will let the scheduler execute the tasks in an order that leads to missed deadlines, which result

in performance degradation, or even safety issues for some applications.

In hard real-time systems, all jobs generated by tasks must finish execution before their corresponding deadlines. However, in some real-time systems, occasional deadline misses are acceptable and do not result in performance degradation, as long as the deadline misses are constrained within an acceptable range [17]. Such systems provide scheduling with more headroom, as the scheduling conditions are relaxed or rewritten in stochastic form.

In contrast to relaxing the conditions of schedulability, more design metrics that specify the requirements to satisfying performance could be added to the constraints of schedulability. End-to-end latency (e.g., reaction time, data age) [94] that quantifies system responsiveness and quality of control is a crucial factor to be considered in real-time systems. A well-performed task system should not only meet all deadlines with respect to response time, but also meet given constraints of design metrics.

1.1 Concepts

In this section, we introduce the terminologies that are widely used in real-time research studies and provide readers with formal definitions.

1.1.1 Task Model

Definition 1.1. Liu and Layland (LL) Periodic Task Model [65]. A task τ is defined by its worst-case execution time (WCET) and period, which is the interval between its two successive job instances.

This well known task model was proposed by C.L.Liu and J.W.Layland in 1973 [65]. The

LL model assumes tasks are periodic with fixed priorities, which reduces the complexity of real-time system design and enables the designers to analyze and verify system schedulability before implementation. With additional constraints on deadlines of tasks, this model can have two variations as follows.

Definition 1.2. Implicit Task Model. The deadline of a task is equal to its period.

The implicit task model works for real-time task systems that deadlines of tasks are naturally aligned with periods. It serves as one of the most basic assumptions and simplifies the schedulability analysis as tasks should finish execution before the occurrence of next task instance.

Definition 1.3. Constrained Task Model. The deadline of a task is smaller or equal to its period.

The constrained task model imposes a stricter constraint on the task deadline. This model applies to time-critical applications, such as robot navigation, automotive, that require timely response due to potential safety issues and ensure tasks finish execution before the occurrences of their next task instances.

These variations in the task model introduce an additional constraint on tasks' deadlines, which allows designers to analyze system schedulability with respect to specific timing constraints for applications.

1.1.2 Schedulability and analysis

Definition 1.4. Schedulability [65]. Given a scheduling algorithm, it is guaranteed that all deadlines of tasks will be met with the generated schedule prior to system run-time.

Definition 1.5. Schedulability Test [65]. Given a scheduling algorithm $\mathcal{A}$, if the task system Γ scheduled by $\mathcal{A}$ meets all deadlines of tasks upon a specified platform, then the system is said to be $\mathcal{A}$-schedulable.

The schedulability test needs the scheduling algorithm, task system, and hardware platform as input. With the schedule specified by the scheduling algorithm, a schedulable system ensures that all occurrences of tasks will finish execution before or at their deadlines.

1.1.3 Priority Assignment

Definition 1.6. Priority Assignment Algorithm. It is the algorithm that assigns priorities to tasks, which determines the importance of a task relative to others.

When a priority ordering over a task set is given, higher-priority tasks are scheduled or executed ahead of lower-priority ones when computing resources are available. A priority assignment algorithm determines the standard to assign priorities. This plays an important role in real-time scheduling, as a poor priority assignment will result in unnecessary missed deadlines that affect system performance.

1.2 Existing approaches, Issues and Challenges

In this section, we provide a selective review of recent studies of fixed-priority assignment problems that motivate our work in this dissertation. We explore two directions around the main topic of priority assignment: 1) global fixed-priority assignment on multiprocessor platform and 2) optimization techniques for priority assignment with constraints variations. We discuss the limitations and challenges of existing work for each direction as follows.

1.2.1 Global Fixed-Priority Assignment Algorithms

The state-of-the-art approaches for priority assignment can be classified into the following categories. An authoritative and complete survey can be found in the study [36].

Meta heuristics

Examples of widely applied metaheuristics are simulated annealing [91] and the genetic algorithm [52]. To our knowledge, the most recent work that involves the machine learning framework [61] also belongs to this category. Meta heuristics has advantages that they are not problem-oriented and applicable to various problem settings. However, these methods cannot guarantee the convergence of the solution and require (possibly heavy) parameter tuning. This limits their usage on increasingly more complex models and analyses.

Heuristics

In recent studies, several static deterministic priority assignment heuristics have been used. This includes DMPO [65], DCMPO [18] and DkC [32]. These methods assign priority to tasks based on the ordering of the calculated values. For example, in DMPO [65], tasks are sorted by the deadline values. Tasks with shorter deadlines will be assigned higher priorities. These heurisitics have the advantage of compatibility with any schedulability test. However, as heuristics, the optimality of the solution is not guaranteed, which can result in poor priority assignment and schedulability degradation.

Standard Optimization Frameworks

Standard optimization frameworks such as BnB [100] and Integer Linear Programming [122] can be applied to problems of priority assignment. However, due to the scalability issues, these frameworks can be very time-consuming.

Domain-specific Optimization Framework

While Audsley's optimal priority assignment (OPA) [5] is not compatible with recent sufficient scheduling tests, the authors of the studies [111, 112] propose frameworks that relax compatibility conditions and still leverage Audsley's polynomial time in the optimization process. However, these studies are limited in the cases where exact schedulability tests are compatible with Audsley's algorithm.

1.2.2 Priority Assignment with Constraints Variations

In aforementioned sections, schedulability of task system is defined as the system meets all deadlines with the given scheduling algorithm. However, in real-time applications, a system deemed schedulable may accept occasional deadline misses or have stricter constraints in addition to deadlines. These variations in constraints may offer the designer more room to adjust the design with respect to requirements of applications. For example, in [92], the task system is still considered schedulable if consecutive deadline misses are restricted to a predefined value k. In study [83], systems are deemed schedulable if the probability deadline is restricted to a small range. These changes relax the schedulability constraints and fit the actual requirements of real applications. In contrast, in other applications, more design metrics need to be considered, in addition to meeting all deadlines, when a system is deemed to be schedulable. Reaction time and data age are considered important factors in real-

time applications as they quantify the responsiveness and control quality of the system to input changes [94]. They can serve as additional constraints or objectives in system design phase. In this dissertation, we explore both cases and aim to find optimal or close-to-optimal priority assignment to different applications.

1.3 Organization

The remainder of the dissertation is organized as follows. In Chapter 2, we consider global fixed priority scheduling on the multiprocessor platform. The state-of-the-art priority assignment algorithms for G-FP can be divided into two main categories. The first category leverages the accuracy of the most recent sufficient schedulability analysis but uses it with a simple heuristic for priority assignment. The reason is that schedulability analyses in this category do not meet the compatibility conditions of the existing optimal priority assignment (OPA) algorithm. However, the second category uses the Audsley's polynomial-time optimal priority assignment algorithm, which is compatible only with a less accurate schedulability analysis. In this chapter, we study this issue and propose a novel algorithm that outperforms both existing categories. To begin with, we propose the concept of response-time estimation range that provides a range of feasible response times. Based on this, we build a new priority assignment framework, which is optimal when working with a more accurate schedulability analysis. The framework is expected to be better than the combination of OPA and its compatible analysis because it is integrated with a much more accurate analysis and proved to be optimal if its compatibility conditions are met. Furthermore, we judiciously combine the framework with the first category to produce a hybrid priority assignment framework that achieves high acceptance ratios under different task utilizations. The effectiveness of the proposed framework is evaluated with various sets of synthetic tasks. It has shown that

the proposed framework always achieves the highest acceptance ratio and outperforms the existing methods by 25% on average.

In Chapter 3, we focus on the scheduling of real-time systems embedded in real-world robotic applications. Robotic applications, such as search and rescue or precision agriculture, require robots to operate in complex, uncertain environments while maintaining safe interactions with human teammates within a specified level of performance. Such applications have unique properties that need to be considered when assigning priorities to tasks. First, due to unpredictable changes in the surrounding environment, the execution times of tasks may vary accordingly. When analyzing response times of task chains, current studies either use worst-case execution times (WCETs) of tasks or assume that execution times of tasks follow specific probability distributions (e.g., Gaussian distribution). However, this is not always true. WCETs consider the upper bounds of execution times, which can be too pessimistic in analysis, and Gaussian distributions can be too ideal for robots that operate in a dynamically changing environment. Second, modern robots are equipped with an increasingly more complex hardware platform for better functionality and efficiency. The hardware platform can be heterogeneous, which contains CPU, GPU, and other accelerators. Tasks assigned to different types of processor may have different execution times. This has an impact on the analysis of response time analysis of task chains. To solve the aforementioned issues, we propose the stochastic heterogeneous parallel conditional DAG (SHPC-DAG) model. It uses conditional DAG nodes to model the execution of computational pipelines based on environmental contexts, while the stochasticity of DAG edges captures the uncertain nature of a system's environment. In addition to this, we use probability distributions to model the execution times of tasks without making any assumptions about the distributions. A new partitioning algorithm, called least latency partitioned (LLP), which considers precedence constraints among nodes during the allocation process, is proposed as one of our major

contributions. Synthetic task sets inspired by real robotics are generated to evaluate the effectiveness of our framework. It is shown that our framework outperforms the state-of-the-art with more flexible representation of timing constraints and ultimately, safety-performance trade-offs.

We consider the priority assignment problem from a different perspective in Chapter 4. In addition to ensuring that the response times of all tasks meet the corresponding deadlines, other design metrics should be considered when assigning priorities. Reaction time and data age are end-to-end latency metrics that are crucial in control-centric applications. Existing studies have not considered such metrics in the Simulink model. In this chapter, we study the optimization problem that minimizes the reaction time and/or data age for the implementation of Simulink models by judiciously assigning task priorities. We first propose the method to calculate the reaction time and data age of task chains in Simulink model. An optimization framework is established that represents the schedulability constraints in compact forms. Experimental evaluation contains a real industrial case study and the results show that our framework runs at least 1000× faster than a direct formulation in integer linear programming.

Lastly, Chapter 5 concludes the dissertation. We summarize our contributions and discuss challenges and limitations that can be explored in future work. This dissertation contains the following studies that are published, or submitted:

1. A. Sifat*, X. Deng*, B. Bharmal, S. Wang, S. Huang, J. Huang, C. Jung, H. Zeng and R. Williams, A safety-performance metric enabling computational awareness in autonomous robots. In *IEEE Robotics and Automation Letters*, 2023.

2. X. Deng, S. Raja, Y. Zhao and H. Zeng, Priority Assignment for Global Fixed Priority Scheduling on Multiprocessors. In *IEEE Transactions on Computer-Aided Design of*

Integrated Circuits and Systems, 2024.

3. X. Deng, A. Sifat, S. Huang, S. Wang, J. Huang, C. Jung, R. Williams, H. Zeng, Partitioned scheduling with safety-performance trade-offs in stochastic conditional DAG models. In *Journal of Systems Architecture*, 2024.

4. X. Deng and H. Zeng, Latency-Aware Priority Assignment for Fixed Priority Scheduling in Multi-rate Simulink Models. *62nd Annual Design Automation Conference*, 2025 (Submitted).

Chapter 2

Priority Assignment for Global Fixed Priority Scheduling on Multiprocessors

2.1 Introduction

The widespread adoption of multiprocessors in real-time systems application has inspired many research studies on scheduling policies for multiprocessor platforms. Among them, global fixed-priority (G-FP) scheduling is a popular choice due to its advantages of application-transparent task migration and load balancing across all processors [13]. In G-FP, each task is assigned a static priority that applies to all its instances. At runtime, tasks are selected for execution based on their priorities, and they are allowed to execute on any processor and migrate from one processor to another [33].

The problem of priority assignment for G-FP faces two intertwined challenges [34]: (1) an analysis to check system schedulability, where the problem of exact schedulability for G-FP is proven to be NP-hard [21], and (2) the identification of the optimal priority order among a total of $n!$ possible ones. For the latter, Audsley's optimal priority assignment (OPA) algorithm [5] is a well-known efficient algorithm that only explores $O(n^2)$ priority orders.

However, it is only "optimal" with respect to the analysis that satisfies its compatibility conditions. Denote the set of all tasks with higher priorities than the current task τ_i as $hp(i)$, and $lp(i)$ as the set of those with lower priorities. The schedulability of a task is based on sets $hp(i)$ and $lp(i)$, but not on the relative orders in these sets [111].

Based on their compatibility with OPA, the schedulability analyses for G-FP can be classified into two categories: (1) OPA-compatible tests or (2) OPA-incompatible ones. Deadline Analysis (DA test) [20] and the improved Deadline Analysis with Limited Carry-in (DA-LC) [33] are two examples under the first category, both of which use the deadline of higher priority tasks to bound their interferences. In contrast, the Response Time Analysis (RTA test) [19] and the improved RTA test with Limited Carry-in (RTA-LC) [48] utilize the response time upper bounds of higher-priority tasks, which require exact priority ordering among tasks. Recently, a new analysis called EPE-ZLL is proposed to improve upon RTA-LC by excluding the parallel execution of higher priority tasks in the calculation of the interference [120]. RTA test, RTA-LC, and EPE-ZLL all violate the compatibility conditions of OPA.

Since many of the schedulability tests are not OPA-compliant, the current proposals for priority assignment in G-FP follow two different directions. The first is to use OPA together with an OPA-compliant analysis. This approach has to sacrifice accuracy in the schedulability analysis in order to leverage OPA. As far as we know, the most accurate analysis that is OPA-compliant is DA-LC [33]. Instead, the second approach settles with a suboptimal priority assignment algorithm such as Deadline Monotonic Priority Ordering (DMPO), Deadline Minus Computation Monotonic (D-CMPO), and DkC [18, 33, 65], in favor of accommodating an OPA-incompatible analysis that is much more accurate than DA-LC.

In this chapter, we propose a novel algorithm for priority assignment in G-FP [37]. We first push the frontier for the OPA-based approach, by developing an optimization framework that is optimal for a broader range of schedulability analysis than OPA. In other words, its

compatibility conditions are much weaker than those of OPA, which makes it possible to work with RTA-LC, a significantly more accurate analysis than DA-LC. We observe that this new frontier for the first approach, called MITER (Maximum Unschedulable response Time Estimation Range) with RTA-LC, is now performing better than the best of the second approach (DkC with EPE-ZLL) under low system utilization. We then develop a hybrid algorithm that works in a similar way as OPA but can use any schedulability test. Specifically, while assigning priority at a certain level, we use MITER with RTA-LC to estimate the priority order of higher priority tasks when the system utilization is low and use DkC with EPE-ZLL when the utilization is high. This judiciously combines the strengths of both approaches, achieving better schedulability than the two across all system settings.

The remainder of this chapter is structured as follows. We provide a selective review of related work on G-FP in section 2.2. In section 2.3, we introduce the system model and preliminaries. Section 2.4 describes the proposed concepts related to the MITER optimization framework. Section 2.5 presents the proposed hybrid priority assignment algorithm for G-FP scheduling. Section 2.6 conducts experiments to compare our approach with the state-of-the-art methods. Finally, Section 2.7 concludes the paper.

2.2 Related Work

We provide a selective review of the most recent and relevant research results for G-FP. For some of the more classical results, the readers are referred to [13, 36].

Compared to single-processor schedulability, the problem of exact schedulability for G-FP on a multiprocessor platform is far more challenging since the critical instant is unknown. As such, the research focus is to develop sufficient-only schedulability analyses and improve their accuracy. Andersson et al. propose a simple response time upper bound for tasks with

constrained deadlines [3]. Bertogna et al. develop the DA test [20] where the interference from a carry-in job is upper bounded using its deadline. RTA test instead uses the carry-in job's worst-case response time to bound its interference [19]. Guan et al. propose RTA-LC [48], an improvement over RTA test by bounding the number of carry-in jobs using the idea from Baruah [8]. Davis et al. [33] use the same idea to improve DA test and derive DA-LC. Zhou et al. present an improvement over RTA-LC called ZLL [119], by observing that part of the carry-in workload needs to be completed earlier, while RTA-LC assumes they are executed as late as possible. Later Zhou et al. propose a new method, henceforth denoted as EPE-ZLL, by excluding the parallel execution in the calculation of all interferences (not just those from carry-in jobs) [120]. There are also other *sufficient-only* schedulability tests in the literature such as [87, 88], but EPE-ZLL is demonstrated to achieve the best performance among them [120].

There are a few *exact* schedulability tests available, but they are either limited to the case of strictly periodic tasks [31], or time- and memory-consuming thus only practical for analyzing small tasksets (no more than 13 tasks) [23]. Cucu and Goossens derive an exact analysis for periodic tasks by simulating the task executions [31]. Baker et al. propose an exact schedulability test for sporadic tasks with brute-force search for feasible system states [6]. Bonifaci et al. improve this work by traversing a state transition graph and searching for an unschedulable state [21]. In [22], Burmyakov et al. improve the work in [21] by cutting down the state space for analysis. In [23], they further exploit the state-pruning idea to obtain a speedup of 2-3 magnitudes compared to Bonifaci's test. As a different approach than [6, 21, 22, 23], Sun et al. model the schedulability of G-FP as a linear hybrid automaton [89].

For priority assignment in G-FP, there is no known algorithm that is both efficient and compatible with the most accurate schedulability test. Cucu uses exhaustive search to find the optimal priority assignment policy for periodic tasks [30], requiring to check all $n!$ possible

priority orderings where n is denoted as the number of tasks. In contrast, the optimal priority assignment proposed by Audsley only checks $O(n^2)$ priority orders [5], but it comes with a set of compatibility conditions [33]. As far as we know, the most accurate analysis that is compatible with OPA is DA-LC test [33]. Also, various heuristic priority assignment policies are applied to G-FP, including DMPO [65], D-CMPO [18], and DkC [32]. In DMPO [65], the smaller the task deadline, the higher the priority. In D-CMPO [18], the smaller the difference between deadline and worst-case execution time, the higher the priority. DkC is a variant of D-CMPO that includes an extra parameter k, which depends on the number of processors [32]. These heuristics [18, 32, 65] can be combined with any schedulability test. Recently, Lee et al. proposed a machine learning (ML) framework [61], which requires the incremental construction of samples. It may become hard to infer a feasible priority assignment when n is large.

If we look beyond G-FP and review the approaches for priority assignment in other types of real-time systems, an authoritative survey can be found in [34]. In particular, when Audsley's algorithm is not optimal (either because the schedulability analysis violates its compatibility conditions, or the problem involves an objective function or other constraints), the state-of-the-art approaches may be classified into following categories. 1) meta-heuristics such as genetic algorithm (e.g., [52]). We compare with ML, possibly the most advanced method in this category and demonstrate that our approach may be better; 2) problem specific heuristics (e.g., [76, 96, 107]). This category often relies on certain properties in the system and cannot easily carry over to G-FP; 3) Standard optimization frameworks, including BnB [100] and Integer Linear Programming (ILP) [122]. However, besides the potential scalability issue, frameworks such as ILP may not be applicable to priority assignment in G-FP. For example, EPE-ZLL lacks an analytical form to calculate the interferences from higher priority tasks [120]; 4) Audsley's algorithm [5] and domain-specific frameworks [97, 110, 111, 112,

113, 114, 116, 117, 118] to optimize real-time systems. However, the status quo is that they either are for other design variables than priority assignment, or are limited to systems where the exact schedulability analysis is still compliant with Audsley's algorithm. Thus, for priority assignment in G-FP, these frameworks in [110, 111, 112, 113, 114, 116, 117, 118] are no longer applicable.

2.3 System Model and Preliminaries

We consider a real-time application consisting of n periodic or sporadic tasks $\Gamma = \{\tau_1, \tau_2, ...\tau_n\}$ scheduled on m identical processors under the global fixed priority scheduling algorithm. We adopt the three-parameter task model, which is a generalization of the Liu and Layland model [13]. The task is defined by a three-parameter tuple $\{C_i, T_i, D_i\}$, where C_i represents the Worst-Case Execution Time (WCET), T_i is the period or minimum inter-arrival time, and D_i is the deadline. We assume that the tasks have constrained deadlines ($D_i \leq T_i$). Since all events in the system occur at integer clock ticks [120], we assume that these parameters are all *positive integers*. The utilization of task τ_i is defined as $U_i = \frac{C_i}{T_i}$ and the total utilization of the system is $U = \sum_{i=1}^{n} U_i$. Each task τ_i is associated with a unique priority level π_i. For two tasks, τ_i and τ_j, if $\pi_i > \pi_j$, then task τ_i has a higher priority than τ_j. The set of tasks assigned higher priorities than τ_i is denoted as $hp(i)$. Similarly, tasks with lower priorities than τ_i are in the set $lp(i)$. Given a taskset Γ, we aim to find a schedulable task priority assignment $\mathbf{P}$ such that all tasks meet their deadline. The worst-case response time for the task is denoted as R_i for each task τ_i.

2.3.1 Schedulability Analysis

We now provide a summary of the response time analyses. We consider *sufficient-only schedulability analysis* methods since all the exact analysis methods for sporadic tasks suffer from the issue of high computational complexity and can only be practical for *analyzing* a small taskset (up to 13 tasks) [23]. The problem of priority assignment may involve analyzing the schedulability of a large number of design alternatives.

Definition 2.1. Workload [19, 48]. The workload $W_j(a, b)$ of a task τ_j in an interval $[a, b)$ is the accumulated execution time of τ_j within the interval $[a, b)$.

Definition 2.2. Interference [19, 48]. The interference $I_j(a, b)$ from a task τ_j on the target task τ_i over a time interval $[a, b)$ is the part of the workload of τ_j that can actually prevent τ_i from executing.

The workload consists of three different parts: *body*, *carry-in*, and *carry-out*. The *body* workload refers to the contribution of all jobs with both release time and deadline in the interval; each job of τ_j contributes to the workload in $[a, b)$ with a complete execution time C_j. The *carry-in* workload is the contribution of at most one job with its release time before a and deadline in $[a, b)$. The *carry-out* workload is the contribution of at most one job with its release time in $[a, b)$ and deadline after b.

DA Test [20]. DA test is based on the observation that the maximum interference could occur when the carry-in job is executed as late as possible and finishes at its deadline. An upper bound on the workload of task τ_j in a time interval of length l is derived as

$$W_j^D(l) = N_j^D(l) \cdot C_j + \min\left(C_j, l + D_j - C_j - N_j^D(l) \cdot T_j\right) \tag{2.1}$$

where $N_j^D(l) = \left\lfloor \frac{l+D_j-C_j}{T_j} \right\rfloor$ denotes the number of jobs whose release time and deadline are

both inside the time interval with length l. For task τ_i, the upper bound on the interference introduced by a higher priority task τ_j within the time interval of length l is given by

$$I_j^D(l, C_i) = \min\left(W_j^D(l), l - C_i + 1\right) \tag{2.2}$$

Then, a sufficient schedulability condition for task τ_i is derived by considering a time interval of length D_i

$$D_i \geq R_i = C_i + \left\lfloor \frac{1}{m} \sum_{j \in hp(i)} I_j^D(D_i, C_i) \right\rfloor \tag{2.3}$$

It is obvious that the $W_j^D(D_i)$ and $I_j^D(D_i, C_i)$ functions only require the knowledge of D_j, T_j, and C_j, which are all independent from τ_j's priority level. Hence, DA-test satisfies the compatibility conditions of Audsley's OPA algorithm [33].

DA-LC Test [33]. DA-LC improves DA test by limiting the interference from carry-in jobs: at most $m-1$ higher priority tasks can contribute to the carry-in workload. This leads to the following schedulability condition for τ_i:

$$\begin{aligned} &D_i \geq R_i = C_i + \\ &\left\lfloor \frac{1}{m} \left\{ \sum_{j \in hp(i)} I_j^N(D_i, C_i) + \sum_{j \in [\![i]\!]_{m-1}} \left\{ I_j^D(D_i, C_i) - I_j^N(D_i, C_i) \right\} \right\} \right\rfloor \end{aligned} \tag{2.4}$$

where $[\![i]\!]_{m-1}$ denotes the set of $m-1$ tasks in $hp(i)$ that have the largest values of $I_j^D(D_i, C_i) - I_j^N(D_i, C_i)$, and $I_j^N(D_i, C_i)$ is the maximum interference from τ_j if it has no carry-in job. For an interval of length l, $I_j^N(l, C_i)$ is

$$I_j^N(l, C_i) = \min\left\{ \left\lfloor \frac{l}{T_j} \right\rfloor C_j + \min(C_j, l - \left\lfloor \frac{l}{T_j} \right\rfloor T_j), l - C_k + 1 \right\} \tag{2.5}$$

Overall, the right-hand side of (2.4) achieves the maximum value over any subset of $m-1$

tasks in $hp(i)$.

Like DA test, DA-LC still only relies on the parameters T_j, C_j, and D_j of a higher priority task τ_j, which are all independent from its priority. DA-LC is the most accurate analysis that is compatible with OPA [33].

RTA Test [19]. RTA test is similar to DA test but with a more accurate way to execute the carry-in job: the latest time that a job can execute is at its worst-case response time rather than at its deadline. Hence, the workload of τ_j given in (2.1) can be more accurately rewritten as

$$W_j^R(l) = N_j^R(l) \cdot C_j + \min\left(C_j, l + R_j - C_j - N_j^R(l) \cdot T_j\right) \tag{2.6}$$

where $N_j^R(l) = \left\lfloor \frac{l+R_j-C_j}{T_j} \right\rfloor$ denotes the number of jobs of τ_j whose entire execution is inside the time interval with length l. The response time of τ_i under G-FP is the least fixed point solution of the following equation

$$R_i = C_i + \left\lfloor \frac{1}{m} \sum_{\tau_j \in hp(i)} I_j^R(R_i, C_i) \right\rfloor \tag{2.7}$$

where the inference $I_j^R(\cdot,\cdot)$ of τ_j now uses the new workload function $W_j^R(\cdot)$

$$I_j^R(l, C_i) = \min\left(W_j^R(l), l - C_i + 1\right) \tag{2.8}$$

Since $I_j^R(R_i, C_i)$ now depends on τ_j's response time R_j, the response time of τ_i not only relies on $hp(i)$ but also on the relative order among the tasks in $hp(i)$. Thus, RTA test is OPA-incompatible [33].

RTA-LC Test [48]. RTA-LC, similar to DA-LC, is based on the idea of limiting the interference from carry-in jobs. RTA-LC uses R_j as the latest completion time of a higher

priority task τ_j as opposed to D_j in DA-LC. This requires the following modification to (2.4):

$$R_i = C_i + \left\lfloor \frac{1}{m} \Omega_i(R_i) \right\rfloor \tag{2.9}$$

where the function $\Omega_i(\cdot)$ is defined as

$$\Omega_i(l) = \sum_{j \in hp(i)} I_j^N(l, C_i) + \sum_{j \in [\![i]\!]_{m-1}} \left\{ I_j^R(l, C_i) - I_j^N(l, C_i) \right\} \tag{2.10}$$

and the functions $I_j^N(\cdot,\cdot)$ and $I_j^R(\cdot,\cdot)$ are defined in (2.5) and (2.8) respectively.

Like RTA, RTA-LC is OPA-incompatible due to its dependency on R_j, the response time of a higher priority task τ_j [33]. However, we show that our enhancement over OPA, the MITER-based framework, is still compatible with RTA and its improvement RTA-LC (see Section 2.4).

EPE-ZLL Test [120]. EPE-ZLL tries to reduce the overestimation of interferences from all parts of the workload (carry-in, body, carry-out), not just those from the carry-in. It derives a lower bound on the accumulative time the target task and higher priority tasks are executed in parallel, which can be excluded from the interference.

To calculate R_i, EPE-ZLL uses an iterative process that gradually increases the value of τ_i's execution time a from one (i.e., $a = 1$) to its real value C_i ($a = C_i$). At each iteration, EPE-ZLL calculates the corresponding response time UR_i^{a+1} based on the value of UR_i^a. Specifically, UR_i^{a+1} is the minimal solution of the following equation

$$x \geq \left\lfloor \frac{\Psi_i(a, x)}{m} \right\rfloor + a + 1 \tag{2.11}$$

where $\Psi_i(a, x)$ is defined as

$$\Psi_i(a,x) = \min\Big\{ \quad m\left(UR_i^a - a\right) + \sum_{j \in hp(i) \cup \{i\}} W_{j,i}^{\max}(a,x), \\ \Omega_i\left(x + UR_k^a\right) \Big\} \tag{2.12}$$

Note that the calculation of $\Omega_i\left(x + UR_i^a\right)$ follows (2.10), and $W_{j,i}^{\max}(a,x)$ is calculated by an algorithm that requires the exact priority ordering of tasks in $hp(i)$ to determine the worst case release times of interfering jobs [120]. Similar to RTA-LC, EPE-ZLL is also incompatible with OPA. In addition, even if we assume that R_i of each task is known, the calculation of $W_{j,i}^{\max}(a,x)$ and hence EPE-ZLL still require the relative priority order in $hp(i)$.

2.4 MITER-guided Optimization Algorithm

We develop a novel optimization framework that advances OPA, termed Maximal Unschedulable response Time Estimation Range guided optimization algorithm (in short, MITER). Like OPA, MITER is optimal *if the associated schedulability analysis satisfies its compatibility conditions*, but those conditions are more relaxed than OPA, thus it can accommodate more accurate analysis that OPA cannot, including RTA and RTA-LC.

Still, MITER is unable to use some of the sophisticated but more accurate analyses such as EPE-ZLL. *In this section, by schedulability we mean that with a given priority assignment, the system is deemed schedulable with respect to the MITER-compatible analysis (and MITER will be able to find it).* It is possible that due to the inaccuracy of the chosen analysis, even if MITER fails to find a schedulable priority assignment, the system may still be deemed schedulable with another more accurate analysis. Similarly, *optimality in this section is also in regard to the used schedulability analysis.*

2.4.1 Response Time Dependency

We first introduce the concept of *response time dependency (in short, RT dependency)*. It is the property that the schedulability analysis shall satisfy in order to be used together with MITER. The response time R_i of each task τ_i can be calculated as follows,

$$R_i = f_i\left(hp(i), \mathbf{R}\right), \quad \forall \tau_i \tag{2.13}$$

where the function $f_i(\cdot)$ takes as inputs $hp(i)$, the set of higher priority tasks, and $\mathbf{R}$, the vector of response time estimations for all tasks. As can be seen from the definition of $hp(i)$, if $hp(i)$ is defined, then $lp(i)$ is also defined. Because the task set Γ consists of current task τ_i, $hp(i)$ and $lp(i)$. That is, $lp(i) \bigcup hp(i) = \Gamma \backslash \{\tau_i\}$. Therefore, for better clarity, we only give the definition of $hp(i)$ in the calculation.

The response time analysis written in the form of (2.13) directly implies the following two assumptions.

- A1: the response time R_i of τ_i, *if the response times of other tasks are known*, depends on the set of higher priority tasks $hp(i)$, but not on their relative order.
- A2: the response time R_i of τ_i, *if the response times of other tasks are known*, depends on the set of lower priority tasks $lp(i)$, but not on their relative order.

We assume (2.13) further satisfies two more conditions

- A3: the response time R_i of τ_i is monotonically non-decreasing with the set of higher priority tasks $hp(i)$. Specifically, given two sets of tasks $hp(i)$ and $hp'(i)$ such that $hp(i) \subseteq hp'(i)$, the response time R_i with $hp'(i)$ as the higher priority tasks is no smaller than that with $hp(i)$.

- A4: the response time R_i of τ_i is monotonically non-decreasing with the increase of the response time R_j of any other task τ_j.

Now we give a formal definition of the RT dependency.

Definition 2.3. The response time analysis of a real-time system Γ is said to be response time dependent (**RT dependent**) if it satisfies the above four assumptions A1-A4.

In comparison, the compatibility conditions of OPA are as follows.

- A1': the response time R_i of τ_i is based on $hp(i)$, the set of tasks of higher priority, but not on their relative order [111].
- A2': the response time R_i of τ_i is based on the set of lower priority tasks $lp(i)$, but not on their relative order [111].
- A3': the same as A3 (it is rewritten differently than [33]).

It is easy to see that A1-A4 are a weaker requirement than A1'-A3', hence MITER may be optimal with a more accurate analysis than Audsley's OPA. In fact, RTA and RTA-LC are both proven to violate A1'-A3' [33], but they can be combined with MITER as they are RT dependent. We formally state that in the following theorem.

Theorem 2.4. *RTA and RTA-LC tests are RT-dependent.*

Proof. We only prove property A4 for the two tests. The rest of the proof is straightforward and can be found in the Appendices A.1 and A.2.

We first show that the workload function $W_j^R(\cdot)$ as in Equation (2.6) will never decrease if the response time of any task τ_j ($j \neq i$) in set increases. For any schedulable system, we must have $C_j \leq T_j$. We first show $W_j^R(l)$ is monotonic with R_j when $l+R_j-C_j \in [K \cdot T_j, (K+1)T_j)$,

where the value of $N_j^R(l)$ is always equal to K. Let $x = l + R_j - C_j - K \cdot T_j$, then $W_j^R(l) = K \cdot C_j + \min(C_j, x)$. We consider two cases.

- Case 1: $l + R_j - C_j \in [K \cdot T_j, K \cdot T_j + C_j]$. Then the value of x will fall inside the interval $[0, C_j]$. The workload is

$$W_j^R(l) = K \cdot C_j + x,\ x \in [0, C_j] \tag{2.14}$$

  which is a linear function of R_j with a positive coefficient 1.

- Case 2: $l + R_j - C_j \in [K \cdot T_j + C_j, (K+1) \cdot T_j)$. x now becomes no smaller than C_j. Hence, the workload is a constant in this case:

$$W_j^R(l) = K \cdot C_j + C_j,\ x \in [C_j, T_j) \tag{2.15}$$

From the above two cases, we can conclude that when $l + R_j - C_j \in [K \cdot T_j, (K+1) \cdot T_j)$, the workload $W_j^R(l)$ is monotonically non-decreasing when R_j increases. Now we prove that when $l + R_j - C_j = (K+1) \cdot T_j$ (i.e., $N_j^R(l) = K+1$), the workload is guaranteed to be larger than or equal to the one when $l + R_j - C_j = (K+1) \cdot T_j - 1$. From the previous discussion, we already know that

$$W_j^R(l) = K \cdot C_j + C_j,\ \text{if } l + R_j - C_j = (K+1) \cdot T_j - 1 \tag{2.16}$$

When R_j increases by one, we have

$$W_j^R(l) = (K+1) \cdot C_j + \min(C_j, 0) \tag{2.17}$$

$$= (K+1) \cdot C_j,\ \text{if } l + R_j - C_j = (K+1) \cdot T_j \tag{2.18}$$

Therefore, the workload $W_j^R(l)$ is monotonically non-decreasing in the interval $[K \cdot T_j, (K+1) \cdot T_j]$ for arbitrary integer K, and consequently it is monotonically non-decreasing with R_j.

This easily lets us conclude that the interference $I_j^R(\cdot,\cdot)$ in (2.8), and the response time for RTA in (2.7) are monotonically non-decreasing with R_j. For RTA-LC, the response time in (2.9) additionally relies on $I_j^N(\cdot,\cdot)$, but this function, as defined in (2.5), is independent from R_j. Hence, condition A4 is satisfied for both RTA and RTA-LC. □

On the contrary, EPE-ZLL test [120] does not satisfy the conditions of RT dependency, since even if we assume R_i of each task is known beforehand, the relative priority orders are still required to determine the worst-case combination of the release times for the interfering jobs.

Theorem 2.5. *EPE-ZLL test is not RT-dependent.*

Proof. We leave the proof to the Appendix A.3 □

2.4.2 Response Time Estimation Range

The analysis in (2.13) violates the conditions of Audsley's OPA. However, if the response times of every task τ_j $(j \neq i)$ is appropriately estimated, then computing R_i in (2.13) will only need information of the sets $hp(i)$ and/or $lp(i)$, but not the information of exact priority orderings in these two sets. Combining with condition A3, this enables us to use OPA *if the response times of all tasks can be appropriately estimated*. We first introduce the concepts and properties related to response time estimations.

Definition 2.6. A **Response time estimation (RTE)** is defined as $\mathcal{E} = \{\langle \tau_1, r_1 \rangle, \ldots \langle \tau_n, r_n \rangle\}$, where r_i represents the estimated response time of task τ_i where $r_i \in [C_i, D_i]$.

For a given $\mathcal{E}$, the *estimation-inferred response time* of τ_i is calculated with the analysis in (2.13) assuming the response times of other tasks follow the estimated value in $\mathcal{E}$.

Definition 2.7. Given a priority assignment $\mathbf{P}$ and a response time estimation $\mathcal{E} = \{\langle \tau_1, r_1 \rangle, ... \langle \tau_n, r_n \rangle\}$, the **estimation-inferred response time** of task τ_i is the least fixed point solution of the following equation, denoted as R_i^E,

$$R_i^E = f_i\left(hp(i), \mathbf{E}_i\right), \quad \forall \tau_i \tag{2.19}$$

where $\mathbf{E}_i$ is a vector: the i-th entry of E_i is the variable R_i^E, and the j-th entry for any $j \neq i$ takes the corresponding given value r_j in $\mathcal{E}$,

$$\mathbf{E}_i = \left[r_1, \ldots, R_i^E, \ldots, r_n\right] \tag{2.20}$$

The vector of estimation-inferred response times R^E is denoted as

$$R^E = \left[R_1^E, R_2^E, \ldots, R_n^E\right] \tag{2.21}$$

Remark 2.8. With a given RTE $\mathcal{E}$, the calculation of estimation-inferred response time R_i^E for task τ_i only depends on the set of higher/lower priority tasks, but not on their relative order.

A desired property of an RTE that secures a schedulable priority assignment is given below (Definition 2.9). Moreover, there must exist an RTE with this property if and only if the system is schedulable (Theorem 2.10).

Definition 2.9. An RTE $\mathcal{E} = \{\langle \tau_1, r_1 \rangle, ... \langle \tau_n, r_n \rangle\}$ is defined as **feasible** if and only if there exists a priority assignment $\mathbf{P}$ such that the estimation-inferred response times are component-wise no larger than $\mathcal{E}$. That is, $\mathcal{E}$ is feasible if and only if

$$\exists \mathbf{P} \ s.t. \ \ \forall i = 1..n, \ R_i^E = f_i\left(hp(i), \mathbf{E}_i\right) \leq r_i \tag{2.22}$$

Theorem 2.10. *A system Γ has a schedulable priority assignment if and only if there exists a feasible RTE $\mathcal{E}$ [110].*

Theorem 2.10 claims that searching for a response time estimation $\mathcal{E}$ that satisfies Equation (2.22) is equivalent to finding a feasible priority assignment. The Audsley's algorithm can be used to check RTE $\mathcal{E}$ against Equation (2.22). As the estimation-inferred response times only requires information of the set $hp(i)$, but not the exact priority ordering inside $hp(i)$, it satisfies the compatibility conditions of Audsley's algorithm.

The total number of response time estimations is $O(\prod_i D_i)$, which is obviously impractical to do an exhaustive search. We propose a search space reduction technique that greatly improves the algorithm efficiency. The idea is that many infeasible RTEs share common reasons that violate definition 2.9 so we can quickly rule out similar infeasible RTEs from the search space to improve efficiency.

Before providing the formal definition, we first explore two conflicting requirements that exist in an RTE. For a given RTE $\mathcal{E} = \{\langle \tau_1, r_1 \rangle, ... \langle \tau_n, r_n \rangle\}$, consider any element $\langle \tau_i, r_i \rangle$, where the estimated response time r_i is used in two distinct places in condition (2.22): (a) r_i acts as an upper bound for a feasible estimation-inferred response time R_i^E of τ_i; (b) when computing the estimation-inferred response time R_j^E $(j \neq i)$, r_i is a constant entry in vector $\mathbf{E}_j$. Obviously for (a), a larger value of r_i is better but for (b), a smaller value of r_i is desirable since the estimation-inferred response time R_j^E is non-decreasing with r_i (property A4). These two conflicting requirements suggest that an infeasible RTE can be generalized to a range of response time estimations such that any of them falling inside this range will be infeasible too.

Instead of using r_i directly, we now split the estimation r_i into an optimistic estimation r_i^l and a pessimistic one r_i^u, where $r_i^l \leq r_i^u$. As discussed above, now we use r_i^u for (a) and

r_i^l for (b), which will result in a weaker condition (2.26) than (2.22). Suppose that this weaker condition (2.26) does not even allow a schedulable priority assignment, then it can be implied that any r_i falling inside the range $[r_i^l, r_i^u]$ would be infeasible for the original condition (2.22).

The formal definitions related to the response time estimation range are given below.

Definition 2.11. A **response time estimation range** $\mathcal{G}$ is a collection of tuple elements $\langle \tau_i, [r_i^l, r_i^u] \rangle$ for each task τ_i, i.e., $\mathcal{G} = \{\langle \tau_1, [r_1^l, r_1^u] \rangle, .. \langle \tau_n, [r_n^l, r_n^u] \rangle\}$, where $C_i \leq r_i^l \leq r_i^u \leq D_i$. It represents a range of possible estimation values for the actual response time R_i of each task τ_i.

For each task τ_i, the range $[r_i^l, r_i^u]$ is restricted to be a subset of range $[C_i, D_i]$. The reason is that for any schedulable system, the response time cannot be smaller than the worst-case execution time, or exceed the deadline.

Definition 2.12. A response time estimation $\mathcal{E}$ is said to be **contained** in a response time estimation range $\mathcal{G}$, denoted as $\mathcal{E} \in \mathcal{G}$, if and only if for each $\langle \tau_i, r_i \rangle$ in $\mathcal{E}$, the corresponding range $\langle \tau_i, [r_i^l, r_i^u] \rangle$ in $\mathcal{G}$ satisfies $r_i \in [r_i^l, r_i^u]$.

Definition 2.13. Given a response time estimation range $\mathcal{G} = \{\langle \tau_1, [r_1^l, r_1^u], ..[r_n^l, r_n^u] \rangle\}$ and a priority assignment $\mathbf{P}$, the **estimation range-inferred response time** of τ_i, denoted as R_i^G, is the least fixed point of the following equation

$$R_i^G = f_i\left(hp(i), \mathbf{G}_i\right) \tag{2.23}$$

where $\mathbf{G}_i$ is a vector constructed by taking the i-th entry as variable R_i^G and any other j-th entry as the value r_j^l from $\mathcal{G}$

$$\mathbf{G}_i = \left[r_1^l, ..., R_i^G, ..., r_n^l\right] \tag{2.24}$$

The **vector of estimation range-inferred response times** is denoted as $\mathbf{R}^G$.

Intuitively, due to property A4, the estimation range-inferred response time is essentially the smallest estimation inferred response time that can possibly be obtained for $\mathcal{E} \in \mathcal{G}$, as shown in the following equation

$$\begin{aligned} &\forall \mathcal{E} \in \mathcal{G},\ \forall i,\ \forall j \neq i,\ r_j \geq r_j^l \\ \Rightarrow\ &\forall \mathcal{E} \in \mathcal{G},\ \forall i,\ R_i^E \geq R_i^G \qquad \text{(by property A4)} \end{aligned} \tag{2.25}$$

Given an estimation range, the analysis of estimation range-inferred response times is based only on set $hp(i)$ but not on the exact priority ordering in $hp(i)$. This satisfies the compatibility conditions of Audsley's algorithm.

The schedulability of response time estimation range is defined as follows.

Definition 2.14. A response time estimation range $\mathcal{G} = \{\langle \tau_1, [r_1^l, r_1^u] \rangle, \ldots \langle \tau_n, [r_n^l, r_n^u] \rangle\}$ is said to be **feasible** if

$$\exists \mathbf{P}\ s.t.\ \ \forall i = 1..n,\ R_i^G = f_i(hp(i), \mathbf{G}_i) \leq r_i^u \tag{2.26}$$

Condition (2.26) is weaker than (2.22): it allows a smaller r_i^l than r_i^u, hence easier to be satisfied than (2.22). Like (2.22), (2.26) can be checked efficiently using Audsley's algorithm.

The proposed concept is useful due to its unique property: all the response time estimations included in an infeasible $\mathcal{G}$ will also be deemed infeasible. This helps us to quickly rule out some infeasible estimations if we find one infeasible $\mathcal{G}$. In contrast, a feasible $\mathcal{G}$ does not have this property and cannot help us speed up the search process of solution.

Theorem 2.15. *Given an infeasible response time estimation range* $\mathcal{G} = \{\langle \tau_1, [r_1^l, r_1^u] \rangle, \ldots \langle \tau_n, [r_n^l, r_n^u] \rangle\}$, *any response time estimation* $\mathcal{E} \in \mathcal{G}$ *is infeasible [110].*

Definition 2.16. A response time estimation range $\mathcal{G}_1 = \{\langle \tau_1, [r_1^{l1}, r_1^{u1}] \rangle, ... \langle \tau_n, [r_n^{l1}, r_n^{u1}] \rangle\}$ is a **subset** of another range $\mathcal{G}_2 = \{\langle \tau_1, [r_1^{l2}, r_1^{u2}] \rangle, ... \langle \tau_n, [r_n^{l2}, r_n^{u2}] \rangle\}$ if

$$\forall i = 1..n,\ r_i^{l1} \geq r_i^{l2}\ and\ r_i^{u1} \leq r_i^{u2} \tag{2.27}$$

$\mathcal{G}_1$ is said to be a **strict subset** of $\mathcal{G}_2$ if and only if $\mathcal{G}_1$ is a subset of $\mathcal{G}_2$ and $\mathcal{G}_1 \neq \mathcal{G}_2$.

We now define a class of infeasible response time estimation ranges that are not a strict subset of any other infeasible ones. This can maximize its contained infeasible RTEs.

Definition 2.17. A response time estimation range $\mathcal{U}$ is a **Maximal Infeasible response Time Estimation Range (MITER)**[1] if and only if it satisfies the following conditions

- $\mathcal{U}$ is infeasible by Definition 2.14;
- For all $\mathcal{G}$ such that $\mathcal{U} \subset \mathcal{G}$, $\mathcal{G}$ is feasible.

Remark 2.18. For the concept of response time estimation range, we shall treat the subset relationship as a partial order, i.e., $\mathcal{G}_1$ is no larger than $\mathcal{G}_2$ if $\mathcal{G}_1 \subseteq \mathcal{G}_2$. Let $\mathbb{S}$ be the set of all *infeasible* response time estimation ranges. A MITER $\mathcal{U}$ by Definition 2.17 is essentially a "maximal" element of $\mathbb{S}$. Since the order among response time estimation ranges is only partial, there may be multiple maximal elements of $\mathbb{S}$, i.e., multiple MITERs.

Intuitively, consider two infeasible response time estimation ranges $\mathcal{G}_1$ and $\mathcal{G}_2$ such that $\mathcal{G}_1 \subseteq \mathcal{G}_2$. $\mathcal{G}_1$ is redundant in the presence of $\mathcal{G}_2$, since the latter contains all infeasible RTEs contained in $\mathcal{G}_1$. In this sense, a MITER $\mathcal{U}$ is more useful than any of its subset $\mathcal{G}$ (i.e., $\mathcal{G} \subseteq \mathcal{U}$), since it is more efficient than $\mathcal{G}$ in capturing infeasible RTEs. We leverage this property to rule out the most infeasible RTEs with the fewest number of infeasible response

Algorithm 1 Algorithm for Computing MITER

1: **Function** MITER (infeasible RTE $\mathcal{E} = \{\langle \tau_1, r_1 \rangle, ... \langle \tau_n, r_n \rangle\}$)
2: $\mathcal{G} = \{\langle \tau_1, [r_1, r_1] \rangle, ... \langle \tau_n, [r_n, r_n] \rangle\}$
3: **for** each $\langle \tau_i, [r_i^l, r_i^u] \rangle \in \mathcal{G}$ **do**
4: Use binary search to find out the smallest value that r_i^l can be decreased to
5: while keeping $\mathcal{G}$ infeasible
6: Use binary search to find out the largest value that r_i^u can be increased to
7: while keeping $\mathcal{G}$ infeasible
8: **return** $\mathcal{G}$

time estimation ranges.

An infeasible RTE $\mathcal{E} = \{\langle \tau_1, r_1 \rangle, ... \langle \tau_n, r_n \rangle\}$ can be generalized into a MITER by Algorithm 1. We assume that the initial input $\mathcal{E}$ satisfies $r_i \in [C_i, D_i]$ for each task τ_i. We will later show in Section 2.4.3 how it can be guaranteed. The algorithm first converts the RTE to a response time estimation range $\mathcal{G} = \{\langle \tau_1, [r_1, r_1] \rangle, ... \langle \tau_n, [r_n, r_n] \rangle\}$ containing only $\mathcal{E}$ itself (Line 2). Then it leverages the property that condition (2.26) is monotonic w.r.t. each r_i^l and r_i^u: increasing r_i^u or decreasing r_i^l can only make (2.26) easier to be satisfied. It uses binary search to find out the minimum value that r_i^l can be decreased to (or the maximum value r_i^u can be increased to) while maintaining the unschedulability of $\mathcal{G}$ (Lines 3–6).

Specifically, Line 4 preserves the values of r_i^u and all other response time estimation ranges $\langle \tau_j, [r_j^l, r_j^u] \rangle, i \neq j$, and uses binary search to decrease r_i^l as much as $\mathcal{G}$ is infeasible. By Definition 2.7, r_i^l must be no smaller than C_i. Also, r_i^l has an initial value r_i which is known to be infeasible. Thus, the initial lower and upper bounds for the binary search of r_i^l are C_i and r_i respectively. The binary search stops when the lower and upper bounds converge (i.e., their difference is no more than one, which is sufficient since all response time estimations are integers). Line 5 is similar except that (a) it increases r_i^u, while keeping r_i^l at the value determined by Line 4; (b) the initial lower and upper bounds for r_i^u are r_i and D_i respectively.

[1]With a slight abuse of terminology, we use MITER to also refer to the MITER-guided optimization framework. The meaning of MITER should be clear from the context.

At Lines 4–5, Audsley's algorithm is used to check if the resulting $\mathcal{G}$ is feasible, i.e., to see if it permits a priority assignment that satisfies (2.26). Note that Audsley's algorithm only needs to explore $O(n^2)$ priority orders, r_i^l is bounded below by $C_i \leq D_i$, and r_i^u is bounded above by D_i, hence Algorithm 1 checks a total of $O(n^2 \log D)$ priority orders to calculate a MITER, where $D = \prod_i D_i$, and n is the number of tasks.

2.4.3 MITER-Guided Framework

By leveraging the concepts of RTE and MITER, we present an optimization algorithm for systems with an RT-dependent response time analysis. Finding a schedulable priority assignment for such systems, including G-FP, is particularly difficult since there is no known tractable procedure like Audsley's algorithm. As in Theorem 2.10, finding a schedulable priority assignment is equivalent to finding a feasible RTE. The latter has the promise to be more scalable for the following unique capability of Algorithm 1: it can efficiently generalize a given infeasible RTE to a set of MITERs, each of which contains a maximal range of infeasible RTEs.

Based on Algorithm 1, we propose a framework to find the optimal assignment of priorities. Instead of directly solving the original problem $\mathbb{O}$, we start with a relaxed problem $\mathbb{X}$ that leaves out all the system schedulability constraints. If the obtained RTE by solving $\mathbb{X}$ is infeasible, we use Algorithm 1 to generalize it to a set of MITERS. The corresponding constraints are then added to problem $\mathbb{X}$ to rule out similar infeasible RTEs, and the updated problem will be solved again. The iterative procedure will terminate (i) if the obtained RTE is feasible, which is guaranteed to be an optimal solution of $\mathbb{O}$, or (ii) $\mathbb{X}$ is deemed infeasible, which implies $\mathbb{O}$ is also infeasible (see Theorem 2.19).

The procedure is illustrated in Figure 2.1. Step 1 checks if the most relaxed response time

estimation range is schedulable. If yes, it enters the loop between Step 2 (solving the relaxed problem $\mathbb{X}$) and Step 3 (computing MITERs). The details for each step is explained as follows.

Step 1. Let R_i^L and R_i^U denote the smallest and greatest values of the response time of each task τ_i in any schedulable priority assignment. In this paper, we assume $R_i^L = C_i$ and $R_i^U = D_i$. Step 1 evaluates whether the response time estimation range $\zeta^M = \{\langle\tau_1, [C_1, D_1]\rangle, ...\langle\tau_n, [C_n, D_n]\rangle\}$ is schedulable. If not, then the systems must be unschedulable by any priority assignment, and the algorithm reports unschedulability and terminates.

Step 2. With all current MITERs (if available), the next step is to find a response time estimation that has not been deemed unschedulable. We exclude all schedulability conditions and replace them with MITERs which represent the constraints in a more compact form. The relaxed version, $\mathbb{X}$, of the original problem is solved to find such a response time estimation. Specifically for each MITER $\mathcal{U} = \{\langle\tau_1, [r_1^l, r_1^u]\rangle, ... \langle\tau_n, [r_n^l, r_n^u]\rangle\}$, by Theorem 2.15 any feasible RTE $\mathcal{E} = \{\langle\tau_1, r_1\rangle, ... \langle\tau_n, r_n\rangle\}$ cannot be contained in $\mathcal{U}$. This implies the following constraint,

$$\mathcal{E} \notin \mathcal{U} \;\Leftrightarrow\; \neg \begin{cases} r_1^l \le r_1 \le r_1^u \\ ... \\ r_n^l \le r_n \le r_n^u \end{cases} \Leftrightarrow \left\| \begin{array}{l} r_1 < r_1^l \| r_1 > r_1^u \\ ... \\ r_n < r_n^l \| r_n > r_n^u \end{array} \right. \tag{2.28}$$

where $\neg$, $\{$, and $\|$ represent the logic-NOT, logic-AND, and logic-OR operations, respectively. Thus, $\mathbb{X}$ just consists of the objective and all the other constraints in $\mathbb{O}$, while replacing the system schedulability constraints with those of (2.28) implied by all currently computed MITERs. $\mathbb{X}$ also includes the response time estimations of all tasks $[r_1, ...r_n]$ as the additional design variables, as well as their initial bounding constraints $C_i \le r_i \le D_i, \forall i$. In the problem

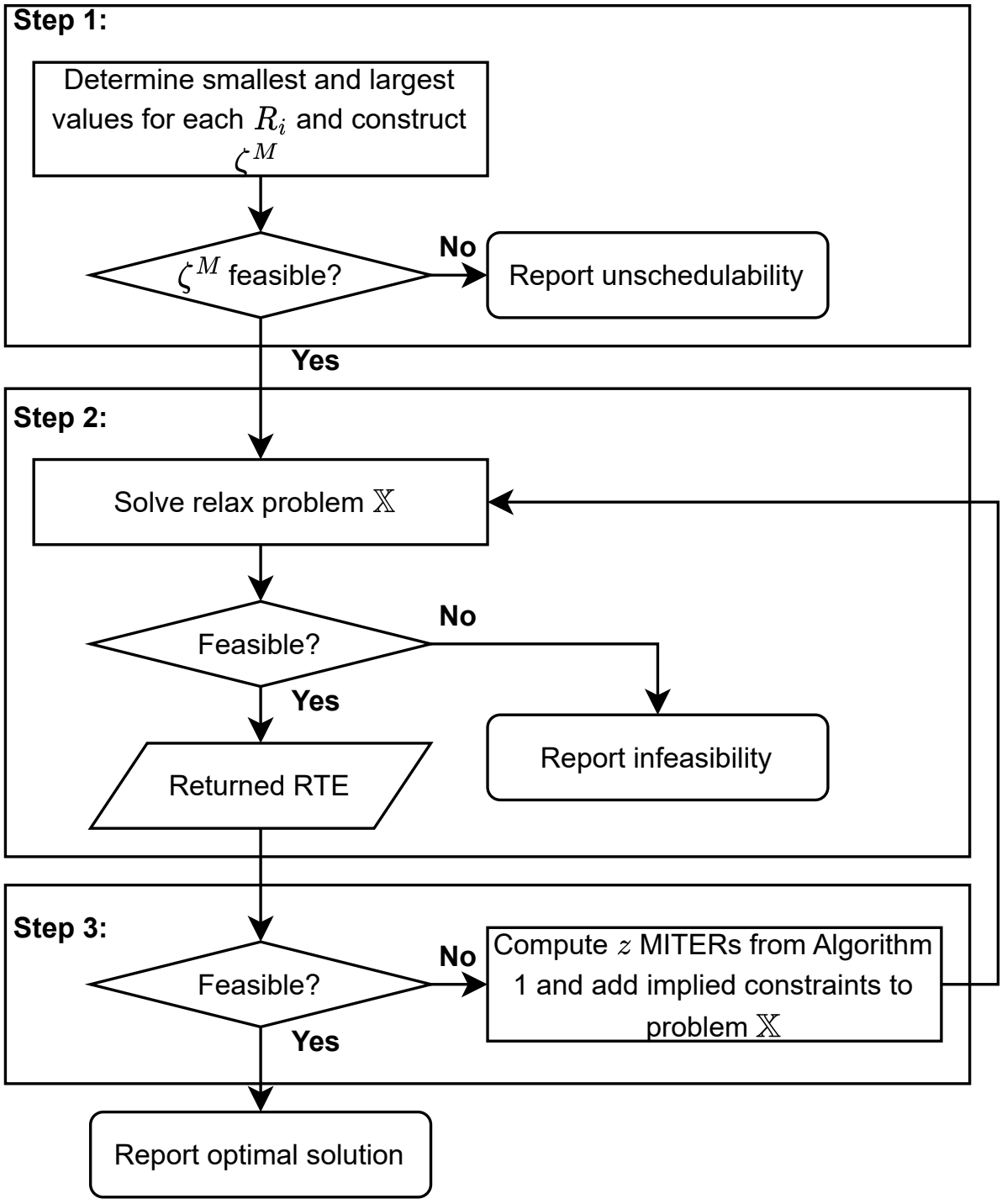


Figure 2.1: MITER-guided priority assignment algorithm

of priority assignment for G-FP, $\mathbb{O}$ has no objective and other constraints, but only the system schedulability constraints. The problem $\mathbb{X}$ is formulated as follows.

$$\begin{aligned} \mathbb{X}: \min \quad & 1 \\ \text{s.t.} \quad & \mathcal{U} \text{ as in (2.28)}, \ \forall\, \mathcal{U} \in \mathbb{U} \\ & C_i \leq r_i \leq D_i, \forall i \end{aligned} \tag{2.29}$$

where $\mathbb{U}$ is the set of currently known MITERs.

Standard ILP solvers (e.g., CPLEX) are applied to solve $\mathbb{X}$. The "big-M" method is used to formulate the logical disjunction constraint in (2.28).

$$\left\| \begin{matrix} r_1 < r_1^l \\ r_1 > r_1^u \end{matrix} \right. \Leftrightarrow \begin{cases} r_1 < r_1^l + M \cdot b_1 \\ r_1 > r_1^u - M \cdot (1 - b_1) \end{cases} \tag{2.30}$$

Here b_1 is an auxiliary *binary* variable, defined as

$$b_1 = \begin{cases} 0, & \text{if } r_1 < r_1^l \\ 1, & \text{if } r_1 > r_1^u \end{cases} \tag{2.31}$$

M is a sufficiently large constant such as D_1.

If $\mathbb{X}$ is infeasible (i.e., no solution satisfies all constraints in $\mathbb{X}$), then the system cannot be schedulable by any RTE (see Theorem 2.19 below). Otherwise, solving $\mathbb{X}$ will return an RTE (composed of the solution values assigned to the decision variables $[r_1, ...r_n]$) that respects all the implied constraints by the set of known MITERs $\mathbb{U}$.

Step 3. This step evaluates the schedulability of the RTE $\mathcal{E}$ returned in Step 2, i.e., whether it satisfies Equation (2.22). If yes, then $\mathcal{E}$ is optimal, and the associated priority assignment $\mathbf{P}$ is an optimal priority assignment (proven in Theorem 2.19). Here the priority assignment

$\mathbf{P}$ can be obtained as a byproduct of applying Audsley's algorithm in checking $\mathcal{E}$ against the condition (2.22).

If $\mathcal{E}$ is infeasible, it is generalized into a number of MITERs using Algorithm 1. The newly calculated MITERs imply compact forms of schedulability constraints and are added to the problem $\mathbb{X}$. The framework goes to Step 2 for the next iteration. The process terminates when it reaches max iterations with no feasible solution, or when an optimal solution is found. Here we set an upper bound z on the number of MITERs generated from a single infeasible RTE. Based on the experimental results, $z = 5$ is considered a good threshold in most cases. A set of examples that explains the definitions and calculation processes of the proposed concepts is included in our previous work (i.e., Examples 1–8) [110].

In Theorem 2.19, we claim the correctness of algorithm in 2.1.

Theorem 2.19. *The algorithm in 2.1 guarantees to* ***terminate*** *in finite iterations. If no feasible solution can be found for $\mathbb{O}$, the algorithm terminates and reports* ***infeasibility****. Otherwise, it returns the* ***optimal*** *solution with respect to the given objective [110].*

Despite that the proposed MITER-guided framework in 2.1 guarantees to terminate, it still has exponential time complexity with respect to the number of tasks. The total number of iterations depends on the tasks deadlines and can be as large as $O(\prod_i D_i^2)$. Therefore, we predefine a constant z to limit the maximum number of distinct MITERs calculated for each iteration. In each iteration, the framework solves an ILP problem $\mathbb{X}$. To conclude, the time complexity of our proposed framework remains exponential in the worst case.

In the problem of priority assignment for G-FP, the main advantage of MITER-guided framework compared to OPA is its optimality with a more accurate analysis RTA-LC than DA-LC, the most accurate analysis that can be used with OPA. In our experiments, we show that MITER with RTA-LC is consistently outperforming OPA with DA-LC (up to 30% in

acceptance ratio), despite that we limit the runtime of the algorithms (see Section 2.6).

2.5 Hybrid Algorithm with MITER

Each of the two categories of priority assignment algorithms for G-FP now has its own advantages: MITER with RTA-LC, our advancement to OPA-based algorithms, can schedule more tasksets when the system utilization is low, and DkC with EPE-ZLL, the best in the second category (heuristic with any analysis that may be more accurate than RTA-LC), works better for high system utilizations. We prudently combine their strengths and present a hybrid algorithm that significantly outperforms both.

We first briefly introduce the priority assignment process in OPA [5]. All tasks are not assigned priorities at the beginning. For each priority level, from lowest to the highest, OPA selects a task τ_i and assigns the current priority level to it. Assuming that all unassigned tasks have a higher priority than τ_i, the schedulability of τ_i is tested with an OPA-compatible test. If it is schedulable, the priority assignment is successful and OPA proceeds to the next higher priority level. If τ_i is unschedulable, the algorithm iteratively selects another unassigned task and repeats the schedulability test. If the whole set of unassigned tasks is traversed and no schedulable task is found, then OPA returns unschedulable. If all tasks are assigned priority levels at the end, the OPA returns schedulable.

Our observation is that OPA is a powerful and efficient algorithm, but it has one major limitation from its compatibility condition A1' (A2' and A3' are actually satisfied by most, if not all, schedulability tests including EPE-ZLL). The proposed idea is that in OPA, when deciding if τ_i is schedulable at a particular priority level, we may find a way to temporarily approximate the priority order of higher priority tasks $hp(i)$. This allows us to estimate the response time R_i of τ_i, even if we use the OPA-incompatible analysis EPE-ZLL.

Algorithm 2 Hybrid Priority Assignment with MITER

1: **Input:** Set of tasks Γ, Utilization threshold Θ
2: **Use:** Set of unassigned tasks Δ, Set of schedulable tasks Γ_s at the given priority level, $\Delta = \Gamma$
3: **for** each priority level p from lowest to highest **do** $\Gamma_s = \emptyset$
4: **for** each unassigned task τ_i in Δ **do** $hp(i) = \Delta \backslash \{\tau_i\}$
5: **if** Utilization of $hp(i) \leq \Theta$ **then**
6: Use MITER to find priority order $\mathbf{P}_{hp(i)}$ for tasks in $hp(i)$
7: **if** Unsuccessful **then** continue to next τ_i
8: **end if**
9: **else**
10: Use DkC to find priority order $\mathbf{P}_{hp(i)}$ for tasks in $hp(i)$
11: Calculate R_i of τ_i at priority p using EPE-ZLL
12: **end if**
13: **if** $R_i \leq D_i$ **then** Check schedulability of Γ using EPE-ZLL
14: **if** Γ is schedulable **then** Return **schedulable**
15: **else** Add τ_i to Γ_s
16: **end if**
17: **end if**
18: **end for**
19: **if** $\Gamma_s \neq \emptyset$ **then** Assign p to τ_i in Γ_s with the largest DkC value
20: $\Delta = \Delta \backslash \{\tau_i\}$
21: **else** Assign p to τ_i in Δ with the smallest lateness $R_i - D_i$ value
22: $\Delta = \Delta \backslash \{\tau_i\}$
23: **end if**
24: **end for**
25: **Return unschedulable**

Our new algorithm, called hybrid priority assignment with MITER (or in short HP-MITER), is detailed in Algorithm 2. Like OPA, it assigns a priority to a task in an iterative manner, starting from the lowest priority and advancing to the highest (lines 3–22). At each priority level p, it calculates the response time of each unassigned task τ_i using EPE-ZLL (Line 11). However, EPE-ZLL requires knowledge of the exact priority ordering of tasks in the set $hp(i)$. Therefore, we leverage the strengths of the two categories of priority assignments for G-FP and use a hybrid of MITER with RTA-LC and DkC with EPE-ZLL to estimate the priority order among tasks in $hp(i)$. When the total utilization of tasks in $hp(i)$ is lower than

a predefined threshold Θ (see next paragraph), MITER with RTA-LC is utilized to get the total order in $hp(i)$ (Lines 5–7). Otherwise, DkC is applied (Lines 9–10). Note this order $\mathbf{P}_{hp(i)}$ within $hp(i)$ is temporary and only used for the purpose of determining the response time R_i of τ_i (Line 11) or system schedulability (Line 13) while trying to assign τ_i at priority level p.

Threshold. The threshold Θ is acting as a dividing line between MITER with RTA-LC and DkC with EPE-ZLL. As shown in [120, Fig. 13], EPE-ZLL is significantly more accurate than RTA-LC when the system utilization is higher than 40%, otherwise, the difference between the two methods is negligible. Since MITER is optimal and better than DkC for MITER-compatible analysis such as RTA-LC, we set the threshold Θ at 40%, with the hope that MITER can compensate the small disadvantage of RTA-LC. Also, on varying the threshold from 0% to 100%, it is found that 40% gives the overall best performance in both implicit and constrained deadline cases. Note that this threshold is chosen for the combination of MITER with RTA-LC and DkC with EPE-ZLL, and it may not be the same for other combinations of schedulability tests and priority assignment policies.

Algorithm 2 has a couple of noticeable designs that are different from OPA, considering that the priority order $\mathbf{P}_{hp(i)}$ may not be the same as in the final solution and the estimated response time R_i may not be accurate. *First*, in Line 13, in the case that τ_i is schedulable at level p, we opportunistically check whether the current priority assignment (the temporary order $\mathbf{P}_{hp(i)}$ for the set $hp(i)$, τ_i at level p, and all other tasks follow the previously fixed priority levels) happens to be schedulable according to EPE-ZLL. If so, the algorithm terminates and returns the current priority assignment. *Second*, if we find a schedulable task at priority level p, instead of terminating immediately and returning schedulability, we simply add this task to Γ_s, the set of schedulable tasks at level p (Line 15). Again, this is because $\mathbf{P}_{hp(i)}$ is not necessarily the order in the final solution. *Third*, if at least one candidate task

is schedulable at the current priority level (i.e., Γ_s is not empty), then we follow the DkC policy and select the one in Γ_s with the largest DkC value (Line 19–20). If none of them is schedulable, instead of reporting unschedulability as in OPA, we select the one with the smallest lateness $R_i - D_i$ among them and continue to the next priority level (Lines 21–22), in the hope that the small lateness may be erased by a different order in $hp(i)$. *Fourth*, in the rare case that MITER is unable to find any schedulable priority order for the set $hp(i)$, we continue on to the next task in the set of unassigned tasks. (Lines 7). *Finally*, the loop terminates at Line 25 returning unschedulable in two cases: (1) if for a particular priority level, MITER is always used to obtain the initial ordering for $hp(i)$ and it is not able to find a feasible ordering for any them, or (2) none of the priority levels lead to Line 13 returning true.

Another point we would like to clarify is the use of schedulability analysis in Algorithm 2. Throughout the algorithm we use EPE-ZLL, the most accurate analysis that is still practical. The only exception is at Line 6, where the MITER-guided framework is leveraged to find the temporary priority order for $hp(i)$. We use RTA-LC at Line 6, the most accurate analysis satisfying the compatibility conditions of MITER.

2.6 Experimental Evaluation

In this section, we present the experimental results of the proposed priority assignment algorithms as well as other representative combinations of schedulability tests and priority assignment algorithms. We perform two sets of experiments, the first is to include most of the state-of-the-art except the machine learning framework [61]. Due to the lack of source code and training data from [61], we were unable to duplicate its results. Hence, we use a second set of experiments following the settings in [61] and make an indirect comparison

with it.

2.6.1 First Experiment

As summarized in Section 3.1, the state-of-the-art can be classified into two categories: an optimal priority assignment algorithm with an analysis that is compatible, and a heuristic algorithm with a more accurate analysis. Hence, we consider the following methods and compare their performances in terms of acceptance ratio, i.e., the percentage of tasksets that are deemed schedulable by each method and the average runtime.

- OPA + DA-LC (Purple line): Deadline Analysis with Limited Carry-in (DA-LC test from [33]) with Audsley's Optimal Priority Assignment. This is the previous state-of-the-art in the first category.
- MITER (Blue line): This is our advancement in the first category. It is optimal with RTA-LC, a more accurate analysis than DA-LC.
- EPE-ZLL (Red, Orange, Green lines) combined with three different heuristics DkC, DMPO, and D-CMPO. They represent the state-of-the-art in the second category.
- HP-MITER (Black line): The hybrid priority assignment algorithm presented in Section 2.5.
- HP-MITER without lateness (Brown line): This is the version of Algorithm 2 without Lines 21–22. This is done to evaluate the effectiveness of using lateness when all tasks are unschedulable at a priority level.

Since MITER has an exponential worst-case time complexity, to avoid excessive runtime while ensuring a fair comparison, we set the time limit of MITER to the same value of 600

seconds both when used in HP-MITER and as a stand-alone approach for comparison. When a timeout occurs, the taskset is deemed unschedulable by MITER.

We consider the following number of processors and tasks: (1) $m = 4, n = 16$, i.e., 4 processors and 16 tasks; (2) $m = 8, n = 32$; (3) $m = 16, n = 64$. For each case, we choose 30 system utilizations in the range of $[0, m]$. For each utilization, 1000 random tasksets are generated, with the following way to set the task parameters:

- Utilization: Task utilizations are generated using the UUnifast-Discard algorithm [33].
- Period: Task periods are generated according to a log-uniform distribution in the range [10, 1000].
- Execution time: Task execution times are calculated based on the generated utilizations and periods: $C_i = U_i \cdot T_i$.
- Deadline: We test the algorithms for two task models: implicit deadline and constrained deadline. In the latter case, task deadlines are set according to a uniform distribution in the range $[C_i, T_i]$.

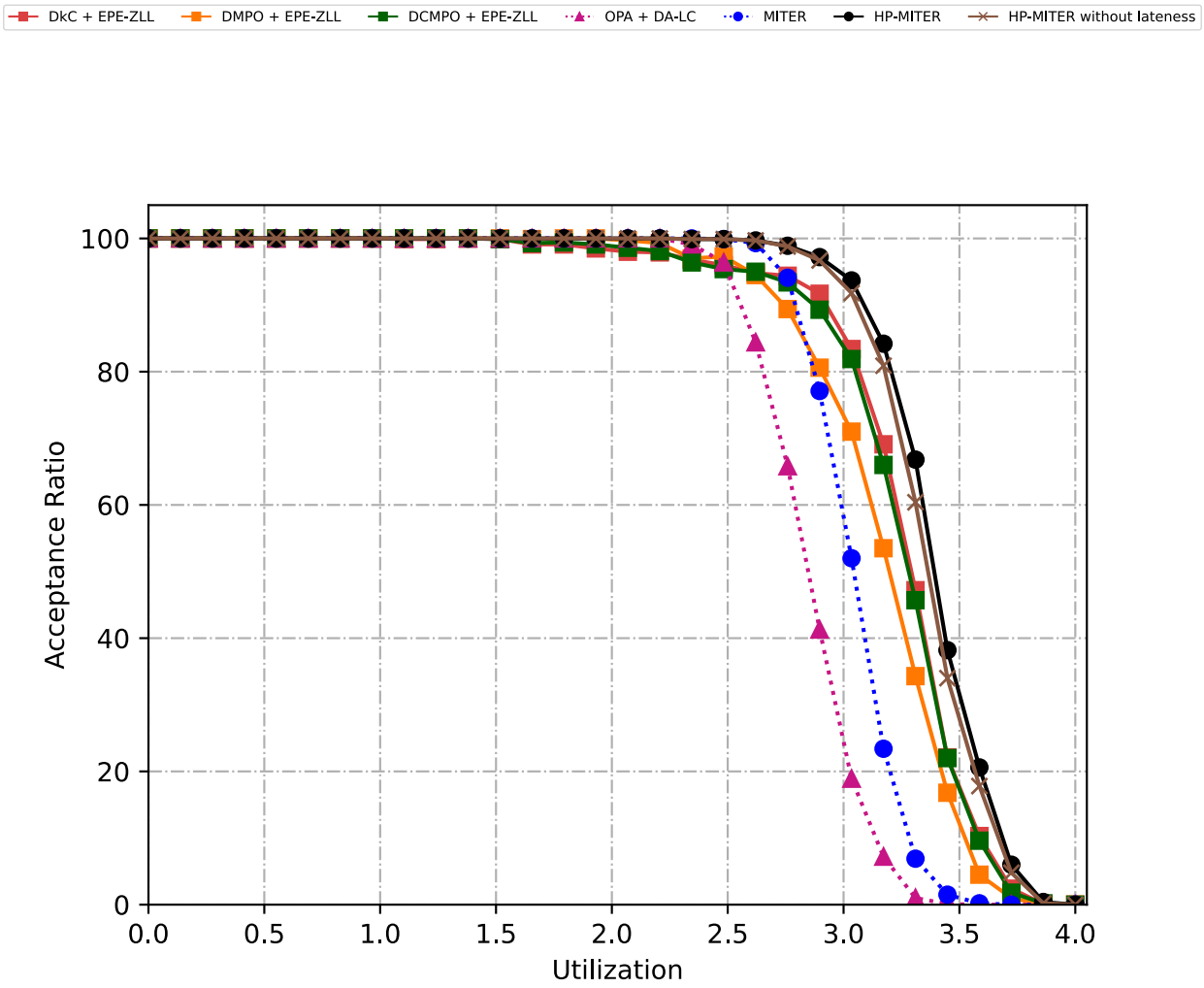


Figure 2.2: $m = 4, n = 16$, implicit deadline

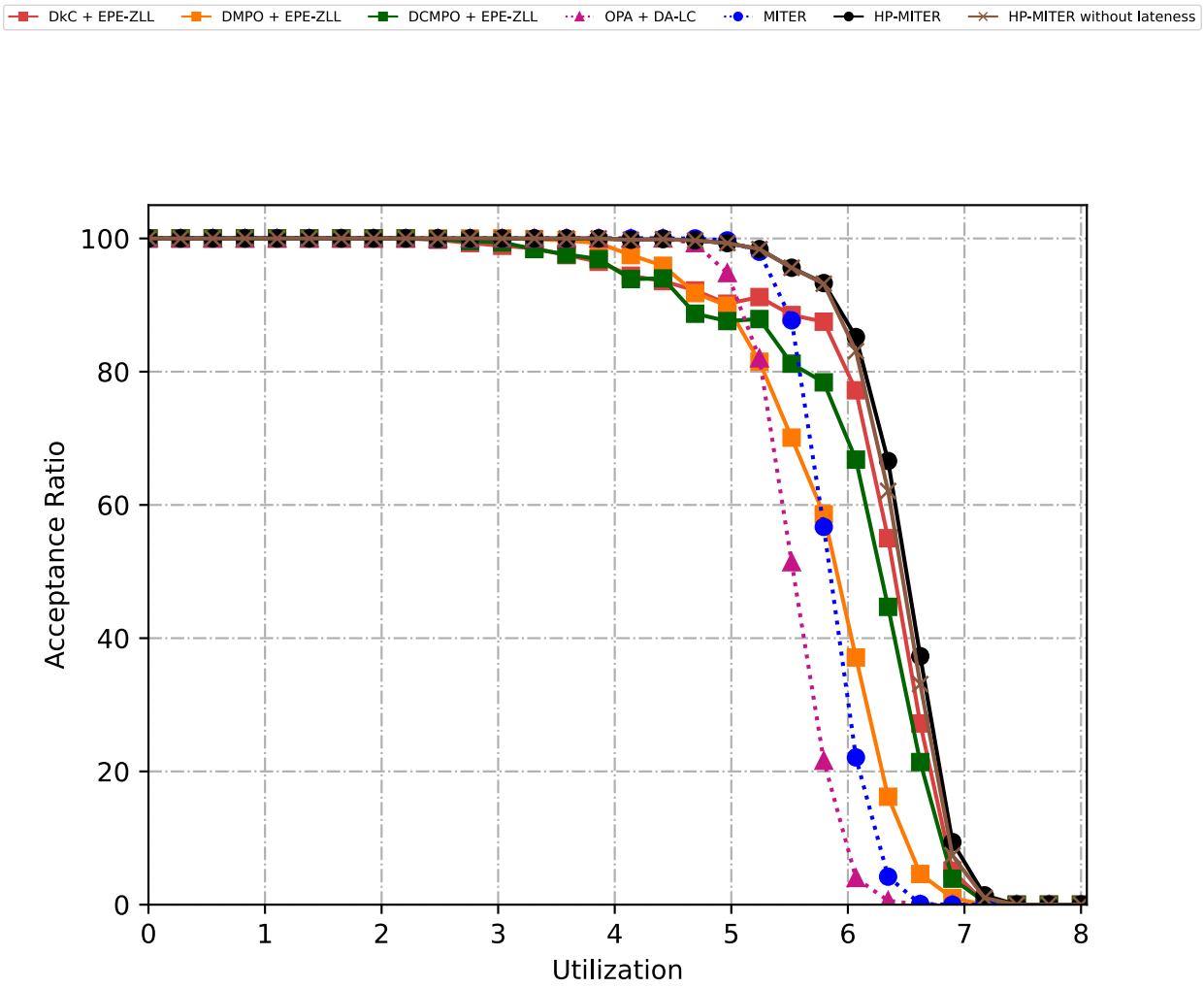


Figure 2.3: $m = 8, n = 32$, implicit deadline

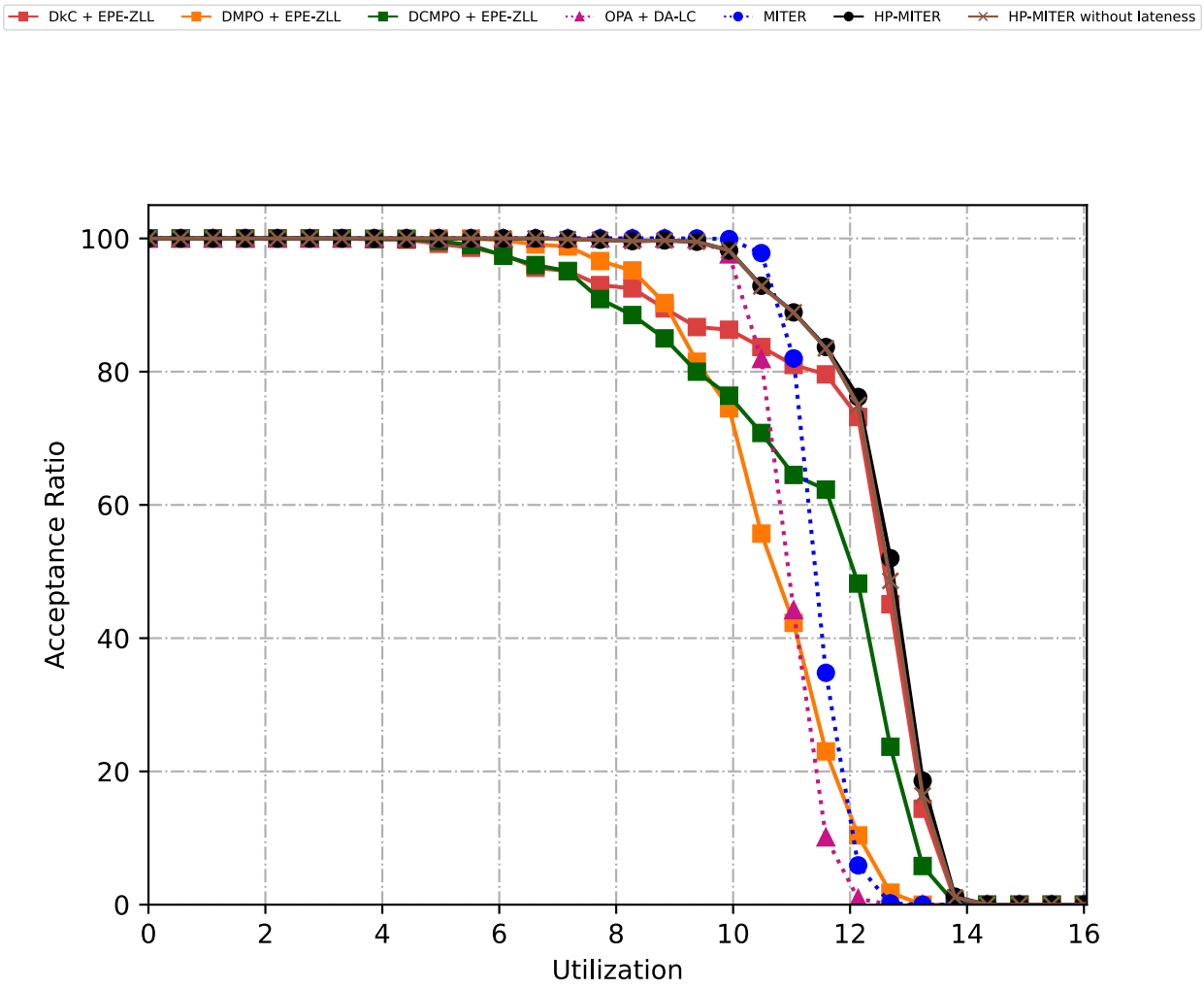


Figure 2.4: $m = 16, n = 64$, implicit deadline

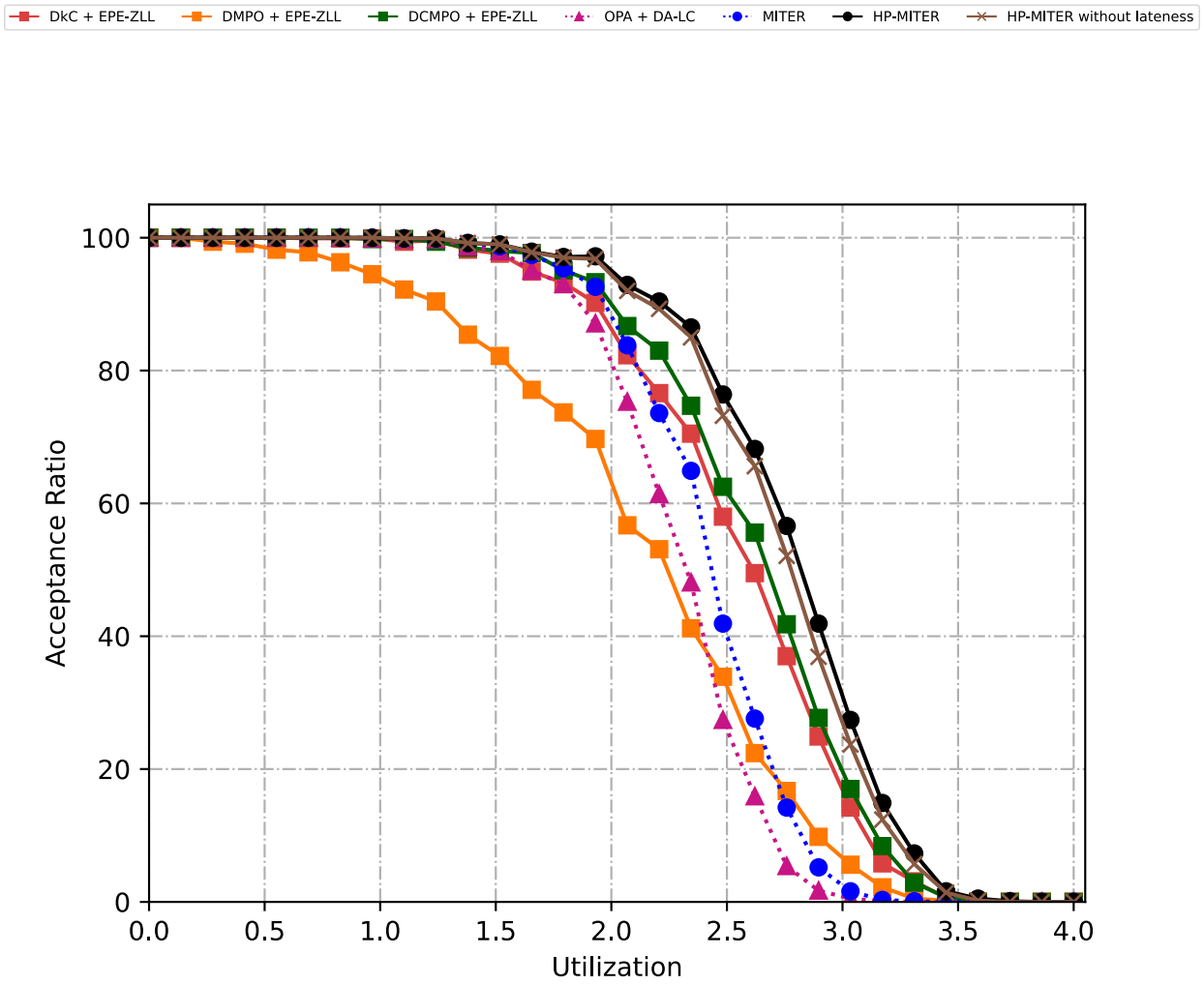


Figure 2.5: $m = 4, n = 16$, constrained deadline

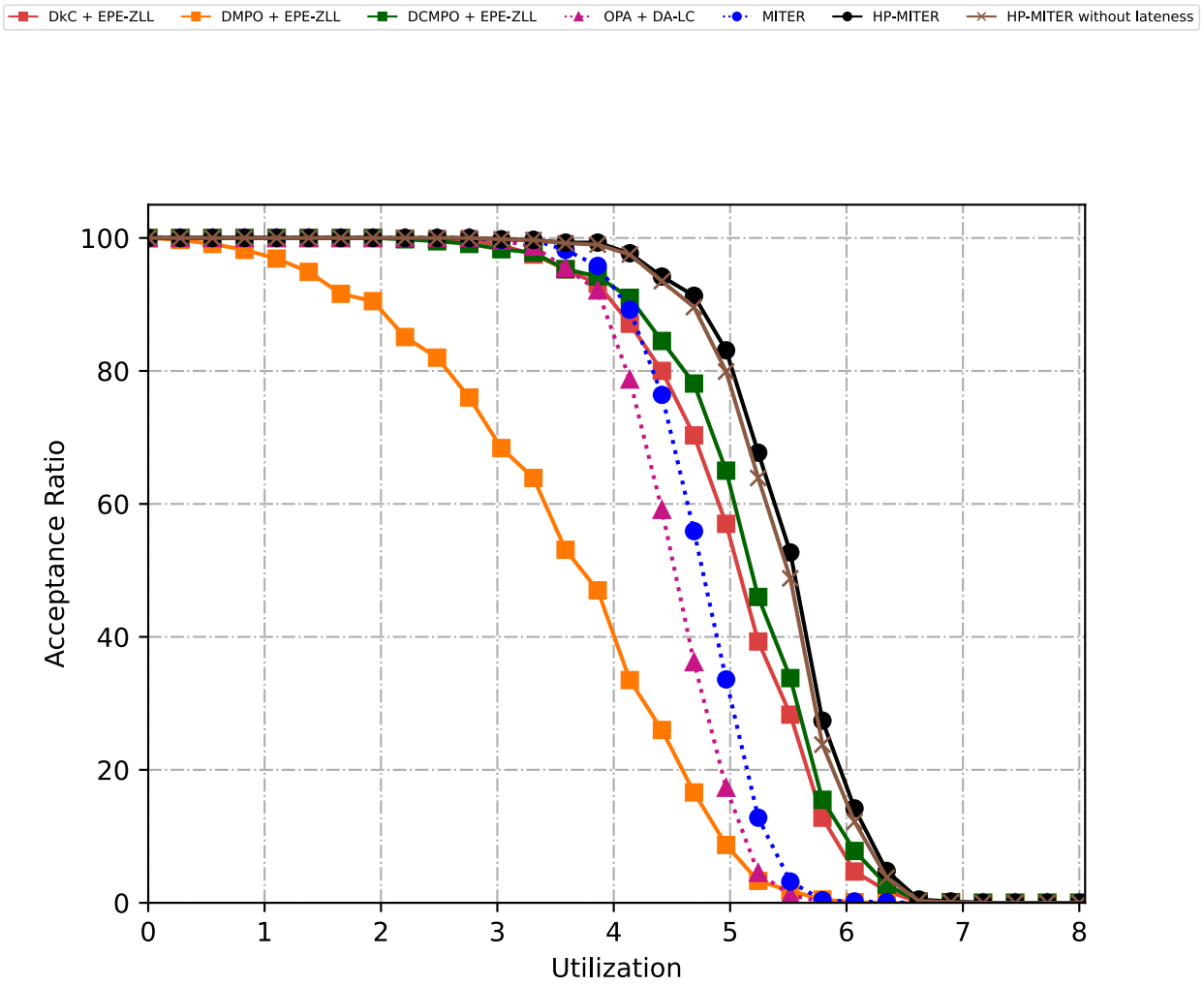


Figure 2.6: $m = 8, n = 32$, constrained deadline

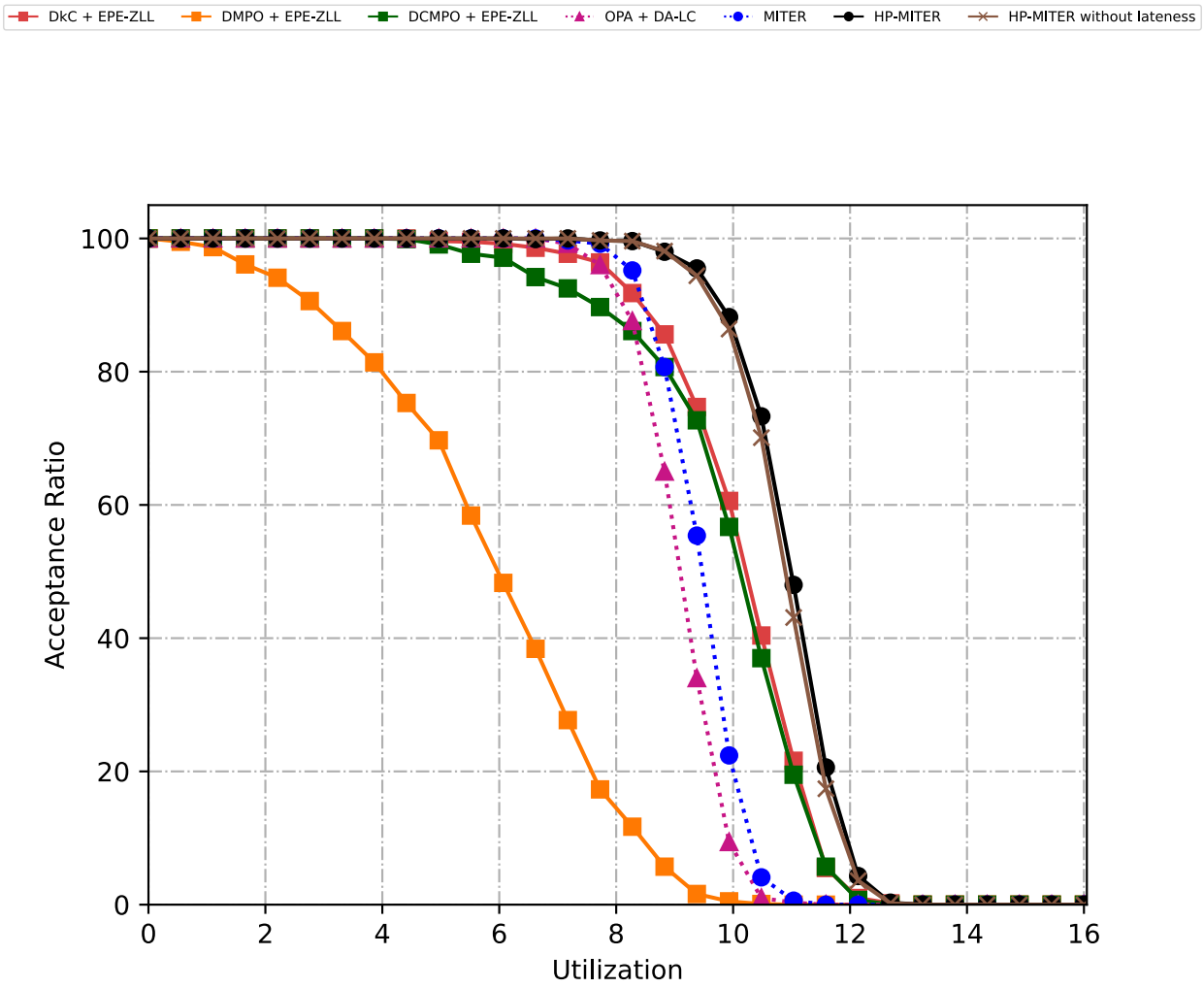


Figure 2.7: $m = 16, n = 64$, constrained deadline

The experimental results are plotted in Figures 2.2 to 2.7, from which we make a few interesting observations. *First*, MITER with RTA-LC is always able to schedule (as much as 30%) more tasksets than OPA + DA-LC. Since both are optimal with respect to their schedulability analyses satisfying the respective compatibility conditions, MITER's superiority comes from its ability to adopt a more accurate analysis than OPA. *Second*, DkC is still a better heuristic than the other two (DMPO and D-CMPO) with the new analysis EPE-ZLL, consistent with the conclusion in [33] that uses RTA-LC as the analysis. However, different than [33], the approaches in the second category, in particular, DkC + EPE-ZLL is now significantly better than the previous state-of-the-art OPA + DA-LC in the first category. Even compared to the new frontier MITER with RTA-LC in the first category, DkC + EPE-ZLL is still able to schedule more tasks at high system utilization. This is mainly because the new advancement in schedulability analysis, EPE-ZLL, is substantially more accurate than RTA-LC or DA-LC, easily compensating the disadvantage of a heuristic priority assignment policy like DkC compared to optimal priority assignment algorithms. *Third*, HP-MITER, taking advantages of both categories, is outperforming the other methods across all settings. For example, it improves OPA + DA-LC by up to 70% for implicit deadline tasks, and by up to 80% for constrained deadline tasks. Compared to DkC + EPE-ZLL, HP-MITER can be better by a margin of 18% for tasks with implicit deadlines. For tasks with constrained deadlines, the improvement of HP-MITER over DkC + EPE-ZLL is even higher, with a maximum difference of 25%. On comparing HP-MITER with the version where lateness is not used, there is a small but noticeable difference which decreases as the number of processors increases. This implies that either (1) the case where none of the tasks have a response time less than deadline is very rare, or (2) whenever that case does occur, the choice of using lateness does not have a large impact on the schedulability.

The average runtimes of all the algorithms are presented in Table 2.1. It can be seen from

Table 2.1: Average Runtime for each Priority Assignment (+ Schedulability Test) in seconds

Type of Deadline	m	DkC + EPE-ZLL	DMPO + EPE-ZLL	DCMPO + EPE-ZLL	OPA + DA-LC	MITER	HP-MITER	HP-MITER without lateness
Implicit	4p	0.7939	1.1108	0.8734	0.0029	42.4201	81.9924	27.1600
	8p	4.7934	6.8357	5.6165	0.0110	96.2297	738.4391	182.1743
	16p	23.4517	29.6310	26.5116	0.0395	118.4041	9824.9771	2299.8825
Constrained	4p	0.2587	0.2995	0.3221	0.0016	21.8565	129.9478	28.5039
	8p	1.8508	1.7981	2.7669	0.0080	67.5505	961.5012	193.7253
	16p	12.4310	8.1328	17.8814	0.0363	93.0550	10274.4882	2284.1387

the table that the combination of OPA and DA-LC is the fastest at the cost of the lowest acceptance ratio. The heuristics with the EPE-ZLL run slower due to the complexity of EPE-ZLL. Our proposed methods, MITER and HP-MITER run several magnitudes slower than other methods. This is expected as in the worst case, the MITER will be called n^2 number of times. Omitting Lines 21–22 (lateness) could result in a better runtime but sacrifice the acceptance for up to 9% when compared to the original version of HP-MITER. As the proposed algorithm is used in an offline setting and the aim is to find more schedulable tasksets that are deemed unschedulable by existing methods, the runtime of HP-MITER remains acceptable for the acceptance ratio that it provides.

2.6.2 Second Experiment

In the second experiment, we make an effort to compare with the ML framework [61]. We follow the same experimental settings as in [61] and use ZLL [119] as a common method for comparison. Specifically, we consider systems with $m = 2, 4, 6$ processors, and vary the number of tasks n as follows: $m = 2, n \in [6, 15]$; $m = 4, n \in [11, 20]$; and $m = 6, n \in [16, 25]$. The task parameters are generated as follows:

- Utilization: When setting the task utilizations in a taskset, first one of the following ten distributions is selected randomly: binomial and exponential distributions, each with constants 0.1, 0.3, 0.5, 0.7, and 0.9. Then for each of the tasks, its utilization is generated according to the same selected distribution.

- Period: it is randomly generated following a log-uniform distribution in the range of [10,1000].
- Execution time: it is calculated by the multiplication of $T_i \times U_i$ for each task τ_i.
- Deadline: all tasks have implicit deadlines, i.e., $D_i = T_i$.

After these task parameters are created, the taskset is tested and discarded if it is (1) schedulable by RTA-LC with the heuristics DkC [32], D-CMPO [18] or DMPO [65], (2) not schedulable according to the C-RTA condition [33], which is a necessary but not sufficient condition for a taskset to be schedulable by RTA-LC [48], and (3) schedulable by DA-LC test [33] with Audsley's OPA [5]. For each (m, n) pair, 1000 tasksets are generated.

Below is the list of methods compared in this experiment:

- ZLL: The baseline in [61] using the three heuristic priority assignment policies (DkC, DMPO, D-CMPO) with ZLL as the schedulability analysis [119]. That is, a taskset is schedulable by this baseline if any of the three policies with ZLL deems it schedulable.
- EPE-ZLL: The same as the above method except using EPE-ZLL [120] as the schedulability analysis instead of ZLL [119].
- MITER: The same as in the previous experiment.
- HP-MITER: The same as in the previous experiment.

We present the results in Figures 2.8 to 2.10, where the y-axis gives the percentage of schedulable tasksets among those that are deemed schedulable by at least one of the four methods. In all cases, across all values of n, HP-MITER is able to find a feasible priority ordering for at least 85% of the time, consistently outperforming all the other methods. The *average* difference between HP-MITER and ZLL is around 10% for $m = 2$, 35% for $m = 4$,

and 55% for $m = 6$. As reported in [61], the ML framework is outperformed by ZLL when (1) $m = 2, n \in [11, 15]$; (2) $m = 4, n = 20$; (3) $m = 6, n = 21, 24, 25$. Even when ML is better than ZLL, the *maximum* advantage is about 40%. These results provide an indirect proof that HP-MITER is potentially able to find schedulable priority assignments for more tasksets than the ML framework. It must be noted that this result is expected as [61] uses RTA-LC, which is a less accurate schedulability test than EPE-ZLL, as its basis.

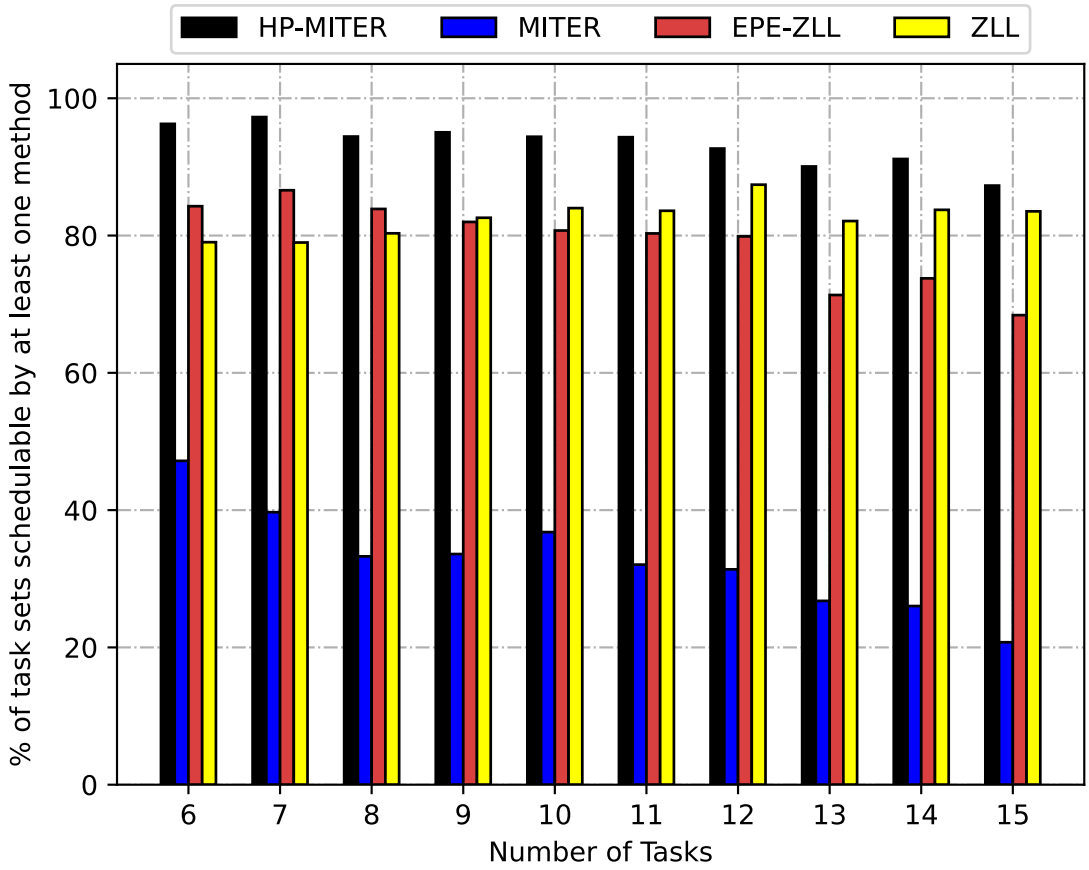


Figure 2.8: $m = 2$ with $6 \leq n \leq 15$

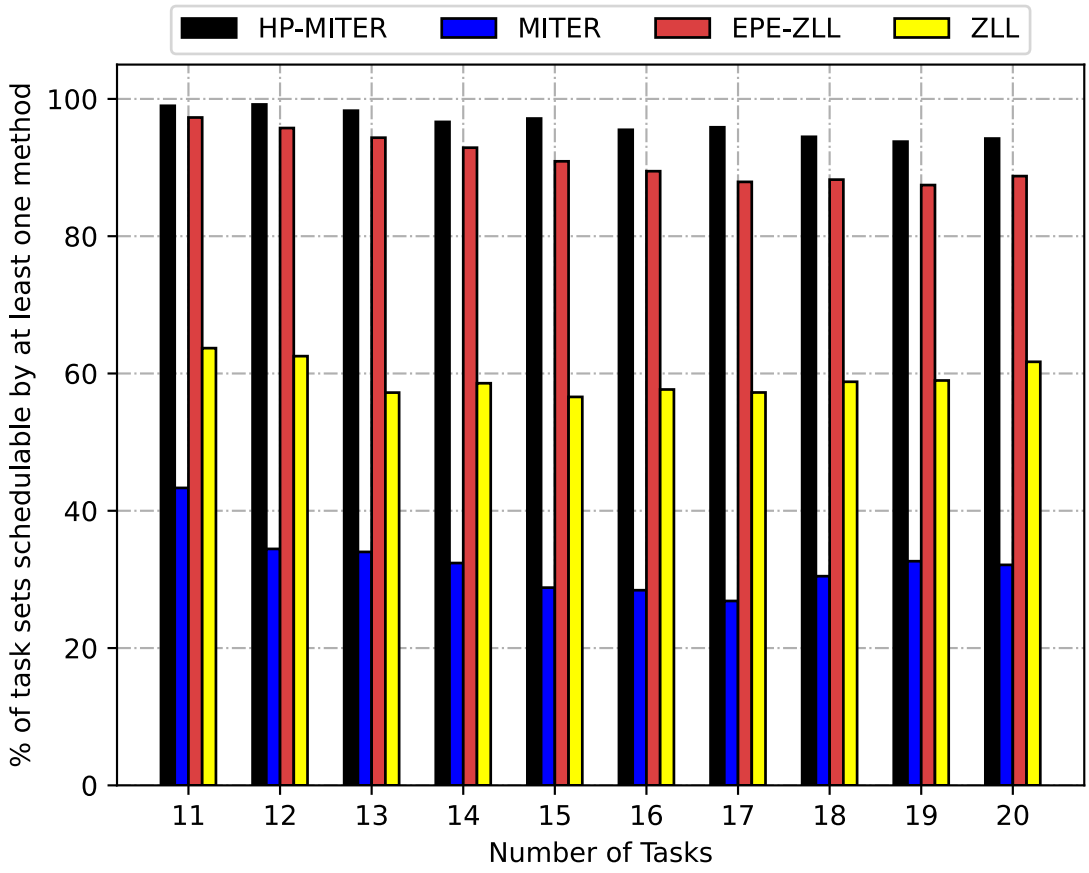


Figure 2.9: $m = 4$ with $11 \leq n \leq 20$

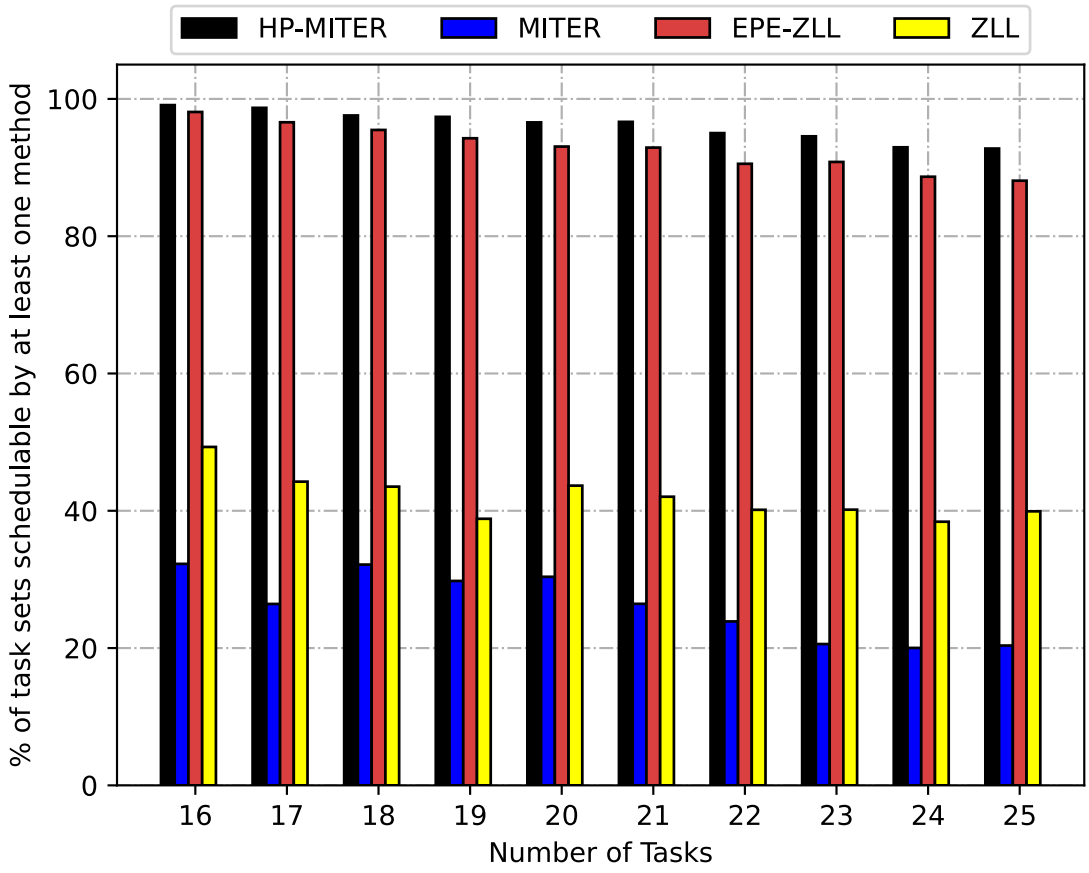


Figure 2.10: $m = 6$ with $16 \leq n \leq 25$

2.7 Conclusion

In this chapter, we consider the problem of priority assignment for global fixed priority scheduling on a multiprocessor platform. We first propose a framework, MITER, that leverages the concept of response time estimation range. It remains optimal for a broader range of response time analysis techniques than Audsley's optimal priority assignment algorithm. We then present an algorithm that judiciously takes advantage of both MITER and heuristic algorithms. Experimental results with various synthetic tasksets show that the proposed ap-

proach significantly outperforms the state-of-the-art algorithms. For future work, we consider extending our optimization framework to different scheduling algorithms and task models, such as global dynamic priority scheduling and the arbitrary deadline task model.

Chapter 3

Partitioned Scheduling with Safety-Performance Trade-Offs in Stochastic Conditional DAG Models

3.1 Introduction

Modeling and scheduling real-time applications on heterogeneous hardware platforms has received significant attention recently. Due to the increasingly complex functionalities of emerging real-time applications (such as human-robot search and rescue (SAR) or precision agriculture [56, 74]), fully utilizing the computing power of heterogeneous hardware becomes an important problem. Directed Acyclic Graphs (DAGs) is one of the powerful tools to model applications with dependencies. In DAGs, tasks can be represented as nodes while edges connect nodes, indicating that there exists dependencies among them [98]. The property of DAGs has the benefit of showing dependency in a straightforward manner [58]. To our knowledge, the study [57] proposes the Heterogeneous Parallel Conditional DAG (HPC-DAG) model, which is known to be the most expressive DAG model [93]. However, HPC-DAGs *do not taken into account the randomness of execution and dynamic changes in environment in real operation.* [9, 57, 83, 93, 103]. We propose the stochastic HPC-DAG model (SHPC-DAG), where nodes and edges are associated with probabilities to capture the

uncertainties at run-time.

Another enhancement in the SHPC-DAG model is the use of probabilistic execution time for each subtask. The majority of scheduling algorithms utilize the worst-case execution times (WCETs) of tasks for analyzing schedulability, which can be overly pessimistic [42]. In addition, the execution times of tasks are often associated with some uncertainty before run-time and may vary due to environmental dynamism and varying sensor inputs [62, 92]. It is then natural to account for the uncertain nature of a system's environment and/or the reliability of its hardware, by representing the execution times of subtasks as random variables and optimizing the expectation of the objective function [62]. However, existing studies on DAG scheduling that model the execution times as random variables [24, 25, 62] all assume that the random execution times follow some well-known probability distributions such as Gaussian or Weibull distributions. Although this assumption allows one to exploit the mathematical properties of the particular distribution to simplify analysis, such distributions do not always hold in real-world applications. For example, if robots are assigned distinct behaviors when they are operating indoors versus outdoors (as in our motivating SAR problem), the execution time of the same task may vary significantly due to environmental dynamism, thus making its probability distribution discrete rather than continuous and difficult to fit by Gaussian or Weibull functions. Another important example has been demonstrated in [44], where the probability distributions of natural image data are shown to be long-tailed, and thus non-Gaussian. Thus, we do not assume a specific type of probability distribution to characterize the random execution times of subtasks.

For the proposed SHPC-DAG system model, we develop a novel partitioned scheduling algorithm called Least Latency Partitioned (LLP) scheduling consisting of a node-to-processor allocation strategy and a scheduling algorithm. Compared with the heuristics proposed in [26, 57], our method utilizes the precedence constraints among subtasks to select the proper

processor for allocation, aiming at maximal parallel execution and minimal end-to-end latency. Importantly, our algorithm has polynomial time complexity in the size of DAG. The Safety-Performance (SP) metric from the study [83] enables a holistic mission objective in safety-critical systems. The SP metric always ensures safety first. The remaining time slack is used to maximize performance, capturing the idea that additional timing slack may not necessarily yield a safer system [83]. For robotic systems with complex mission objectives in dynamic environments, the system may not always need to prioritize safety concerns, especially when no human is involved. Instead, broader mission objectives that also take into account performance may better describe the system needs, which leads to the key idea of SP metric to capture such safety-performance trade-offs. Our proposed scheduling algorithm utilizes such metrics in both the allocation and scheduling processes.

This chapter studies the stochastic version of the restricted assignment scheduling problem (RASP) [9]. A specific model with similar assumptions has also been posed by the 2021 edition of the Real-time Systems Symposium (RTSS) Industry challenge. To the best of our knowledge, this chapter is the first to consider the stochastic version of the RASP problem (S-RASP) and makes the following contributions.

- We introduce a novel SHPC-DAG task model that captures the stochasticity of the uncertain environment.
- We propose a novel partitioned scheduling algorithm LLP to optimize the SP metric. It leverages the precedence constraints among subtasks to maximize parallel execution. We show the algorithm has polynomial time complexity in the size of the DAG.
- The proposed method LLP outperforms the scheduling algorithm in HPC-DAG [57] under hard timing constraints, and performs even better when changing hard timing constraints to probabilistic constraints. Our simulation results show an improvement

of up to 50% in the acceptance ratio on random synthetic DAG tasksets.

The chapter is organized as follows. Section 3.3 describes the proposed SHPC-DAG model. Section 3.4 gives a brief introduction to the SP metric. Section 3.5 and 3.6 illustrate the proposed allocation strategy and scheduling algorithm. Section 3.7 evaluates the proposed algorithms. Section 2.2 reviews the related work. Finally, Section 3.8 concludes our paper with future research directions.

3.2 Related Work

Existing work on the DAG task model mostly focuses on introducing extensions to the DAG task to describe the complexity of real-time applications [93]. Conditional DAG (C-DAG) introduces a conditional node that represents if-then-else structures for conditional execution [14, 45, 67, 70, 86]. Heterogeneous DAG, is another extension of the DAG task model. It associates each subtask with a tag to specify the kind of processing unit that each subtask should run upon [9, 28, 54, 57, 78, 102]. However, the stochastic nature of unknown execution times during run-time is absent in these studies, and they all use the WCETs of computational subtasks for response-time analysis, which may be overly pessimistic [42].

While the stochastic DAG scheduling problem has been studied in the literature [24, 25, 62, 84], assumptions about specific probability distributions (e.g., the Gaussian distribution) are made for analysis. Ueter et. al. considered a probabilistic version of the C-DAG in [92]. In contrast with our present work, the probabilistic aspects of [92] only focus on the outgoing edges of the conditional node. They still consider the WCETs of nodes in the scheduling process with the assumption of an identical multi-core system, which is different from our S-RASP problem. In our work, we do not make assumptions of specific probability distributions for random execution times of subtasks. In addition, we consider the probabilistic

nature both in the computing and conditional nodes, and propose a general solution to the scheduling problem on heterogeneous platforms.

Searching for an optimal schedule for tasks modeled by DAGs is considered to be NP-hard, therefore most existing scheduling algorithms are heuristics [58]. Recently, Baruah [9] formulated the scheduling problem as an integer linear programming (ILP) problem, which is an exact test with time complexity $\mathcal{O}(n^3)$ that may serve as a baseline against which the performance of heuristics can be compared. However, the stochastic nature of execution times of computational kernels is absent in this recent work as well.

3.3 System Model

3.3.1 Motivating Applications

Many applications of real-time systems, such as autonomous robots, operate in changing environmental conditions (called *contexts*) [71, 79, 81]. To clarify our work, we consider the following definition:

Definition 3.1. (Context ($\mathcal{CTX}$)). A context of operation for a real-time system is a set of detectable environment features that are associated with a set of computational tasks and timing requirements. Each distinct context (including the unknown context) is associated with an integer index that takes values in a finite set,

$$ctx \in \mathcal{N}, \mathcal{N} = \{0, 1, 2, \dots\} \tag{3.1}$$

Contexts are required system behavior (computation and timing requirements) based on the specific nature of the operational environment. We argue that although operational

environments can contain various uncertainties and unknowns (especially for robots) as well as continuous inputs to a system that are not easily quantizable, for many applications there still remain detectable environment features that dictate a system's desired behavior and timing requirements. For instance, when a mobile robot traverses an environment with humans nearby, the responsiveness of vision-based detection and robot motion controllers is critical as collisions with humans are unacceptable [63]. On the other hand, a recent trend in robotics is that not all collisions are bad [64], and thus if a robot is operating with no human presence the requirements on vision and motion responsiveness can be relaxed. In this example, the vision-based detection of nearby humans (e.g., see [29]) indicates the nature of the operational environment, and from this (potentially imperfect) detection the required system behavior can be set. We aim to capture such relationships in the models and algorithms proposed in this work.

Safety standards for software development in safety critical systems vary in different domains [47]. Development of software for general safety critical systems usually follows IEC 61508 [15, 47], which performs risk analysis and specifies the appropriate safety integrity level (SIL) for each function. Another example is the RTCA DO-178B that is applied to airborne software development [60]. In this work, we define the notion of time sensitivity level (TSL), which refers to the safety levels of subtasks and reflects any priorities that may exist between functionalities and subsystems under a specified safety standard.

Definition 3.2. (Time sensitivity level (TSL)). TSL is the criticality of the subtask regarding its functionality, determined by software safety standard and application requirements. It takes values in the set of positive integers,

$$tsl \in \mathcal{Z}^+, \mathcal{Z}^+ = \{1, 2, 3, ...\} \tag{3.2}$$

According to application-dependent criticalities, TSLs of all subtasks help to form an *ordering* in the taskset. The larger the TSL, the earlier the subtask should be guaranteed to schedule for meeting application requirements. For example, subtasks along a computational path for pedestrian detection would certainly be scheduled as early as possible for autonomous applications [108], without violating precedence constraints. Note that if the criticalities of the subtasks (i.e., functionalities) are not specified in the application requirements and safety standards, we set all TSLs to the same values by default.

3.3.2 Assumptions

We state our assumptions first for better clarity. We study a single-rate DAG with the event-driven activation model, which assumes the same activation periods for all subtasks in the DAG [95]. Contrary to the multi-rate DAG in [94] where all subtasks activate periodically and independently, the event-driven activation model is a single-rate model where subtasks are activated simultaneously and execute in the order constrained by data dependencies. Therefore, the intra-task interference could be eliminated as in single-rate DAG, each subtask is executed only once by following the precedence constraints. In [94], a method is proposed to convert a multi-rate DAG with timing constraints to a single-rate DAG. Specifically, each subtask executes a number of jobs within one hyper-period according to its period, and synchronization constructs are inserted in the DAG to guarantee that the original multi-rate DAG and the converted single-rate DAG have equivalent temporal requirements. Due to the existence of this conversion [94], our results for single-rate DAG could be applied to multi-rate DAGs without loss of generality. Finally, we assume DAG subtasks are executed *non-preemptively*.

3.3.3 Stochastic Heterogeneous Parallel Conditional DAGs

We propose a new DAG task model, the *Stochastic Heterogeneous Parallel Conditional Directed Acyclic Graph (SHPC-DAG)*, which incorporates both timing and resource constraints for safety-critical systems. Our observation is that computational pipelines may be executed conditionally due to uncertainty, recognizing that not all outcomes of computational subtasks can be perfectly predicted in real-world applications (e.g., extreme events). Also, the execution times of tasks may not be perfectly known before run-time and may vary significantly *online* due to environmental dynamism and conditions of the hardware platform. Scheduling according to the worst-case execution time (WCET) may be overly pessimistic. Hence, the SHPC-DAG model contains two significant extensions compared to the most recent HPC-DAG model. The first is that the execution time of a computational subtask is assumed to follow a probability distribution rather than the WCET. The second is that stochastic conditional nodes and probabilities associated with their outgoing edges are added to model computational path selection.

Definition 3.3. A real-time application can be modeled by a *SHPC-DAG*, denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, f, Type, Tag, TSL)$, where:

- $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ is the set of nodes in the graph. Each of the n nodes $v_i \in \mathcal{V}$ is a subtask which represents a block of code that executes on a specific type of processor. Hence, there are n subtasks in the application.
- $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges among subtasks that indicates the data dependencies, with associated probabilities indicating the likelihood of edge traversal during execution. This will be introduced in the next subsection along with the stochastic conditional node.
- $f = \{f_1, f_2, \ldots, f_n\}$ is the set of probability distribution functions that characterize

the possible values of the subtasks' execution times. Each f_i can either be discrete (i.e., probability mass function (PMF)) or continuous (i.e., probability density function (PDF)). For both PMF and PDF, the inputs are the random execution times of subtasks. For PMF, the execution times take discrete values in the set of positive integer numbers $Z^+ = \{1, 2, 3, ...\}$. Each discrete execution time will be associated with a probability (i.e., the function output) and the sum of all probabilities equal to one. For PDFs, the random execution times take continuous values in a certain range with lower and upper bounds. The integration of probability density over this range equals to one.

- $Type = \{type_1, type_2, \ldots, type_n\}$ is the set of operation types of all nodes in $\mathcal{V}$. This may include stochastic *Conditional* nodes, for which the formal definition is given in section 3.3.4. For applications such as autonomous driving, other potential node types may include *Sensor* and *Sync*, which represent the sensor inputs and synchronization constructs of the application, respectively.

- $Tag = \{tag_1, tag_2, \ldots, tag_n\}$ indicates the type of processors that each task in $\mathcal{V}$ should run on, e.g., CPU, GPU, etc.

- $TSL = \{tsl_1, tsl_2, \ldots, tsl_n\}$ is the set of TSLs for all nodes in the graph.

Lastly, we adopt the formal definitions of safety-critical paths and nodes in [83] to equip SHPC-DAG with safety-critical features.

Definition 3.4. (Safety-critical path [83]). A safety-critical path is a task chain that achieves critical functionality/feature regarding the safety aspect of the application (also known as a critical cause-effect chain).

Let $\mathcal{C}$ denote the set of DAG paths that are safety-critical where each path is indexed as

ℓ_i, $\ell_i \in \mathcal{C}$. There may exist many paths in a SHPC-DAG but not all of them are critical to the safety aspect of the application. The determination of safety-critical paths is highly dependent on the application functionality and requirements.

Definition 3.5. (Safety-critical node [83]). Safety-critical nodes are subtasks along computational paths that achieve some safety-critical functionality of the application.

$$s = \begin{cases} True, \text{safety-critical node} \\ False, \text{non-safety-critical node} \end{cases}$$

Note that the deadline of a safety-critical node may be independent from the timing requirements of the paths it is associated with. For some autonomous applications, meeting the deadlines of the application or computational path alone cannot guarantee the safety in real operation. For example, in Simultaneous Localization and Mapping (SLAM), mapping should always be updated on time to ensure that subsequent tasks operate with a correct map of the environment, which is critical for safety [77]. Thus, the mapping subtask itself will have an individual deadline in addition to the deadline of the entire computational path. As in the above definition, an additional set of node attributes $S = \{s_1, s_2, \ldots, s_n\}$ with boolean values are used to indicate whether the nodes are safety-critical or not. The set of safety-critical nodes is denoted as $\mathcal{V}_c$, and we have $\mathcal{V}_c \subseteq \mathcal{V}$.

An illustrative system modeled by the proposed DAG framework is given in Figure 3.1. Such a computational setup may be seen as a subsystem in, for example, autonomous driving or robotic search and rescue. Robotic systems are equipped with sensors (e.g., cameras, lidars), which take measurements of the surrounding environment. The sensors' measurements could be asynchronous due to different sample rates. In the Robot Operating System (ROS) framework, it provides several configurable sensor fusion packages that can handle asynchronous multi-rate measurements [75], which is denoted as the sensor fusion node in

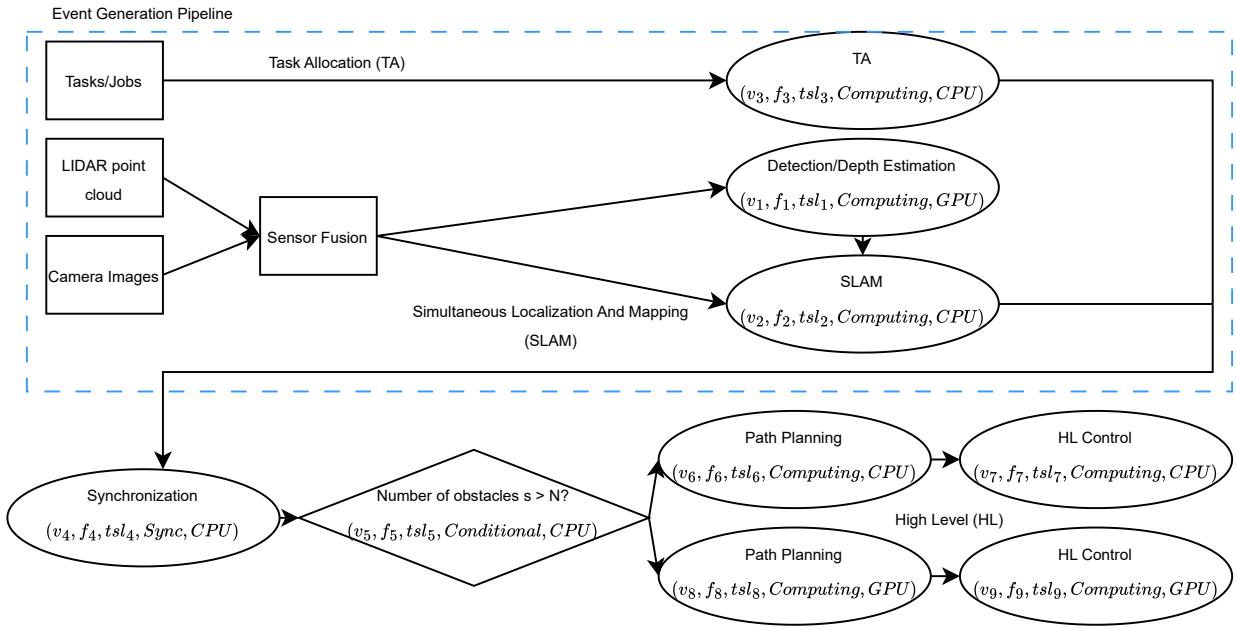


Figure 3.1: An illustrative example of our *SHPC-DAG* model for subsystems seen in autonomous driving.

Figure 3.1. The combined inputs are then passed to the computational subtasks (e.g., path planning, localization and mapping) for further processing. Since the whole subsystem about path planning is crucial for safety problems in autonomous driving, all the path planning subtasks in both branches are labeled as safety-critical nodes. Based on the environment that is of a cluttered, uncertain nature, the conditional node (denoted as a diamond box in the figure) selects which computational path is used for path planning. We explain that in the next subsection.

3.3.4 Stochastic Conditional Nodes in DAG Models

SHPC-DAG contains a novel stochastic conditional node structure to capture the stochastic nature of real-time applications during run-time. This type of node selects the execution paths in DAGs based on the *contexts* in which the systems operate. Let m denote the number

of possible contexts that can occur during execution (e.g, environmental conditions), and let $\mathcal{CTX} = \{ctx_1, ctx_2, \ldots, ctx_m\}$ denote the set of all contexts. The specific context that the application will have during execution is unknown before run-time. During execution, the context will be determined by real-time information (e.g., sensor inputs) and the embedded detector. In most existing works, they consider the context returned by the detector as a single deterministic context across $\mathcal{CTX}$. However, due to the imperfectness of the detector, it may misjudge the current context. To consider such case, we propose to use a probability distribution to represent the probability of occurrence of all possible contexts rather than a single deterministic result. The attributes (e.g., execution time, processor tag) of the nodes in the *SHPC-DAG* will depend on the corresponding context.

Definition 3.6. (Probability Distribution of Context Detection) The detected context of the real-time application will follow a probability distribution $P(ctx),\ ctx \in \mathcal{CTX}$ based on real-time information. The sum of probabilities associated with all possible contexts will be 1, i.e., $\sum_{i=1}^{m} P(ctx_i) = 1$, where m is the number of distinct contexts.

Let k denote the number of all possible branches (i.e., outgoing edges) that a stochastic conditional node has and $\mathcal{B} = \{b_1, \ldots, b_k\}$ denote the set of all branches. Each branch is associated with a distinct context ctx. We assume $k \leq m$ to make sure that each branch represents a distinct context and execution path that will not duplicate. During execution the stochastic conditional node will select only one branch to execute with probability $P(b_i)$, thus $\sum_{i=1}^{k} P(b_i) = 1$.

The probability that a real-time application executes along a specific execution branch depends on the detected context during run-time. When receiving the input data from sensors, the embedded detector may determine the current context is ctx_i. However, due to the imperfectness of context detection, the actual context could be ctx_j $(j \neq i)$. This stochasticity needs to be considered when calculating the probability $P(b_i)$ of selecting each branch.

Definition 3.7. (Probability for Each Branch of Conditional Node). Suppose a stochastic conditional node has k outgoing branches under the assumption of m possible contexts in total and $k \leq m$. The branch b_i will be selected only if the detected context ctx_j satisfies the *condition* of branch b_i (determined by the conditional statement in conditional node). The probability $P(b_i)$ of selecting branch b_i is

$$P(b_i) = \sum_{j=1}^{m} P(b_i|ctx_j) \cdot P(ctx_j) \tag{3.3}$$

which should satisfy

$$\begin{cases} \sum_{i=1}^{k} P(b_i) = 1 \\ \sum_{j=1}^{m} P(ctx_j) = 1 \end{cases}$$

Note that the $P(b_i|ctx_j)$ term will be determined by the accuracy of the detector[1]. The classification of the context is conducted by the embedded detector in robots [29].

Figure 3.2 illustrates the idea of stochastic conditional node considering two possible contexts: illuminated environment (i.e., ctx_1) and dark environment (i.e., ctx_2). The conditional statement embedded in the conditional node determines whether the data collected is of high quality or low quality (e.g., the collected images are clear or not), and selects execution branch accordingly. We consider the stochasticity by associating edges of conditional nodes with conditional probabilites. For example, it is highly likely that robot could collect clear images of surroundings in a well-illuminated environment (i.e., $P(b_1|ctx_1)$). However, the context detector is imperfect. The result of detection could be a dark environment, but still with good quality images (i.e., $P(b_1|ctx_2)$). Either case will lead to the upper execution

[1] In cases where perfect context knowledge exists, our definition for the stochastic conditional node collapses to a classical conditional node with fixed probabilities associated with each outgoing edge, as in [92].

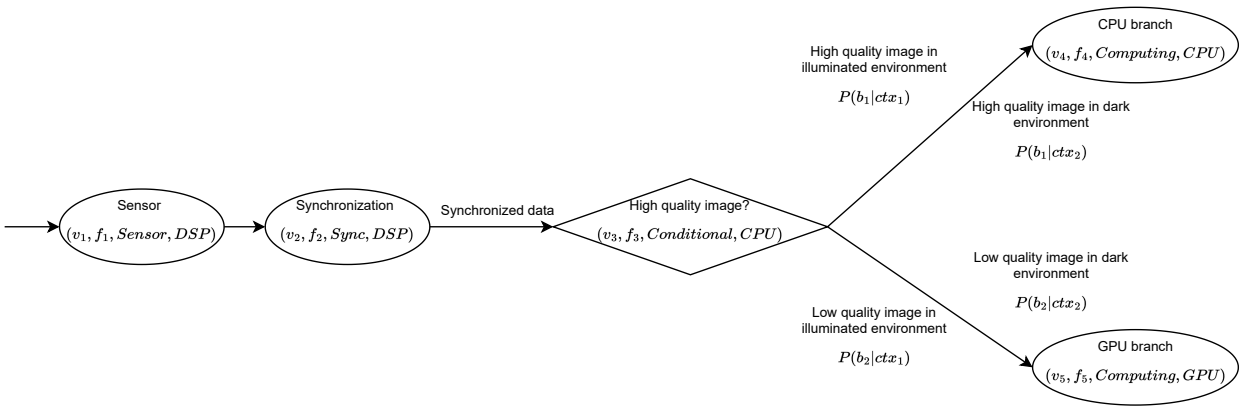


Figure 3.2: An illustrative example of the structure of our proposed stochastic conditional node.

branch.

3.4 Safety-Performance Metric

Our proposed scheduling approach (Sections 3.5 and 3.6) requires an objective function to optimize when selecting appropriate schedules. While a typical function such as makespan can be used, we consider a metric that identifies safety-critical nodes and paths. We aim to design a scheduler that always prioritizes safety and takes advantage of timing headroom to improve system performance. Specifically, we exploited the Safety Performance (SP) metric of the study [83], which defines a series of penalties/rewards based on violating/satisfying timing constraints in a DAG. In this work, the term *safety* refers to the concept of *nominal safety*, where the timing of software modules below predefined thresholds can contribute to unsafe behaviors in autonomous systems. Its definition has been well cited in the autonomous vehicles (AV) area [80, 108].

Below we briefly summarize the definition of SP metric, starting with the calculation of the response time probability distributions for the nodes and paths in the graph model.

3.4.1 Response time probability distributions

When calculating the response time distribution of nodes and safety-critical paths, we need to consider the maximum and summation operations of independent random execution times characterized by probability distributions. For many data-driven projects in practice, the analytic form of the probability density function of execution time may not be available. Instead we have measurement data from experiment or simulation in different environments, i.e., a probability mass function (PMF) defined at discrete points. The subtask will be executed multiple times on the target hardware according to several distinct context of operations and measure execution times under each of the context [2]. Therefore, in this work we analyze the random variables characterized by PMFs.

Consider X and Y as two independent random variables with PMFs respectively. Let Z be the maximum of X and Y, i.e., $Z = \max(X, Y)$, S be the summation of X and Y. We define the operations as follows.

Maximum Operation. Consider a node in SHPC-DAG that has multiple predecessors (i.e., the nodes from which there exists a path to the current node) of which the response times are characterized as probability distributions. The earliest possible time instant of this node to start execution is determined by the maximum response times among all its predecessors, denoted as starting time (ST). From basic probability theory, the cumulative distribution function (CDF) $F_Z(z)$ of maximum of two independent random variables X and Y is given as,

$$\begin{aligned} F_Z(z) &= P(\max(X, Y) \leq z) = P(X \leq z, Y \leq z) \\ &= P(X \leq z)P(Y \leq z) = F_X(z)F_Y(z) \end{aligned} \tag{3.4}$$

Based on CDFs of X and Y, we could find the PMF of z by,

$$P(z) = F_X(z)F_Y(z) - F_X(z-1)F_Y(z-1) \tag{3.5}$$

where $z \geq 1$ and takes only integer values, this process terminates when $F_Z(z) = 1$ since $F_Z(z) = \sum_{z=1} P(z)$. For node with multiple predecessors (e.g., more than two), (3.4) and (3.5) can be extended in the same way.

Summation Operation. The density function of the summation $\mathcal{Z}$ of two independent random variables $\mathcal{X}$ and $\mathcal{Y}$ is the convolution of the density functions of the individual random variables [66, 68].

$$P(\mathcal{Z} = z) = \sum_{k=-\infty}^{\infty} P(\mathcal{X} = k)P(\mathcal{Y} = z - k) \tag{3.6}$$

For discrete convolution in the time domain, we have each probability mass represented as a data point in an array of length N. To speed up the calculation, the Fast Fourier transform (FFT) is applied to efficiently compute the above multiplication when N is a power of 2, yielding improved time complexity $O(Nlog(N))$. It is known that the convolution of two functions is the inverse Fast Fourier transform (IFFT) of the product of their individual Fourier transforms, therefore we could obtain the PMF of $\mathcal{Z}$ by taking the IFFT of the product.

Note that the response time distribution of safety-critical paths is not simply the linear summation of the nodes' execution times along the path. Instead, the precedence constraints in DAG and resource constraints should both be taken into account when analyzing the response time distributions, which is different from the work [66] that also considers tasks with probabilistic parameters.

Definition 3.8. (Starting time). The probability distribution of node v_i's starting time (ST) is determined by the maximum response time (RT) distributions, defined by (3.4) and (3.5), among all its predecessors and the nodes scheduled ahead of it on the same processor,

$$ST_i = \max_{v_j \in pred(v_i) \cup ahead(v_i)} RT_j \tag{3.7}$$

Definition 3.9. (Response time). The probability distribution of node v_i's response time (RT) is the convolution of its starting time (ST) and execution time (ET), both characterized by probability distributions, where * denotes the convolution operation

$$RT_i = ST_i * ET_i \tag{3.8}$$

The response time of a safety-critical path is equal to the response time of its ending subtask.

3.4.2 Safety-Performance Metric

To begin with, we give the formal definition of penalty in the SP metric when safety-critical paths and/or critical nodes violate their timing constraints [83].

Definition 3.10. For a given schedule $\mathcal{S}$, the penalty for safety-critical paths that violate their timing constraints is given by:

$$f_{\text{cp}}^{\text{p}}(\mathcal{S}) = \sum_{\ell_i \in \mathcal{C}_{\text{us}}} p_{\text{cp}}(P(R_{\ell_i} > \tau_{\ell_i}) - \lambda_{\ell_i}) \tag{3.9}$$

Here $\mathcal{C}_{\text{us}}$ denotes the set of safety-critical paths that violate a *probabilistic* timing constraint, i.e., $P(R_{\ell_i} > \tau_{\ell_i}) > \lambda_{\ell_i}$. The random variable R_{ℓ_i} denotes the response time of a critical path ℓ_i. τ_{ℓ_i} is the *nominally safe* response time (i.e., deadline) for path ℓ_i, and λ_{ℓ_i} is the

probabilistic timing constraint for ℓ_i. $f^{\mathrm{p}}_{\mathrm{cp}}(\mathcal{S})$ represents a penalty term (p) for critical path violations (cp) based on schedule $\mathcal{S}$ with a generic penalty function $p_{\mathrm{cp}}(\cdot)$.

Equation (3.9) can be interpreted as penalizing every violating critical path based on the *deviation* from its probabilistic timing constraint. This equation can also deal with systems that have hard timing constraint by setting $\lambda_{\ell_i} = 0$, which requires that the response time will always meet the nominally safe response time, i.e., $R_{\ell_i} \leq \tau_{\ell_i}$. If none of the safety-critical paths violate their timing constraints with schedule $\mathcal{S}$, the penalty $f^{\mathrm{p}}_{\mathrm{cp}}(\mathcal{S}) = 0$.

Definition 3.11. For a schedule $\mathcal{S}$, the penalty for the safety-critical nodes that violate their timing constraints is given by:

$$f^{\mathrm{p}}_{\mathrm{cn}}(\mathcal{S}) = \sum_{v_i \in \mathcal{V}_{\mathrm{us}}} p_{\mathrm{cn}}(P(R_{v_i} > \tau_{v_i}) - \lambda_{v_i}) \tag{3.10}$$

Here $\mathcal{V}_{\mathrm{us}}$ denotes the set of safety-critical nodes that violate a *probabilistic* timing constraint, i.e., $P(R_{v_i} > \tau_{v_i}) > \lambda_{v_i}$. Similar to the terms in the definition of penalty for critical paths, R_{v_i}, τ_{v_i} and λ_{v_i} denote the response time, nominally safe response time, and probabilistic timing constraint of v_i, respectively. Given schedule $\mathcal{S}$, if all safety-critical nodes can meet their corresponding timing constraints, then $\mathcal{V}_{\mathrm{us}} = \emptyset$ and $f^{\mathrm{p}}_{\mathrm{cn}}(\mathcal{S}) = 0$, yielding no penalty.

Next, we introduce rewards gained when timing constraints for safety-critical paths are all satisfied. The reward term can be non-zero only when there exists no critical path or node that violates its timing constraints.

Definition 3.12 (*Path Reward*)**.** The reward for all paths that satisfies safety-critical timing constraints, is defined as:

$$f^{\mathrm{r}}_{\mathrm{path}}(\mathcal{S}) = \Big[\sum_{\ell_i \in \mathcal{C}} r^{\mathrm{s},\ell_i}_{\mathrm{path}}(\lambda_{\ell_i} - P(R_{\ell_i} > \tau_{\ell_i})) + r^{\mathrm{p},\ell_i}_{\mathrm{path}}(R_{\ell_i})\Big] + \sum_{\ell_i \notin \mathcal{C}} r^{\mathrm{p},\ell_i}_{\mathrm{path}}(R_{\ell_i}) \tag{3.11}$$

Definition 3.13 (*Node Reward*)**.** The reward for all nodes that satisfies safety-critical timing constraints, is defined as:

$$f^{\mathrm{r}}_{\mathrm{node}}(\mathcal{S}) = \Big[\sum_{v_i \in \mathcal{V}_c} r^{\mathrm{s},v_i}_{\mathrm{node}}(\lambda_{v_i} - P(R_{v_i} > \tau_{v_i})) + r^{\mathrm{p},v_i}_{\mathrm{node}}(R_{v_i}) \Big] + \sum_{v_i \notin \mathcal{V}_c} r^{\mathrm{p},v_i}_{\mathrm{node}}(R_{v_i}) \tag{3.12}$$

Here $f^{\mathrm{r}}_{\mathrm{path}}(\mathcal{S})$ and $f^{\mathrm{r}}_{\mathrm{node}}(\mathcal{S})$ represent the reward term (r) for *all* paths and nodes respectively, regardless of whether they are safety-critical or not. The generic reward functions $r^{\mathrm{s},\ell_i}_{\mathrm{path}}(\cdot), r^{\mathrm{p},\ell_i}_{\mathrm{path}}(\cdot)$, $r^{\mathrm{s},v_i}_{\mathrm{node}}(\cdot), r^{\mathrm{p},v_i}_{\mathrm{node}}(\cdot)$ define the weights to reward safety margins (s) and system performance (p). As task chains and node functionality are both important in evaluating the overall system performance, two reward functions are given for each category.

Finally, with all terms defined, we design our scheduler to optimize the overall safety-performance metric defined as a weighted sum of terms (3.9)-(3.12). This yields a schedule that trade off safety and system performance relative to probabilistic timing constraints.

An example for the penalty functions is a linear operator, e.g., $p_{\mathrm{cp}}(x) = \gamma x$, yielding a weighted penalty for timing constraint violation. For the safety reward functions, an example is a *sigmoid function*, e.g., $r^{\mathrm{s}}_{\mathrm{path}}(x) = 1/(1 + \exp^{-x})$, which models diminishing returns of rewards for increasing safety margins. For the non-critical, performance-based reward functions one could consider the right-hand tail of the response time distribution, e.g., $r^{\mathrm{s}}_{\mathrm{path}}(R_{v_i}) = P(R_{v_i} > x)$, with x defined as a response time threshold that yields higher quality computational output. One clear case of such a timing-performance relationship can be seen in the *traveling salesperson problem (TSP)* [4]. Finally, it is important to note that instead of transforming the conditional DAG into an ordinary DAG (e.g., [86]), we directly analyze the conditional DAG on a *path-basis* for calculating the SP metric (details in Section 3.6). This avoids the cost of additional DAG transformation and helps focus on the conditional branching of safety-critical paths.

3.5 The LLP Scheduler: Subtask Allocation

We consider partitioned scheduling for the *SHPC-DAG* task model on heterogeneous hardware platforms. Partitioned scheduling has low run-time overhead due to the absence of thread migration compared to global scheduling [57, 59], and the proper partitioning (i.e., allocating subtasks onto the processors) is typically determined before scheduling [26]. In this section, we propose a novel partitioned scheduling algorithm called Least Latency Partitioned (LLP). It utilizes the proposed *SHPC-DAG* task model and SP metric to maximize the probabilistic reward of all safety-critical paths and nodes, while meeting their timing and performance requirements.

The existing work of HPC-DAG proposed two different allocation methods that allocate the subtasks on a minimal number of processors and schedule with Earliest Deadline First (EDF) algorithm on each processor [57]: Sequential Allocation (HPC-s) and Parallel Allocation (HPC-p). In HPC-s, subtasks with the same processor tags could be allocated on a single processor if the schedulability test is passed. If HPC-s fails, HPC-p will remove one subtask at a time and reallocate it on another available processor until the remaining subtasks could be schedulable on a single processor.

Our proposed LLP allocation method takes additional factors into account to maximize parallel execution of subtasks and to reduce end-to-end path response times. First, LLP avoids situations where processors remain idle during run-time when subtasks are ready to execute. Since we consider a non-preemptive model, the computing power of all available processors should be utilized to maximize parallel execution. Second, LLP considers the precedence constraints among subtasks during allocation, while HPC-DAG uses simple heuristics (i.e., best-fit (BF) or worst-fit (WF)) to determine the allocation. Third, in addition to precedence constraints, LLP further sorts the subtasks according to the their TSLs, while in HPC-DAG

the orderings of subtasks that do not belong to the critical path (i.e., the path with the longest response time) are not specified.

The main idea of LLP is to utilize precedence constraints among subtasks and TSLs in both allocation and scheduling processes to reduce end-to-end latency and maximize the SP metric of safety-critical paths and nodes. The LLP allocation process can be divided into the following steps.

Assign time sensitivity levels (TSLs) to subtasks. The TSLs of subtasks are highly dependent on the application requirements and the applied safety standards. Based on the functionality regarding the safety aspect of the application, each path will be assigned a TSL according to the specific safety standard (e.g., IEC 61508) that is applied to the system. For safety-critical nodes along the path, they will be assigned a higher TSL. Note that for subtasks that are predecessors of the ones along the safety-critical paths, we use the *backward propagation* to update their TSLs by inheriting the maximal TSL among all their successors. If the TSLs of the subtasks are not specified in the application requirements, we sort all the safety-critical paths in the *SHPC-DAG* by expected value of lengths in *decreasing* order. Subtasks along the safety-critical path with largest expected value of length will be assigned highest TSL. The subtasks along shorter safety-critical paths will be assigned incrementally smaller TSLs until all subtasks are traversed.

Partition SHPC-DAG based on tags. Based on the processor tags (e.g., CPU, GPU) associated with subtasks, we divide the SHPC-DAG into different subgraphs. Each processor tag will be associated with a set of subtasks and processors, respectively, that have the same tags. Subtasks will only be allocated to their corresponding set of processors.

Sort subtasks according to dependencies and TSLs. To respect the precedence constraints in *SHPC-DAG*, we use topological sort to obtain complete linear ordering(s) of the subtasks. If two subtasks are independent with each other, the relative order is determined by the TSLs of the subtasks. The larger the TSL, the earlier the subtask appear in the ordering. This is to guarantee that the more time-sensitive nodes are selected to allocate ahead when computing resources are less occupied.

Allocate subtasks to processors by LLP. The procedure of allocating subtasks to computing resources is summarized in Algorithm 3. The subtasks are traversed from the beginning of each ordering (Line 1-3). Only the processors with the same processor tag as the current node will be considered at each iteration.

If the current subtask v_i has no predecessor with the same tag, we traverse all the processors with the same tag as v_i and use the worst-fit (WF) strategy to select the target processor for allocation (Line 4-5). The reason is that if v_i has no predecessor with the same tag, then it is ready to execute once all its predecessors (with different tags if any) finish execution and we want to find the least occupied processor for it to start as soon as possible.

If v_i has exactly one predecessor $pred_i$ with the same tag, and $pred_i$ has no other successors with the same tag (i.e., *siblings* of v_i), we allocate v_i to the same processor p_i of $pred_i$ (Line 6-9). Since v_i can only start execution after $pred_i$ due to the precedence constraint, allocating v_i to a different processor p_j will force it to remain idle until $pred_i$ finishes execution, while during this interval, other subtasks without dependencies between v_i could execute on p_j to maximize parallelism.

If v_i does have one predecessor with the same tag and one of its siblings has been allocated to the same processor of the predecessor, then we traverse all processors with the same tag

as v_i and find the target processor by WF to allow v_i to start execution as soon as possible (Line 10).

If v_i has multiple predecessors, due to the dependencies, v_i can only start execution once all of its predecessors finish execution. Among all the processors on which v_i's predecessors have been allocated, find the most occupied one by the Best-fit (BF) strategy (Line 11-14) among all processors with the same tag.

Algorithm 3 LLP Allocation Algorithm

Input:
T: Set of processor tags.
L: Set of sorted orderings for all types of processor tags.
n: Cardinality of the set $n = |T| = |L|$.

```
1:  Initilaize index k ← 1.
2:  while k ≤ n do t ← T_k, l ← L_k
3:      for (v_i : l) do
4:          if v_i has no predecessor in l then
5:              Allocate v_i to the processor with tag t by WF.
6:          else if v_i has one predecessor in l then
7:              if None of v_i's siblings has been allocated then
8:                  Allocate v_i to the same processor
9:                  of its predecessor.
10:             else Allocate v_i to the processor with tag t by WF.
11:             end if
12:         else if v_i has multiple predecessors in l then
13:             Among all the processors onto which v_i's
14:             predecessors have been allocated, allocate v_i to
15:             the processor with tag t by BF.
16:         else
17:                 The allocation of v_i fails. Terminate and return failure.
18:             end if
19:         end for
20:     end while
```

We provide a sketch of proof by cases to demonstrate that LLP will always find an allocation with smaller or equal end-to-end latency than HPC-s and HPC-p methods in [57].

Proof. If for a specific tag t, the platform only has one corresponding processor with the same

tag, then both LLP and methods of HPC-DAG will try to allocate all nodes on the single processor. The results of allocation will be the same. Else if there are multiple processors with tag t, allocation based on HPC-s and HPC-p will try to fill processors sequentially and may leave remaining processors idle during the execution, while LLP tries to utilize the computing power of all available processors to reduce latency. Assuming the same scheduling algorithm for each single processor for both methods, in the cases where allocation by HPC-p occupies all processors as well, LLP additonally utilizes the precedence constraints and parallelism among subtasks to maximize parallel execution, thus generating an allocation with less end-to-end latency for safety-critical paths. □

3.6 The LLP Scheduler: Scheduling

3.6.1 Scheduling Algorithm

For a single-rate (explained in Section 3.3.2) and non-preemptive SHPC-DAG, we propose a scheduling algorithm coupled with the allocation algorithm. In the work about HPC-DAG [57], the authors proposed that Earliest Deadline First (EDF) and Deadline Monotonic (DM) could be a representative scheduler for each processor, given that the intermediate deadlines of subtasks have been assigned. Their methods respect the precedence constraints in DAG automatically but lack the considerations of TSL and stochasticity in our problem.

We illustrate the idea of LLP scheduling algorithm. A feasible schedule on each processor should satisfy two requirements: (1) the precedence constraints among subtasks should not be violated and (2) subtasks with higher TSLs should be guaranteed to execute ahead. As stated in Section 3.5, the SHPC-DAG is partitioned by processor tags. For each processor tag, the topological sorting is applied to obtain the linear ordering Θ of all subtasks per tag.

For each processor, the set of subtasks allocated upon it is denoted as V. For these subtasks, the initial schedule S_i is obtained by sorting subtasks according to Θ with the same tag (Line 3-7). The schedule is then adjusted by re-sorting subtasks with higher TSLs to the front of the schedule without violating the precedence constraints (Line 8-13). Pseudo code of the algorithm is given in Algorithm 4.

Algorithm 4 LLP Scheduling Algorithm

Input:

Θ: Linear ordering of subtasks with the same tag as current processor.
V: Set of subtasks allocated to the current processor.
N: Cardinality of the ordering Θ, $N = |\Theta|$.
n: Cardinality of the set V, $n = |V|$.

Output:

S: Final schedule of the current processor determined by LLP.

```
1: Initialize indices i, j, k ← 1.
2: Initial schedule S_i = ∅, final schedule S = ∅.
3: for i = 1:N do
4:     Search each subtask v_i in Θ from the beginning.
5:     if v_i ∈ V then
6:         S_i.push_back(v_i).
7:         remove v_i from Θ.
8:     end if
9: end for
10: Adjust initial schedule S_i by LLP.
11: for j = 1:n do
12:     for k = 1:j do
13:         if  S_i[k].TSL ≤ S_i[k+1].TSL
14:              &&(!is_predecessor(S_i[k],S_i[k+1])) then
15:             swap(S_i[k], S_i[k + 1]).
16:         end if
17:     end for
18: end for
19: S = S_i.
20: return S.
```

3.6.2 Schedulability analysis

In this section, we assume that subtasks have already been assigned TSLs and safety-critical labels, and they have been allocated on the corresponding types of processors. We present the schedulability analysis to test if all tasks respect their deadlines when scheduled by the LLP algorithm and SP metric.

Definition 3.14. Let $\mathcal{G}$ be a *SHPC-DAG* that is allocated to the hardware platform and the obtained schedule is S under LLP algorithm. Let $\mathcal{V}$ be the set of all safety-critical nodes and $\mathcal{L}$ be the set of all safety-critical paths. Denote τ_{v_i} and τ_{l_i} as the deadlines of the safety-critical nodes and paths respectively. The λ_{v_i} and λ_{l_i} are the probabilistic timing constraints used in SP metric.Task $\mathcal{G}$ is schedulable by LLP if and only if,

$$\begin{cases} P(R_{v_i} > \tau_{v_i}) - \lambda_{v_i} < 0, \forall v_i \in \mathcal{V} \\ P(R_{l_i} > \tau_{l_i}) - \lambda_{l_i} < 0, \forall l_i \in \mathcal{C} \\ (f^r_{node}(S) + f^r_{path}(S)) - (f^p_{cn}(S) + f^p_{cp}(S)) > 0 \end{cases}$$

Based on the definitions in Section 3.4.2, the SP metric is defined as the sum of all penalties and rewards. If the timing constraints of safety-critical nodes/paths are violated, the whole taskset is deemed as unschedulable. The constraints of these safety-critical nodes/paths can be independent from those of the whole taskset and must be met to ensure safety. Based on this, if the sum of rewards (i.e., $f^r_{node}(S)$ for nodes and $f^r_{path}(S)$ for paths) is greater than penalties (i.e., $f^p_{cn}(S) + f^p_{cp}(S)$), then task $\mathcal{G}$ is deemed as schedulable under LLP algorithm and SP metric.

3.6.3 Time complexity analysis

We give the time complexity analysis of the proposed partitioned scheduling algorithm which includes both the allocation and scheduling parts. Denote N as the number of subtasks in SHPC-DAG, and n_t as the number of subtasks for one specific tag (e.g., CPU). Similarly, denote K as the number of all processors and k_t as the number of processor with a specific tag. In general, we have $N \geq n_t$ and $K \geq k_t$ since a SHPC-DAG task might contain computing subtasks that should be executed upon different types of processors.

For the LLP allocation algorithm, the first step is to update the TSLs of each subtask and the subtasks will inherit the TSLs of their successors, if any, in the SHPC-DAG. Since the safety-critical paths and nodes are predefined by users, for each subtask v_i, we need to check the dependency between every subtask and v_i. If the precedence constrains exist, the TSL will be updated accordingly. For each tag this process will have complexity $\mathcal{O}(N * (N-1)/2) = \mathcal{O}(N^2)$. The time complexity of topological sort is $\mathcal{O}(N+E)$, where N is the number of subtasks in graph and E is the number of edges connecting subtasks. It is upper bounded by $\mathcal{O}(N^2)$. During step 4 the worst case will require the subtask to traverse all the processors with the same tag to find the most suitable one, which requires $k_t * \mathcal{O}(n)$ at most. The overall time complexity of allocation is $\mathcal{O}(N^2) + \mathcal{O}(N^2) + k_t * \mathcal{O}(n_t) = \mathcal{O}(N^2)$. For the scheduling part on each processor, the overall time complexity will be upper bounded by $K * \mathcal{O}((n_t)^2)$. In a real application, the number of processor on platform K is relatively small compared to the number of subtasks N in application. Therefore the overall time complexity of the proposed partitioned scheduling algorithm is $\mathcal{O}(N^2)$.

3.7 Experimental Evaluation

We evaluate the performance of our proposed allocation and scheduling algorithm against the state-of-the-art. The test platform simulates NVIDIA AGX, a widely used hardware in robotic applications. It contains 8 CPU processors and 2 GPU processors. Each processor is considered as a single computing resource [57], i.e., only one subtask at a time is executed on each processor.

We compared the proposed LLP algorithm with recent works that are closely related to multiprocessor DAG scheduling.

- **HPC-s** and **HPC-p**. These are two partitioned scheduling methods originally developed for HPC-DAG models on heterogeneous platforms [10, 57].

- **AG-partition**. A partitioning heuristic for DAG scheduling on multicore platforms with identical cores [16]. We partition the DAG task graph into subgraphs by the processor tags of subtasks first, then apply AG-partition to each subgraph.

- **UGC**. A hierarchical scheduling algorithm based on reservation systems for DAG scheduling on homogeneous multi-processor platforms in study [92]. For fair comparison, the number of reservation servers is set to the number of processors per processor tag. The execution time budget is set to the deadline of DAG for the spinning reservation system.

- A **simulated annealing (SA)** method.

Table 3.1: Exemplary computational kernels for realistic robotic systems (time in seconds)

Name	Deadline (s)	Tag	WCET	Avg.ET
LOAM SLAM	15	CPU	26.5	12.7
RRT Path Planning	50	CPU	20	15
Task allocation	50	CPU	45	30
MPC HL Control	0.01	CPU	0.16	0.01
Midas Depth Estimation	0.5	GPU	0.35	0.32
PWC-net Optical Flow	0.5	GPU	0.57	0.5

3.7.1 Taskset Generation

To validate our task model and scheduler, we conduct experiments on both synthetic tasksets and a real-world robotic application.

Synthetic Tasksets Generation

Since a robotic system typically contains a base set of computational subtasks that are widespread in robotic applications, we consider a list of such subtasks in the design of experiments. A comprehensive data analysis of execution times for such subtasks has been carried out in the study [82]. We take the data from that study to define the execution time probability distributions needed for generating synthetic tasksets, lending practical relevance to our experimental design. Table 3.1 summarizes the execution time statistics of those representative subtasks. It also gives the other temporal characteristics including subtask deadline and tag.

We select the autonomous robot navigation, one of the most important problems in robotics, as the test scenario. We define two *contexts* that are used to construct the stochastic conditional node. As the robot could operate in either a crowded environment with the interference from obstacles and pedestrian, or a spatial space with only few obstacles, the contexts can be defined as *crowded* or *spatial*. The generation process of taskset is based on well-known

methods in the real-time systems literature, as detailed below.

- **Type.** The type of each subtask is initialized as *Computing*. The type can be changed to *Conditional* in following step if the subtask is selected as stochastic conditional node.

- **Tag.** The processor tag assigned to each subtask could either be CPU or GPU, each with an equal probability of 0.5.

- **Probabilities.** The probability distributions that characterize the execution times of nodes are generated as discrete probability mass functions (PMFs). The robotic system is assumed to operate in two contexts with equal probabilities. Therefore, $P(ctx_1) = P(ctx_2) = 0.5$.
  The four conditional probabilities (refer to Fig. 3.2), $P(b_1|ctx_1)$, $P(b_1|ctx_2)$, $P(b_2|ctx_1)$, $P(b_2|ctx_2)$ associated with outgoing edges of conditional node are randomly generated such that they satisfy equation (3.3), i.e., $P(b_i) = \sum_{j=1}^{2} P(b_i|ctx_j) \cdot P(ctx_j)$ and $\sum_{i=1}^{2} P(b_i) = 1$.

- **Execution times.** Based on two defined contexts *crowded* (ctx_1) and *spatial* (ctx_2), the two possible execution times (ET_i) of each subtask are set to its WCET for ctx_1 and $0.5 \cdot$ WCET for ctx_2.

- **Task Utilization.** The utilization ($U_i = C_i/T_i$) of each subtask is defined as the ratio of expected execution time over period, where the expected execution time is calculated as $\sum_{i=1}^{2} P(ctx_i) \cdot ET_i$. For each subtask, it is randomly generated as one of the six computational kernels provided in Table 3.1, with corresponding parameters. For each input utilization, the tasksets are generated using the *UUnifast-Discard* algorithm [35].

- **Dependencies.** We follow the method in He et al.'s experiments [55] to generate edges

among nodes. A random number is uniformly generated in interval [0,1] and compared with threshold (i.e., parallelism factor in [55]) to decide whether an edge should be added between two nodes. For experiments, the default setting of the threshold is 0.4.

- **Safety-critical nodes/paths assignments**: Depending on real application requirements, the LOAM SLAM kernel and RRT path planning kernel are crucial for safety during navigation [77]. Therefore, these two kernels are assigned highest TSLs and are selected as safety-critical nodes. For synthetic tasksets, each node is generated either as safety-critical node (s = True and TSL = 2) or non-safety-critical node (s = False and TSL = 1). Any paths that contain safety-critical nodes are labeled as safety-critical paths. The threshold τ_{l_i} of a safety-critical path takes value from a range in $(200, 300)$.

- **Stochastic Conditional node assignment** :As the robot could operate in two distinct environments (crowded or spatial space), we first find the set of subtasks in the DAG with exactly two successors, each corresponds to a distinct execution path. Within this set, random subtasks are selected and assigned as stochastic conditional nodes, and their *types* are switched from *computing* to *conditional*. Based on our testing scenario, probabilities $P(b_1|ctx_1)$, $P(b_1|ctx_2)$, $P(b_2|ctx_1)$, $P(b_2|ctx_2)$ are generated such that constraints in the previous steps are satisified.

For each input utilization, the experiments are conducted 5 times with 1000 DAG tasksets each time. The sum of every per-tag utilization is upper bounded by the number of processors.

Real Taskset

In Figure 3.1, an exemplary robotic system for autonomous driving has been presented. The parameters of these computational kernels are given in Table 3.1. The path planning kernels in both branches are labeled as safety-critical nodes with highest TSLs since they are crucial for safety problems in autonomous driving. The conditional node controls the computational paths used for planning based on the cluttered, uncertain nature of the environment.

3.7.2 Results Analysis

We first evaluate the effectiveness of our proposed LLP algorithm over synthetic tasksets. The comparison consists of three parts: acceptance ratio, average value of safety-performance metric and run-time. We then evaluate the method on a real robotic subsystem.

Experiments on Synthetic Tasksets

In real-world operations, occasional deadline misses of subtasks are acceptable within a certain range. In addition, the execution times of subtasks are usually unknown beforehand and vary during run-time. Our experiments take this stochastic nature of subtasks into account by relaxing the probabilistic timing constraint λ in the SP metric (See (3.9) to (3.12) in Section 3.4.2).

Remark 3.15. When $\lambda = 0$, the system is equivalent to a hard real-time system.

As discussed in Section 3.4.2, when $\lambda = 0$, it means that for a task to be deemed schedulable, both the response times of safety-critical path/node have to be smaller than the thresholds and there is no probability tolerance for the timing constraints.

When λ is non-zero, the timing constraints of the real-time system are relaxed and becomes stochastic. For example, if $\lambda = 0.01$, it means that the task will be deemed schedulable if it has a $1 - 0.01 = 99\%$ cumulative probability that the response times of its safety-critical path/node are smaller than the predefined thresholds based on their probability distributions.

Remark 3.16. The acceptable range of the probability tolerance coefficient λ is set to [0, 0.1].

Relaxing the probability tolerance λ in a small range improves the schedulability. Probability tolerances are selected in a small range to simulate *weakly-hard* systems where only small deviations from timing requirements are permissible.

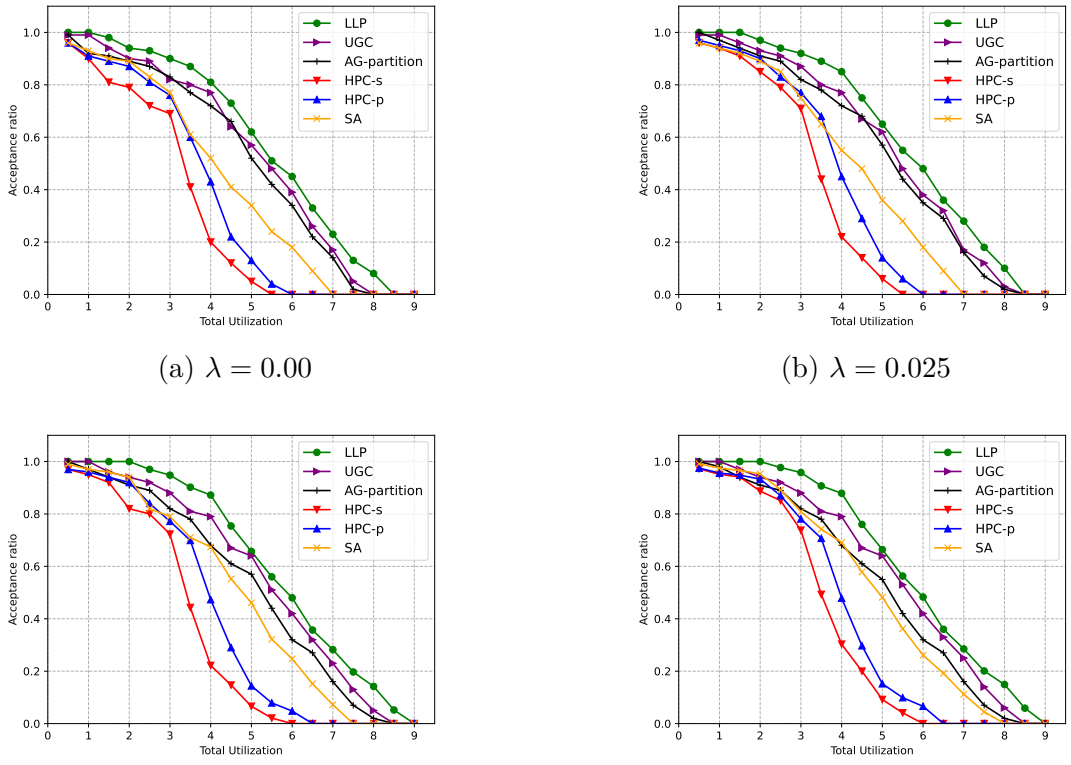
Comparison of Acceptance Ratio In the first subplot of Figure 3.3, when the probability tolerance λ is set to 0, the timing constraints represent a *hard* real-time system. In addition, the SP metric is equivalent to a conventional metric *makespan* in this case, since there is no tolerance for deadline miss and the response times have to be smaller than deadlines for schedulable tasks. The LLP method outperforms other methods because it aims to maximize SP metric, which is equivalent to makespan for the *hard* real-time systems in this case, in the allocation and scheduling process. In addition, it utilizes TSLs and precedence constraints to further optimize the schedules.

When timing constraints are relaxed by probability tolerance λ, the system becomes *weakly-hard* and LLP still keeps a much higher schedulability than other methods when the utilization of subtasks increases. When the probability tolerance λ increases, the schedulability improves as more tasks will be deemed schedulable if the probability of occasional deadline misses remain within a small range characterized by λ. For the combination of HPC-s allocation and EDF scheduling, when the utilization of task increases, allocating all subtasks

on a single processor per tag will more likely results in an infeasible schedule. Therefore, the acceptance ratio of HPC-s significantly drops as utilization increases. HPC-p is based on HPC-s by removing and reallocating one subtask at a time if a processor is fully occupied. It ignores the relative ordering of subtasks' TSLs, which could cause the safety-critical nodes/paths to miss their own independent deadlines. When the maximum iteration of SA is limited to 2000 (due to timing constraints in real operation), it cannot approach the global optimum fast enough which causes the performance gap between LLP and SA. AG-partition method traverses the available cores and allocate the nodes to the core with minimum cost. Although it also considers precedence constraints to reduce latency, it does not guarantee the safety-critical nodes to be scheduled first when precedence constraints are not violated. UGC needs to verify the feasibility of schedule on parallel servers beforehand, if the service cannot be guaranteed, the process is terminated and returns failure. It assumes all servers are identical and allow nodes to execute on any of the parallel reservation servers that are idle (i.e., property 2 in study [92]). However, the precedence constraints among nodes are not utilized when assigning nodes to available servers, which results in a difference between UGC and LLP with respect to acceptance ratio.

Comparison of Safety-Performance Metric In Figure 3.4, we evaluate the average safety-performance (SP) metric obtained by different scheduling algorithm. SP metric can be interpreted as rewarding for each safety-critical path and node, a balance of exceeding timing constraints (safety margin) and system performance related to improved response time. The larger the SP value, the more that schedule could trade exceeding safety margin for better performance or focus on improving safety by sacrificing performance without violating either safety or performance requirement.

In Figure 3.4, the more relaxed the timing constraints, the larger the difference between



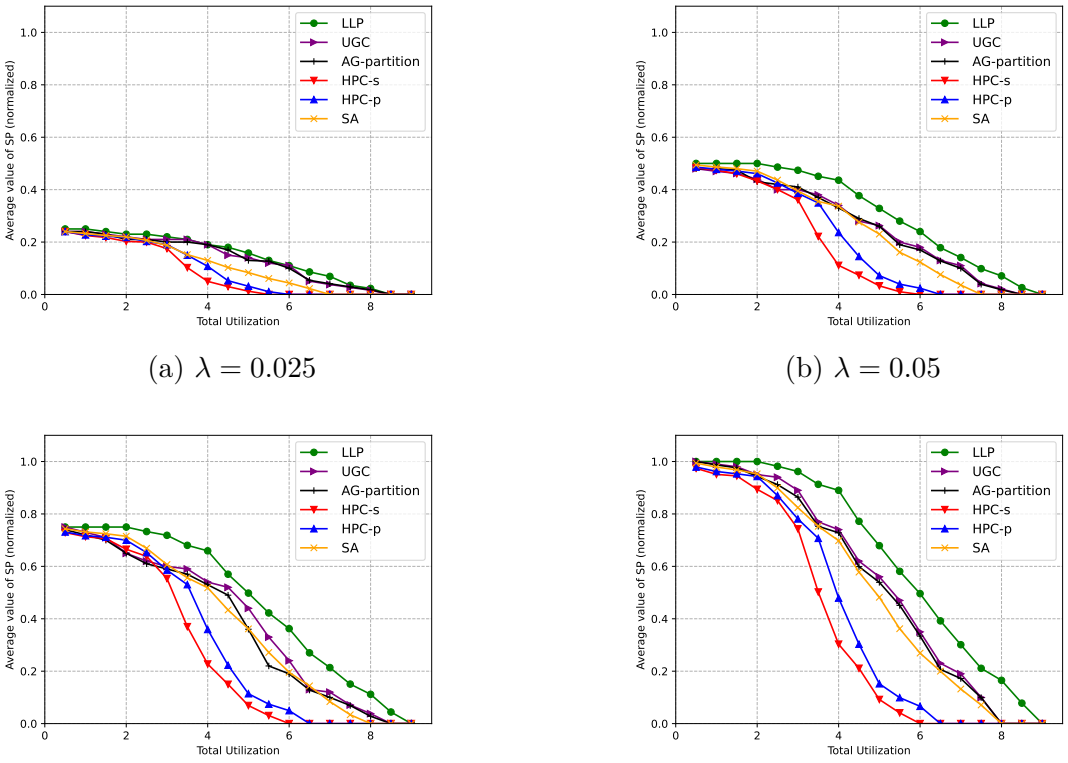
(a) $\lambda = 0.00$ (b) $\lambda = 0.025$

(c) $\lambda = 0.05$ (d) $\lambda = 0.075$

Figure 3.3: Acceptance ratio comparison with varied probability tolerance λ.

LLP and other methods. SA outperforms HPC-s and HPC-p since it uses SP as part of the objective function to optimize. LLP outpeforms AG-partition because (1) it tries to minimize the end-to-end latency in allocation process and prioritize the subtasks with higher TSLs in both allocation and scheduling, which results in more schedulable tasks; (2) LLP runs faster than AG-partition (shown in Table 3.2), it is less unlikely for a task to be deemed as unschedulable due to timeout in LLP. The reasons are the same for the SA method. LLP outperforms UGC because the execution times are represented as distributions instead of worst-case execution times in our experiments and UGC are not originally designed for this case. In addition, TSLs of nodes are not specified and utilized in UGC method.

For better clarity, we omit the hard timing constraint case where $\lambda = 0$ since the calculated



(a) $\lambda = 0.025$ (b) $\lambda = 0.05$

(c) $\lambda = 0.075$ (d) $\lambda = 0.1$

Figure 3.4: Average SP value comparison with varied probability tolerance λ.

SP will be zero. We pick $\alpha_{l_i} = \alpha_{n_i} = 0.5$ and show the results in Fig 3.4. The balancing parameters α_{l_i}, α_{n_i} could be tuned within a certain range (i.e., without violating the minimal requirement for safety-related timing constraints) based on the application. The trends are similar for various balancing parameters, we only provide one as representative due to the space constraint.

All the SP values in figures are normalized with 1000 for better readability.

Comparison of Time Complexity We compare the efficiency of different methods by setting fixed probability tolerance λ and safety-performance balancing parameters α_{l_i}, α_{n_i}. Since probability tolerance λ will affect the acceptance ratio and balancing parameters will

Table 3.2: Average Execution Time for Different Methods

(time in seconds)

Name/Number of nodes	5	10	15	20	25	30	35	40	45	50
LLP	2.24E-3	7.45E-3	1.41E-2	2.16E-2	3.07E-2	3.99E-2	5.67E-2	7.54E-2	1.14E-1	1.69E-1
UGC	2.07E-3	7.82E-3	1.25E-2	1.69E-2	2.22E-2	2.79E-2	8.14E-3	1.27E-2	1.45E-2	1.67E-2
AG-partition	3.85E-3	7.66E-3	1.59E-2	2.61E-2	3.92E-2	5.74E-2	6.13E-2	9.72E-2	1.52E-1	4.23E-1
HPC-s + EDF	4.47E-3	1.11E-2	1.84E-2	1.33E-2	2.84E-3	5.02E-3	7.12E-3	1.13E-2	1.89E-2	3.27E-2
HPC-p + EDF	4.47E-3	1.11E-2	1.84E-2	2.59E-2	3.36E-2	4.43E-2	5.43E-2	7.04E-2	9.38E-2	1.05E-1
SA	1.57E-1	4.12E-1	6.70E-1	8.20E-1	1.25	1.62	2.72	3.78	5.59	11.9

affect the SP value only, we pick $\lambda = 0.01$ and $\alpha_{l_i} = \alpha_{n_i} = 0.9$ to present a representative result in Table 3.2.

LLP runs around two orders of magnitude faster than SA in Table 3.2 because SA is a meta-heuristic that approximates global optimum with a large number of iterations. Compared with the heuristics for HPC-DAG [57], LLP runs with the same level of speed but provides a much higher acceptance ratio. LLP runs around 30% faster than AG-partition on average. The reason is that for each subtask, AG-partition will first compute its cost when assigned to a processor. Since this partitioning part requires the calculation of cost function for all processors, it will be cost-expensive when the number of subtasks increases. LLP runs as fast as UGC when the size of DAG is relatively small (i.e., number of nodes is smaller than 25). When the size of DAG increases, more DAG tasks are deemed as unschedulable by UGC because the execution time budget of the reservation system is static [92], and the promised service must be verified. If a feasible schedule that guarantees the service cannot be found beforehand, the process is terminated and returns failure directly, which saves scheduling time. To conclude, LLP performs better than heuristic methods (e.g., HPC-s/p, SA etc.) for real-time systems that require efficient scheduling. At the cost of slower speed, LLP achieves better performance than UGC according to SP metric.

Experiments on Real Tasksets

We utilize the exemplary robotic system in Figure 3.1 as the real taskset for the experiments. The probability mass function (PMF) of execution time for each computational kernels is constructed based on the parameters in Table 3.1, where the PMF consists of the worst-case execution time (WCET), minimum execution time and average execution time with corresponding probabilities reported in the study [82]. Two path planning nodes are labeled as safety-critical nodes based on the application requirement.

Similar to the experiments of synthetic tasksets, we apply different scheduling algorithm to this specific DAG taskset and run the experiments for 1000 times with various deadlines and probability tolerance λ. The deadline of the application takes value in the interval [75,120], which corresponds to various operating context and timing requirements.

Table 3.3: Average run-time of varied scheduling algorithms (time in seconds)

LLP	UGC	AG-partition	HPC-s	HPC-p	SA
5.01E-4	4.86E-4	6.12E-4	7.47E-4	7.42E-4	2.29E-3

Table 3.4: Average SP metric with varied probability tolerance λ

	LLP	UGC	AG-partition	HPC-s	HPC-p	SA
$\lambda = 0.025$	0.682	0.479	0.473	0.079	0.099	0.475
$\lambda = 0.050$	0.696	0.481	0.472	0.099	0.099	0.468
$\lambda = 0.075$	0.712	0.492	0.474	0.100	0.109	0.495

As can be seen from Table 3.3, LLP runs as fast as UGC method with a difference of 3%, which aligns with results in Table 3.2 because the size of real taskset is relatively small. LLP runs faster than other methods with a minimum of 20% run-time improvement. For varied probability tolerance λ in Table 3.4, the SP metric schedule under LLP outperforms other methods with a significant margin. Although LLP runs slightly slower than UGC method in Table 3.3, it achieves much higher SP value with an average improvement of 42%.

3.8 Conclusion

This paper proposed a partitioned scheduling algorithm called LLP which consists of novel allocation and scheduling algorithm for stochastic heterogeneous conditional DAG scheduling, which takes the stochastic nature of execution time and conditional nodes within the DAG tasks into account. We use a multi-objective metric called the Safety-Performance (SP) metric to explore the trade-offs between timing constraints and system performance for safety-critical systems. Furthermore, unlike existing works for stochastic scheduling, we do not make any assumptions about the specific distributions of random execution times, instead we propose a convolution based method to evaluate the response times of safety-critical paths and nodes against the stochastic timing constraints. The proposed algorithm LLP outperforms the state-of-the-art DAG scheduling algorithms for multi-processor systems. In future work, we plan to extend our work with multiple DAGs and SP metric with a third objective function of software reliability.

Chapter 4

Latency-Aware Priority Assignment for Fixed Priority Scheduling in Multi-rate Simulink Models

4.1 Introduction

Simulink/Stateflow is a model-based design framework that is widespread in the development of control-centric embedded systems applications [90]. The underlying formalism is based on the "zero execution time" semantics of the Synchronous Reactive (SR) model of computation, whose deterministic dataflow facilitates testing and formal verification of the model. Specifically, dataflow determinism means that for a communication link (possibly between two blocks with different periods), the reader job always reads from the same write job.

The execution times of blocks will be non-zero at run-time, so it is important to ensure the semantics preservation of the software implementation, i.e., the software is predictably equivalent to the "zero execution time" assumption of the SR model. The solution in the code generation tools Simulink Coder/Embedded Coder is to use a special wait-free communication buffer, Rate Transition (RT) blocks in the multi-task implementation [90]. An RT

block is added between each pair of communicating blocks that have different periods. It leverages the fixed-priority scheduling policy adopted by Simulink Coder/Embedded Coder, to enforce the execution order of the blocks and provide deterministic data transfer according to the "zero execution time" semantics. However, RT blocks come with the price of extra memory and, as the focus of this paper, worse end-to-end latencies.

Reaction time (RXN) and data age (DA) are important end-to-end latency metrics for control-centric embedded systems. Intuitively, for an end-to-end path (typically from sensor-to-actuator, also called a cause-effect chain) in the system, RXN measures the responsiveness of the system to an input event along the path, and DA quantifies the data freshness used in the decision making [51]. Both RXN and DA have significant impact on the quality of control [94]. In this paper, we study the optimization of RXN/DA in the implementation of Simulink models. We first review the related work.

4.2 Related Work.

There has been a rich set of research results on the multi-task software implementation of SR model, and we only provide a selective review below. Caspi et al. presented a specialized wait-free buffer as the semantics-preserving communication mechanism for multi-rate SR models [27]. The studies in [39, 121] proposed to decompose Stateflow blocks (i.e., those modeled with synchronous finite state machines) into multiple software tasks, and presented efficient policies to ensure semantics preservation and associated analysis on timing and memory. Also the real-time schedulability analysis techniques for such task systems were developed in [73, 104], leveraging research advances from the digraph real-time task model and max-plus algebra [72, 85, 105, 106]. On the optimization of semantics-preserving software implementation, Di Natale et al. developed a branch-and-bound (BnB) algorithm

that optimizes the memory overhead from RT blocks of Simulink models [69]. In [40], they formulated the problem as an efficient integer linear program (ILP) that can be solved much faster than BnB. Such a problem is also studied for other types of scheduling policies (such as EDF, Earliest Deadline First [1] and mixed-criticality scheduling [109]) or hardware platforms (including multicore [7] and distributed systems [41, 53]). Similar to our paper, Zhao et al. leveraged the idea of using an abstraction layer to hide the real-time schedulability constraints. They designed efficient optimization techniques for implementing Simulink models on multicore platforms [7] and the models with Stateflow blocks [115]. However, to the best of our knowledge, *none of the existing studies, including [7] [115], consider reaction time and/or data age in the implementation of SR models.* Since the optimization variables, constraints, and/or objective are different than [7] [115] (or [111, 112, 118] that have similar concepts and optimization techniques to those in [7, 115]), this work studies a uniquely challenging problem that requires a complete redesign of the optimization framework.

Regarding the existing work on reaction time and data age, we refer the readers to a recent tutorial for a more comprehensive review [51]. The metrics of data age and reaction time were first formally defined by Feiertag et al. [43]. Since then, different definitions of the metrics were proposed in [49, 50]. Xiong et al. presented a scheduling algorithm for minimizing the data age of transactions [101]. More recently, Verucchi et al. developed an analysis to calculate the upper bound of data age and reaction time [94]. Gohari et al. calculated the lower and upper bounds on the data age of multi-rate task chains with a job-level fixed-priority scheduling [46]. Wang et al. [99] optimizes the end-to-end latency metrics (including reaction time, data age, and time disparity) for the model-based design using the logical execution time paradigm. Still, *there is no work that considers the analysis of reaction time or data age for the implementation of Simulink models, not to mention the optimization of these two metrics.* We find that the RT blocks pose useful properties that can be exploited

for the analysis and optimization of Simulink models.

Contributions. This work is the first to consider the problem of optimizing RXN/DA in the implementation of multi-rate Simulink models. We first exploit the property of RT block in Simulink to calculate the reaction time and data age of the task chain. Then we propose an abstraction layer that hides the complexity of schedulability analysis. We develop an optimization framework to find the task priority assignment that minimizes either reaction time or data age.

4.3 Simulink Model Semantics

A Simulink model can be represented as a *Direct Graph* Γ [90]. The set of nodes in Γ is denoted as $\mathcal{N} = \{\tau_1, \tau_2, \ldots, \tau_{|\mathcal{N}|}\}$, where each node is a functional block that is triggered periodically. The set of edges $\mathcal{L} = \{L_1, L_2, \ldots, L_{|\mathcal{L}|}\}$ represents the communication links among the blocks. For each block τ_i, it is associated with a nominal period T_i with no offset. Hence, all blocks are triggered at the same time 0, and the release pattern of the blocks is repeated in each *hyperperiod* (with length equal to the least common multiple of all periods). For each edge L_i, it connects two blocks by feeding the output of the preceding block (writer) to the input of the succeeding block (reader). For each communication link, Simulink assumes the *harmonic* periods of the writer and the reader blocks, i.e., the period of one block is evenly divisible by the other. The k-th job instance of block i is denoted as $\tau_{i,k}$. Simulink Coder/Embedded Coder adopts a fixed-priority scheduling policy, where all instances of a block are assigned with the same priority. The priority of block τ_i is denoted as π_i. The smaller the value of π, the higher the priority.

The actual non-zero execution time implementation of the system may be significantly different from the zero-semantics execution of the SR model [38]. For a multi-rate model (i.e.,

the periods of the blocks may not be the same), the implementation may suffer from consistency and time determinism issue between two connecting blocks [38]. To solve such issues, Simulink adopts a Rate Transition (RT) block that behaves like zero-order hold block or a unit delay block plus a hold block to maintain consistency of shared buffers and time determinism. The RT blocks fully controls the timing of data transfer in a *predictable* way, but in the case of unit-delay communication, they add functional delays that prolong the end-to-end latencies.

Specifically, there are two types of communication between Simulink blocks: *direct feedthrough* and *unit-delay*. For each pair of two blocks (τ_i, τ_j) connected by a communication link, if the output of the reader τ_j is directly dependent on the most recently triggered instance from the writer τ_i, then the link is associated with a precedence constraint, which is referred to as *direct feedthrough*. Denote $i_j(k)$ as the input to the k-th job instance of the reader τ_j, and $o_i(m)$ as the output of the most recently triggered job instance of the writer τ_i, which is triggered no later than the k-th job instance of the reader N_j,

$$i_j(k) = o_i(m), \text{where } m = \max\{x | t_i(x) \leq t_j(k)\} \tag{4.1}$$

Here $t_i(x)$ is the trigger time of the x-th instance of τ_i, and $t_j(k)$ is similarly defined.

According to the synchronous reactive (SR) semantics, Simulink also allows delayed communication. The reader will not read data from the most recently triggered instance of the writer, but instead, from the previous instance of the writer. The delay is limited to one unit (i.e., one instance of the writer) in Simulink and denoted as *unit-delay* communication

$$i_j(k) = o_i(m-1), \text{where } m = \max\{x | t_i(x) \leq t_j(k)\} \tag{4.2}$$

4.4 Reaction time and Data age of task chain

The reaction time (RXN) measures the responsiveness of a system to a change in the input and the data age (DA) quantifies for how long an input data can affect the output of the system [51]. We follow Verucchi et al. [94] and provide the formal definitions of the two metrics.

Definition 4.1. [94] For a communicating task chain $\tau_1, \tau_2, ..., \tau_n$, consider any instance $\tau_{1,i}$ of τ_1, and assume $\tau_{n,j}$ as the *first* among all instances of τ_n such that its result depends on the output of $\tau_{1,i}$. The reaction time (RXN) of the task chain is defined as the maximum interval between the trigger time of $\tau_{1,i}$ and the completion time of $\tau_{n,j}$.

Definition 4.2. [94] For a communicating task chain $\tau_1, \tau_2, ..., \tau_n$, consider any instance $\tau_{1,i}$ of τ_1, and assume $\tau_{n,j}$ as the *last* among all instances of τ_n such that its result depends on the output of $\tau_{1,i}$. The data age (DA) of the task chain is defined as the maximum interval between the trigger time of $\tau_{1,i}$ and the completion time of $\tau_{n,j}$.

4.4.1 Calculate reaction time and data age for one edge

We start the calculation of reaction time and data age in Simulink model by considering one single link. Without loss of generality, we consider two communicating blocks τ_1 and τ_2. The two broader cases that can cause issues about data consistency and time-determinism in the data transfer are: (a) a high priority block driving a low priority one; (b) a low priority block driving a high priority one. Also consider the fact that RT blocks are only needed between blocks with different periods, this leads to four different combinations depending on the priorities and periods of the two blocks. We illustrate all four cases as follows.

Case 1

$\pi_1 < \pi_2,\ T_1 < T_2$.

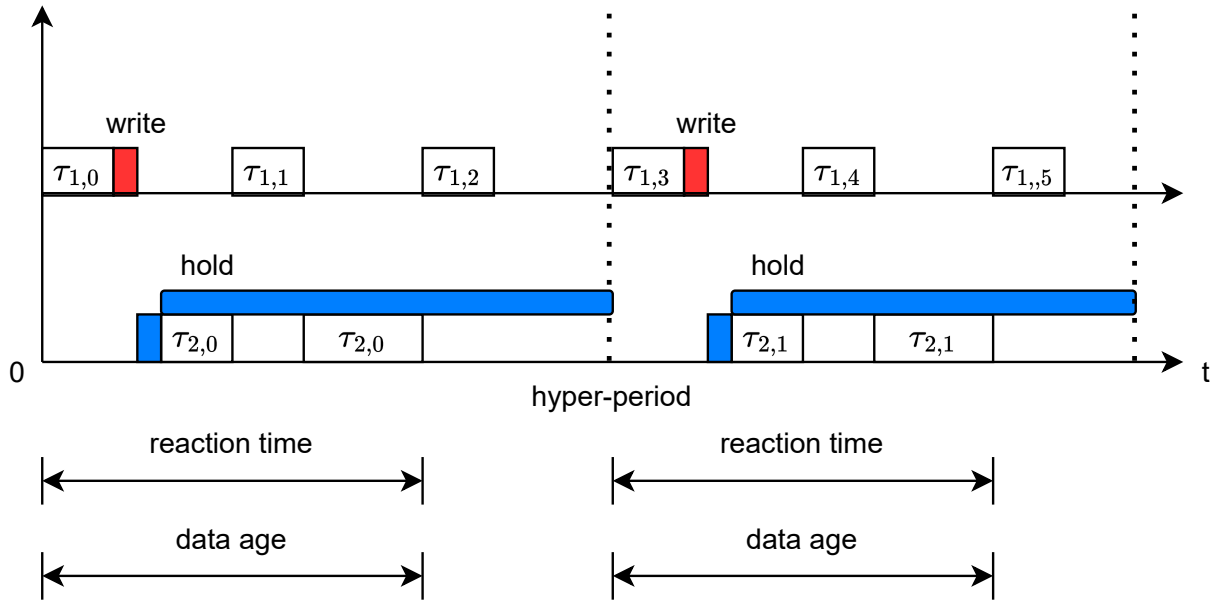


Figure 4.1: High priority block τ_1 driving low prority block τ_2, $T_1 < T_2$.

Fig. 4.1 shows an example where the block with smaller period and higher priority drives the other one. Due to the existence of RT blocks (i.e., the red boxes on the top line and vertical blue boxes on the bottom line), in the first hyperperiod only the output of job $\tau_{1,0}$ will be copied by the RT block (the vertical blue box before $\tau_{2,0}$) and fed to the reader. Once the RT block copies the data, it will be held for the entire hyper-period. The RT blocks have the same periods as the blocks with larger periods. The output of the other two writer job instances $\tau_{1,1}$ and $\tau_{1,2}$ are ignored. This pattern will be repeated in each hyperperiod.

In this case, both reaction time and data age are equal to the worst-case response time (WCRT) of instance $\tau_{2,0}$. Here we use the WCRT of the block τ_2 as a safe overestimation of

the WCRT of the instance $\tau_{2,0}$.

$$RXN = DA = WCRT_2 \tag{4.3}$$

For any block τ_i, its WCRT can be calculated as the smallest fixed point solution of the following formula

$$WCRT_i = C_i + \sum_{\forall j \in hp(i)} \lceil \frac{WCRT_i}{T_j} \rceil C_j \tag{4.4}$$

where C_i is the worst case exectuion time of block τ_i, and $hp(i)$ is the set of blocks that have higher priorities than τ_i.

Case 2

$\pi_1 < \pi_2,\ T_1 > T_2$.

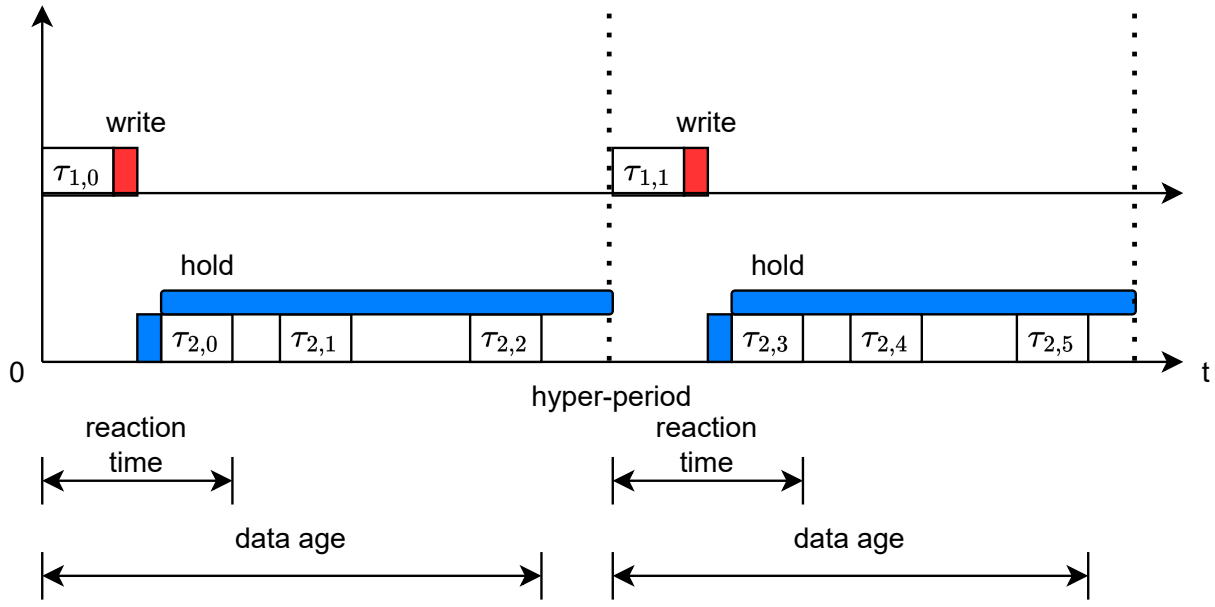


Figure 4.2: High priority block τ_1 driving low prority block τ_2, $T_1 > T_2$.

In this case, the first and last job instance that reacts to the data output written by $\tau_{1,0}$ are

different. The reaction time in this case will be the response time of instance $\tau_{2,0}$. The data age will be the difference between the release times of instance $\tau_{2,2}$ and instance $\tau_{1,0}$ plus the response time of $\tau_{2,2}$.

$$RXN = WCRT_2 \tag{4.5}$$

$$DA = T_1 - T_2 + WCRT_2 \tag{4.6}$$

Case 3

$\pi_1 > \pi_2,\ T_1 < T_2$.

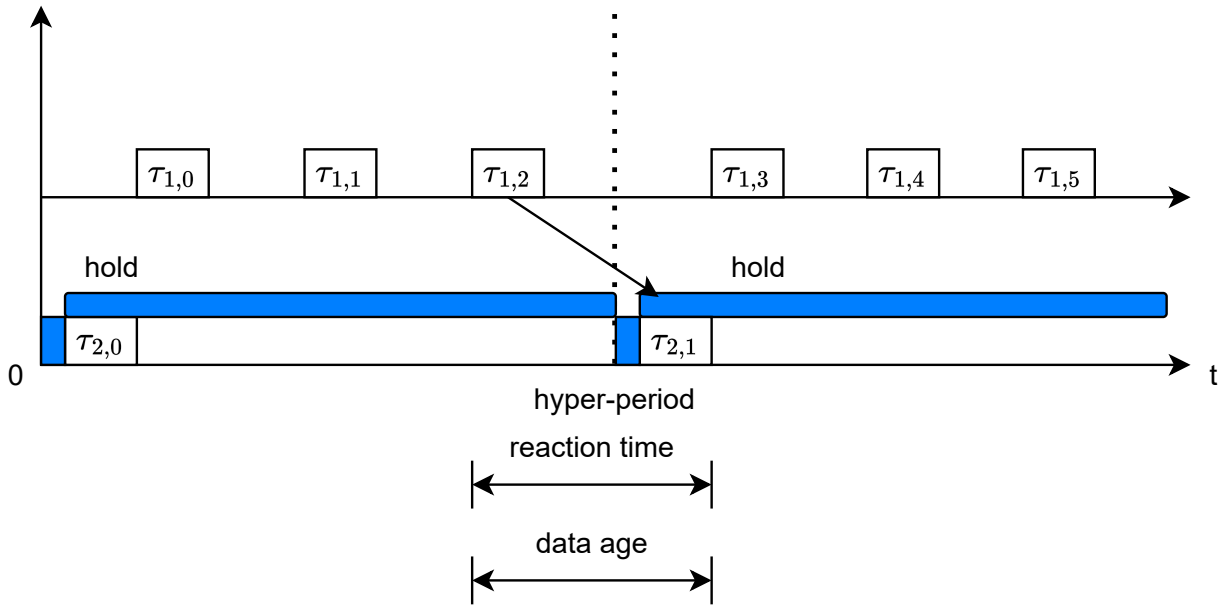


Figure 4.3: Low priority block τ_1 driving high priority block τ_2, $T_1 < T_2$.

Fig. 4.3 represents the case where unit-delay exists. The RT blocks introduce unit-delay that the instances $\tau_{2,0}$ and $\tau_{2,1}$ will read the previous instances of block τ_1 with one unit delay. In this case, both the reaction time and data age will be equal to the sum of block τ_1's period

(i.e., one unit-delay) and response time of block τ_2's instance,

$$RXN = DA = T_1 + WCRT_2 \tag{4.7}$$

Case 4

$\pi_1 > \pi_2,\ T_1 > T_2$.

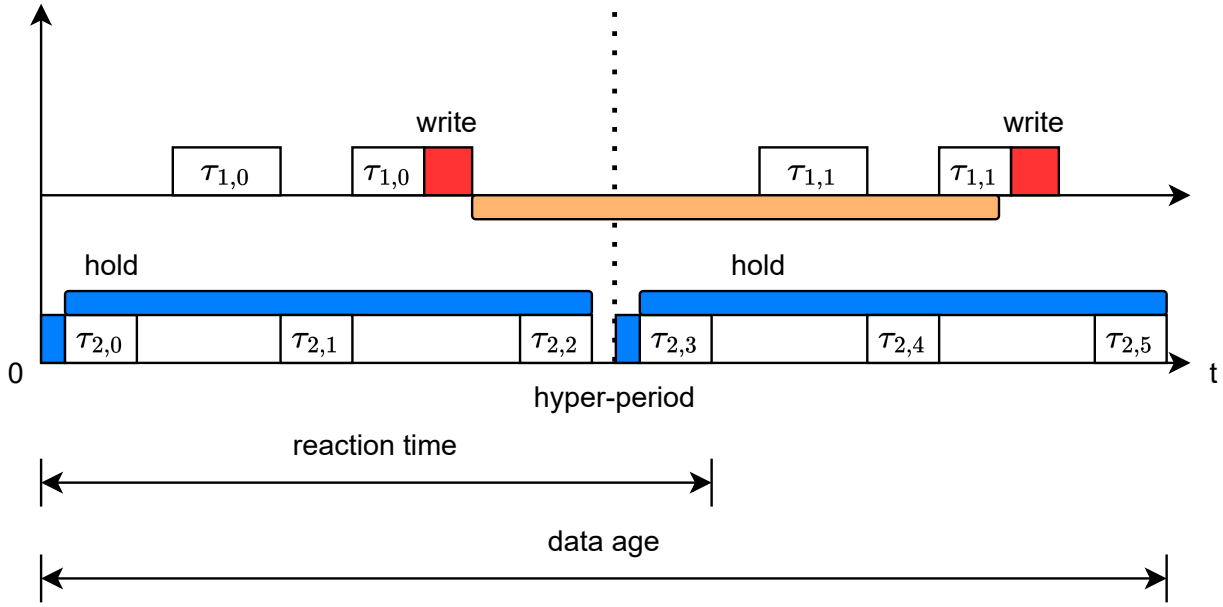


Figure 4.4: Low priority block τ_1 driving high priority block τ_2, $T_1 > T_2$.

Fig. 4.4 shows a case where $\pi_1 > \pi_2$, and the unit-delay introduced by RT block forces the instance $\tau_{2,3}$ to read the data produced by previous instance $\tau_{1,0}$. As the release time of $\tau_{1,0}$ is at time 0, the reaction time will be the sum of larger period of two blocks and the response time of $\tau_{2,3}$. The data age will be the sum of release time of last instance of block τ_2 in the next hyper period and its response time,

$$RXN = T_1 + WCRT_2 \tag{4.8}$$

$$DA = 2 \cdot T_1 - T_2 + WCRT_2 \tag{4.9}$$

4.4.2 Calculate the upper bound of RXN/DA of task chain

We now consider the general case where a task chain have multiple blocks (i.e., more than two). Our method calculates the upper bounds of reaction time and data age, but the overestimation can be bounded within an acceptable range.

To unify the notation and calculation of RXN and DA, we first present the formal definition of release time interval (rti). RXN_i and DA_i are denoted as the reaction time and data age for two connected blocks τ_i and τ_{i+1} in the chain.

Definition 4.3 (Release Time Interval (rti))**.** rti is the difference between the release times of two job instances.

The *rti* in *RXN*. When high priority block drives low priority block, the *rti* of jobs released by two connected blocks is 0 as two blocks both release job instances at the beginning of their largest common multiple (LCM). In the cases of low priority block drives high priority block, the *rti* of two job instances is the the period of the starting block due to the unit delay.

$$rti = \begin{cases} 0, \pi_i < \pi_{i+1} \\ T_i, \pi_i > \pi_{i+1} \end{cases}$$

The *rtí* in *DA*. As the indices of job instances that correspond to RXN and DA calculations are different in some cases, we use rti'_i to indicate this difference from rti. To calculate *rtí* in DA, information of both relative priority orders and the relative periods between two blocks are needed.

In the cases of high priority block drives the low priority block, if T_1 is smaller than T_2, the *rtí* will be the 0 as in figure 4.1. Otherwise, the *rtí* will be $T_1 - T_2$. In the cases of low priority block drives high priority block, if T_1 is smaller than T_2, the *rtí* is $2 * T_1 - T_2$. If $T_1 > T_2$, the *rtí* is T_1.

$$rti = \begin{cases} 0, if\ \pi_i < \pi_{i+1}, T_i < T_{i+1} \\ T_i - T_{i+1}, if\ \pi_i < \pi_{i+1}, T_i > T_{i+1} \\ 2 * T_i - T_{i+1}, if\ \pi_i > \pi_{i+1}, T_i > T_{i+1} \\ T_i, if\ \pi_i > \pi_{i+1}, T_i < T_{i+1} \end{cases}$$

We use three blocks where an additional block is connected after block τ_2 to illustrate the calculation process. When we simply sum up the reaction time of each pair of two adjacent blocks, i.e., $RXN_1 + RXN_2$, the WCRT of the block in the middle (i.e., $WCRT_2$) will be added twice. The reason is, RXN_1 is defined as $rti_1 + WRCT_2$, and RXN_2 starts from the release time of τ_2, and ends after block τ_3's response. The overlap between RXN_1 and RXN_2 starts at the release time of τ_2 and ends after its response. When we add RXN_1 and RXN_2, the $WCRT_2$ is added twice.

$$RXN_1 + RXN_2 = rti_1 + WCRT_2 + rti_2 + WCRT_3 \tag{4.10}$$

By definition, the RXN of the task chain will be the sum of the rti between instances of first and third block and the response time of the last instance,

$$RXN = rti_1 + rti_2 + WCRT_3 \tag{4.11}$$

Therefore, by subtracting the $WCRT_2$ from (4.10), we can derive the upper bound of the

RXN of the task chain. By induction, the formula for calculating the RXN of task chain can be generalized as,

$$RXN = \sum_{i=1}^{n-1} RXN_i - \sum_{j=2}^{n-1} WCRT_j = \sum_{i=1}^{n-1} rti_i + WCRT_n \tag{4.12}$$

Similarly, the calculation of DA can be written as,

$$DA = \sum_{i=1}^{n-1} DA_i - \sum_{j=2}^{n-1} WCRT_j = \sum_{i=1}^{n-1} rti'_i + WCRT_n \tag{4.13}$$

Where n denotes the number of blocks in the task chain.

4.4.3 Bound of overestimation

Due to space limit, a sketch proof of overestimation in the above calculations is provided. For the RXN calculation, the calculation of rti will not introduce any pessimism as by definition, we always try to find the first job instance in the hyper-period that reacts to the data produced by writer. The release time of reader will either be 0 or the start time of next hyper-period. However, as we use WCRT to estimate the response time of last block, this might introduce overestimation as in (4.12) it assumes that the job instance of last block will suffer from *all* job instances from $hp(i)$. The error of estimation is bounded by,

$$UB_{overest}(RXN) = WCRT_n - C_n \tag{4.14}$$

For the DA calculation, the indices of job instances for each pair of two blocks that measure the DA might not be the same when we traverse the task chain due to RT block. Therefore, the maximum error could be $|T_i - T_{i+1}|$ as the release time difference between first and last

instance of the block. Therefore, the overestimation could be bounded by,

$$UB_{overest}(DA) = \sum_{i=1}^{n-1} |T_i - T_{i+1}| \tag{4.15}$$

4.5 The concept of MUDPA

We propose the formal definition of Maximal Unschedulable Deadline and Priority Assignment (MUDPA), which is an irreducible representation of both the deadline and priority assignment that causes the system unschedulable.

Definition 4.4 (Relative Priority Order (RPO))**.** The RPO is a binary variable that indicates the relative priority order between two blocks τ_i and τ_j, denoted as $r_{i,j}$,

$$r_{i,j} = \begin{cases} 1, \pi_i < \pi_j \\ 0, \pi_i > \pi_j \end{cases}$$

The RPO set $\mathcal{R} = \{r_{i_1,j_1}, r_{i_2,j_2}, \cdot, r_{i_k,j_k}\}$ is the collection of all relative priority orders that satisfy the condition $r_{i,j} \equiv 1$. It serves as constraints on relative priorities, where block τ_i is required to have higher priority than block τ_j if $r_{i,j}$ exists in the set $\mathcal{R}$. The more elements in set $\mathcal{R}$, the more constraints on relative priorities among blocks, which results in a smaller search space for feasible solution.

The concept of *virtual deadline* has been proposed in [12] for scheduling at run-time. We exploit this concept but use it differently at design phase.

Definition 4.5 (Virtual Deadline (VD))**.** A VD d_i is not larger than its corresponding deadline D_i. It serves as a tighter constraint on the deadline as $d_i \leq D_i$.

Definition 4.6 (Deadline and Priority Assignment (DPA))**.** DPA is a collection of con-

straints on (i) virtual deadline assignments for each block (ii) relative priorities among blocks, denoted as $\mathcal{M}$,

$$\mathcal{M} = \{< \tau_1, d_1 >, < \tau_2, d_2 >, \cdots < \tau_n, d_n >, \mathcal{R}\} \tag{4.16}$$

The $\mathcal{M}$ represents the constraints on both deadlines and priorities. Firstly, the response time of each block should meet their corresponding virtual deadline (VD). Secondly, the priority assignment $\mathcal{P}$ should respect the relative priority constraints included in the RPO set $\mathcal{R}$.

To formally describe the relationships between two DPAs, we note that the WCRT analysis in Equation (4.4) is sustainable [11]. This means that a schedulable system should remain schedulable with (a) less relative priority constraints in $\mathcal{R}$ and/or (b) element-wise larger virtual deadlines.

Definition 4.7 (Dominance)**.** Given two DPAs $\mathcal{M}_1$ and $\mathcal{M}_2$, $\mathcal{M}_1$ is said to *dominate* $\mathcal{M}_2$ if and only if $\mathcal{M}_1$ represents element-wise *looser* or *the same* constraints on relative deadlines and priorities than counterparts of $\mathcal{M}_2$, denoted as $\mathcal{M}_1 \succeq \mathcal{M}_2$.

Example 1. We consider an example of a task chain consists of four blocks and three DPAs.

$$\begin{aligned}
\mathcal{M}_1 &= \{< \tau_1, 8 >, < \tau_2, 8 >, < \tau_3, 15 >, < \tau_4, 15 >, \\
&\quad \{r_{1,2}, r_{3,4}\}\} \\
\mathcal{M}_2 &= \{< \tau_1, 8 >, < \tau_2, 8 >, < \tau_3, 15 >, < \tau_4, 15 >, \\
&\quad \{r_{1,2}, r_{2,3}, r_{3,4}\}\} \\
\mathcal{M}_3 &= \{< \tau_1, 8 >, < \tau_2, 8 >, < \tau_3, 16 >, < \tau_4, 18 > \\
&\quad \{r_{1,2}\}\}
\end{aligned}$$

We claim that $\mathcal{M}_1$ dominates $\mathcal{M}_2$ as $\mathcal{M}_1$ has less constraints on relative priorities. $\mathcal{M}_3$

dominates other two DPAs because it has the least constraints on priorities, and the VDs of it are no smaller than their counterparts in $\mathcal{M}_1$ and $\mathcal{M}_2$.

Denote the taskset as Γ and the a given priority assignment as P, separately. We define the schedulability of the system as follows.

Definition 4.8 (Schedulability)**.** Γ is $\mathcal{M}$-schedulable if and only if P satisfies the relative priority constraints R and the WCRTs of tasks meet all VDs and in $\mathcal{M}$.

Algorithm 5 Algorithm for calculating MUDPA

Input:

An unschedulable DPA $\mathcal{M}$
$\mathcal{M} = \{< \tau_1, d_1 >, \cdots < \tau_n, d_m >, \mathcal{R}\}$
Set of all tasks Γ
Set of concerned tasks Ω
Virtual deadlines $d_i, i \in [1, m]$ for all tasks $\in \Omega$

1: **for** each element r in relative priority order $\mathcal{R}$ **do**
2: **if** taskset Γ is **not** $\mathcal{R}\backslash\{r\}$-schedulable **then** exclude r from $\mathcal{R}$
3: **end if**
4: **end for**
5: **for** each element in $< \tau_i, d_i >$ **do**
6: Use binary search to find the largest d_i while keeping other elements unchanged and
7: Γ unschedulable.
8: If the value can be increased to its upper bound and taskset remains unschedulable,
9: update it with the upper bound value.
10: **end for**
11: Return the *MUDPA*

We provide the definition of *Maximal Unschedulale Deadline and Priority Assignment* and explain its calculation in Algorithm 4.5.

Theorem 4.9. *A deadline and priority assignment (DPA) $\mathcal{M}$ is a Maximal Unschedulable DPA (MUDPA) if and only if (i) taskset Γ is not $\mathcal{M}$-schedulable and (ii) for any other DPA $\mathcal{R}$ strictly dominates $\mathcal{M}$, $\mathcal{R} \succ \mathcal{M}$, Γ is $\mathcal{R}$-schedulable.*

4.6 MUDPA-Guided Optimization Framework

We design the optimization framework based on the MUDPA concept. The objective is to minimize the weighted sum of reaction times or data ages of all concerned task chains. The constraints consist of (i) upper bound of reaction time or data age; (ii) schedulability of the system, i.e., response times of all tasks do not exceed their corresponding virtual deadlines; (iii) priority constraints specified in MUDPA and (iv) design constraints such as upper bound of memory consumption. The optimization problem is to find the optimal sets of virtual deadline assignment variables *and* priorities that satisfy the constraints.

Formally, let Γ be the taskset and $\mathcal{U}$ be the set consists of all MUDPAs. Denote $\mathcal{M}$ as any MUDPA in $\mathcal{U}$, $\mathcal{M} \in \mathcal{U}$. The set of all task chains we concern is $\mathcal{L}$. For each chain l in $\mathcal{L}$, it has an assigned weight β. The number of chains is represented as m. The set of tasks contained in $\mathcal{L}$ is Ω. All the design constrains, including upper bounds of RXN and DA, are represented as $G(X) \leq 0$. We formulate the optimization problem as follows,

$$
\begin{aligned}
&\min \sum_{l_i \in \mathcal{L}, i=1}^{m} \beta_i \cdot RXN_i \text{ (or } DA_i) \\
s.t.\ & C_j \leq d_j \leq D_j, \forall \tau_j \in \Omega \\
& WCRT_j = d_j, \forall \tau_j \in \Omega \\
& G(X) \leq 0 \\
& \Gamma \ is\ \mathcal{M}\text{-schedulable}, \forall \mathcal{M} \in \mathcal{U}
\end{aligned}
$$

We propose a MUDPA-guided optimization framework to solve the above problem.

Step 1 (Initial problem)

Initialize the ILP formulation $\mathcal{I}$ by excluding all the schedulability constraints in the original optimization problem. i.e., $\mathcal{U}$ is initialized as $\emptyset$,

Step 2 (Solve problem)

Solve the problem $\mathcal{I}$. If $\mathcal{I}$ is not feasible, report unschedulability. The original problem remains infeasible as its relaxed version $\mathcal{I}$ is infeasible. If $\mathcal{I}$ has a feasible solution $\mathcal{S}$, we obtain the values of virtual deadline assignments and priorities from $\mathcal{S}$.

Step 3 (Test schedulability)

The revised Audsley's algorithm in [112] is used to determine if taskset Γ is $\mathcal{M}$-schedulable. If the $\mathcal{S}$ is feasible for the original problem, the framework terminates and returns optimal solution. Otherwise, it goes to step 4.

Step 4 (MUDPA calculation)

We compute the MUDPAs by following algorithm 1. The problem $\mathcal{I}$ is updated by adding MUDPAs back as induced schedulability constraints. The framework returns to step 2 with updated MUDPA set $\mathcal{U}$.

4.7 Experimental Evaluation

We use an industrial task system which implements a Simulink model of fuel injection system from the studies [40, 112] to evaluate the effectiveness of the proposed method. It contains

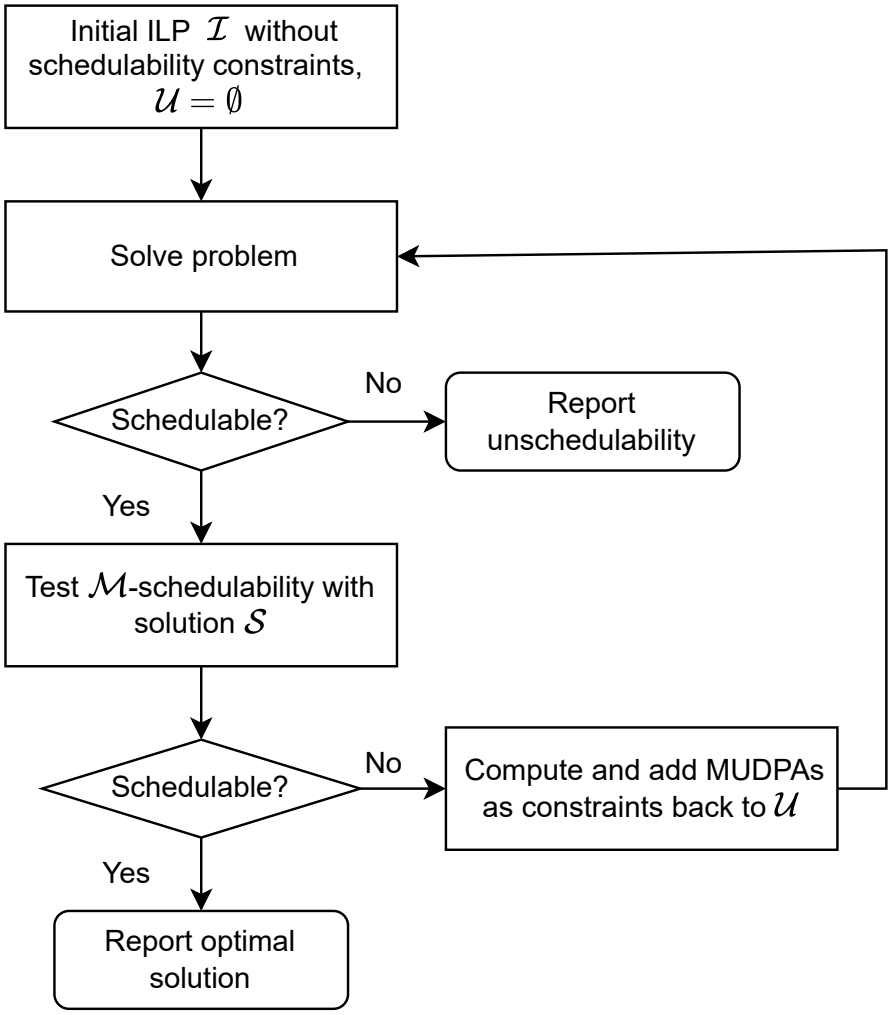


Figure 4.5: The MUDPA-guided optimization framework.

90 blocks and 106 communication links among them. The periods of blocks range between 4 to 1000 ms, and are harmonic as required by Simulink. Blocks are modeled as implicit tasks where deadlines are equal to periods. Utilization of the whole system is 94.1%. Weights of tasks vary between [26, 9768] according to the weighted case in [112].

The optimization problem is to minimize the weighted sum of reaction times or data ages of concerned taks chains in $\mathcal{L}$. We compare MUDPA-guided framework with an Integer Linear

Programming formulation (denoted as ILP) method. To compare runtimes and optimalities of different methods, we vary the values of design constraints, i.e., upper bounds of reaction time (RXN) or data age (DA).

The parameters of the experiments are set as follows. The set of concerned task chains $\mathcal{L}$ start from the source blocks which do not have any preceding blocks. Traverse the successors of these blocks and select the first block that has not been included in any existing concerned task chains as the next block. This step is repeated until the task chain reaches the sink block which does not have any successors. Therefore, each concerned task chain is unique and do not have any overlapped blocks with other concerned task chains in $\mathcal{L}$. The weight β of each concerned task chain is set as the weight of the source task. This metric has the benefit of reflecting the importance of the input data of each task chain. The weights are divided by the largest weight (i.e., 9768) and normalized into the range of (0,1]. The objective is to minimize the weighted sum of RXN or DA of all concerned task chains in $\mathcal{L}$. The upper bound of RXN or DA is set to 10^4 milliseconds. We find that keep increasing this upper bound will not significantly improve the objective value or the runtime for this task system. We set the maximal runtime of both methods to 12 hours ($4.32 \cdot 10^7$ in milliseconds). If the ILP method uses up all assigned time but still does not find optimal solution, it will return the status as *timeout* and terminate. If any method finds the optimal solution, it will return the status as *optimal* and terminate.

We evaluate the effectiveness of the proposed framework by optimizing RXN and DA separately. It can be seen from table 1 that the MUDPA-Guided optimization framework can find the optimal solution with varying upper bound constraints. In contrast, the ILP method cannot find any feasible solutions (denoted as N/A) within the assigned time if the constraints are tight. ILP method can find the solution for the two most relaxed cases, however, the solution is suboptimal to its counterpart of MUDPA-Guided, and it is at least 10^6 slower.

The results are similar in the DA case where ILP can only find feasible solution with the most relaxed constraints.

Table 4.1: Results with varying RXN upper bounds (in milliseconds)

RXN upper bound	**MUDPA-Guided**			**ILP**		
	RXN	***Runtime***	***Status***	***RXN***	***Runtime***	***Status***
8500	4636.92	11925.03	Optimal	N/A	4.32E7	Timeout
9000	4636.85	1632.25	Optimal	N/A	4.32E7	Timeout
9500	4629.56	8.91	Optimal	7150.03	4.32E7	Timeout
10000	4629.56	7.62	Optimal	6353.69	4.32E7	Timeout

Table 4.2: Results with varying DA upper bounds (in milliseconds)

DA upper bound	**MUDPA-Guided**			**ILP**		
	DA	***Runtime***	***Status***	***DA***	***Runtime***	***Status***
8500	7150.72	12042.75	Optimal	N/A	4.32E7	Timeout
9000	7150.55	1640.21	Optimal	N/A	4.32E7	Timeout
9500	7143.28	8.41	Optimal	N/A	4.32E7	Timeout
10000	7143.28	9.58	Optimal	9914.73	4.32E7	Timeout

4.8 Conclusion

We focus on the optimization problem of priority assignment that minimizes reaction time or data age in the Simulink model. We first propose a method to calculate reaction time and data age in the Simulink model. We then present the concept of deadline and priority assignment (DPA) to provide a compact representation of the schedulability constraints. An

optimization framework based on the proposed concept is proposed and compared with a direct ILP formulation. Experimental results show that our framework runs at least 1000× faster and keeps the same level of optimality as ILP.

Chapter 5

Conclusions

5.1 Summary

In this dissertation, we propose several algorithms around the main topic of priority assignment. In our studies, different applications and hardware platforms are considered, which have specific requirements of run-time and schedulability. We propose both heuristic methods and an optimization framework for priority assignment and aim to achieve better schedulability (i.e., acceptance ratio of the system) and efficiency (i.e., run-time).

In Chapter 2, we consider the problem of priority assignment for global fixed priority scheduling (G-FP) on a multiprocessor platform. We first propose the concept of response time estimation range, which serves as the estimation of actual response time and enables a broader range of response time analysis to be leveraged than the Audsley's optimal priority assignment algorithm. We then establish a framework called the Maximum Infeasible Response Time Estimation Range (MITER), which leverages the concept of the response time estimation range and rules out infeasible priority assignments in a more efficient way. Lastly, we take advantage of both the optimization framework (i.e., MITER) and the heuristics (i.e., DkC, DCMPO etc.) to perform better system schedulability and run-time.

We then consider the problem of priority assignment of a real-world robotic application, whose hardware platform is more complex and has specific metrics for safety and perfor-

mance in Chapter 3. To begin with, we propose the stochastic heterogeneous parallel conditional DAG (SHPC-DAG) model. It extends the model in [57] by introducing probabilities/probability distributions associated with edges and DAG nodes. A novel metric, the Safety-Performance (SP) metric of the study [83], is used to explore the trade-offs between timing constraints and system performance for safety-critical systems. Furthermore, we propose a partitioned scheduling algorithm called Least Latency Priority (LLP) assignment which consists of novel allocation and scheduling algorithms for stochastic heterogeneous conditional DAG scheduling. To the best of our knowledge, this is the first work that does not make any assumptions about the specific distributions of random execution times and analyzes the distribution of the response times of safety-critical paths and nodes using a convolution-based method.

Finally, in Chapter 4, we focus on the priority assignment optimization problem that minimizes reaction time or data age in the Simulink model. Due to the existence of the Rate-Transition (RT) block, the data transfer in the Simulink model can be analyzed in a predictable way. Using the concepts of reaction time (RXN) and data age (DA) in the study [94], we propose the method to calculate the RXN and DA of task chains in the Simulink model. We then present the concept of deadline and priority assignment (DPA) to provide a compact representation of the schedulability constraints. An optimization framework based on the proposed concept is proposed and compared with a direct ILP formulation. The experimental results show that our framework runs several magnitudes faster than a direct ILP formulation while maintaining the same level of optimality as ILP.

5.2 Future Work

In this dissertation, we focus on the topic of priority assignment in various settings of the task system. The experimental results have shown that our proposed methods can perform better with respect to the acceptance ratio and/or run-time. However, challenges still exist in our studies and can be explored further. We summarize the main challenges or limitations that can be addressed in future work.

In Chapter 2, we propose a hybrid framework which judiciously takes advantage of both optimization technique and heuristics. Our method achieves a better acceptance ratio by finding a more feasible priority assignment for the task system. However, this is at the cost of a much slower run-time. For G-FP, there might exist a heuristic method which runs much faster and maintains the same level of optimality that is close to the optimization framework. If we consider a scheduling algorithm other than G-FP for future work, we may be able to extend our optimization framework to different scheduling algorithms and task models, such as global dynamic priority scheduling and the arbitrary deadline task model.

For our second work in Chapter 3, the major limitation is that we make assumptions about a single DAG. We plan to extend our work with multiple DAGs and SP metric with a third objective function of software reliability.

Finally, in Chapter 4, we conduct our experiments under the assumption of a uniprocessor platform. However, the optimization process and calculation of RXN/DA on the multiprocessor platform may be very different from what we currently have. The allocation strategy for assigning tasks to processors may also have an impact on the calculation of RXN/DA. Our work can be further extended to take these challenges into account.

Appendices

Appendix A

RT dependency of schedulability tests

A.1 Proof of RTA Test's compatibility with RT dependency

The RTA test [19] proposes that for carry-in jobs, the last time a job can execute is at its worst-case response time rather than at its deadline.

The workload of task τ_j can be written as,

$$W_j^R(l) = N_j^R(l) \cdot C_j + \min\left(C_j, l + R_j - C_j - N_j^R(l) \cdot T_j\right) \tag{A.1}$$

where $N_j^R(l) = \left\lfloor \frac{l+R_j-C_j}{T_j} \right\rfloor$ denotes the number of jobs whose entire execution is inside the time interval with length l. The R_i under G-FP is the least fixed point solution of A.2,

$$R_i = C_i + \left\lfloor \frac{1}{m} \sum_{\tau_j \in hp(i)} I_j^R(R_i, C_i) \right\rfloor \tag{A.2}$$

where the inference $I_j^R(\cdot,\cdot)$ of τ_j depends on the workload function $W_j^R(\cdot)$,

$$I_j^R(l, C_i) = \min\left(W_j^R(l), l - C_i + 1\right) \tag{A.3}$$

The calculation of interference $I_j^R(R_i, C_i)$ depends on the τ_j's response time R_j, which is based on both $hp(i)$ and the relative order among the tasks in $hp(i)$.

Proof. If we assume that the response time of each task τ_j in $hp(i)$ is known, the calculation of $W_j^R(l)$ and $I_j^R(l, C_i)$ will only require knowledge of C_j and T_j, which are independent of the priority level of task τ_j. Hence, the calculation of the response time of task τ_i relies on $hp(i)$ and $lp(i)$, but not on the relative priority order among tasks, which satisfies conditions A1 and A2 of the RT-dependency.

Now we prove that R_i of task τ_i is monotonically nondecreasing with the set $hp(i)$ (i.e., condition A3). As can be seen from (A.3), the interference function $I_j^R(l, C_i)$ is *non-negative*: (i) the minimal value of interval l (i.e., the response time) is C_i. Hence, $l - C_i + 1$ is non-negative; (ii) the term $l + R_j - C_j - N_j^R(l) \cdot T_j$ in (A.1) is non-negative because,

$$N_j^R(l) \cdot T_j = \left\lfloor \frac{l + R_j - C_j}{T_j} \right\rfloor \cdot T_j \leq l + R_j - C_j \tag{A.4}$$

Therefore, the interference function $I_j^R(l, C_i)$ is always non-negative. When the taskset $hp(i)$ increases, the newly added task τ_j in $hp(i)$ introduces more interference $I_j^R(l, C_i)$ in task τ_i, thus increasing the sum of the second term in (A.2). Hence, the response time R_i is nondecreasing with the set $hp(i)$, which satisfies condition A3 of the RT-dependency.

We now prove that the workload function $W_j^R(\cdot)$ as in Equation (A.1) will be monotonically nondecreasing if the response time of other task τ_j $(j \neq i)$ increases. This easily leads to the result that the interference $I_j^R(\cdot, \cdot)$ in (A.3) and response time R_i in (A.2) are also monotonically nondecreasing with R_j, and condition A4 is satisfied.

For any schedulable system, we must have $C_j \leq T_j$. We first show $W_j^R(l)$ is monotonic for $l + R_j - C_j \in [k \cdot T_j, (k+1)T_j)$, where the value of $N_j^R(l)$ is always equal to k. Let

$x = l + R_j - C_j - k \cdot T_j$, then $W_j^R(l) = k \cdot C_j + \min(C_j, x)$. We consider two cases.

- Case 1: $l + R_j - C_j \in [k \cdot T_j, k \cdot T_j + C_j]$. Then the value of x will fall inside the interval $[0, C_j]$. The workload is

$$W_j^R(l) = k \cdot C_j + x,\ x \in [0, C_j] \tag{A.5}$$

- Case 2: $l + R_j - C_j \in [k \cdot T_j + C_j, (k+1) \cdot T_j)$. x now becomes no smaller than C_j. Hence, the workload load is a constant in this case:

$$W_j^R(l) = k \cdot C_j + C_j \tag{A.6}$$

From the above two cases, we can conclude that when $l + R_j - C_j \in [k \cdot T_j, (k+1) \cdot T_j)$, the workload $W_j^R(l)$ is monotonically non-decreasing when R_j increases. Now we prove that when $l + R_j - C_j = (k+1) \cdot T_j$ (i.e., $N_j^R(l) = k+1$), the workload is guaranteed to be larger than or equal to the one when $l + R_j - C_j = (k+1) \cdot T_j - 1$. From the previous discussion, we already know that

$$W_j^R(l) = k \cdot C_j + C_j,\ \text{when } l + R_j - C_j = (k+1) \cdot T_j - 1 \tag{A.7}$$

When $l + R_j - C_j$ increases by 1, we have

$$W_j^R(l) = (k+1) \cdot C_j + \min(C_j, 0) \tag{A.8}$$

$$= (k+1) \cdot C_j,\ \text{when } l + R_j - C_j = (k+1) \cdot T_j \tag{A.9}$$

Therefore, the workload $W_j^R(l)$ is monotonically non-decreasing in the interval $[k \cdot T_j, (k+1) \cdot T_j]$ for arbitrary integer k, and consequently it is monotonically non-decreasing with R_j.

As the RTA test satisfies all the conditions A1-A4 of RT-dependency, we prove that it is RT-dependent. □

A.2 Proof of RTA-LC Test's compatibility with RT dependency

RTA-LC [48] limits the interference from carry-in jobs, which uses the response time R_j as the latest completion time of a higher priority task τ_j. The response time of task τ_i is the least fixed point solution of A.10,

$$R_i = C_i + \left\lfloor \frac{1}{m} \Omega_i(R_i) \right\rfloor \tag{A.10}$$

where the function $\Omega_i(\cdot)$ is defined as,

$$\Omega_i(l) = \sum_{j \in hp(i)} I_j^N(l, C_i) + \sum_{j \in [\![i]\!]_{m-1}} \left\{ I_j^R(l, C_i) - I_j^N(l, C_i) \right\} \tag{A.11}$$

where $[\![i]\!]_{m-1}$ denotes the set of $m-1$ tasks in $hp(i)$ that have the largest values of $I_j^R(l, C_i) - I_j^N(l, C_i)$, $I_j^R(l, C_i)$ is defined in (A.3) and,

$$I_j^N(l, C_i) = \min \left\{ \left\lfloor \frac{l}{T_j} \right\rfloor C_j + \min(C_j, l - \left\lfloor \frac{l}{T_j} \right\rfloor T_j), l - C_i + 1 \right\} \tag{A.12}$$

Proof. If we assume the response time R_j of each task τ_j in taskset $hp(i)$ is known, the calculation of workload $W_j^R(l)$ and the interference $I_j^R(l, C_i)$ will only require the knowledge of C_j, T_j and C_i, which are all independent from task τ_j's priority level. The interference $I_j^N(l, C_i)$ does not require knowledge of task τ_j's response time. Hence, the calculation of the

function $\Omega_i(l)$ and thus the calculation of response time R_i will be independent from task τ_j's priority level in the taskset $hp(i)$. Therefore, the conditions A1 and A2 of RT-dependency are satisfied.

If the taskset $hp(i)$ increases, the first term in (A.11) will be non-decreasing as more higher-priority tasks are involved and $I_j^N(l, C_i)$ is non-negative. The second term in (A.11) will either stay the same or increase as only $m-1$ tasks with the largest values of $I_j^R(l, C_i) - I_j^N(l, C_i)$ will be selected. Hence, both $\Omega_i(l)$ and response time R_i will be nondecreasing when $hp(i)$ increases. The condition A3 is satisfied.

If the response time R_j of any task in the set $hp(i)$ increases, it will not affect the values of $I_j^N(l, C_i)$ as it does not require knowledge of R_j. However, the value of $I_j^R(l, C_i)$ will be monotonically non-decreasing when R_j increases according to (A.1) and (A.3), which is proved in the previous case of RTA test. Hence, the value of response time R_i will also be non-decreasing if R_j increases.

As RTA-LC test is RT-dependent as it satisfies all the four conditions. □

A.3 Proof of EPE-ZLL's incompatibility with RT dependency

EPE-ZLL tries to reduce the overestimation of interferences from all parts of the workload (carry-in, body, carry-out), not just those from the carry-in. In this section, we prove that the EPE-ZLL test does not satisfy the conditions of RT-dependency.

To calculate the response time of τ_i, EPE-ZLL uses an iterative process that gradually increases the value of its execution time a from one (i.e., $a = 1$) to its real value C_i ($a = C_i$). At each iteration, it calculates the corresponding response time UR_i^{a+1} based on the value

of UR_i^a. Specifically, UR_i^{a+1} is the minimal solution of the following equation,

$$x \geq \left\lfloor \frac{\Psi_i(a,x)}{m} \right\rfloor + a + 1 \tag{A.13}$$

where $\Psi_i(a,x)$ is defined as

$$\Psi_i(a,x) = \min\Big\{ \quad m\left(UR_i^a - a\right) + \sum_{j \in hp(i) \cup i} W_{j,i}^{\max}(a,x), \\ \Omega_i\left(x + UR_i^a\right) \Big\} \tag{A.14}$$

The calculation of $\Omega_i\left(x + UR_i^a\right)$ follows the definition in (A.11). The term $W_{j,i}^{\max}(a,x)$ requires the knowledge of relative order among tasks in the set $hp(i)$ from the study [120].

Proof. The main idea of EPE-ZLL test is to first calculate UR_i^1 by an existing RTA test and then gradually increase C_i to calculate UR_i^a under increasing value of a ($1 \leq a \leq C_i$). For each iteration, the value of UR_i^a from previous step will be known as a constant when we calculate the value of UR_i^{a+1}. Therefore, the first term $m(UR_i^a - a)$ in (A.14) is known as a constant. As we proved in RTA and RTA-LC test, if we assume the response time R_j of each task in $hp(i)$ is known, the calculation of $\Omega_i\left(x + UR_i^a\right)$ will be independent from τ_j's priority level. To prove whether EPE-ZLL test satisfies the conditions A1 and A2 of RT dependency, we only need to focus on the the term $\sum_{j \in hp(i) \cup i} W_{j,i}^{\max}(a,x)$.

In [120], the term $W_{j,i}^{\max}(a,x)$ is calculated by an algorithm that requires the relative priority order among tasks in the set $hp(i)$ to determine the worst case release times of interfering jobs. First, we introduce a set of definitions proposed in the study [120].

Let $\mathbf{L}, \mathbf{P}$ be two time intervals respectively,

$$\mathbf{L} = [r_{i,l}, r_{i,l} + UR_i^a] \tag{A.15}$$

$$\mathbf{P} = [r_{i,l} + UR_i^a, x) \tag{A.16}$$

where (i) $r_{i,l}$ denotes the release time of the job of τ_i whose response time is equal to the WCRT of τ_i (ii) $r_{j,k}$ denotes the k-th job released by task τ_j and (iii) x denotes the worst case response time.

Let $c = \min\{r_{i,l} + x - r_{j,k}, x\}$, $b = r_{i,l} + UR_i^a - r_{j,k}$. The calculations of interference from carry-in (i.e., $I_i^R(UR_i^a, a)$) and non-carry-in (i.e., $I_i^N(UR_i^a, a)$) jobs follow the definitions in (A.3) and (A.12). The total interference $\Omega_i(\cdot, z)$ follow the definition in (A.11), where $z = m-1$ or $m-2$ depends on whether τ_j is a carry-in task or non-carry-in task over $\mathbf{L}$.

Lemma A.1. *The two lower bounds of the interference on the target job $J_{i,l}$ by any higher priority task τ_j in $\mathbf{L}$ are,*

$$I_i^R(x, a) = m(UR_i^a - a) - \Omega_i(a, m-2) \tag{A.17}$$

$$I_i^N(x, a) = m(UR_i^a - a) - \Omega_i(a, m-1) \tag{A.18}$$

Lemma A.2. *In [120], there are two lower bounds of the accumulative time for parallel execution in $\boldsymbol{L}$. If $J_{j,k}$ has execution after $r_{i,l} + UR_i^a$, the lower bound will be,*

$$\vartheta_{j,i}(a, c) = \max\{x' | UR_i^{x'} \leq c\} \tag{A.19}$$

If $J_{j,k}$ has no execution after $r_{i,l} + UR_k^a$, the lower bound will be,

$$\vartheta'_{j,i}(a, c) = [\![C'_j + \max\{x' | UR_i^{x'} \leq c\} - b]\!]_0 \tag{A.20}$$

where C'_i is the real execution time of $J_{j,k}$.

Theorem A.3. *The workload of τ_j in* **L**, *depends on whether $J_{j,k}$ has execution after* **L** *or not,*

$$W^R_{j,i}(a) = I^R_j(a) + \vartheta_{j,i}(a,c) \tag{A.21}$$

$$W^N_{j,i}(a) = I^N_j(a) + \vartheta_{j,i}(a,c) \tag{A.22}$$

$$\overline{W^R_{j,i}}(a) = I^R_j(a) + \vartheta'_{j,i}(a,c) \tag{A.23}$$

$$\overline{W^R_{j,i}}(a) = I^N_j(a) + \vartheta'_{j,i}(a,c) \tag{A.24}$$

To calculate the maximum workload of τ_j in **P**, the loop iterates from $b = 0$ to $b = T_j$. To derive $W_{j,i}(a,b,x)$,

$$W_{j,i}(a,b,x) = \begin{cases} [\![C_j - W^R_{j,i}(a)]\!]^\lambda & \text{, if } b > UR^a_i \\ [\![[\![W_1]\!]^d_{W2}]\!]^\lambda & \text{, if } b \leq UR^a_i \end{cases} \tag{A.25}$$

where $d = C_j - \vartheta_{j,i}(a,c)$. Other parameters and terms are defined in [120]. When calculating the workload $W^R_{j,i}(a)$ in (A.25), we require the value of c. Note that $c = \min\{r_{i,l} + x - r_{j,k}, x\}$ where $r_{j,k}$ represents the release time of last job (of a higher priority task τ_j than τ_i) released no later than $r_{i,l} + UR^a_i$.

If we assume the response times R_j of all other tasks (other than task τ_i) are known, we still need the information about relative priority order among tasks in the set $hp(i)$ to determine the release patterns of jobs generated by tasks in $hp(i)$. Specifically, the release time of $r_{j,k}$ highly depends on the exact priority ordering of tasks in the set $hp(i)$. If we swap the priorities of task τ_j with any other task in $hp(i)$, the release pattern of jobs might change and thus affecting the values of $W^{\max}_{j,i}(a,x)$ and the calculation of response time R_i. This violates

conditions A1 and A2 of RT-dependency because we need the exact priority ordering of set $hp(i)$ to calculate the response time R_i. Therefore, EPE-ZLL is not RT-dependent. □

Bibliography

[1] Zaid Al-bayati, Haibo Zeng, Marco Di Natale, and Zonghua Gu. Multitask implementation of synchronous reactive models with earliest deadline first scheduling. In *2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 168–177. IEEE, 2013.

[2] Slim Ben Amor. *Scheduling of Dependent Tasks with Probabilistic Execution Times on Multi-core Processors*. PhD thesis, Sorbonne Université, 2020.

[3] B. Andersson and J. Jonsson. Some insights on fixed-priority preemptive non-partitioned multiprocessor scheduling. In *IEEE Real-Time Systems Symposium, Work-in-Progress Session*, pages 19–23, 2000.

[4] David L Applegate, Robert E Bixby, Vašek Chvatál, and William J Cook. *A Computational Study*. Princeton University Press, 2006.

[5] N.C. Audsley. On priority assignment in fixed priority scheduling. *Information Processing Letters*, 79(1):39–44, 2001. ISSN 0020-0190. doi: https://doi.org/10.1016/S0020-0190(00)00165-4. URL https://www.sciencedirect.com/science/article/pii/S0020019000001654.

[6] T. Baker and M. Cirinei. Brute-force determination of multiprocessor schedulability for sets of sporadic hard-deadline tasks. In *International Conference On Principles Of DIstributed Systems*, pages 62–75. Springer, 2007.

[7] Shamit Bansal, Yecheng Zhao, Haibo Zeng, and Kehua Yang. Optimal implemen-

tation of simulink models on multicore architectures with partitioned fixed priority scheduling. In *IEEE Real-Time Systems Symposium*, pages 242–253, 2018.

[8] S. Baruah. Techniques for multiprocessor global schedulability analysis. In *IEEE Real-Time Systems Symposium*, pages 119–128, 2007. doi: 10.1109/RTSS.2007.35.

[9] Sanjoy Baruah. An ilp representation of a dag scheduling problem. *Real-Time Systems*, pages 1–18, 2021.

[10] Sanjoy Baruah. An ilp representation of a dag scheduling problem. *Real-Time Systems*, pages 1–18, 2021.

[11] Sanjoy Baruah and Alan Burns. Sustainable scheduling analysis. In *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*, pages 159–168. IEEE, 2006.

[12] Sanjoy Baruah, Vincenzo Bonifaci, Gianlorenzo DAngelo, Haohan Li, Alberto Marchetti-Spaccamela, Suzanne Van Der Ster, and Leen Stougie. The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *2012 24th Euromicro Conference on Real-Time Systems*, pages 145–154. IEEE, 2012.

[13] Sanjoy Baruah, Marko Bertogna, and Giorgio Buttazzo. *Multiprocessor scheduling for real-time systems*. Springer, 2015.

[14] Sanjoy Baruah, Vincenzo Bonifaci, and Alberto Marchetti-Spaccamela. The global edf scheduling of systems of conditional sporadic dag tasks. In *2015 27th Euromicro Conference on Real-Time Systems*, pages 222–231, 2015. doi: 10.1109/ECRTS.2015.27.

[15] Ron Bell. Introduction to iec 61508. In *Acm international conference proceeding series*, volume 162, pages 3–12. Citeseer, 2006.

[16] Slim Ben-Amor and Liliana Cucu-Grosjean. Graph reductions and partitioning heuristics for multicore dag scheduling. *Journal of Systems Architecture*, 124:102359, 2022.

[17] Guillem Bernat, Alan Burns, and Albert Liamosi. Weakly hard real-time systems. *IEEE transactions on Computers*, 50(4):308–321, 2001.

[18] M. Bertogna. *Real-Time Scheduling Analysis for Multiprocessor Platforms*. PhD thesis, Scuola Superiore Sant'Anna, Pisa, 2007.

[19] M. Bertogna and M. Cirinei. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In *28th IEEE International Real-Time Systems Symposium*, pages 149–160, 2007. doi: 10.1109/RTSS.2007.31.

[20] M. Bertogna, M. Cirinei, and G. Lipari. Schedulability analysis of global scheduling algorithms on multiprocessor platforms. *IEEE Transactions on Parallel and Distributed Systems*, 20(4):553–566, 2009. doi: 10.1109/TPDS.2008.129.

[21] V. Bonifaci and A. Marchetti-Spaccamela. Feasibility analysis of sporadic real-time multiprocessor task systems. *Algorithmica*, 63:763–780, 2012.

[22] A. Burmyakov, E. Bini, and E. Tovar. An exact schedulability test for global fp using state space pruning. In *23rd International Conference on Real Time and Networks Systems*, pages 225–234, 2015.

[23] A. Burmyakov, E. Bini, and C. Lee. Towards a tractable exact test for global multiprocessor fixed priority scheduling. *IEEE Transactions on Computers*, 71(11):2955–2967, 2022. doi: 10.1109/TC.2022.3142540.

[24] Louis-Claude Canon and Emmanuel Jeannot. Evaluation and optimization of the robustness of dag schedules in heterogeneous environments. *IEEE Transactions on Parallel and Distributed Systems*, 21(4):532–546, 2010. doi: 10.1109/TPDS.2009.84.

[25] Kun Cao, Junlong Zhou, Peijin Cong, Liying Li, Tongquan Wei, Mingsong Chen, Shiyan Hu, and Xiaobo Sharon Hu. Affinity-driven modeling and scheduling for makespan optimization in heterogeneous multiprocessor systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(7):1189–1202, 2019. doi: 10.1109/TCAD.2018.2846650.

[26] Daniel Casini, Alessandro Biondi, Geoffrey Nelissen, and Giorgio Buttazzo. Partitioned fixed-priority scheduling of parallel tasks without preemptions. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 421–433. IEEE, 2018.

[27] Paul Caspi, Norman Scaife, Christos Sofronis, and Stavros Tripakis. Semantics-preserving multitask implementation of synchronous programs. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(2):1–40, 2008.

[28] Shuangshuang Chang, Xufeng Zhao, Zhenyu Liu, and Qingxu Deng. Real-time scheduling and analysis of parallel tasks on heterogeneous multi-cores. *Journal of Systems Architecture*, 105:101704, 2020.

[29] Long Chen, Shaobo Lin, Xiankai Lu, Dongpu Cao, Hangbin Wu, Chi Guo, Chun Liu, and Fei-Yue Wang. Deep neural network based vehicle and pedestrian detection for autonomous driving: A survey. *IEEE Trans. Intell. Transp. Syst.*, 22(6):3234–3246, June 2021.

[30] Liliana Cucu. Optimal priority assignment for periodic tasks on unrelated processors. In *Euromicro Conference on Real-Time Systems, WIP session*, 2008.

[31] Liliana Cucu and Joel Goossens. Feasibility intervals for multiprocessor fixed-priority scheduling of arbitrary deadline periodic systems. In *Design, Automation & Test in Europe Conference*, pages 1–6, 2007.

[32] R. Davis and A. Burns. Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In *30th IEEE Real-Time Systems Symposium*, pages 398–409, 2009.

[33] R. Davis and A. Burns. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real-Time Systems*, 47(1): 1–40, 2011.

[34] R. Davis, L. Cucu-Grosjean, M. Bertogna, and A. Burns. A review of priority assignment in real-time systems. *Journal of systems architecture*, 65:64–82, 2016.

[35] Robert I. Davis and Alan Burns. Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In *IEEE Real-Time Systems Symposium*, 2009. doi: 10.1109/RTSS.2009.31.

[36] Robert I Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM computing surveys*, 43(4):1–44, 2011.

[37] Xuanliang Deng, Shriram Raja, Yecheng Zhao, and Haibo Zeng. Priority assignment for global fixed priority scheduling on multiprocessors, © 2024 ieee. reprinted, with permission. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(9):2538–2550, 2024. doi: 10.1109/TCAD.2024.3376588.

[38] Marco Di Natale. Optimizing the multitask implementation of multirate simulink models. In *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, pages 335–346. IEEE, 2006.

[39] Marco Di Natale and Haibo Zeng. Task implementation of synchronous finite state machines. In *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 206–211. IEEE, 2012.

[40] Marco Di Natale, Liangpeng Guo, Haibo Zeng, and Alberto Sangiovanni-Vincentelli. Synthesis of multitask implementations of simulink models with minimum delays. *IEEE Transactions on Industrial Informatics*, 6(4):637–651, 2010.

[41] Marco Di Natale, Qi Zhu, Alberto Sangiovanni-Vincentelli, and Stavros Tripakis. Optimized implementation of synchronous models on industrial ltta systems. *Journal of Systems Architecture*, 60(4):315–328, 2014.

[42] S. Edgar and A. Burns. Statistical analysis of wcet for scheduling. In *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001) (Cat. No.01PR1420)*, pages 215–224, 2001. doi: 10.1109/REAL.2001.990614.

[43] Nico Feiertag, Kai Richter, Johan Nordlander, and Jan Jonsson. A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics. In *IEEE Real-Time Systems Symposium: 30/11/2009-03/12/2009*. IEEE Communications Society, 2009.

[44] Vitaly Feldman and Chiyuan Zhang. What neural networks memorize and why: Discovering the long tail via influence estimation. *arXiv preprint arXiv:2008.03703*, 2020.

[45] José Carlos Fonseca, Vincent Nélis, Gurulingesh Raravi, and Luís Miguel Pinho. A multi-dag model for real-time parallel applications with conditional execution. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 1925–1932, 2015.

[46] Pourya Gohari, Mitra Nasri, and Jeroen Voeten. Data-age analysis for multi-rate task chains under timing uncertainty. In *Proceedings of the 30th International Conference on Real-Time Networks and Systems*, pages 24–35, 2022.

[47] Patrick Graydon and Iain Bate. Safety assurance driven problem formulation for mixed-criticality scheduling. *Proc. WMC, RTSS*, pages 19–24, 2013.

[48] N. Guan, M. Stigge, W. Yi, and G. Yu. New response time bounds for fixed priority multiprocessor scheduling. In *30th IEEE Real-Time Systems Symposium*, pages 387–397, 2009. doi: 10.1109/RTSS.2009.11.

[49] Mario Günzel, Kuan-Hsun Chen, Niklas Ueter, Georg von der Brüggen, Marco Dürr, and Jian-Jia Chen. Timing analysis of asynchronized distributed cause-effect chains. In *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 40–52. IEEE, 2021.

[50] Mario Günzel, Kuan-Hsun Chen, Niklas Ueter, Georg von der Brüggen, Marco Dürr, and Jian-Jia Chen. Compositional timing analysis of asynchronized distributed cause-effect chains. *ACM Transactions on Embedded Computing Systems*, 22(4):1–34, 2023.

[51] Mario Günzel, Harun Teper, Georg von der Brueggen, and Jian-Jia Chen. End-to-end latency of cause-effect chains: A tutorial. *ACM Transactions on Embedded Computing Systems*, 2024.

[52] A. Hamann, M. Jersak, K. Richter, and R. Ernst. Design space exploration and system optimization with symta/s - symbolic timing analysis for systems. In *IEEE Real-Time Systems Symposium*, 2004.

[53] Gang Han, Marco Di Natale, Haibo Zeng, Xue Liu, and Wenhua Dou. Optimizing the implementation of real-time simulink models onto distributed automotive architectures. *Journal of Systems Architecture*, 59(10):1115–1127, 2013.

[54] Meiling Han, Nan Guan, Jinghao Sun, Qingqiang He, Qingxu Deng, and Weichen Liu.

Response time bounds for typed dag parallel tasks on heterogeneous multi-cores. *IEEE Transactions on Parallel and Distributed Systems*, 30(11):2567–2581, 2019.

[55] Qingqiang He, Mingsong Lv, and Nan Guan. Response time bounds for dag tasks with arbitrary intra-task priority assignment. In *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[56] Larkin Heintzman, Amanda Hashimoto, Nicole Abaid, and Ryan K Williams. Anticipatory planning and dynamic lost person models for Human-Robot search and rescue. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8252–8258, May 2021.

[57] Zahaf Houssam-Eddine, Nicola Capodieci, Roberto Cavicchioli, Giuseppe Lipari, and Marko Bertogna. The hpc-dag task model for heterogeneous real-time systems. *IEEE Transactions on Computers*, 70(10):1747–1761, 2021. doi: 10.1109/TC.2020.3023169.

[58] Jing Huang, Renfa Li, Xun Jiao, Yu Jiang, and Wanli Chang. Dynamic dag scheduling on multiprocessor systems: Reliability, energy, and makespan. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):3336–3347, 2020. doi: 10.1109/TCAD.2020.3013045.

[59] Wen-Hung Huang, Maolin Yang, and Jian-Jia Chen. Resource-oriented partitioned scheduling in multiprocessor systems: How to partition and how to share? In *2016 IEEE Real-Time Systems Symposium (RTSS)*, pages 111–122. IEEE, 2016.

[60] Leslie A Johnson et al. Do-178b, software considerations in airborne systems and equipment certification. *Crosstalk, October*, 199:11–20, 1998.

[61] S. Lee, H. Baek, H. Woo, K. Shin, and J. Lee. Ml for rt: Priority assignment using

machine learning. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 118–130, 2021. doi: 10.1109/RTAS52030.2021.00018.

[62] Kenli Li, Xiaoyong Tang, Bharadwaj Veeravalli, and Keqin Li. Scheduling precedence constrained stochastic tasks on heterogeneous cluster systems. *IEEE Transactions on computers*, 64(1):191–204, 2013.

[63] Keyu Li, Yangxin Xu, Jiankun Wang, and Max Q-H Meng. SARL*: Deep reinforcement learning based Human-Aware navigation for mobile robot in indoor environments. In *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 688–694, December 2019.

[64] P Lin. The robot car of tomorrow may just be programmed to hit you. *Machine Ethics and Robot Ethics*, 2020.

[65] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, January 1973. ISSN 0004-5411.

[66] Dorin Maxim and Liliana Cucu-Grosjean. Response time analysis for fixed-priority tasks with multiple probabilistic parameters. In *2013 IEEE 34th Real-Time Systems Symposium*, pages 224–235. IEEE, 2013.

[67] Alessandra Melani, Marko Bertogna, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Giorgio C Buttazzo. Response-time analysis of conditional dag tasks in multiprocessor systems. In *2015 27th Euromicro Conference on Real-Time Systems*, pages 211–221. IEEE, 2015.

[68] Alexander McFarlane Mood. Introduction to the theory of statistics. 1950.

[69] Marco Di Natale and Valerio Pappalardo. Buffer optimization in multitask imple-

mentations of simulink models. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):1–32, 2008.

[70] Risat Pathan, Petros Voudouris, and Per Stenström. Scheduling parallel real-time recurrent tasks on multicore platforms. *IEEE Transactions on Parallel and Distributed Systems*, 29(4):915–928, 2017.

[71] Paulo Pedro and Alan Burns. Schedulability analysis for mode changes in flexible real-time systems. In *EUROMICRO Workshop on Real-Time Systems*, 1998.

[72] Chao Peng and Haibo Zeng. Response time analysis of digraph real-time tasks scheduled with static priority: generalization, approximation, and improvement. *Real-Time Systems*, 54(1):91–131, 2018.

[73] Chao Peng, Haibo Zeng, and Marco Di Natale. A comparison of schedulability analysis methods using state and digraph models for the schedulability analysis of synchronous fsms. *Real-Time Systems*, 55:598–638, 2019.

[74] M Rangwala, J Liu, K S Ahluwalia, S Ghajar, H S Dhami, and others. DeepPaSTL: Spatio-Temporal deep learning methods for predicting Long-Term pasture terrains using synthetic datasets. *Agronomy*, 2021.

[75] Denise Ratasich, Bernhard Frömel, Oliver Höftberger, and Radu Grosu. Generic sensor fusion package for ros. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 286–291. IEEE, 2015.

[76] M. Saksena and Yun Wang. Scalable real-time system design using preemption thresholds. In *IEEE Real-Time Systems Symposium*, 2000.

[77] Lukas Schmid, Victor Reijgwart, Lionel Ott, Juan Nieto, Roland Siegwart, and Cesar Cadena. A unified approach for autonomous volumetric exploration of large scale

environments under severe odometry drift. *IEEE Robotics and Automation Letters*, 6 (3):4504–4511, 2021.

[78] Maria A Serrano and Eduardo Quinones. Response-time analysis of dag tasks supporting heterogeneous computing. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018.

[79] Ali Shafti, Pavel Orlov, and A Aldo Faisal. Gaze-based, context-aware robotic system for assisted reaching and grasping. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 863–869. IEEE, 2019.

[80] Shai Shalev-shwartz, Shaked Shammah, and Amnon Shashua. On a Formal Model of Safe and Scalable Self-driving Cars. pages 1–37, 2017.

[81] Mali Shen, Yun Gu, Ning Liu, and Guang-Zhong Yang. Context-aware depth and pose estimation for bronchoscopic navigation. *IEEE Robotics and Automation Letters*, 4(2): 732–739, 2019.

[82] Ashrarul H Sifat, Burhanuddin Bharmal, Haibo Zeng, Jia-Bin Huang, Changhee Jung, and Ryan K Williams. Towards computational awareness in autonomous robots: an empirical study of computational kernels. *Complex & Intelligent Systems*, pages 1–27, 2023.

[83] Ashrarul H. Sifat, Xuanliang Deng, Burhanuddin Bharmal, Sen Wang, Shaoyu Huang, Jiabin Huang, Changhee Jung, Haibo Zeng, and Ryan Williams. A safety-performance metric enabling computational awareness in autonomous robots. *IEEE Robotics and Automation Letters*, 8(9):5727–5734, 2023. doi: 10.1109/LRA.2023.3300251.

[84] BEN-AMOR Slim, CUCU-GROSJEAN Liliana, and Dorin Maxim. Worst-case response time analysis for partitioned fixed-priority dag tasks on identical processors.

In *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1423–1426. IEEE, 2019.

[85] Martin Stigge, Pontus Ekberg, Nan Guan, and Wang Yi. The digraph real-time task model. In *2011 17th IEEE real-time and embedded technology and applications symposium*, pages 71–80. IEEE, 2011.

[86] Jinghao Sun, Nan Guan, Jingchang Sun, and Yaoyao Chi. Calculating response-time bounds for openmp task systems with conditional branches. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 169–181. IEEE, 2019.

[87] Y. Sun and M. Di Natale. Assessing the pessimism of current multicore global fixed-priority schedulability analysis. In *33rd ACM Symposium on Applied Computing*, pages 575–583, 2018.

[88] Y. Sun and M. Di Natale. Pessimism in multicore global schedulability analysis. *Journal of Systems Architecture*, 97:142–152, 2019.

[89] Y. Sun and G. Lipari. A pre-order relation for exact schedulability test of sporadic tasks on multiprocessor global fixed-priority scheduling. *Real-Time Systems*, 52:323–355, 2016.

[90] The MathWorks. The mathworks simulink and stateflow user's manuals. web page: http://www.mathworks.com.

[91] Ken W Tindell, Alan Burns, and Andy J. Wellings. Allocating hard real-time tasks: an np-hard problem made easy. *Real-Time Systems*, 4(2):145–165, 1992.

[92] Niklas Ueter, Mario Günzel, and Jian-Jia Chen. Response-time analysis and optimiza-

tion for probabilistic conditional parallel dag tasks. In *2021 IEEE Real-Time Systems Symposium (RTSS)*, pages 380–392, 2021. doi: 10.1109/RTSS52674.2021.00042.

[93] Micaela VERUCCHI and Marko BERTOGNA. A comprehensive analysis of dag tasks: solutions for modern real-time embedded systems. *PhD Thesis*, 2021.

[94] Micaela Verucchi, Mirco Theile, Marco Caccamo, and Marko Bertogna. Latency-aware generation of single-rate dags from multi-rate task sets. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 226–238. IEEE, 2020.

[95] AS Vincentelli, Paolo Giusto, Claudio Pinello, Wei Zheng, and MD Natale. Optimizing end-to-end latencies by adaptation of the activation events in distributed automotive systems. In *13th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'07)*, pages 293–302. IEEE, 2007.

[96] C. Wang, Z. Gu, and H. Zeng. Global fixed priority scheduling with preemption threshold: Schedulability analysis and stack size minimization. *IEEE Trans. Parallel and Distributed Systems*, 27(11):3242–3255, Nov. 2016.

[97] Sen Wang, Ryan K Williams, and Haibo Zeng. A general and scalable method for optimizing real-time systems with continuous variables. In *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 119–132. IEEE, 2023.

[98] Sen Wang, Dong Li, Shao-Yu Huang, Xuanliang Deng, Ashrarul H Sifat, Jia-Bin Huang, Changhee Jung, Ryan Williams, and Haibo Zeng. Time-triggered scheduling for nonpreemptive real-time dag tasks using 1-opt local search. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(11):3650–3661, 2024.

[99] Sen Wang, Dong Li, Ashrarul H Sifat, Shao-Yu Huang, Xuanliang Deng, Changhee Jung, Ryan Williams, and Haibo Zeng. Optimizing logical execution time model for both determinism and low latency. In *2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 135–148. IEEE, 2024.

[100] Yun Wang and M. Saksena. Scheduling fixed-priority tasks with preemption threshold. In *International Conference on Real-Time Computing Systems and Applications*, 1999.

[101] Ming Xiong, Song Han, Kam-Yiu Lam, and Deji Chen. Deferrable scheduling for maintaining real-time data freshness: Algorithms, analysis, and results. *IEEE Transactions on Computers*, 57(7):952–964, 2008.

[102] Kecheng Yang, Ming Yang, and James H Anderson. Reducing response-time bounds for dag-based task systems on heterogeneous multicore platforms. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, pages 349–358, 2016.

[103] Houssam-Eddine Zahaf, Nicola Capodieci, Roberto Cavicchioli, Marko Bertogna, and Giuseppe Lipari. A c-dag task model for scheduling complex real-time tasks on heterogeneous platforms: preemption matters. *arXiv preprint arXiv:1901.02450*, 2019.

[104] Haibo Zeng and Marco Di Natale. Schedulability analysis of periodic tasks implementing synchronous finite state machines. In *2012 24th Euromicro Conference on Real-Time Systems*, pages 353–362. IEEE, 2012.

[105] Haibo Zeng and Marco Di Natale. Using max-plus algebra to improve the analysis of non-cyclic task models. In *2013 25th Euromicro Conference on Real-Time Systems*, pages 205–214. IEEE, 2013.

[106] Haibo Zeng and Marco Di Natale. Computing periodic request functions to speed-up the analysis of non-cyclic task models. *Real-Time Systems*, 51:360–394, 2015.

[107] Haibo Zeng, Marco Di Natale, and Qi Zhu. Minimizing stack and communication memory usage in real-time embedded applications. *ACM Trans. Embed. Comput. Syst.*, 13(5s):1–25, July 2014.

[108] Hengyu Zhao, Yubo Zhang, Pingfan Meng, Hui Shi, Li Erran Li, Tiancheng Lou, and Jishen Zhao. Safety score: A quantitative approach to guiding safety-aware autonomous vehicle computing system design. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1479–1485, 2020. doi: 10.1109/IV47402.2020.9304602.

[109] Qingling Zhao, Zaid Al-Bayati, Zonghua Gu, and Haibo Zeng. Optimized implementation of multirate mixed-criticality synchronous reactive models. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 22(2):1–25, 2016.

[110] Y. Zhao and H. Zeng. The concept of response time estimation range for optimizing systems scheduled with fixed priority. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 283–294, 2018. doi: 10.1109/RTAS.2018.00036.

[111] Yecheng Zhao and Haibo Zeng. The concept of unschedulability core for optimizing priority assignment in real-time systems. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 232–237. IEEE, 2017.

[112] Yecheng Zhao and Haibo Zeng. The virtual deadline based optimization algorithm for priority assignment in fixed-priority scheduling. In *2017 IEEE Real-Time Systems Symposium (RTSS)*, pages 116–127. IEEE, 2017.

[113] Yecheng Zhao and Haibo Zeng. The concept of unschedulability core for optimizing

real-time systems with fixed-priority scheduling. *IEEE Transactions on Computers*, 68(6):926–938, 2018.

[114] Yecheng Zhao and Haibo Zeng. The concept of maximal unschedulable deadline assignment for optimization in fixed-priority scheduled real-time systems. *Real-Time Systems*, 55:667–707, 2019.

[115] Yecheng Zhao, Chao Peng, Haibo Zeng, and Zonghua Gu. Optimization of real-time software implementing multi-rate synchronous finite state machines. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):1–21, 2017.

[116] Yecheng Zhao, Vinit Gala, and Haibo Zeng. A unified framework for period and priority optimization in distributed hard real-time systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2188–2199, 2018.

[117] Yecheng Zhao, Runzhi Zhou, and Haibo Zeng. An optimization framework for real-time systems with sustainable schedulability analysis. In *2020 IEEE Real-Time Systems Symposium (RTSS)*, pages 333–344. IEEE, 2020.

[118] Yecheng Zhao, Runzhi Zhou, and Haibo Zeng. Design optimization for real-time systems with sustainable schedulability analysis. *Real-Time Systems*, 58(3):275–312, 2022.

[119] Q. Zhou, G. Li, and J. Li. Improved carry-in workload estimation for global multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 28(9): 2527–2538, 2017. doi: 10.1109/TPDS.2017.2679195.

[120] Q. Zhou, J. Li, and G. Li. Excluding parallel execution to improve global fixed priority response time analysis. *ACM Transactions on Embedded Computing Systems*, 20(5s): 1–24, 2021.

[121] Qi Zhu, Peng Deng, Marco Di Natale, and Haibo Zeng. Robust and extensible task implementations of synchronous finite state machines. In *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1319–1324. IEEE, 2013.

[122] Qi Zhu, Haibo Zeng, Wei Zheng, Marco Di Natale, and Alberto Sangiovanni-Vincentelli. Optimization of task allocation and priority assignment in hard real-time distributed systems. *ACM Trans. Embed. Comput. Syst.*, 11(4):1–30, January 2013.