TECHNISCHE UNIVERSITÄT MÜNCHEN
Lehrstuhl für Echtzeitsysteme und Robotik

Schedulability Analysis of General Task Model and Demand Aware Scheduling in Mixed-Criticality Systems

Biao Hu

Vollständiger Abdruck der von der Fakultät der Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Prof. Dr. Hans Michael Gerndt
Pr¨ufer der Dissertation: 1. Prof. Dr.-Ing. habil. Alois Knoll
2. Prof. Dr. Lothar Thiele, ETH Zürich/Schweiz

Die Dissertation wurde am 07.12.2016 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 25.04.2017 angenommen.

Abstract

Nowadays, the embedded systems are undergoing an unprecedented trend towards integrating components or tasks of different criticality levels onto a common computing platform, as the task integration can reduce the "SWaP" (Size, Weight, and Power) related costs. Those systems, commonly referred to mixed-criticality systems, consist of functionalities with two or more distinct criticality levels, e.g. safety criticality and mission criticality; it is of vital importance for systems to meet tasks' requirements corresponding to their own criticality levels. To achieve the mixed-criticality guarantee, less criticality tasks are assumed to be degraded or dropped once they overrun their given execution budgets, and tasks are commonly considered being activated sporadically because sporadic activation model is easier to be analyzed than other complex activation models like sporadic burst model. These two assumptions are however pessimistic because in fact the system may have some slacks to allow tasks overrun; and today's real-time systems are embracing a growing variety of activation patterns whose activation features may deviate a lot from the sporadic activation assumption.

In this thesis, we focus on adaptively postponing the mode-switch online by exploiting the system's static and runtime slacks; and we address the problem of schedulability analysis towards a more general task model in mixed-criticality systems. Specifically, we first propose an on-the-fly fast overrun budgeting mode-switch scheme to online postpone the mode-switch. Then, we extend the current sporadic task model to the arbitrary activation task model (arrival curve) in mixed-criticality systems, based on which we furthermore propose an approach that can adaptively shape the arriving events of task activation. We also present a case study showing the application of some basic mixed-criticality scheduling concepts in an autonomous driving system.

Zusammenfassung

Derzeit gibt es bei der Entwicklung eingebetteter Systeme einen deutlichen Trend Integration von Komponenten mit unterschiedlichen Kritikalitätsstufen auf einer gemeinsamen Berechnungsplattform, da dies die sogenannten SWaP-Kosten (Size, Weight, Power) reduzieren kann. Solche Systeme, die im Allgemeinen als Systeme mit gemischter Kritikalität bezeichnet werden, stellen Funktionalitäten bereit, die mindestens zwei Kritikalitätsstufen zuzuordnen sind, z.B. sicherheitskritische oder missionskritische. Es ist dabei besonders wichtig, dass die Systeme die Anforderungen der Aufgaben gemäß der jeweiligen Kritikalitätsstufe erfüllt. Wir nehmen an, dass Aufgaben auf weniger kritischen Stufen, die ihre Ausführungszeit überschreiten, entweder degradiert oder verworfen werden, um die Anforderungen des Gesamtsystems zu erfüllen. Aufgaben werden dabei sporadisch aktiviert, da dieses Aktivierungsmodell einfacher zu analysieren ist, als z.B. das komplexere sporadische Burst-Modell. Diese beiden Annahmen sind jedoch pessimistisch, da es zu Leerlaufzeiten und somit zu vermeidbaren Situationen kommt, in denen Aufgaben ihr Zeitbudget überschreiten. Zudem werden vermehrt neue Aktivierungstypen entwickelt, deren Eigenschaften sich unter Umständen stark von der sporadischen Aktivierung unterscheiden.

In dieser Arbeit konzentrieren wir uns darauf, den „Modus-Wechsel“ während der Ausführung zu verschieben, indem wir die statischen Leerlaufzeiten des Systems ausnutzen. Hierbei wird ein Analyseverfahren für ein generisches Aufgabenmodell in Systemen mit gemischter Kritikalität entwickelt und genutzt, um ein Online-Verfahren zu entwerfen, das kurzfristig zur Laufzeit ein Zeitbudget für Überläufe nutzt, um so einen Modus-Wechsel zu verschieben. Weiterhin wird das sporadische Aufgabenmodell erweitert, um willkürlich Aufgaben aktivieren zu können. Dies wird genutzt, um eine adaptive Methode zu entwerfen, die den Aktivierungsverlauf von Aufgaben steuern kann.

Außerdem evaluieren wir unseren Ansatz in einer Fallstudie mit gemischter Kritikalität zur Regelung eines autonomen Antriebssystems.

Acknowledgements

First of all, I would like to express my sincere gratitude to Prof. Dr. Alois Knoll for offering me the opportunity of a PhD study. Without his support, this thesis would have not been possible. I would like to thank Prof. Dr. Lothar Thiele for his guidance in his group and being the second supervisor of this thesis.

I would also like to thank: Prof. Dr. Kai Huang for his guidance and inspirations on the research direction; Dr. Gang Chen for giving me the chance to participant his research; my colleague Mr. Long Cheng for the research cooperation; Mr. Pengcheng Huang for the research cooperation and future advice; Dr. Dongkun Han for the help on writing and research encouragement. Furthermore, I would like to thank all my former and current colleagues of the whole Robotics and Embedded System group for their company and support, especially Dr. Hardik Shah and Dr. Martin Eder for their nice experience sharing. I also want to thank all the students whom I ever worked with.

Finally, my dearest thanks go to my wife Wenhe Wang for her continuous support, trust, passion, and encouragement. During the past four years, her love helped me overcome many difficulties. I am thankful to my parents and brother for their trust and encouragement on my study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Timing guarantee is important in many embedded systems, especially in the safety-critical system like vehicle driving system and airplane flight-control system. Real-time scheduling is responsible to schedule task executions so that their timing requirements can be met. In the last few decades, many thousands of research papers have been published on how to perform the optimal schedule so that the system resource can be ingeniously utilized to meet task timing requirements. However, with the increasing complexity of an embedded system architecture, previous scheduling approaches now face more challenges.

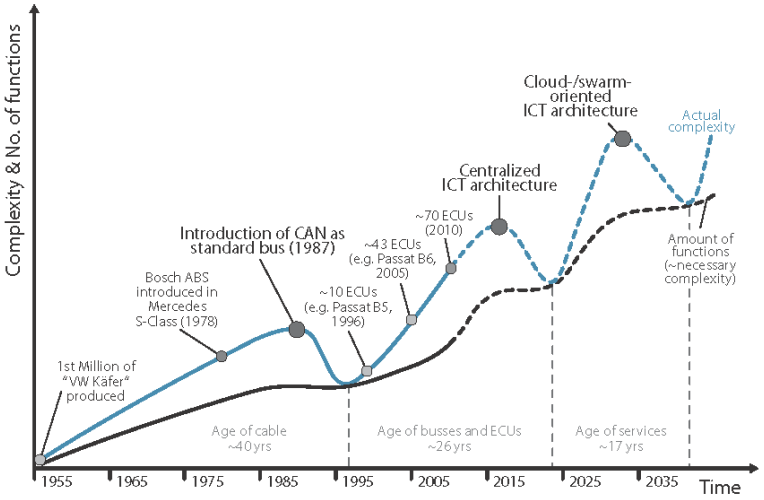


Figure 1.1: ECUs evolution in modern cars (figure from [1])

Typically, in the automotive system, it was reported that German premium vehicles have 70 to 100 electrical control units (ECUs) [3] and it is predicted that more ECUs will be mounted

Level	Failure Condition	Interpretation
A	Catastrophic	Failure may cause a crash.
B	Hazardous	Failure has a large negative impact on safety or performance, or reduces the ability of the crew to operate the plane due to physical distress or a higher workload, or causes serious or fatal injuries among the passengers.
C	Major	Failure is significant, but has a lesser impact than Hazardous failure (for example, leads to passenger discomfort rather than injuries).
D	Minor	Failure is noticeable, but has a lesser impact than a Major failure (for example, causing passenger inconvenience a routine flight plan change).
E	No effect	Failure has no impact on safety, aircraft operation, or crew workload.

Table 1.1: DO-178B software certification standard

on the modern car in order to perform more advanced functionalities [1], as shown in Figure 1.1. The increasing ECUs will impose more weight and costs on the car, and the complex networking among those ECUs makes it difficult to schedule their uses. To reduce the complexity and the scheduling difficulty along with increasing ECUs, a powerful centralized ECU is suggested to replace all ECUs. As a result, different components or tasks of different criticalities are integrated together to share a same processing unit [1]. A new arising research question under this integration is how to reconcile the conflicting requirements of partition for safety assurance and sharing for efficient resource usage in a disciplined way [4]. This type of system that consists of different criticality tasks is called *mixed-criticality systems* (MCSs).

1.1 Mixed-Criticality Systems

Many embedded systems, especially safety-critical systems, are subject to *certification* requirements: their functionalities must be guaranteed to a certain level based on some industry standards. For example, for avionics vehicles, the RTCA DO-178B avionics software standard divides task criticalities into five assurance levels ranging from level A to level E, as listed in Table 1.1. In detail, the failure of A-criticality tasks is catastrophic, whereas the failure of E-criticality tasks has no effect on the airplane safety. Thus, A-criticality tasks need to be given more stringent certification guarantee at runtime.

From the real-time perspective, the timing guarantee level of more critical tasks should be higher than that of less critical tasks. To address this concern, Vestal [5] proposed a multi-criticality task model that models *worst-case execution times* (WCETs) on different criticality levels, with the one on a higher criticality level being more pessimistic; i.e., for the same piece of code, it will have a higher WCET if it is safety-critical than it would be if it is non-critical. By allowing multiple level WCETs of one task, the task timeliness guarantee level at runtime then depends on the level that task executions run into. In particular, the system will guarantee all tasks meet their deadlines if all tasks run within their lowest level WCETs, and the system only guarantees the highest-critical tasks if any single task runs over a certain level WCET.

This distinguishing feature that tasks' WCETs are dependent on the criticality level of the task makes the conventional optimal scheduling algorithms not optimal in mixed-criticality systems. It was presented in [5] that the deadline monotonic priority assignment, which is known to be optimal for implicit tasks (relative deadlines are the same as their periods), is not optimal for the multi-criticality scheduling problem. Besides, the earliest deadline first (EDF) schedule, known as the optimal algorithm in dynamic priority assignment system, is also not optimal for scheduling multi-criticality tasks.

Since Vestal proposed this type of multi-criticality task model [5], there already exists some good results for scheduling multi-criticality tasks. It was proved in [6] that Audsley algorithm [7] is optimal to assign task priorities under the multi-criticality scheduling model. Audsley algorithm starts with no task being assigned a priority, then priorities are assigned from the lowest to the highest in a way that at each step a task with the lowest priority is picked out. This procedure continues to pick the left tasks until no task is left. Vestal's model was also improved to enhance the system schedulability by dropping tasks whose criticalities are lower than the system current guarantee level [8]. The EDF schedule is also successfully applied to schedule multi-criticality tasks by applying a technique called earliest deadline first with virtual deadlines (EDF-VD) [9]. In addition to the uniprocessor system, Vestal's model was extended to multi-core systems. Most scheduling approaches in multi-core systems are based on the global scheduling model or partition scheduling model by allowing tasks to share the resources simultaneously. Shared resources, however, undermine the system predictability because jobs that run concurrently pay unpredictable performance penalties due to contention in accessing shared resources. To overcome this problem, a novel scheduling model called isolation scheduling was proposed that only allows tasks with the same criticality to share the system resource at a time slot [10].

The mixed-criticality scheduling approaches were also evaluated by implementing them in real platforms. Towards the five levels of tasks defined in RTCA DO-178B, in a multi-core system, a framework called MC^2 used a cyclic executive (static schedule) for level-A, partitioned preemptive EDF for level-B, global preemptive EDF for levels C and D and finally global best-effort for level-E [11–14]. Besides MC^2, other research evaluates the system performance by implementing some state-of-the-art scheduling approaches in real platforms [15,16]. In addition to the above listed results, there are other research results in mixed-criticality systems. We recommend the interested readers to read a comprehensive review report [4] for more details.

1.2 Motivations

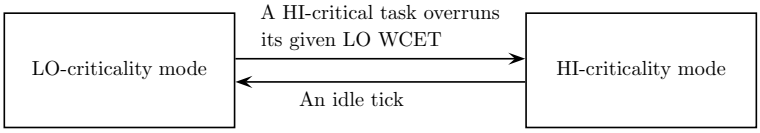


Figure 1.2: Criticality switch flow

The concept of scheduling the multi-criticality task in a centralized platform provides enormous potential to reduce the system "SWaP" (Size, Weight, and Power) related costs. Although a vast of papers have been published focusing on this goal, there remains a lot of challenges on how, in a disciplined way, to reconcile the conflicting requirements of partitioning for safety assurance and sharing for efficient resource usage. One challenge is how to verify the timing guarantee for multi-criticality tasks. In contrast to the conventional hard real-time system that needs to guarantee all tasks timing requirements in any case, the timing of multi-criticality tasks should be guaranteed to their own levels. The definition of timing guarantee to task's criticality level is not clear.

The dual-criticality system is a classic mixed-criticality system model that was often adopted in many research paper (e.g. [17–21]), because an approach that is tested being effective in dual-criticality system can be easily extended to systems of two more criticality levels. We suppose a dual-criticality task set $\tau = \{\tau_1, ..., \tau_n\}$ is given to be scheduled on a uniprocessor. For this task set, the standard model follows the below scheduling principles:

1. Each task, τ_i, is supposed to a sporadic task characterized by a minimal inter-arrival time T_i, relative deadline D_i ($D_i \leq T_i$), WCET $\mathbf{C_i}$ and criticality $\mathcal{X}_i$, where $\mathbf{C_i} = (C_i^L, C_i^H)$

and $\mathcal{X}_i = \{LO, HI\}$. Each LO-critical task only has a LO WCET C_i^L, and each HI-critical task has a LO WCET C_i^L and a HI WCET C_i^H. For HI-critical tasks, their HI WCETs are not smaller compared to their LO WCETs, i.e., $C_i^L \leq C_i^H \leq D_i$.

2. As shown in Fig. 1.2, the system executes in one of two modes. It starts in LO-criticality mode, where all tasks are assumed to not exceed/overrun their LO WCETs and are guaranteed to meet their deadlines.

3. If any HI-critical job exceeds its LO WCET, then the system transits immediately to the HI-criticality mode, where all LO-critical jobs are abandoned or degraded and HI-critical tasks are guaranteed to meet their deadlines if they do not exceed their HI WCETs.

4. If any LO-critical job executes for its LO WCET without completion, it is immediately aborted.

5. When the system is in HI-criticality mode, an idle tick will trigger the system to switch back to the LO-criticality mode.

This standard system model is called *mixed-criticality schedulable* (i.e. tasks are guaranteed to their own criticality level) if and only if the following two properties are guaranteed:

- Property 1: All jobs that are released and completed in LO-criticality mode, are guaranteed to meet their deadlines.

- Property 2: HI-critical jobs released at any time are guaranteed to meet their deadlines.

Although this standard model provides us a clear verification on how to provide a task the timing guarantee to its criticality level, it has a lot of problems.

1.2.1 Standard Mixed-Criticality Mode-Switch

In the standard model, whenever a HI-critical task overruns its LO WCET, the system will immediately switch to HI-criticality mode, in which LO-critical tasks will be dropped or their services are degraded; and whenever a LO-critical task overruns its LO WCET, it is immediately aborted. Although such a mode-switch scheme is effective in guaranteeing timeliness of HI-critical tasks, it is abrupt and pessimistic - abrupt in the sense that LO-critical tasks are suddenly dropped/degraded after a single HI-critical task overruns, and pessimistic in the sense that the system may naturally have an overrun budget due to free slacks to allow some tasks to overrun. Such slacks exist either because the system is underloaded (static slacks), or

because tasks will most likely finish before their WCETs at runtime, giving space for other tasks to execute (dynamic slacks). This, however, is not fully exploited in existing mode-switch schemes, to keep the system "away" from the HI-criticality mode where LO-critical tasks are abandoned or degraded. The question here is:

"How to efficiently utilize the system static and dynamic slack to postpone the mode-switch, thus improving the quality of service to LO-critical tasks."

1.2.2 Standard Task Model

To simplify the scheduling problem, tasks in mixed-criticality systems are often modeled as the sporadic tasks that only define a minimum inter-activation interval (also called period, see the aforementioned standard model). This sporadic task model can represent many *nondeterministic* activation patterns by assuming the task be activated in every period and thus helps us to get some insights on the scheduling property. However, with a growing variety of activation patterns in today's embedded systems, some of them capture arbitrary activation patterns, and the assumed sporadic activation patterns in most previous scheduling approaches have sometimes become inapplicable or ineffective. For instance, a simple approach to deal a periodic task with a jitter release pattern is to transform it into a new sporadic task with a shorter period [22]. While this approach is safe, the transformation can lead to overly schedulability loss. If this shorter period is smaller than the task WCET, it is impossible to schedule this task by modeling it as a sporadic task, because the sporadic model assumes that the task will be activated in every shorter period. The real situation is that the task cannot be activated in every shorter period. This task may be mixed-criticality schedulable, which however will be tested unschedulable after modelling it as a sporadic task.

The arbitrary activation patterns are not uncommon in real-time systems. Periodic tasks with jittery releases or tasks with burst activations, often exist in many reactive embedded systems. The jitter may come from release-delay overheads induced by tick-driven scheduling [23], execution of interrupt service routines [22], or I/O overheads. The delays by scheduling and data dependencies may also cause the jitter. In ARINC avionics systems, different tasks scheduling partitions are connected over a switched Ethernet. Due to the network delay, tasks in a partition are not always released strictly periodically, but with a certain jitter [24]. In the automotive systems [25], a lot of event streams that are used to activate tasks suggest the use of more general event stream models than the classical sporadic event model. Hence, this gives rise to a question:

"How to model an arbitrarily activated task. Regarding to this model, how to test its schedulability under the context of mixed-criticality scheduling."

1.2.3 Run-Time Adaptability

The standard model advocates the conditional guarantee for LO-critical tasks. The condition is that the executions of LO-critical tasks should never endanger the timing guarantee for HI-critical tasks. To achieve this goal, based on the offline schedulability test, the workload of LO-critical tasks are constrained to a certain bound. This offline workload bound (i.e. task period and WCET), however, is often too pessimistic because it makes a worst-case assumption on the demand of HI-critical tasks, where their actual demand is often lower than the worst-case assumption. To improve the system utilization and provide better service to LO-critical tasks, the actual demand of HI-critical tasks at runtime needs to be considered. This gives rise to another question:

"How to make use of the run-time demand of HI-critical tasks to adaptively manage the executions of LO-critical tasks at runtime, meanwhile the timeliness of HI-critical tasks is still guaranteed."

1.2.4 Practical Evaluations on Real-Life Systems

Mixed-criticality scheduling makes a lot of assumptions on the task and the system, based on which many approaches have been proposed [4]. Most of those approaches are only evaluated from the perspectives of theoretic analysis or simulations. Although there are some implementation evaluations [11–16], those evaluations are not based on a real-life system. Tasks and system settings in those implementations are artificially generated to meet the mixed-criticality scheduling assumptions, in which way scheduling approaches can be successfully deployed. However, real-life tasks and systems may not meet those assumptions and it is unknown to which extent that mixed-criticality scheduling can be performed on real-life systems. Besides, the doubt on the practical value of the mixed-criticality scheduling is rising recently [26, 27], making it urgent to evaluate the effectiveness by applying mixed-criticality scheduling to real-life systems. Then, the question is:

"What kind of real-life system can be used to evaluate the mixed-criticality system. How to apply the mixed-criticality scheduling concept to this system and how to evaluate its performance. What problems or bottlenecks will the current mixed-criticality scheduling have in real-life implementations."

1.3 Thesis Outline and Contributions

This thesis provides partial answers to the above listed questions. The contributions of this thesis are in extending the current mixed-criticality task model to a more general task model and relaxing the scheduling at runtime to be flexible and adaptable, so that the schedulability test becomes more effective and the system performance is improved. This thesis is subdivided into 6 chapters to present those contributions.

1. In chapter 2, we propose an *on-the-fly fast overrun budgeting* (FFOB) scheme for the mixed-criticality system to online postpone the mode-switch. A feature called *automatic schedulability guarantee* is explored that greatly reduces the computation overhead of FFOB scheme. We evaluate the FFOB scheme by simulations in MATLAB and implementations in a framework called SF3P [28].

2. In chapter 3, we analyze the schedulability of dual-criticality system with arbitrarily activated tasks. By using the *arrival curve* to represent the upper bound of task activations, we integrate the well-established results from Real-Time Calculus to the mixed-criticality schedulability analysis. Compared to previous schedulability analysis, our proposed approaches can handle more general tasks with blocking, jitter, and arbitrary deadlines.

3. In chapter 4, we present an adaptive scheme for managing the workload of low criticality tasks online. Two online workload management policies, namely priority-adjustment policy and workload-shaping policy, are investigated. We also propose a lightweight approach with the complexity of $O(n \cdot log(n))$ to reduce the online workload management overhead.

4. In chapter 5, we present a case study by applying some basic mixed-criticality scheduling concepts to an autonomous driving system. All running tasks in this system have specific functionalities related to autonomously driving. We develop and evaluate the time-triggered and the event-triggered scheduling approaches, both applying mixed-criticality scheduling concepts.

5. In chapter 6, we summarize this thesis and discuss about future research directions based on this thesis.

Chapter 2

On-the-Fly Fast Overrun Budgeting Mechanism

State-of-the-art mixed-criticality scheduling techniques commonly assume to switch system mode and drop all less critical tasks whenever any single critical task overruns. Despite many efforts in reducing the pessimism of this approach, postponing the mode-switch to improve system guarantees by exploring system slacks dynamically online, remains an unsolved problem for mixed-criticality systems. Such a problem is important as mode-switch procrastination naturally helps to improve the system performance.

We propose an online mode-switch procrastination technique called on-the-fly fast overrun budgeting in this chapter for both FP- and EDF-scheduled MCSs. The proposed approach has a feature of automatic schedulability guarantee that transfers the problem of mixed-criticality schedulability guarantee online to the counterpart of conventional real-time systems. With a routinely updated shared resource pool of overrun budgets, the system allows tasks to overrun, which thus postpones the mode-switch as long as possible. Extensive simulations and real platform implementations confirm that our proposed technique significantly improves the system QoS over the state-of-the-art, while at the same time permitting light-weight online deployments.

2.1 Overview

In mixed-criticality systems (MCSs), it is of vital importance for tasks to meet the requirements corresponding to their own criticality levels. For example, from the real-time perspective, the timing of high-criticality tasks should be more rigorously guaranteed than that of less

critical tasks. To address this concern, a common assumption of mixed-criticality scheduling is to guarantee the schedulability of all tasks when no tasks overrun (normal mode) and the schedulability of only high-criticality tasks when they overrun (critical mode).

Following this assumption, a plethora of scheduling techniques have been proposed in the literature, see [4] for an excellent survey. Those techniques, e.g. [8, 18, 19, 29–31], all advocate a mode switched scheduling, where less or no resources are provided to low-criticality tasks whenever a *single* high-criticality task overruns its given execution time threshold (i.e. system enters critical mode); as a result, high-criticality tasks can still be guaranteed even if they overrun. Such a scheme, although being effective in guaranteeing most critical tasks, can be abrupt and pessimistic in practice - abrupt in the sense that low-criticality tasks are suddenly dropped/degraded after a *single* high-criticality task overruns, and pessimistic in the sense that the system may naturally have some slack time to accommodate overrunning tasks. Such slack time arises either because the system is underloaded (static slacks), or because tasks finish earlier before their worst-case execution times (WCETs) at runtime, giving space for other tasks to execute (dynamic slacks). Furthermore, due to this pessimism, less critical tasks can receive no/little resources (i.e. experience degradation) very often, impairing system functionality or even system safety [32, 33]. Consequently, to reduce pessimism and to improve the *Quality-of-Service* (QoS) for low-criticality tasks, the mode-switch should be postponed as long as possible.

Indeed, several scheduling techniques have already been proposed to keep the system "away" from the critical mode. However, they can only explore static slacks available offline, while run-time slacks due to the less loading and task finishing before their WCETs are neglected. In [31], instead of procrastinating the transition to the critical mode, the authors proposed a bailout protocol to timely switch the system back to the normal mode. In [34], Santy et al. presented a method to compute offline the static margins/allowances of high-criticality tasks, using which they can overrun without triggering the mode-switch. By further adopting sensitivity analysis [35], this method was proposed in [36] to more efficiently explore the statically available system slacks.

In this chapter, for both earliest-deadline-first (EDF) scheduled and fixed-priority (FP) scheduled systems, we propose an *on-the-fly fast overrun budgeting* (FFOB) mode-switch scheme. FFOB relies on the offline analysis and run-time information of tasks to reclaim available slacks (both static and dynamic, denoted as the overrun budget), which all tasks can spend on overrunning without triggering the transition to the critical mode. The design and analysis of

FFOB mode-switch scheme, however, is nontrivial. The reason is multi-fold. First, the mode-switch scheme should exploit free run-time slacks as much as possible in order to increase its efficiency. Second, the procrastination of mode-switch should not hamper *dynamic guarantees* in MCSs, i.e. all tasks must be schedulable in normal mode and high criticality tasks must be schedulable in both modes. Testing the system schedulability with multi-criticality tasks is already a NP-hard problem [37], and finding the maximal overrun allowance and computing when to conduct the mode-switch with runtime information makes this problem more complicated. Last, the timing overhead to make the mode-switch decision should be kept a minimum. Any mode-switch would be useless if its timing overhead is more than the allowance of task overrun.

To this end, we propose the FFOB mode-switch scheme to address these concerns. The proposed scheme is inspired by the task procrastination techniques in dynamic power management [38–40], where the processing of incoming tasks are deliberately postponed such that the processor can reside in a sleep status to reduce energy consumption. Analogously, task overrun in this chapter is considered as a procrastination on the system, in the sense that it delays resources available to tasks that do not overrun. While a lot of effective procrastination techniques are proposed for conventional real-time systems, none of them can provide dynamic timing guarantees for MCSs. To solve this problem, a distinguishing feature that makes the existing procrastination techniques applicable in MCSs is explored. This feature, called *automatic schedulability guarantee*, can guarantee that if the system is schedulable in both modes by offline analysis, then the schedulability of normal mode at runtime automatically guarantees the schedulability of critical mode. This way, the schedulability guarantee of dual-criticality systems is transformed to the schedulability guarantee of conventional real-time systems. Besides, FFOB only needs to use a timer to manage the overrun budget, which can be efficiently implemented in many embedded systems. This timer can be renewed once it is depleted, which can further explore the existing slack in the system to schedule overrun tasks. In detail, the contributions of this chapter are summarized as follows:

1. We propose an on-the-fly mode-switch scheme for both EDF-scheduled and FP-scheduled MCSs that can effectively keep the system in normal mode by allowing tasks to overrun. This scheme is able to make use of the static slack and adaptively reclaim the dynamic slack at runtime to postpone the mode-switch as long as possible.

2. We explore the automatic schedulability guarantee feature, reducing the dual-criticality schedulability guarantee to the schedulability guarantee of conventional systems. The automatic schedulability guarantee feature enables us to apply existing task procrastination techniques of conventional real-time systems to the MCS.

3. We develop several options in implementing the FFOB scheme. Especially for FP-scheduled MCS, applying FFOB scheme is intrinsically unsuitable by the state-of-the-art approaches. To overcome this problem, we propose a RTI-FP method that combines the concept of virtual deadlines and real-time interface analysis [41, 42] to enable the automatic schedulability guarantee feature, thus making FFOB applicable in FP-scheduled MCS.

4. We present concrete proof showing the correctness of the FFOB scheme. Extensive simulations and embedded-platform implementation demonstrate the FFOB scheme outperforms the state-of-the-art approaches in improving the system performance to a large extent. The implementation also demonstrates that the FFOB scheme is lightweight in the aspect of computation overhead.

Organizations. The remainder of this chapter is structured as follows. Section 2.2 briefly reviews relevant work. Section 2.3 presents our system settings. Section 2.4 presents the working flow of FFOB scheme. Section 2.5 and Section 2.6 provide the use of FFOB in EDF-scheduled and FP-scheduled system, respectively. Section 2.7 proves the correctness of FFOB scheme. The evaluation results are presented and discussed in Section 2.8 and Section 2.9 concludes this chapter.

2.2 Related Work

Mixed-criticality scheduling advocates using limited resources to provide enough guarantee for a task set of multiple criticality levels. It stems from the seminal paper [5] and is drawing increasing interest from both the research community [4] and the industry [43]. To date, the real-time community has mostly focused on providing different timing guarantees on different criticality levels. We survey the relevant results in single processor systems as this chapter only studies single processor system; discussions of other results can be found in a comprehensive review by Burns *et al.*, see [4].

Under EDF schedule, an algorithm named EDF-VD (virtual deadline) was first proposed to meet the dynamic guarantees on mixed-criticality task sets by fairly shortening the deadline of high-criticality tasks in normal mode and resuming their deadlines in critical mode [17]. Later work [9] provided a more strict bound on the schedulability test of EDF-VD. The schedulability of EDF-VD can be significantly improved if the deadlines can be shortened individually by taking the task demand into the schedulability test [18, 30].

Under FP schedule, Baruah *et al.* [8] proposed the adaptive mixed-criticality scheme to provide heterogeneous timing guarantees for tasks of two different criticality levels. They presented two response-time analyzing approaches called AMCrtb and AMCmax, to analyze the system schedulability. Experimental results demonstrated that AMCmax is slightly tighter than AMCrtb in testing the system schedulability. Based on AMCrtb analysis, the strictness for scheduling a mixed-criticality task set was relaxed by increasing task execution time thresholds [34], exceeding which the mode-switch will be triggered. As a result, the critical system mode will be more unlikely due to the relaxation. Further improvements on relaxing the mixed-criticality scheduling can be achieved by integrating the Audsley's priority assignment scheme [7, 36]. On top of AMCrtb and the relaxing approach, a bailout protocol [31] was proposed, which can further utilize the offline slack to timely switch the system back to the normal mode. Once again, all aforementioned works are based on the offline analysis, which is different from this chapter that utilizes the runtime slack to improve system performances.

2.3 Models

In this section, we formally introduce the mixed-criticality system and task models.

System Model. We adopt the classic dual-criticality system model [4, 8, 18, 19, 29–31]. A dual-criticality task set $\tau = \{\tau_1, ..., \tau_n\}$ is to be deployed on a uniprocessor under fixed-priority scheduling. All tasks are independent. Each task, τ_i, is characterized by a minimal inter-arrival time T_i, a relative deadline D_i, a WCET $\mathbf{C_i}$ and a criticality $\mathcal{X}_i$, where $\mathbf{C_i} = (C_i^L, C_i^H)$ and $D_i = T_i$. A task can either have high (HI) or low (LO) criticality. Each LO-critical task only has a LO level WCET C_i^L, and each HI-critical task has a LO level WCET C_i^L and a HI level WCET C_i^H. For HI-critical tasks, their HI level WCETs are not smaller than their LO level WCETs, i.e., $C_i^L \leq C_i^H \leq D_i$. The rationale behind is that the execution time estimation on a higher criticality level is more conservative. At runtime, since C_i^L is less conservative, some

tasks (including LO-critical tasks) may overrun their LO level WCETs. However, we assume no HI-critical tasks can overrun their given HI level WCETs.

Based on the above assumptions, a standard mode-switch scheme [4,8,18,19,29,30] exists to schedule the system, as presented in Section 1.2. Besides, a system is defined as *mixed-criticality schedulable* if the two properties presented in Section 1.2 hold.

Task Workload Model. The task release pattern can be modeled as an *arrival function* [44] that specifies the maximum number of released jobs within any time interval of a length Δ. For a sporadic task τ_i with a minimum release distance T_i, the arrival function is

$$\alpha(\tau_i, \Delta) = \left\lfloor \frac{\Delta + T_i}{T_i} \right\rfloor, \ \forall\ \Delta > 0, \tag{2.1}$$

with $\alpha(\tau_i, 0) = 0$.

The *workload bound function* (WBF) models the number of execution time units requested by a task over any interval of length Δ. Suppose the WCET of τ_i is C_i, then the WBF of τ_i is given by:

$$\mathrm{wbf}(\tau_i, \Delta) = \left\lfloor \frac{\Delta + T_i}{T_i} \right\rfloor \cdot C_i, \ \forall\ \Delta > 0, \tag{2.2}$$

with $\mathrm{wbf}(\tau_i, 0) = 0$.

Task Demand Model. Analogous to WBF, the minimum execution time units over any interval of a length Δ that must be provided to a set of tasks to ensure their schedulability, is modeled by the *demand bound function* (DBF). For a single task τ_i with relative deadline D_i, the demand bound function is [45]

$$\mathrm{dbf}(\tau_i, \Delta) = \left\lfloor \frac{\Delta + T_i - D_i}{T_i} \right\rfloor \cdot C_i, \ \forall\ \Delta \geq 0. \tag{2.3}$$

Task Resource Model. The resource that the system provides is modeled by the *supply bound function* (SBF), denoting the minimum number of execution time units available over any time interval of fixed length. In this thesis, the total system resource is simply a dedicated uniprocessor with a unit-speed. We can further model the resource provided to a task or a task set over any interval of fixed length. For example, for a task set τ, the SBF provided to τ in any interval of length Δ is represented by the following function:

$$\mathrm{sbf}(\tau, \Delta) = \Delta, \ \forall\ \Delta \geq 0. \tag{2.4}$$

Short Notations. For ease of expression in the sequel, we adopt some short notations. We denote the subset of all LO-critical tasks and all HI-critical tasks in τ as $\tau^L = \{\tau_i \in \tau | \mathcal{X}_i = \mathrm{LO}\}$ and $\tau^H = \{\tau_i \in \tau | \mathcal{X}_i = \mathrm{HI}\}$, respectively. Furthermore, we use $[\![a]\!]_b$ to represent $\max(a, b)$ and $[\![a]\!]^c$ to denote $\min(a, c)$.

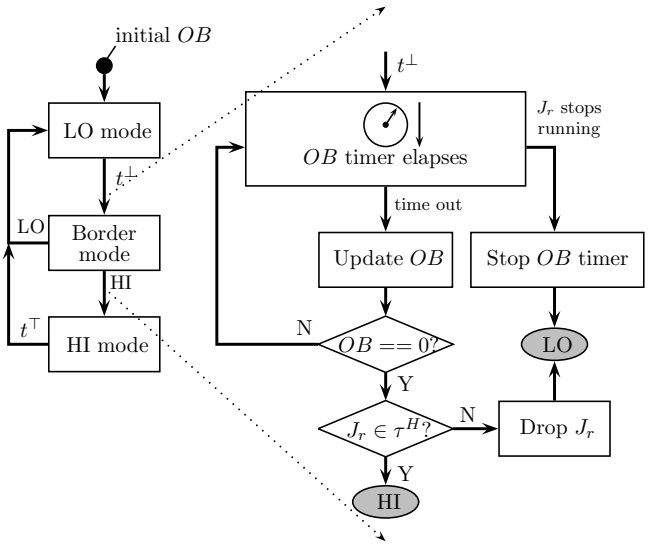


Figure 2.1: Overview of FFOB in the MCS

2.4 FFOB Mode-Switch Scheme

In this section, we present the FFOB mode-switch scheme and two relevant key problems. We then provide an example to further explain this scheme.

2.4.1 The Working Flow of FFOB Scheme

In the standard model, LO-critical jobs are not allowed to exceed their LO WCET and a HI-critical job's overrun of its LO WCET immediately triggers the mode-switch. However, in FFOB, by relying on an overrun budget, all jobs are allowed to run over their LO WCETs, without being dropped or triggering the mode-switch. To denote the system state that a job overruns while not being in HI mode, the Border mode is introduced. In detail, the FFOB scheme maintains a common resource pool at runtime (the overrun budget), with which the system allows tasks to overrun by residing in Border mode.

An overview of FFOB is presented in Fig. 2.1. With an initial assignment of the overrun budget (represented as the OB timer), online updating of OB goes through different system modes as follows.

LO mode: The system starts with an initial overrun budget, which intuitively represents the static slack in the system and always exists as long as the system is underloaded. In particular, our online scheme performs the following.

(i) The online scheduler holds the initial overrun budget OB at the beginning.

(ii) While all jobs do not execute over their LO level WCETs, the system remains in LO mode.

(iii) If any job executes beyond its LO WCET (at time instant $t^{\perp}$), the system enters the Border mode.

Border mode: In this mode, we allow all tasks to overrun and delay entering to HI mode by using the overrun budget; we replenish online OB by exploring dynamic slacks, which arise as jobs take less than their WCETs to finish. Detailed designs of our Border mode scheme are as follows.

(iv) Whenever any job (denoted as J_r) overruns, the OB timer is decremented by the same amount as its overrun time.

(v) If J_r finishes before OB is depleted, the OB timer stops decrementing and the system goes back to LO mode.

(vi) Once OB is depleted, the scheduler calls an update procedure, in which the overrun budget is replenished based on the current state of task executions. If the updated OB is nonzero, further job overrun is allowed and the OB timer will decrement accordingly. If the updated OB is zero, another decision procedure is called – when the overrun job is LO-critical, it is dropped and the system goes back to LO mode; otherwise, the system transits to HI mode.

HI mode: The FFOB scheme acts as follows.

(vii) Only the timing requirements of HI-critical tasks are guaranteed in the system.

(viii) An idle tick (at time instant $t^{\top}$) will reset the OB timer to its initial value and trigger the switching back to LO mode.

Note that, in all modes, an idle tick will reset the system mode (to LO mode) and the OB timer (to the initial value).

2.4.2 Two Relevant Problems

The system behavior in Border mode demonstrates that the Border mode can be considered as an "extended" LO mode as all pending tasks are kept and the system schedule and task parameters are the same as LO mode. Therefore, to prove a system with FFOB scheme schedulable, we need to guarantee the two properties presented in Section 2.3 and further guarantee that all jobs that are completed in Border mode will not miss their deadlines. If OB is always zero, the Border mode then does not exist and the system is a standard model. The FFOB scheme will not have any advantage compared to previous approaches in this case. The

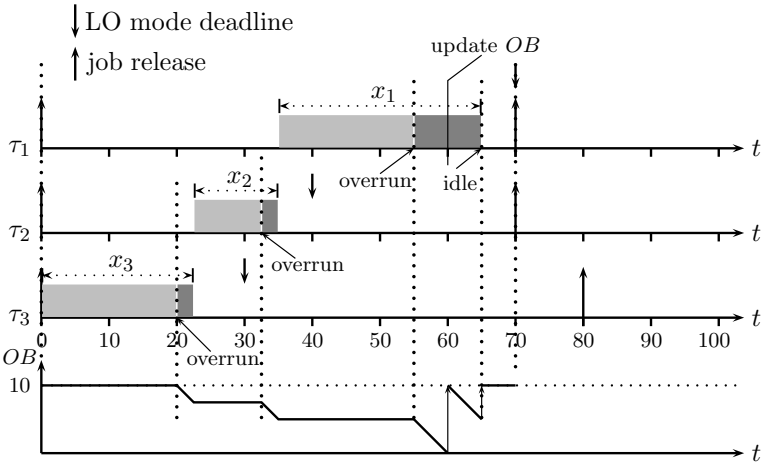


Figure 2.2: Illustration of task execution with FFOB

system will be different from the standard model only when OB is nonzero. Since OB can be changed only by the setting of initial OB and the updating procedure, to guarantee the system mixed-criticality schedulable, the two key problems are how to find an feasible initial OB and how to update the OB at runtime. In the following, we will present the solutions for these two problems in both EDF-scheduled and FP-scheduled systems.

2.4.3 An Example

The following example illustrates how the FFOB mode-switch scheme works in EDF-scheduled system. Note that, in order to apply EDF algorithm to schedule mixed-criticality tasks, a technique called EDF-VD is used that artificially shortens the HI-critical task deadlines in LO and Border modes and resume their original deadlines in HI mode. More details about it are presented in Section 2.5.

Example 1. *In a uniprocessor system, three tasks are scheduled by the EDF-VD algorithm. Task properties are shown as follows.*

τ_i	χ_i	C_i^L	C_i^H	D_i^L	D_i	T_i
τ_1	LO	20	-	70	70	70
τ_2	HI	10	20	40	70	70
τ_3	HI	20	40	30	80	80

Fig. 2.5 illustrates the system runtime behavior under FFOB. Before the system runs, the OB timer is initialized to 10. Suppose that the first jobs of all tasks are released at time zero. Task τ_3 will run first. Once τ_3 overruns at $t = 20$, the OB timer starts to elapse. When τ_3

finishes, the OB timer will hold its current value and stop elapsing. τ_2 starts to execute. After τ_2 executes over 10, it overruns and triggers the OB timer to elapse. Similarly, the OB timer will stop when τ_2 finishes. Then τ_1 runs and further overruns to the extent that OB timer elapses to 0. The OB updating procedure is called and OB is updated to 10 by the approach in Section 2.5.2.3. After that, τ_1 runs until it finishes. After τ_1 finishes, the system returns to LO mode as there is an idle tick; in the meantime OB is reset.

The FFOB mode-switch scheme outperforms the other known methods that fairly increase the allowance of task overrun offline in two aspects. First, OB can be flexibly used by all tasks. In the static method, the overrun allowance is assigned to each individual task, and the mode-switch is triggered once a task exceeds its own overrun allowance. Such a scheme is not flexible as it cannot use the remaining allowances of other tasks. Second, OB is updated at runtime in FFOB, which can often replenish the overrun allowances and postpone the mode-switch by exploring dynamic slacks. Such slacks naturally exist as tasks will most likely take less than their WCETs to finish. The updated OB is able to collect those slacks to postpone the mode-switch. Furthermore, since the remaining OB is still valid with the time going on and OB will be automatically replenished to the initial value whenever an idle tick emerges, FFOB does not need to use a complex way to update OB every time a task overruns.

2.5 FFOB under EDF schedule

In this section, we present how to integrate the FFOB scheme into EDF-scheduled system. We first introduce the EDF-VD technique, on top of which we present how to initialize and update the overrun budget.

2.5.1 EDF-VD Technique

Earliest-Deadline-First Virtual-Deadlines (EDF-VD) [9,18] is a scheduling technique that makes the conventional EDF applicable in the MCS. In this chapter, the proposed FFOB mode-switch scheme relies on the EDF-VD algorithm.

A key feature of EDF-VD is to artificially shorten the deadlines of HI-critical tasks when the system is in LO mode. In this way, HI-critical tasks will finish earlier so that there is enough time slack for them to catch their actual deadlines after switching to the HI mode.

2.5.1.1 DBF in LO and HI modes

In order to schedule HI-critical tasks in MCSs, the relative deadline D_i of a HI-critical task is artificially shortened in LO mode and returns to D_i after the system switches to the HI mode. We name the deadline in LO mode as the LO mode deadline, and denote it as D_i^L. Note that, for LO-critical tasks, their deadlines do not need to be shortened, thus $D_i^L = D_i,\ \forall \tau_i \in \tau^L$.

When the system is in LO mode, each task τ_i behaves as a normal sporadic task with parameters C_i^L, D_i^L and T_i. A DBF of such a task is known [45]:

$$\mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta) = \left\lfloor \frac{\Delta + T_i - D_i^L}{T_i} \right\rfloor C_i^L. \tag{2.5}$$

When the system is in HI mode, LO-critical tasks are abandoned, thus only the demands of HI-critical tasks need to be considered. The DBF of a HI-critical task τ_i in HI mode is that [18]:

$$\begin{aligned} \mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta) &= \left\lfloor \frac{\Delta + T_i - (D_i - D_i^L)}{T_i} \right\rfloor C_i^H - \mathrm{done}(\tau_i, \Delta), \\ \mathrm{done}(\tau_i, \Delta) &= \begin{cases} [\![C_i^L - l + D_i - D_i^L]\!]_0, & if D_i > l \geq D_i - D_i^L \\ 0, \text{otherwise}, \end{cases} \end{aligned} \tag{2.6}$$

where $l = \Delta \bmod T_i$.

In EDF-scheduled MCS, the DBF of a system is the sum of DBFs of all tasks in this system [18]. That is,

$$\begin{aligned} \mathrm{dbf}_{\mathrm{LO}}(\tau, \Delta) &= \sum_{\forall \tau_i \in \tau} \mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta), \\ \mathrm{dbf}_{\mathrm{HI}}(\tau^{\mathrm{H}}, \Delta) &= \sum_{\forall \tau_i \in \tau^H} \mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta). \end{aligned} \tag{2.7}$$

2.5.1.2 Schedulability analysis

The following proposition presents the sufficient conditions that can guarantee all tasks to meet their deadlines in LO mode and all HI-critical tasks to meet their deadlines in both LO and HI modes.

Proposition 1. *[From [18]]: In* MCS*s, the taskset is schedulable if the* DBF*s of LO and HI modes are not greater than the* SBF*s of this system, i.e.,* $\forall \Delta \geq 0$*,*

$$\text{EDF-LO}: \quad \mathrm{dbf}_{\mathrm{LO}}(\tau, \Delta) \leq \mathrm{sbf}(\tau, \Delta) = \Delta, \tag{2.8a}$$

$$\text{EDF-HI}: \quad \mathrm{dbf}_{\mathrm{HI}}(\tau^{\mathrm{H}}, \Delta) \leq \mathrm{sbf}(\tau^H, \Delta) = \Delta. \tag{2.8b}$$

2.5.2 Initialize and Update Overrun Budget

Based on the EDF-VD technique, we present how to initialize the overrun budget and update it at runtime. We first present the schedulability conditions with task procrastination online, based on which the initialization and updating of an overrun budget are presented.

2.5.2.1 Schedulability Analysis at Runtime

At runtime, the task information may be different with the offline worst-case assumption. To denote the task information at runtime, we extend our notations and say that, from any time t online and onward, the actual demand and supply bound functions are $\mathrm{dbf}(\tau_i, \Delta, t)$ and $\mathrm{sbf}(\tau_i, \Delta, t)$, respectively. In fact, the offline analysis is equivalent to analyze the system schedulability at the beginning, i.e., we have $\mathrm{dbf}(\tau_i, \Delta) = \mathrm{dbf}(\tau_i, \Delta, 0)$ and $\mathrm{sbf}(\tau_i, \Delta) = \mathrm{sbf}(\tau_i, \Delta, 0)$. We use the subscript LO and HI in the following to represent the corresponding bound functions in LO and HI modes.

Task Procrastination. Suppose at a time t when the MCS is in LO mode, all tasks in τ are delayed for a time length $\rho(t)$ to be executed, then the SBF of τ after t is that [46]

$$\mathrm{sbf}_{\mathrm{LO}}(\tau, \Delta, t) = [\![\Delta - \rho(t)]\!]_0. \tag{2.9}$$

Denote t_{ms} as the time instant of a mode-switch and $\mathrm{sbf}_{\mathrm{HI}}(\tau^H, \Delta, t_{ms})$ as the SBF of τ^H in HI mode after t_{ms}. Straightforwardly extended from Proposition 1, we have the following schedulability conditions.

Proposition 2. *The schedulability conditions of LO and HI modes at runtime are that,* $\forall\, t,\, t_{ms},\, \Delta \geq 0$*,*

$$\text{EDF-LO-t} : \mathrm{dbf}_{\mathrm{LO}}(\tau, \Delta, t) \leq \mathrm{sbf}_{\mathrm{LO}}(\tau, \Delta, t), \tag{2.10a}$$

$$\text{EDF-HI-t} : \mathrm{dbf}_{\mathrm{HI}}(\tau^H, \Delta, t_{ms}) \leq \mathrm{sbf}_{\mathrm{HI}}(\tau^H, \Delta, t_{ms}). \tag{2.10b}$$

where $\mathrm{dbf}_{\mathrm{LO}}(\tau, \Delta, \mathrm{t})$ *gives the upper bound on the maximum possible execution demand of a task set* τ *over any time interval of length* Δ *from time* t *in LO mode. Similarly,* $\mathrm{dbf}_{\mathrm{HI}}(\tau^{\mathrm{H}}, \Delta, \mathrm{t_{ms}})$ *gives the upper bound on the maximum possible execution demand of tasks* τ^H *over any time interval of length* Δ *from time* t_{ms} *in HI mode.*

The two conditions correspond to the two properties of being mixed-criticality schedulable in Section 2.3, where the condition EDF-LO-t corresponds to Property 1, and the conditions EDF-LO-t and EDF-HI-t together correspond to Property 2. In addition, condition EDF-LO-t can further guarantee the property that completed tasks in Border mode can also meet their LO mode deadlines. This property will be proved in Section 2.7.

Intuition. In the conventional hard real-time system, task executions can be delayed for a certain time length without missing any deadlines. To get a feasible time length, we have the following lemma.

Lemma 1. *[From [47]] Suppose* $\mathrm{dbf}(\tau, \Delta, t)$ *denote the* DBF *of a task set* τ *from time* t*. If there is a* ρ *(*$\rho > 0$*) that satisfies*

$$\forall \Delta > 0 : \mathrm{dbf}(\tau, \Delta, t) \leq [\![\Delta - \rho]\!]_0, \tag{2.11}$$

then the executions of all tasks can be immediately delayed for ρ *and there will be no deadline misses after* t*.*

The FFOB scheme is inspired by Lemma 1 that all tasks are allowed to overrun within an overrun budget, by ensuring that this overrun budget is not greater than the feasible task procrastination time length (a time length that task executions are delayed). In the following, we introduce how to apply Lemma 1 to initialize OB and update OB at runtime.

2.5.2.2 Initialize OB

In order to get the initial OB, we compute the largest procrastination interval that the system at the beginning can accept. Suppose there is a initial procrastination interval ρ on the processor when the system starts. Then, the service bound function becomes $[\![\Delta - \rho]\!]_0$. The longest procrastination interval is defined as follows [39].

Definition 1 (Longest Procrastination Interval)**.** *The longest procrastination interval* ρ^* *with respect to a given* DBF $\mathrm{dbf}_{\mathrm{LO}}(\tau, \Delta)$ *is*

$$\rho^* = \max\Big\{\rho : [\![\Delta - \rho]\!]_0 \geq \mathrm{dbf}_{\mathrm{LO}}(\tau, \Delta),\ \forall \Delta \geq 0\Big\}. \tag{2.12}$$

Therefore, OB is initialized to the longest procrastination interval with respect to $\mathrm{dbf}_{\mathrm{LO}}(\tau, \Delta)$. This longest procrastination interval is denoted as $\rho^*(t_0)$. For the task set in the motivational example, the initial OB is set to 10 based on Eq. 2.12, as shown in Fig. 2.3.

2.5.2.3 Update OB at runtime

While tasks are overrunning, OB may elapse to 0. The system may still be able to postpone the mode-switch, because the actual overrun allowance at this moment may be greater than 0 based on the current tasks' execution state, i.e., dynamic slacks are available. We now derive the actual DBF of a task τ_i at any time t.

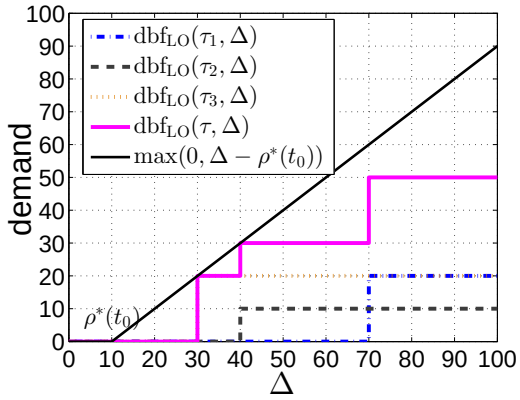


Figure 2.3: Initial *OB* in the motivational example

Lemma 2. *At any time t, for a task τ_i that has no backlogged job at t, its LO mode* DBF *is*

$$\mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta, t) = \mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta). \tag{2.13}$$

For a task τ_i that has one backlogged job at t, its LO mode DBF *is*

$$\mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta, t) = \max\left(\mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta),\ \mathrm{Dmd}^{\mathrm{bk}}(\tau_i, \Delta, t)\right), \tag{2.14}$$

where $\mathrm{Dmd}^{\mathrm{bk}}(\tau_i, \Delta, t)$ *is derived as follows*

$$\begin{aligned}\mathrm{Dmd}^{\mathrm{bk}}(\tau_i, \Delta, t) = &\left[\!\left[\left\lfloor \frac{\Delta}{r_i(t) + D_i^L - t} \right\rfloor\right]\!\right]^1 \cdot \left[\!\left[C_i^L - e_i(t)\right]\!\right]_0 \\ &+ \left[\!\left[\left\lfloor \frac{\Delta + \min\left(T_i, t - r_i(t)\right) - D_i^L}{T_i} \right\rfloor\right]\!\right]_0 C_i^L,\end{aligned} \tag{2.15}$$

and $r_i(t)$, $e_i(t)$ are the release time and the already execution time of the latest released job of τ_i at the moment t, respectively.

Proof. Since the DBF for a task τ_i must upper-bound the maximum execution demand of jobs from τ_i within any scheduling interval after time t, the DBF will include the demand from jobs that are backlogged and the future jobs. As shown in Fig. 2.4, the release time of the latest released job is $r_i(t)$, and at time t, the released job may have finished or may not. Therefore, a task may have backlogged job or may not. We consider the demand of the two cases, respectively.

First, we consider that the released job has been finished. Since the released job has been finished, there is no demand from this job in future. The DBF will only bound the demand of future jobs. We assume future jobs are released as early as possible. At time $r_i(t) + T_i$, the jobs' release pattern is the same as the offline assumption. Then, the maximum demand within an interval will be the same as the DBF within the same interval in the offline analysis, as the demand within the interval Δ' seen in Fig. 2.4. Therefore, we prove that $\mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta, t) = \mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta)$.

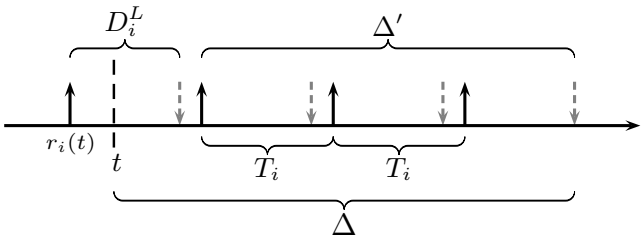


Figure 2.4: Bounding the demand of a task

Second, we consider that there is a backlogged job. The demand of task τ_i within an interval may include the demand of the backlogged job, or may not include this demand. If the demand of the backlogged job is not included, the upper bound of such a demand is the same as $\mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta)$. Therefore, $\mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta, t) \geq \mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta)$. Now we consider the upper bound on the demand that includes the demand of the backlogged job. To include the demand of this backlogged job, an interval Δ should start from t and end at $t + \Delta$. We use $\mathrm{Dmd}^{\mathrm{bk}}(\tau_{\mathrm{i}}, \Delta, \mathrm{t})$ to denote the upper bound of the demand that includes the backlogged job. The backlogged job may overrun or may not overrun. If the backlogged job does not overrun, it will demand $C_i^L - e_i(t)$. Otherwise, its demand is 0, because instead of contributing to the system demand, overrun in our technique is considered as the processing procrastination. In short, we use $[\![C_i^L - e_i(t)]\!]_0$ to denote the demand of the backlogged job. This backlogged job should be given $[\![C_i^L - e_i(t)]\!]_0$ before $r_i(t) + D_i^L$. Future jobs are assumed to be released as early as possible in order to maximize its demand. The request demand of every future job is C_i^L. Within Δ, the maximum number of arrival events is $\lfloor \big(\Delta + \min\big(T_i, t - r_i(t)\big)\big)/T_i \rfloor$, and those jobs should be given their requested demand no later than their LO mode deadlines. In summary, $\mathrm{Dmd}^{\mathrm{bk}}(\tau_{\mathrm{i}}, \Delta, \mathrm{t})$ is represented by Eq. 2.15, where its first part is the demand of the backlogged job and its second part is the demand of its future jobs.

Since $\mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta, t) \geq \mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta)$ and $\mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta, t) \geq \mathrm{Dmd}^{\mathrm{bk}}(\tau_i, \Delta, t)$, Eq. 2.14 holds. □

With Eqs. 2.14 and 2.14, we can get the $\mathrm{dbf}_{\mathrm{LO}}(\tau, \Delta, \mathrm{t})$ at runtime. By applying the following equation (similar to Eq. 2.12)

$$\rho^*(t) = \max\Big\{\rho : [\![\Delta - \rho]\!]_0 \geq \mathrm{dbf}_{\mathrm{LO}}(\tau, \Delta, \mathrm{t}),\ \forall \Delta \geq 0\Big\}, \tag{2.16}$$

where $\mathrm{dbf}_{\mathrm{LO}}(\tau, \Delta, t) = \sum_{\tau_i \in \tau} \mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta, t)$, we get $\rho^*(t)$. This $\rho^*(t)$ is used to renew the overrun budget OB at runtime whenever it elapses to zero.

2.5.2.4 Setting LO mode deadlines

From Eqs. 2.12 and 2.16, we can see that the initial OB and the updated OB depend on the task offline and online demand bound functions, hence depend on their LO mode deadlines. We proceed how to optimize OB by configuring tasks LO mode deadlines.

There can be a lot of options in setting D_i^L for every HI-critical task. Different options may have different runtime effects. To illustrate this problem, we consider the task set in the Example 1. There are two options of D_i^L, as shown in the following.

- Option 1: $D_2^L = 40$ and $D_3^L = 30$, which is the same as the example.
- Option 2: $D_2^L = 60$ and $D_3^L = 40$, which is different with the example.

In both options, the schedulability of this system in the offline analysis is guaranteed. However, the second option can initialize OB to 20, while the first option can only initialize OB to 10. Option 2 is expected to have a better performance because the initial OB is greater and D_2^L, D_3^L of option 2 are greater than those of option 1. When we update an OB, the $\rho^*(t)$ of using option 2 must be equal to or greater than that of using option 1.

Motivated by this example, we set the target to maximize the initial overrun budget by choosing D_i^L for every HI-critical task so that OB will be replenished the most in every update, i.e.,

$$\textbf{Constraint}: \ Eqs.\ 2.19a,\ 2.19b,$$
$$Target: \ \textbf{maximize} \ \ \rho^*.$$

where Eqs. 2.19a, 2.19b (Section 2.5.1) guarantee the schedulability of the system in both modes.

2.6 FFOB under FP schedule

In this section, we present how to integrate the FFOB scheme into FP-scheduled system. In order to achieve this goal, we propose an algorithm named RTI-FP that acts a similar role as EDF-VD in EDF-scheduled system. RTI-FP algorithm is based on the real-time interface analysis [41,42], which is a part of the framework of Real-Time Calculus [44,48]. In the following, we first present RTI-FP algorithm, then present how to initialize and update an overrun budget based on RTI-FP algorithm.

2.6.1 RTI-FP Algorithm

In this section, we introduce the RTI-FP algorithm that extends the standard fixed-priority based mixed-criticality scheduling technique; we further present the corresponding schedulability analysis and how to configure RTI-FP.

2.6.1.1 Real-Time Interface Analysis

For fixed-priority scheduled systems, we rely on the real-time interface analysis [41] to derive the supply bound function and demand bound function of a task (or a task set), and to analyze the system schedulability. The principle of real-time interface analysis is to test the schedulability by comparing the remaining system resource (SBF) with the task demand (DBF) of a task on every priority level. According to the real-time calculus [48], the SBF for lower priority tasks models the resources left over after processing all higher-priority tasks. Suppose task priorities are ordered in a descending order, i.e., τ_i has a higher priority than τ_j when $i < j$. Then, we have

$$\begin{aligned} \mathrm{sbf}(\tau_{i+1}, \Delta) &= \mathrm{RT}(\mathrm{sbf}(\tau, \Delta), \mathrm{wbf}(\tau_1^i, \Delta)) \\ &= \sup_{0 \le \lambda \le \Delta} \{\mathrm{sbf}(\tau, \lambda) - \mathrm{wbf}(\tau_1^i, \lambda)\}, \ \forall i \in \{1..n-1\}. \end{aligned} \tag{2.17}$$

where

$$\mathrm{sbf}(\tau, \Delta) = \Delta, \ \mathrm{wbf}(\tau_1^i, \Delta) = \sum_{j=1}^{i} \mathrm{wbf}(\tau_j, \Delta).$$

To guarantee the schedulability of every task, its SBF must be equal to or greater than its DBF, i.e.,

$$\mathrm{sbf}(\tau_i, \Delta) \ge \mathrm{dbf}(\tau_i, \Delta), \ \forall \Delta, \ \forall i \in \{1..n\}. \tag{2.18}$$

2.6.1.2 RTI-FP Algorithm

In order to facilitate runtime overrun budgeting, we would need more explicit control of task behaviors under fixed-priority scheduling. Inspired by the use of virtual deadlines in EDF-scheduled system where a task can be modeled in LO and HI modes individually, we also introduce virtual deadlines to HI-critical tasks in FP-scheduled systems. The use of virtual deadlines allows us to apply the real-time interface to analyze the two modes schedulability respectively, which further enables the feature of automatic schedulability guarantee.

Let us still denote the LO mode deadline as $D_i^L, \forall \tau_i \in \tau^H$. Based on the above real-time interface analysis, for each task, once its LO mode deadline is determined, we can derive the task workloads and demands in LO and HI modes, i.e., for LO mode we have $\mathrm{wbf}_{\mathrm{LO}}(\tau_i, \Delta)$ and $\mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta)$, and for HI mode we have $\mathrm{wbf}_{\mathrm{HI}}(\tau_i, \Delta)$ and $\mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta)$. Then, based on the real-time interface analysis, we have the following result.

Proposition 3. *A mixed-criticality task set is schedulable by fixed priority on a platform with* SBF $\mathrm{sbf}_{\mathrm{LO}}(\tau, \Delta) = \mathrm{sbf}_{\mathrm{HI}}(\tau^H, \Delta) = \Delta$ *if both of the following conditions hold:*

$$\text{FP-LO} : \forall \Delta \ge 0, \ \tau_i \in \tau : \mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta) \le \mathrm{sbf}_{\mathrm{LO}}(\tau_i, \Delta), \tag{2.19a}$$

$$\text{FP-HI} : \forall \Delta \ge 0, \ \tau_i \in \tau^H : \mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta) \le \mathrm{sbf}_{\mathrm{HI}}(\tau_i, \Delta). \tag{2.19b}$$

Proposition 3 provides a schedulability test for LO (FP-LO) and HI (FP-HI) modes individually. We proceed to show how to derive the workload and demand bound functions for all tasks, first for LO mode and then for HI mode.

LO Mode Schedulability

The runtime behavior of a mixed-criticality system in LO mode is the same as that of a conventional real-time system, assuming LO mode task parameters. To apply condition FP-LO of Eq. 2.19a to test LO mode schedulability, we compute the workloads and demand bounds of tasks as follows (see the definitions in Section 2.3).

$$\begin{aligned} \mathrm{wbf}_{\mathrm{LO}}(\tau_i, \Delta) &= \left\lfloor \frac{\Delta + T_i}{T_i} \right\rfloor \cdot C_i^L, \ \forall\ \Delta \geq 0, \\ \mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta) &= \left\lfloor \frac{\Delta + T_i - D_i^L}{T_i} \right\rfloor \cdot C_i^L, \ \forall\ \Delta \geq 0 \end{aligned} \quad . \tag{2.20}$$

With $\mathrm{wbf}_{\mathrm{LO}}(\tau_i, \Delta)$ and $\mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta)$, the system schedulability can be analyzed by the real-time interface analysis.

HI Mode Schedulability

Suppose all tasks can meet their corresponding deadlines in LO mode, we can now derive $\mathrm{wbf}_{\mathrm{HI}}(\tau_{\mathrm{i}}, \Delta)$ and $\mathrm{dbf}_{\mathrm{HI}}(\tau_{\mathrm{i}}, \Delta)$. A tight $\mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta)$ is already presented in Eq. 2.6. The WBF of a HI-critical task τ_i must upper-bound the maximum workload arriving within any interval of length Δ in HI mode. We derive $\mathrm{wbf}_{\mathrm{HI}}(\tau_i, \Delta)$ by relying on the fact that any task τ_i is deemed to finish C_i^L before its LO mode deadline as τ_i is schedulable in LO mode.

Lemma 3. *If a HI-critical task τ_i with a shortened deadline D_i^L is schedulable in LO mode, its* WBF *in HI mode is*

$$\begin{aligned} \mathrm{wbf}_{\mathrm{HI}}(\tau_i, \Delta) &= \left\lfloor \frac{\Delta + D_i^L + T_i}{T_i} \right\rfloor C_i^H - \mathrm{done}(\tau_{\mathrm{i}}, \Delta), \\ \mathrm{done}(\tau_i, \Delta) &= \begin{cases} [\![C_i^L - l + D_i - D_i^L]\!]_0, \ if D_i > l \geq D_i - D_i^L \\ 0, \mathrm{otherwise}, \end{cases} \end{aligned} \quad , \tag{2.21}$$

where $\Delta > 0$ *and* $\mathrm{wbf}_{\mathrm{HI}}(\tau_i, 0) = 0$ *and* $l = \Delta \bmod T_i$.

Proof. To derive the workload bound function in HI mode, we need to rely on the condition that any HI-critical task will not miss its LO mode deadline before the system switches to HI mode. Then, to denote the workload of a task after the mode-switch, we need to introduce the *carry-over* job. The carry-over job is a job from a HI-critical task that is active (released, but not finished) at the time of the switch to HI mode [18].

There is a fact (lemma III.3 of [18]).

Fact 1: If a job can definitely meet its LO mode deadline D_i^L before system switching to HI mode and this job has l time units left until its HI deadline D_i^H at the moment of system

switching to HI mode, then (1): this job has finished if $l < D_i^H - D_i^L$. (2): this job has finished at least $[\![C_i^L - \big(l - (D_i^H - D_i^L)\big)]\!]_0$ if $l \geq D_i^H - D_i^L$.

Since any HI-critical task will not miss its LO mode deadline before the system switching to HI mode, there must be at least a slack of $T_i - D_i^L$ to release the next job after the mode-switch. Thus, suppose the workload of the carry-over job is C_i^H whenever a mode-switch happens, we can bound the workload as follows.

$$\mathrm{Wrd}_{\mathrm{HI}}^{\mathrm{full}}(\tau_i, \Delta) = \left\lfloor \frac{\Delta + D_i^L + T_i}{T_i} \right\rfloor C_i^H, \ \forall \Delta \geq 0. \tag{2.22}$$

However, to make the workload bound function tighter, we need to consider the fact 1, i.e., the carry-over job may have finished some part of its execution before the mode-switch. Now, we derive how to accommodate such finished part within an interval of δ in $\mathrm{Wrd}_{\mathrm{HI}}^{\mathrm{full}}$.

Based on the fact 1, we can derive the lower bound of the finished execution time within an interval of δ in $\mathrm{Wrd}_{\mathrm{HI}}^{\mathrm{full}}$ as follows.

$$\mathrm{done}(\tau_i, \Delta) = \begin{cases} [\![C_i^L - l + D_i - D_i^L]\!]_0, \ if D_i > l \geq D_i - D_i^L \\ 0, \mathrm{otherwise}, \end{cases},$$

Therefore, a tight workload bound function is Eq. 2.6. □

With $\mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta)$ and $\mathrm{wbf}_{\mathrm{HI}}(\tau_i, \Delta)$, the system HI mode schedulability can be determined by applying the real-time interface to test the condition FP-HI.

2.6.1.3 Optimal LO Mode Deadline Assignment

We have shown our schedulability analysis by constructing the workload and demand bound functions in all modes. Since $\mathrm{wbf}_{\mathrm{HI}}(\tau_i, \Delta)$ and $\mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta)$ are functions related to LO mode deadlines, the HI mode system schedulability depends on LO mode deadlines. Therefore, choosing appropriate LO mode deadlines is critical for passing the RTI-FP test. To ensure LO mode schedulability, we know that $D_i^L \geq R_i^L$ (R_i^L being the response time of τ_i in LO mode and can be computed by using the standard response time analysis [49]). Furthermore, according to Eqs. 2.6, 2.21, decreasing D_i^L makes both $\mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta)$ and $\mathrm{wbf}_{\mathrm{HI}}(\tau_i, \Delta)$ smaller. As a result, the system remains schedulable in HI mode as there are less workloads/demands. Therefore, for HI-critical tasks, it is optimal to set their LO mode deadlines as small as possible, as long as they are schedulable in LO mode, i.e., $D_i^L = R_i^L, \forall \tau_i \in \tau^H$. We summarize this into the following theorem.

Theorem 1. *For our schedulability test as presented in Proposition 3, an optimal assignment of the* LO *mode deadline for a* HI*-critical task τ_i is that $D_i^L = R_i^L$. In other words, if the system is tested unschedulable by setting $D_i^L = R_i^L$, it will be tested unschedulable whatever D_i^L is.*

Proof. This directly follows from our discussions above. □

2.6.2 Initializing/Updating the Overrun Budget

We have presented the RTI-FP algorithm that acts a similar role as EDF-VD in EDF-scheduled system. Under the RTI-FP algorithm, we can follow a similar procedure of Section 2.5.2 on how to initialize and update the overrun budget.

2.6.2.1 Initialize OB

The initial budget denotes the overrun allowance that can be already identified offline. Recall that we view task overrun as a delay of resources provided to "normal" executions of tasks. Thus, analogous to our offline test, we can update the resource model as $\text{sbf}_{\text{LO}}(\tau, \Delta) = [\![\Delta - \rho]\!]_0$ $(\rho > 0)$, which means that the resource supply for LO mode is delayed not longer than ρ. Consequently, the real-time interface based schedulability test can be directly applied to find the maximum ρ while permitting system schedulability, as shown below (proof is in Section 2.7).

Lemma 4. *In a* FP*-scheduled system, the overrun budget can be initialized to* ρ^* *for a task set* τ *at the beginning and every idle tick:*

$$\rho^* = \max\Big\{\rho : \text{sbf}_{\text{LO}}(\tau, \Delta) = [\![\Delta - \rho]\!]_0,\ \forall \Delta \geq 0\Big\}, \tag{2.23}$$

such that under the real-time interface analysis, $\text{sbf}_{\text{LO}}(\tau, \Delta)$ *guarantees that the condition FP-LO of Eq. 2.19a is satisfied.*

2.6.2.2 Updating OB

At runtime, the system can generate dynamic slacks when tasks do not use their full WCETs; thus, we can update *OB* based on those runtime slacks. To achieve that, we first need the actual workload and demand bound functions at runtime. We still use $\text{wbf}_{\text{LO}}(\tau_i, \Delta, t)$ and $\text{dbf}_{\text{LO}}(\tau_i, \Delta, t)$ to denote the actual workload and demand bound functions from any time t online and onward. Note that we use the LO mode subscript since from the analysis point of view, the Border mode is just an extended LO mode. Now, we can formally introduce the runtime overrun budget (proof is in Section 2.7).

Lemma 5. *In* FP*-scheduled systems, the overrun budget can be renewed to* $\rho(t)$ *with the task runtime model* $\text{wbf}_{\text{LO}}(\tau_i, \Delta, t)$ *and* $\text{dbf}_{\text{LO}}(\tau_i, \Delta, t)$

$$\rho(t) = \max\Big\{\rho(t) : \text{sbf}_{\text{LO}}(\tau, \Delta, t) = [\![\Delta - \rho(t)]\!]_0,\ \forall \Delta \geq 0\Big\}, \tag{2.24}$$

such that under the real-time interface based analysis, the above $\text{sbf}_{\text{LO}}(\tau, \Delta, t)$ *guarantees*

$$\forall \Delta \geq 0,\ \tau_i \in \tau : \text{dbf}_{\text{LO}}(\tau_i, \Delta, t) \leq \text{sbf}_{\text{LO}}(\tau_i, \Delta, t).$$

To get $\rho(t)$, we need to know both $\mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta, t)$ and $\mathrm{wbf}_{\mathrm{LO}}(\tau_i, \Delta, t)$. $\mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta, t)$ has been derived in Lemma 2. In a similar way, $\mathrm{wbf}_{\mathrm{LO}}(\tau_i, \Delta, t)$ can be derived as follows.

Lemma 6. *At any time t, for a task τ_i that has no backlogged job at t, we can derive*

$$\mathrm{wbf}_{\mathrm{LO}}(\tau_i, \Delta, t) = \mathrm{wbf}_{\mathrm{LO}}(\tau_i, \Delta). \tag{2.25}$$

For a task τ_i that has one backlogged job at t, we have

$$\mathrm{wbf}_{\mathrm{LO}}(\tau_i, \Delta, t) = \max\left(\mathrm{wbf}_{\mathrm{LO}}(\tau_i, \Delta),\ \mathrm{Wrd}^{\mathrm{bk}}(\tau_i, \Delta, t)\right), \tag{2.26}$$

where $\mathrm{Wrd}^{\mathrm{bk}}(\tau_i, \Delta, t)$ is defined as

$$\begin{aligned}\mathrm{Wrd}^{\mathrm{bk}}(\tau_i, \Delta, t) = &\left[\!\left[\left\lfloor \frac{\Delta}{r_i(t) + T_i - t} \right\rfloor\right]\!\right]^1 \cdot \left[\!\left[C_i^L - e_i(t)\right]\!\right]_0 \\ &+ \left[\!\left[\left\lfloor \frac{\Delta + \min\left(T_i, t - r_i(t)\right) - T_i}{T_i} \right\rfloor\right]\!\right]_0 C_i^L,\end{aligned} \tag{2.27}$$

and $r_i(t)$ and $e_i(t)$ are the release time and the already execution time of the latest released job of τ_i at the moment t.

The proof of $\mathrm{wbf}_{\mathrm{LO}}(\tau_i, \Delta, t)$ follows a similar procedure as the proof of $\mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta, t)$. It is easy to verify and thus is omitted.

2.6.2.3 Setting LO mode deadlines

In order to postpone the transition to HI mode as long as possible, we need to maximize the initial overrun budget, i.e., we need to search LO mode deadlines so that they can help a task set τ to pass the schedulability test of proposition 3 and also make ρ^* of Eq. 2.23 the largest. We now present a method that can effectively find the maximum ρ^*. This method works in two steps: first it narrows down the search space of feasible LO mode deadlines and the initial overrun budget; it then applies binary search to find the maximum initial OB.

The feasible LO mode deadlines should meet two requirements: the task LO mode deadline should not be smaller than its worst-case response time because of the schedulability guarantee in LO mode, i.e., $D_i^L \geq R_i^L$; the task LO mode deadline should not be greater than $D_i - (C_i^H - C_i^L)$ in order to leave space to accommodate its overrun $(C_i^H - C_i^L)$ before its real deadline. In summary, the feasible domain of D_i^L is constrained within the following range:

$$R_i^L \leq D_i^L \leq D_i - (C_i^H - C_i^L),\ \forall \tau_i \in \tau^H. \tag{2.28}$$

Regarding the feasible domain of the longest procrastination interval ρ^*, we have

$$0 \leq \rho^* \leq \min(D_i - C_i^x),\ \forall\ \tau_i \in \tau, \tag{2.29}$$

Algorithm 1 Find the maximal ρ^*

Input: $ParaSet = \{T_i,\, D_i,\, C_i^L,\, C_i^H,\, \mathcal{X}_i\}$
Output: ρ^*

```
1: ρ_l = 0, ρ_r = min(D_i − C_i^x), ∀ τ_i ∈ τ
2: while ρ_r − ρ_l > ε do
3:     ρ* = (ρ_r + ρ_l)/2
4:     D_i^L=FindMinimalLODeadline(ParaSet, ρ*)
5:     if any one of Eqs. 2.19a, 2.19b, 2.28 fails then
6:         ρ_r = ρ*
7:     else
8:         ρ_l = ρ*
9:     end if
10: end while
11: function FindMinimalLODeadline(ParaSet, ρ*)
12:     Compute sbf_LO(τ_i, Δ), ∀τ_i ∈ τ                    ▷ forward analysis
13:     Search the minimal D_i^L that satisfies sbf_LO(τ_i, Δ) ≥ dbf_LO(τ_i, Δ), ∀τ_i ∈ τ^H    ▷ apply
    binary search
14: end function
```

where $x = L$ if $\tau_i \in \tau^L$ and $x = H$ if $\tau_i \in \tau^H$. This is because any task should receive enough execution before its deadline. Now, a complete formulation of the initial overrun budget maximization problem is as follows:

$$\begin{aligned} &\textbf{maximize}\ \ \rho^* \text{ of } Eq.\ 2.12, \\ &\textbf{s.t.}\ \ Eqs.\ 2.19a,\ 2.19b,\ 2.28. \end{aligned} \tag{2.30}$$

With the gathered overrun budget and the delayed system resource, $\mathrm{sbf}_{\mathrm{LO}}(\tau,\, \Delta) = [\![\Delta - \rho]\!]_0$, the smallest feasible LO mode deadline D_i^L corresponding to the delayed resource must not be smaller than R_i^L to make the system feasible. Actually, if $\rho_1 > \rho_2$, the smallest feasible LO mode deadlines corresponding to ρ_1 must not be smaller than those corresponding to ρ_2, as in the former case we have less resources available. Such a monotonic relationship leads us to a binary search based algorithm to maximize the initial overrun budget.

We now continue to explain Algorithm 1. In this algorithm, once we pick up a possible ρ^*, we can derive $\mathrm{sbf}_{\mathrm{LO}}(\tau_i, \Delta)$ for every task τ_i by using the real-time interface analysis. Based on $\mathrm{sbf}_{\mathrm{LO}}(\tau_i, \Delta)$, we can get the minimal feasible D_i^L. Then, the computed D_i^L will be used to test whether it satisfies the schedulability constraints.

2.7 Correctness of FFOB

This section proves the correctness of the FFOB mode-switch scheme, i.e., the two properties presented in Section 2.3 are guaranteed, and the completed jobs in Border mode also meet their LO mode deadlines. The following proof is based on a system resource abstraction that is the same for EDF- and FP-scheduled systems. Therefore, this proof is valid for both of them.

First of all, we need to clarify some denotations on SBF. $\mathrm{sbf}_{\mathrm{LO}}(\tau, \Delta, \mathrm{t})$ and $\mathrm{sbf}_{\mathrm{HI}}(\tau, \Delta, \mathrm{t})$ define the system SBF for the task set τ in LO mode and in HI mode from time t and onward, respectively. The system SBF for the task set τ across all modes is denoted as $\mathrm{sbf}(\tau, \Delta)$, and $\mathrm{sbf}(\tau, \Delta) = \Delta$ because the processor constantly provides full processing service.

2.7.1 All Tasks Meet their LO Mode Deadlines in LO Mode

We prove the property that all tasks meet their deadlines in LO mode by dividing the time interval into the busy interval and the idle interval. The busy interval is an interval during which the system is executing jobs. The idle interval is an interval in which no jobs are executed. In the idle interval, no jobs will miss their deadlines as no jobs exist under a work-conserving scheduler, i.e., EDF or FP. Hence, we need to guarantee this property only for all busy intervals. In the following, we first prove that FFOB guarantees this property in the first busy interval by Lemma 7. We then show that the system behavior in other busy intervals is similar to the first busy interval by Lemma 8 and Theorem 2.

For ease of the proof, we provide some denotations on time instants. Denote t_0 as the starting time of the system. Denote $[t_{sn}, t_{en})$ as the n-th busy interval where t_{sn} and t_{en} are the starting and ending times of a busy interval, respectively. We have $t_{s1} = t_0$. In any busy interval, the system will be switched to HI mode at most once because otherwise there must be an idle tick to switch the system from HI to the LO mode. If the system switches to HI mode, we denote $t_{ms,n}$ as the time instant within $[t_{sn}, t_{en})$ that the system switches to the HI mode. If the system does not switch to HI mode, we set $t_{ms,n} = t_{en}$ for completeness.

We prove that Property 1 holds in the first busy interval. The proof of other busy intervals are similar. During the first busy interval, OB is initialized to $\rho^*(t_0)$ at t_{s1} and the system will be in LO mode only within $[t_0, t_{ms,1}]$.

Lemma 7. *No tasks in LO mode will miss their LO mode deadlines during* $[t_0, t_{ms,1}]$.

Proof. According to FFOB mechanism in Fig. 2.1, LO mode is sometimes interfered by Border mode, while such interference is constrained by OB. Based on OB, we consider the following cases that may happen during $[t_0, t_{ms,1}]$.

- Case 1: OB does not elapse to 0.

If OB does not elapse to 0, the maximum accumulated time of the system in Border mode within $[t_0, t_{ms,1}]$ is less than $\rho^*(t_0)$, which means that the lower bound of resources available within this interval for LO mode is greater than $[\![\Delta - \rho^*(t_0)]\!]_0$. Since

$$\mathrm{dbf}_{\mathrm{LO}}(\tau, \Delta) \leq [\![\Delta - \rho^*(t_0)]\!]_0, \forall \Delta \geq 0, \tag{2.31}$$

we conclude that the lower bound of provided resources to the task set τ within $[t_0, t_{ms,1}]$ is greater than its demand, thus no tasks in τ will miss their deadlines in this case.

- Case 2: OB first elapses to 0 at a time $t_{c1,1}$. At the time $t_{c1,1}$, OB is updated to a new value $OB(t_{c1,1})$.

According to the analysis of case 1, we know that no tasks will miss their deadlines from time t_0 to time $t_{c1,1}$. At the time $t_{c1,1}$, $OB = OB(t_{c1,1})$ that satisfies

$$\mathrm{dbf}_{\mathrm{LO}}(\tau, \Delta, t_{c1,1}) \leq [\![\Delta - OB(t_{c1,1})]\!]_0, \ \forall \Delta > 0. \tag{2.32}$$

Analogously, from time $t_{c1,1}$ to the next time that OB elapses to 0 or the system is idled, the maximum accumulated time in Border mode is not greater than $OB(t_{c1,1})$. Thus, the lower bound of resources available for the task set τ in LO mode is greater than its LO mode DBF. This way, tasks will not miss their LO mode deadlines.

Other cases just repeat case 1 and case 2.

Therefore, all LO mode deadlines can be met in LO mode during the interval $[t_0, t_{ms,1}]$. □

All LO mode deadlines have been proved to be met in LO mode in the first busy interval. From the case 2 in the above proof, we find that the OB updating procedure in FFOB is sufficient to guarantee LO mode deadlines. In other busy intervals, the FFOB can have an impact on the system only by resetting OB to $\rho^*(t_0)$ at the starting time of every busy interval and the OB updating procedure during the busy interval. Therefore, we only need to prove that this reset is sufficient to guarantee that LO mode deadlines can be met in other busy intervals. We prove this by considering the system demand bound functions.

Lemma 8. *The* DBF *and* WBF *of a task τ_i in LO mode at time t_{sn}, $n \in \mathbb{N}^+$ are the same as its* DBF *and* WBF *in LO mode at time t_{s1}.*

Proof. At time t_{sn}, there are no backlogged jobs. According to Lemma 2 and Lemma 6, the LO mode DBF and WBF at time t_{sn} are the same as those at at time t_{s1}. □

Theorem 2. *At any time t_{sn}, $n \in \mathbb{N}^+$, OB can be reset to a value $\rho^*(t_0)$ that guarantees all tasks can meet their LO mode deadlines in LO mode.*

Proof. Since the LO mode DBF and WBF at time t_{sn} are the same as those at at time t_{s1} and the system can keep schedulable in LO mode during $[t_0, t_{ms,1}]$ by setting OB to $\rho^*(t_0)$, the system can also keep schedulable in LO mode during $[t_{sn}, t_{ms,n}]$ by setting OB to $\rho^*(t_0)$ at time t_{sn}. □

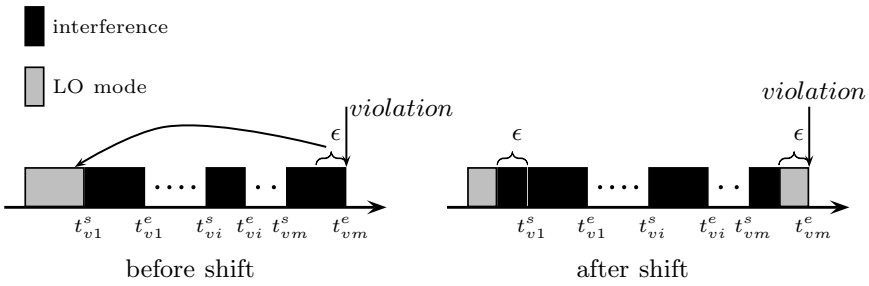


Figure 2.5: Contradiction illustration

2.7.2 Deadline Guarantees in Border mode

An important property in Border mode is that the overrun job in Border mode can also meet its LO mode deadline. Such property will be used in proving Property 2 in the following.

Theorem 3. *Any job finished in Border mode meets its LO mode deadline.*

Proof. Its proof relies on a useful fact.

Fact 1: Suppose $\mathrm{sbf_{LO}}(\tau,\, \Delta,\, \mathrm{t}) = [\![\Delta - \rho(t)]\!]_0$, $\forall \Delta \geq 0$ at time t guarantees that

$$\forall \Delta \geq 0 : \mathrm{dbf_{LO}}(\tau, \Delta, t) \leq \mathrm{sbf_{LO}}(\tau, \Delta, t),$$
$$\textit{or } \forall \Delta \geq 0,\, \tau_i \in \tau : \mathrm{dbf_{LO}}(\tau_i, \Delta, t) \leq \mathrm{sbf_{LO}}(\tau_i, \Delta, t).$$

Then from time t, no matter when the interference occurs, LO mode can be kept schedulable until the accumulated interference is greater than $\rho(t)$. This fact directly comes from the definition of supply bound function.

Now we prove this property by contradiction. Suppose at time t_v, a job J_i^v from task τ_i violates its LO mode deadline in Border mode. This job must have executed beyond its LO WCET C_i^L at t_v. We denote the time instant that the execution time of J_i^v reaches to C_i^L as t_{C_i}. As shown in Fig. 2.5, we further denote a time interval $[t^s_{vi}, t^e_{vi}]$, $i \in \mathbb{N}^+$ as the time interval that J_i^v runs before its timing violation, where $t^s_{v1} = t_{C_i}$, $t^e_{vm} = t_v$[1]. The interval of $[t^s_{vi}, t^e_{vi}]$, $i \in \mathbb{N}^+$ is the interference on LO mode. Now, we make an assumption on the interval of $[t^s_{vi}, t^e_{vi}]$, $i \in \mathbb{N}^+$. Assume $[t^s_{vi}, t^e_{vi}]$, $i \in \mathbb{N}^+$ as slacks that do not run J_i^v or any other jobs. We shift a slack of length ϵ from $[t^e_{vm} - \epsilon, t^e_{vm}]$ to $[t_{C_i} - \epsilon, t_{C_i}]$. Then, the system will be in LO mode within a interval of $[t^e_{vm} - \epsilon, t^e_{vm}]$. The LO mode deadline of J_i^v will be violated in this way, which is contradicted with the fact 1 that the LO mode can be schedulable whenever the interference occurs. □

2.7.3 HI-Critical Tasks Meet their Deadlines in any Mode

We have already proved that all HI-critical tasks meet their LO mode deadlines[2] in LO and Border modes. Based on this property, we now prove that HI-critical tasks can also meet their

[1] t^s_{v1} may not be equal to t_{C_i} in reality, but it does not change the proof later on.
[2] LO mode deadline is smaller than the original deadline

deadlines in HI mode.

In Section 2.5.2.4 and Section 2.6.2.3, the LO mode deadlines have been chosen to guarantee that the system is schedulable in HI mode if FFOB is not applied. To be specific, the LO mode deadlines guarantee that $\forall \Delta \geq 0 : \mathrm{dbf}_{\mathrm{HI}}(\tau^H, \Delta) \leq \Delta$ in EDF system and $\forall \Delta \geq 0$, $\tau_i \in \tau^H : \mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta) \leq \mathrm{sbf}_{\mathrm{HI}}(\tau_i, \Delta)$ in FP system. In the following, we show that the demand bound function of a task in the system with FFOB will be smaller than $\mathrm{dbf}_{\mathrm{HI}}(\tau^H, \Delta)$. Besides, the HI mode supply bound function of a system with FFOB is still Δ because the system HI mode is not interfered by any other modes. Based on these analysis, we prove that HI-critical tasks can meet their deadlines under the FFOB scheme.

Theorem 4. *FFOB guarantees that* $\mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta, t_{ms}) \leq \mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta)$ *and* $\mathrm{wbf}_{\mathrm{HI}}(\tau_i, \Delta, t_{ms}) \leq \mathrm{wbf}_{\mathrm{HI}}(\tau_i, \Delta)$*, where* $\tau_i \in \tau^H$*.*

Proof. In this proof, we need to introduce the *carry-over* job again. A carry-over job is a job from a HI-critical task that is active (released, but not finished) at the instant that the system switches to HI mode [18].

The smallest time interval that a carry-over job's demand must be met is $D_i^H - D_i^L$ as any HI-critical task will not miss its LO mode deadline before the system switches to the HI mode. Except the carry-over jobs, other jobs in HI mode have a time length D_i^H ($D_i^H = T_i$) to meet their deadlines. It indicates that the smallest time interval in which the demand of k jobs must be met is $(k-1) \cdot T_i + D_i^H - D_i^L$. Then, for any HI-critical task τ_i, its HI mode DBF is bounded by

$$\mathrm{Dmd}_{\mathrm{HI}}^{\mathrm{full}} = \left\lfloor \frac{\Delta + T_i - (D_i - D_i^L)}{T_i} \right\rfloor C_i^H, \ \forall \Delta \geq 0. \tag{2.33}$$

The carry-over job may have been executed for some time before the system switched to HI mode. To get its least execution time, we rely on the following fact.

- Fact 1: If a job meets its LO mode deadline D_i^L before the system switches to the HI mode and this job has n time units left until its HI mode deadline D_i^H at the moment of the mode-switch, then (1): this job has finished if $l < D_i^H - D_i^L$; (2): this job has finished at least $[\![C_i^L - \big(l - (D_i^H - D_i^L)\big)]\!]_0$ if $l \geq D_i^H - D_i^L$.

This fact comes from Lemma III.3 of [18]; it is proposed for the carry-over job that does not overrun its LO WCET. For a carry-over job that has overrun its LO WCET, however, its execution time before the mode-switch to HI mode is more than the execution time in Fact 1. Thus, this fact also holds for the FFOB mode-switched system.

Now, we analyze how to get such l in Fact 1. As shown in Fig. 2.6, for a time interval Δ, time units left for the carry-over job are at most $l = \Delta \bmod T_i$ ($0 \leq l < D_i^H = T_i$). If $l < D_i^H - D_i^L$, this carry-over job must have finished. Otherwise, it has finished $[\![\epsilon + C_i^L - \big(l - (D_i^H - D_i^L)\big)]\!]_0$, where ϵ is the time length that a job overruns. The lower bound of finished execution time

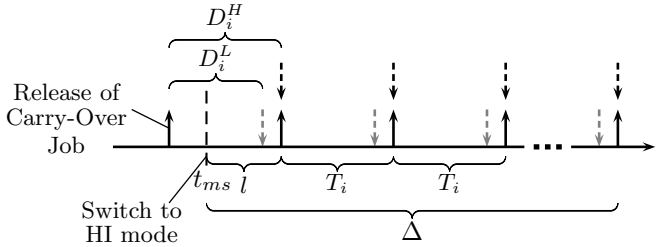


Figure 2.6: Illustration of the HI mode DBF of a HI-critical task

units in Δ is then,

$$\mathrm{Ext}_{\mathrm{HI}}^{\mathrm{done}} = \begin{cases} [\![\epsilon + C_i^L - l + D_i - D_i^L]\!]_0, \ if\ D_i > l \geq D_i - D_i^L \\ 0, \text{otherwise}, \end{cases}, \tag{2.34}$$

where $\epsilon \geq 0$ and $l = \Delta \bmod T_i$.

Therefore, the HI mode DBF of a task τ_i from time t_{ms} is

$$\mathrm{dbf}_{\mathrm{HI}}(\tau_{\mathrm{i}}, \Delta, \mathrm{t_{ms}}) = \mathrm{Dmd}_{\mathrm{HI}}^{\mathrm{full}} - \mathrm{Ext}_{\mathrm{HI}}^{\mathrm{done}}. \tag{2.35}$$

Compared to $\mathrm{dbf}_{\mathrm{HI}}(\tau_{\mathrm{i}}, \Delta)$ in Eq. 2.6, we know for any HI-critical task τ_i, $\mathrm{dbf}_{\mathrm{HI}}(\tau_{\mathrm{i}}, \Delta) \geq \mathrm{dbf}_{\mathrm{HI}}(\tau_{\mathrm{i}}, \Delta, \mathrm{t_{ms}})$ because $\epsilon \geq 0$. Therefore, for a set of HI-critical tasks, we have $\mathrm{dbf}_{\mathrm{HI}}(\tau^H, \Delta, t_{ms}) \leq \mathrm{dbf}_{\mathrm{HI}}(\tau^H, \Delta)$.

In a similar procedure, we can prove that $\mathrm{wbf}_{\mathrm{HI}}(\tau_i, \Delta, t_{ms}) \leq \mathrm{wbf}_{\mathrm{HI}}(\tau_i, \Delta)$. □

Theorem 5. *In* EDF *system with* FFOB*, if* LO *mode deadlines make Eqs. 2.19a, 2.19b hold,* HI*-critical tasks can be guaranteed to meet their deadlines by guaranteeing* HI*-critical tasks to meet their* LO *mode deadlines in* LO *and Border modes. Analogously, in* FP *system, if* LO *mode deadlines make Eqs. 2.19a, 2.19b hold,* HI*-critical tasks can also be guaranteed to meet their deadlines by guaranteeing* HI*-critical tasks to meet their* LO *mode deadlines in* LO *and Border modes.*

Proof. Since the system HI mode cannot be interfered by LO or Border mode, we have $\mathrm{sbf}_{\mathrm{HI}}(\tau^{\mathrm{H}}, \Delta, \mathrm{t_{ms}}) = \mathrm{sbf}_{\mathrm{HI}}(\tau^{\mathrm{H}}, \Delta) = \Delta$.

If the system LO mode is schedulable online, according to Theorem 4, we have $\mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta, t_{ms}) \leq \mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta)$ and $\mathrm{wbf}_{\mathrm{HI}}(\tau_i, \Delta, t_{ms}) \leq \mathrm{wbf}_{\mathrm{HI}}(\tau_i, \Delta)$, where $\tau_i \in \tau^H$. Thus, as $\mathrm{sbf}_{\mathrm{HI}}(\tau^{\mathrm{H}}, \Delta) = \Delta$ guarantees that the system modeled by $\mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta)$ and $\mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta)\ \forall\ \tau_i \in \tau^H$ is schedulable, $\mathrm{sbf}_{\mathrm{HI}}(\tau^{\mathrm{H}}, \Delta, \mathrm{t_{ms}}) = \Delta$ also guarantees that the system HI mode online is schedulable. □

2.7.4 Automatic Schedulability Guarantee.

From the above we conclude that, if the schedulability of HI mode needs to be guaranteed, we need to, first choose appropriate LO mode deadlines offline that can make conditions EDF (FP)-LO and EDF (FP)-HI hold, and second keep the system being schedulable in LO mode

at runtime because its schedulability in HI mode can be automatically guaranteed according to Theorem 5.

2.8 Experimental Evaluation

In this section, we compare the performance of our proposed techniques with state-of-the-art scheduling techniques by extensive simulations and implementations in an embedded platform.

2.8.1 Compared Approaches and Evaluation Metrics

We have proposed the FFOB mechanism for EDF and FP systems respectively. For each system, from the perspective of practical implementations and the control of the runtime overheads, we could have some different options. For example, we may use the FFOB mechanism to only exploit the static slack by removing its updating OB step, thus making its runtime complexity $O(1)$. In particular, our proposed approaches can be implemented in following ways:

- EDF-S (S means simple): The EDF scheduling with a simple FFOB mode-switch scheme that resets OB to its initial value whenever there is an idle tick. This approach does not update OB at runtime when OB is depleted to 0, in order to have $O(1)$ online complexity.
- EDF-A (A means advanced): The EDF scheduling with an advanced FFOB mode-switch scheme that updates OB at runtime when OB is depleted to 0 and resets to its initial value whenever an idle tick emerges.
- RTI-FP-S: Similar to EDF-S, this approach is a simple implementation of FFOB in the FP scheduling that only resets OB to its initial value when there is an idle tick. Its online complexity is also $O(1)$.
- RTI-FP-A: Similar to EDF-A, this approach is implemented in a FP-scheduled system that resets its OB when an idle tick emerges and updates its OB when the OB is depleted to 0.
- RTI-FP-L (L means lightweight): The RTI-FP-A approach uses an exact real-time interface analysis to update OB, which is complex and may prohibit its use online. To reduce the computation overheads, we introduce a lightweight approach RTI-FP-L that uses an approximation method to fast update OB. Details of this approach are in Section 4.7.

In order to evaluate the performances of our proposed approaches, we introduce three state-of-the-art scheduling techniques in mixed-criticality systems.

- EDF-VD: The basic EDF-VD scheduling that forces the mode-switch whenever a HI-critical task overruns.
- AMCmax: The basic fixed-priority scheduling with the AMCmax schedulability test. Details of this test can be seen in [8].
- Bailout: The purpose of this protocol is to timely switch the system from HI mode back to LO mode, instead of waiting for an idle tick. Details of this approach are in [31].

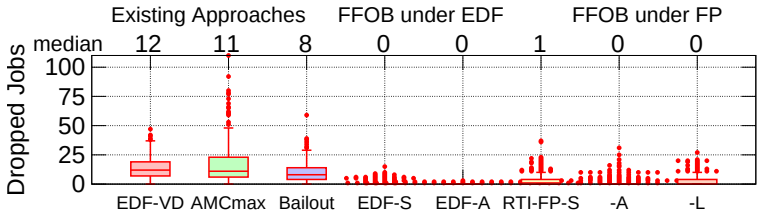
The following metrics are used to evaluate the performance of the listed approaches above.

- Dropped Jobs of LO-Critical Tasks: This metric represents the number of dropped jobs of LO-critical tasks in an interval. Note that, no HI-critical jobs will be dropped in all compared approaches.
- Timing Length of HI-mode: This metric represents the time length that the system stays in HI mode during an interval. The smaller this metric is, the better overall system QoS is.
- Number of Mode-Switch: This metric represents the number of mode-switches in an interval.

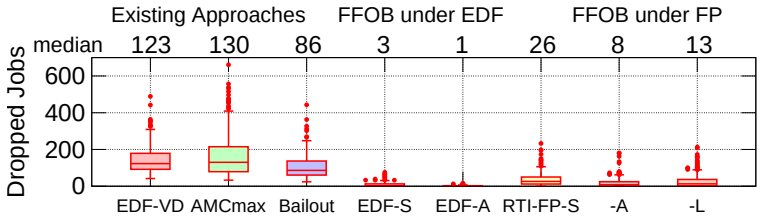
2.8.2 Random Task Set Generation

We use synthetic task sets for our experiments. In particular, we generate random sporadic task sets similar to [8], at each system utilization point (U). Any generated task set consists of 20 tasks, with task parameters chosen as follows.

- *Period and Deadline* - The period of each task is set as $25 \cdot w$, with w being a random integer within [1, 40]. Its timing unit is millisecond (ms). Each task's deadline equals its period. These periods are consistent with the periods typically found in automotive and avionics systems [31, 50].
- *Criticality* - A task is HI-critical with a probability 0.5.



(a) OP $= 10^{-5}$



(b) OP $= 10^{-4}$

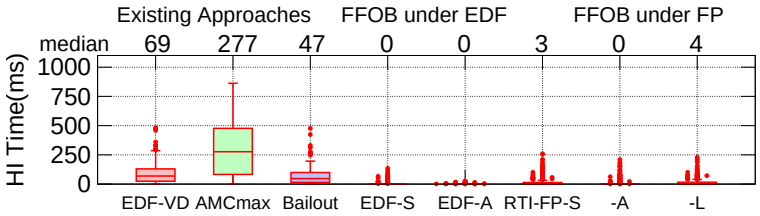
Figure 2.7: Boxplot – number of dropped jobs with different overrun probabilities (OP); Top number of each subfigure show the medians across all approaches; -A, -L in the right plots represent the approaches RTI-FP-A and RTI-FP-L

- *Worst-Case Execution Times in LO and HI modes* - The LO level WCETs are determined according to the UUniFast algorithm [51], which ensures that utilizations are distributed to tasks without bias. For a task τ_i, with utilization U_i generated, its LO level WCET is simply $C_i^L = U_i \cdot T_i$. If this task is HI-critical, its HI level WCET is then $C_i^H = 2 \cdot C_i^L$. The system utilization in LO mode is chosen as 0.7, in which way the system utilization in the case that all tasks run their highest WCETs will exceed 1 and will need a mode-switch scheme to make this system be mixed-criticality schedulable.

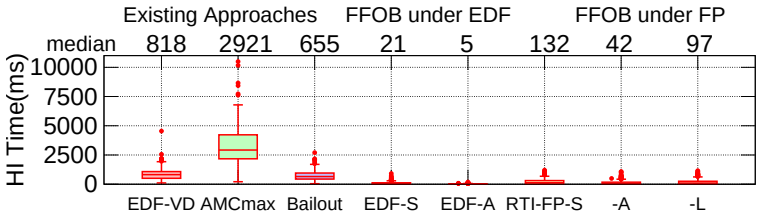
2.8.3 Simulation Results

We now simulate all the above approaches on randomly generated task sets. The simulator is implemented in Matlab on a host with Intel Q9300 processor and 5GB RAM. During our simulations, every job has a probability OP to overrun its LO level WCET. If a task does not overrun, its actual execution time is randomly set within $[0.6 \cdot C_i^L, C_i^L]$. If a task overruns, its actual execution time is randomly drawn in $(C_i^L, 3 \cdot C_i^L]$. We choose OP to 10^{-5} or 10^{-4}, and simulate each task set for 10^7 ms. Our results are summarized in Figs. 2.7, 2.8 and 2.9.

We first compare the number of dropped jobs among those approaches. The numbers of dropped jobs under different overrun probabilities (OP) are presented in Figure 2.7. Here,



(a) OP $= 10^{-5}$

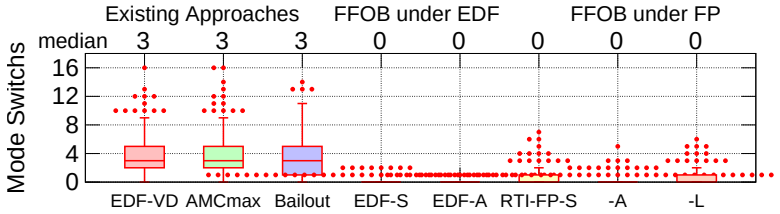


(b) OP $= 10^{-4}$

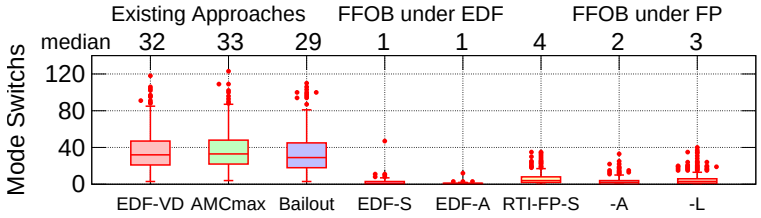
Figure 2.8: Boxplot – HI mode time length with different overrun probabilities (OP). The label rule is the same as Fig. 2.7

we use boxplot to show the distribution of the number of dropped jobs for 500 different task sets. The left plots show the number of dropped jobs of existing approaches, i.e., EDF-VD, AMCmax, Bailout; the middle plots show the result of our proposed FFOB approaches in EDF schedule, i.e., EDF-S and EDF-A; the right plots show the result of FFOB in FP schedule, i.e., RTI-FP-S, RTI-FP-A and RTI-FP-L. Overall, the results confirm a significant reduction of dropped jobs by using the FFOB mode-switch scheme. The FFOB approaches achieve 5-10 folds reductions compared to the existing approaches. The most two effective approaches are EDF-A and EDF-S whose medians are close to 0 for OP $= 10^{-5}/10^{-4}$. In FP schedules, the most effective approach is RTI-FP-A whose median is two-thirds of RTI-FP-L and one-third of RTI-FP-S when OP $= 10^{-4}$. EDF-S and RTI-FP-S are less effective than EDF-A and RTI-FP-A because EDF-S and RTI-FP-S only exploit the system static slack. However, both simple implementations still outperform AMCmax and Bailout protocol to a large extent, which is a significant result considering their $O(1)$ online complexity. RTI-FP-L is less effective than RTI-FP-L but more effective than RTI-FP-S because RTI-FP-L uses an approximation approach to reclaim the dynamic slack. However, as we will show later, RTI-FP-L indeed incurs less runtime overheads due to adopting this approximation.

We proceed to present the time length of HI mode during 10^7 ms simulations in Figure 2.8. The left plots show that, compared to AMCmax, both EDF-VD and Bailout are effective in



(a) OP $= 10^{-5}$



(b) OP $= 10^{-4}$

Figure 2.9: Boxplot – number of mode switch times with different overrun probabilities (OP). The label rule is the same as Fig. 2.7

reducing such time length in HI mode. However, all of them are worse than FFOB approaches. Specifically, the medians of HI mode time length of EDF-S and EDF-A are 40 and 164 folds smaller than those of EDF-VD; 31 and 131 folds smaller than Bailout, for OP $= 10^{-4}$. The right plots also confirm that the performances for the different RTI-FP options are significantly improved. The order from the best to the lowest performance is RTI-FP-A, RTI-FP-L and RTI-FP-S. Our results here demonstrate that, even with a simple implementation of FFOB (e.g. EDF-S, RTI-FP-S or RTI-FP-L), it can still greatly improve the QoS of LO-critical tasks.

The above conclusions also fit for the number of mode-switch. As confirmed in Fig. 2.9, the FFOB mode-switch scheme, in both simple and complex implementations, reduces many folds of mode-switch times compared to the existing approaches.

2.8.4 Implementation Results

In addition to extensive simulations, we implement all the compared approaches on top of a well designed framework called SF3P [28]. All implementations are tested on a host of Raspberry Pi 3 board with a 1.2 GHz ARMv8 CPU [52].

A total of 40 task sets were simulated and each task was simulated for 15 minutes. In order to have a distinguishing result among all compared approaches in a short time (15 minutes), we have to artificially set the overrun probability high enough. We set OP to one of $0.001, 0.01, 0.1$.

Table 2.1: Results of the compared approaches w.r.t. $OP = 0.001, 0.01, 0.1$

$OP = 0.001$	EDF-VD	AMCmax	Bailout	EDF-S
Dropped Jobs	436	451	337	15
HI Mode Time (ms)	1759	6157	1539	47
Mode-Switches	109	117	103	3
$OP = 0.001$	EDF-A	RTI-FP-S	RTI-FP-A	RTI-FP-L
Dropped Jobs	4	156	13	41
HI Mode Time (ms)	14	1386	139	398
Mode-Switches	1	31	4	9
$OP = 0.01$	EDF-VD	AMCmax	Bailout	EDF-S
Dropped Jobs	2079	2184	1767	63
HI Mode Time (ms)	10046	34136	8285	246
Mode-Switches	577	625	561	17
$OP = 0.01$	EDF-A	RTI-FP-S	RTI-FP-A	RTI-FP-L
Dropped Jobs	17	693	69	136
HI Mode Time (ms)	58	6450	849	1672
Mode-Switches	5	138	15	32
$OP = 0.1$	EDF-VD	AMCmax	Bailout	EDF-S
Dropped Jobs	8834	9456	8126	296
HI Mode Time (ms)	39468	142083	36432	1196
Mode-Switches	2025	2152	1977	73
$OP = 0.1$	EDF-A	RTI-FP-S	RTI-FP-A	RTI-FP-L
Dropped Jobs	73	4308	608	1136
HI Mode Time (ms)	262	42003	11057	14166
Mode-Switches	16	829	133	199

The results are shown in Table 2.1, where each cell number represents the median value of the measured metric on all task sets. We observe that the most effective two approaches are still EDF-S and EDF-A that outperform the EDF-VD to many folds in reducing all provided metrics. The RTI-FP-A is still the best approach among the three FFOB implementations in FP-scheduled system. Another point to mention is, that the numbers compared between the different overrun probabilities are decreasing by about the factor 0.2, whereas the OP decreases by the factor 0.1. One possibility could be, the smaller OP is, the less jobs will overrun. Furthermore it can be assumed that if less jobs overrun, the count of resets of the overrun budget will be more, because an idle tick occurred is larger and therefore the QoS increases.

We present the computation overheads of our overrun budgeting schemes on a logarithmic scale in Fig. 2.10. Since simple implementations of FFOB, i.e., EDF-S and RTI-FP-S have $O(1)$

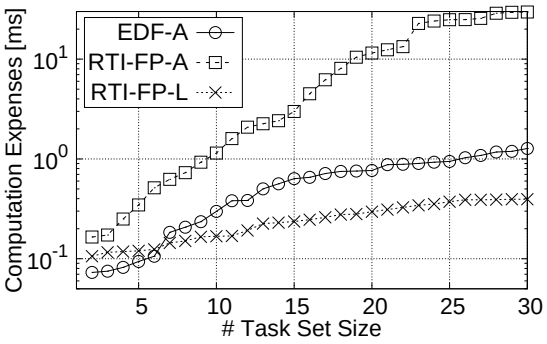


Figure 2.10: Computation overheads evaluation

online complexity, we do not need to measure their computation overheads. We plot the average computation overheads of every update as a function of the task set size for the EDF-A, RTI-FP-F and RTI-FP-A where each task set size was measured 200 times. As one can observe, the timing overhead of RTI-FP-F is the largest, which can be 2 orders of magnitude higher than that of RTI-FP-L. This result confirms that RTI-FP-L is very effective in reducing the runtime overhead; the exact real-time interface analysis needs a longer time in updating and is thus hard to be implemented online. Besides, we observe that EDF-A has less computation expense than RTI-FP-L when the task set size is small, but bears up to 3 times more computation expense compared to RTI-FP-L when the task set size is large. However, both of them are lightweight because their computation expenses are only about 0.4 and 1.2 ms even when the task set size is 30.

2.9 Summary

State-of-the-art mixed-criticality scheduling techniques commonly assume to switch system mode and drop all less critical tasks whenever any critical task overruns. Despite many efforts in reducing the pessimism of this approach, postponing the mode-switch to improve system guarantees by exploring system slacks dynamically online, remains unsolved for mixed-criticality systems. Such a problem is important as mode-switch procrastination naturally helps to improve the system performance.

We propose an online mode-switch procrastination technique called on-the-fly fast overrun budgeting in this chapter for both FP- and EDF-scheduled MCSs. The proposed approach has a feature of automatic schedulability guarantee that transfers the problem of mixed-criticality schedulability guarantee online to the counterpart of conventional real-time systems. With a

routinely updated shared resource pool of overrun budgets, the system allows tasks to overrun, which thus postpones the mode-switch as long as possible. Extensive simulations and real platform implementations confirm that our proposed technique significantly improves the system's QoS over state-of-the-art schedulers, while at the same time permitting light-weight online deployments.

Chapter 3

Schedulability Analysis on Arbitrarily Activated Tasks

In an embedded system, tasks are often referred to some specific-purpose programs that can be repeatedly activated. The task activation can be categorized into two types: *time-triggered* and *event-triggered*. Time-triggered tasks are usually periodically activated tasks whose activation periods have been decided at the system-design stage. Periodic executions are often found in some deterministic or static systems, while can also be seen in systems where periodic polling is used to obtain deterministic behavior. For periodically activated tasks, the optimal scheduling algorithm is the deadline-monotonic scheduling in static priority system (task priorities cannot be changed at runtime) and the earliest-deadline-first scheduling in dynamic priority system (task priorities can be changed at runtime). The event-triggered tasks are not activated by a timer event, but usually be activated upon a signal reception from other sources. For example, the interrupt routine service task can be activated only by an interrupt signal or a communication driver can be activated only by the reception of a protocol frame from an external bus. Event-triggered tasks are often modeled as sporadic tasks (often called asynchronous periodic) whose successive activations are separated by a known minimum time gap, also called its period. It has been proven that the deadline-monotonic scheduling algorithm and earliest-deadline-first scheduling algorithm are also optimal for sporadic tasks. In addition to the periodic and sporadic tasks, two other popular activation models are periodic activations with jitter and sporadically periodic events (also called "sporadic bursts") [53].

The schedulability analysis on the aforementioned task activation models has been discussed in a vast amount of paper in past decades. Most of their results are subject to some specific task

models. Analogously, in mixed-criticality scheduling, a lot of approaches have been presented on how to perform the effective schedule to address the problem of providing the timing guarantee to each task's criticality level, while none of them can be applied to analyze tasks that can be activated arbitrarily. Recent development of system performance analysis frameworks [53–55] that are applicable to any kind of task activations provide us some inspirations on how to make the scheduling analysis applicable to more general task activation patterns. That is, there may be a solution of using some results or some approaches from the system performance analysis to analyze the system schedulability of multi-criticality tasks that are activated arbitrarily.

3.1 Overview

Multi-criticality tasks are often modeled as sporadic tasks. Based on this model, a lot of scheduling approaches [6, 8] have been proposed to verify the schedulability of a task set in MCSs. Although the sporadic task model can represent many nondeterministic activation patterns by assuming that the task can be activated in every period, such representation may not be effective in practice. For instance, a simple approach to dealing a periodic task with a jittery release is to transform it into a new sporadic task with a shorter period [56]. While this approach is safe, the transformation can lead to overly pessimistic schedulability analysis results. In particular, if this shorter period is smaller than the worst-case execution time (WCET) of this task, it is impossible to schedule this task by modeling it as a sporadic task, because the deadlines deem to be missed in the case that the sporadic task is activated in every shorter period. The real situation is that the task cannot be activated in every shorter period. This task may be schedulable in MCSs, which however will be tested unschedulable by modelling it as a sporadic task.

The periodic task with a jittery release or the task with burst activations, often exists in many reactive embedded systems. The jitter may come from release-delay overheads induced by tick-driven scheduling [57], execution of interrupt service routines [56], or I/O overheads. The delays by scheduling and data dependencies may also cause the jitter. In ARINC avionics systems, different tasks scheduling partitions are connected over a switched Ethernet. Due to the network delay, tasks in a partition are not always released strictly periodically, but with a certain jitter [58]. In the automotive systems [25], a lot of event streams that are used to activate tasks suggest the use of more general event stream models than the classical sporadic model. In the traditional real-time systems, complex activation patterns are often modeled as

the *arrival curve* or the *minimum distance function* to compute the system throughput and delay by applying the Modular Performance Analysis [55] (under the framework of Real-Time Calculus [44, 48]) or the Compositional Performance Analysis [54]. In contrast with certifying the system throughput and delay on one level, the throughput and delay need to be certified based on the criticality of tasks in MCSs, which complicates the analysis.

In this chapter, the schedulability of dual-criticality system with arbitrarily activated tasks is analyzed. The schedulability analysis towards the arbitrarily activated tasks in MCSs, however, is nontrivial. In contrast with that at most one carry-over job (released but not finished) exists at the time of system mode switch for each sporadic task, there may be several carry-over jobs at once for every arbitrarily activated task. The exact number of carry-over jobs is difficult to obtain because the carry-over jobs depend on the specific activation patterns of this task and all higher priority tasks. This complicates the derivation of a tight bound of the worst-case response time (WCRT) of a task. Furthermore, in a FP-scheduled system, a given task set need to be assigned their priorities and the exhaustive searching over all possibilities is time-consuming. Hence, a more effective approach should be used to assign priorities.

Being aware of the above, we propose a new schedulability test to extend the classical sporadic task model to the arbitrarily activated task model in FP-scheduled and EDF-scheduled MCSs, respectively. The detailed contributions are as follows:

- In FP-scheduled systems, we extend the classic sporadic task model to the arbitrarily activated task model by presenting a necessary and two sufficient schedulability tests. We also show that Audsley's algorithm is applicable in those tests. Besides, the schedlability test on arbitrarily activated tasks is also extended to the EDF-scheduled systems.

- By using the arrival curve to present the upper bound of task activations, we integrate the well-established results from Real-Time Calculus to analyze the schedulability of arbitrarily activated tasks in MCSs.

- By using the minimum distance function to model task activations and using the busy-window analysis to compute the WCRT of tasks, we present a tighter sufficient schedulability test than the schedulability test of using Real-Time Calculus framework.

- It is demonstrated that, for the sporadic tasks, our proposed tests can achieve the same schedulability as the two state-of-the-art approaches: AMC-max in FP schedule [8] and EDF-VD in EDF schedule. However, compared with them, our tests can handle the

tasks with blocking, jitter, and arbitrary deadlines. Besides, experimental results show that the increase of jitter decreases the system schedulability, while the increase of relative deadlines increases the system schedulability.

The remainder of this chapter is structured as follows. Next section reviews related work. Section 3.3 presents the system model, settings and a motivation example. Section 3.4 and Section 3.5 present the schedulability analysis on FP-scheduled and EDF-scheduled MCSs, respectively. Section 3.6 provides the experimental results and the last section concludes this chapter.

3.2 Related Work

Since the first paper on the verification of a proposed MCS in 2007 [5], the response time and the schedulability analysis on MCSs have ranged from the uniprocessor to the multiprocessor, and the scheduling policies mainly focused on the fixed-job-priority, fixed-task-priority, and earliest-deadline-first. In the following, we only review the related work in uniprocessor system.

In [5], Vestal proposed to use Audsley's algorithm [7] to assign the priorities in MCSs with fixed-priority policy. Audsley's algorithm was proven in [6] to be optimal for assigning priorities to sporadic tasks with different criticality levels. In Vestal's approach, the priorities of tasks with different criticalities are allowed to be interleaved, leading all tasks to be evaluated as if they were of the highest criticality. By implementing the MCS in Ada, it was reported in [59] that higher resource usage can be achieved by monitoring task execution time and preventing execution time overruns. With such a platform that can monitor how long individual jobs have been executed, two new schemes called AMC-rtb and AMC-max that dominate Vestal's approach were proposed in [8, 29]. The AMC-max can schedule more tasks by discarding the low-criticality tasks when a high-criticality task overruns.

All the aforementioned work is based on the task model that is activated sporadically and the assumption that its WCRT is less than the minimum activation interval. With this assumption, there is at most one carry-over job (released, but not finished) when the system mode switch is invoked. The computation of WCRT and the derivation of demand bound function are often based on this carry-over job [8, 18, 29]. Without this assumption, the previous methods of computing WCRT and deriving demand bound function thus cannot be used. Therefore, in this chapter, new scheduling analysis is proposed for the arbitrarily activated tasks.

The sporadic task model was extended to include the release jitter in [6], based on which the sensitivity analysis was presented. However, there are fundamental differences between this analysis and the one in this chapter. First, the analysis in [6] does not consider the prevention of execution time over-runs. Second, although with a jitter, it still assumes that tasks are activated periodically, which is not enough to represent the arbitrarily activated tasks. Third, it assumes that tasks have constrained-deadlines that relative deadlines are not greater than their periods, while our schedulability tests do not need such assumption. Furthermore, the detailed influence of jitter on the system schedulability is not evaluated in [6].

Except the scheduling analysis in FP systems, there is also a lot of work in analyzing the schedulability in EDF systems. In [9,17], an approach called EDF-VD was proposed to schedule the implicit-deadline mixed-criticality sporadic tasks. By using smaller virtual deadlines, the system schedulability keeps unchanged across the system mode switch. In [18], based on the assumption that the system is schedulable in normal mode, the demand bound function towards the individual task was derived. Thus, a new approach called the Greedy Tuning was proposed to shape the demand bound function to comply with the supply of the computing platform by reducing virtual deadlines. This approach can schedule more tasks than the EDF-VD as clearly demonstrated in [60]. In [30], a new demand-based schedulability test and a new deadline tightening strategy were proposed by collectively bounding the demands of multiple tasks. In this chapter, to reduce the complexity, we focus on bounding the demands towards the individual task with arbitrary activations.

Despite of the unchanged period of sporadic tasks that are often modeled in MCSs, a task model in which the period differs among different criticality modes, instead of the WCET, was introduced in [9]. While this setting allows the period transformation, the sporadic task model is not changed in every mode. In [61, 62], by prioritizing all low-criticality tasks over high-criticality tasks in MCSs, two new monitoring approaches were proposed to monitor the workload of low-criticality tasks at runtime. Although the proposed monitoring approaches consider the arbitrarily activated tasks, there is no mode switch in their MCSs. Thus, it is still not clear how to monitor the workload in MCSs if a mode switch exists. In [63], a method was proposed for the scheduling analysis of adaptive multi-mode systems that supports any event stream model. By deriving the workload arrival curve of individual task over all modes, this method can only verify the schedulability of the task set whose priorities are already assigned. For the task set whose priorities are to be decided, the verification of schedulability is different.

3.3 System Model and Motivations

In this section, we present the task activation event model, our system settings, and a motivation example to show the inadequacy of the existing approaches.

3.3.1 Event Model

In the framework of system performance analysis [53–55], task activation is modeled by events, regardless whether time-triggered or event-triggered, periodic or sporadic. This way, task activations can be expressed as an event stream. A trace of such an event stream is described by means of an arrival function $R[s,t)$ that denotes the sum of events arrived in the time interval $[s,t)$, with $R[s,s) = 0$, $\forall s,t \in \mathbb{R}$. While any R always describes one concrete trace, a 2-tuple $\overline{\alpha}(\Delta) = [\overline{\alpha}^u(\Delta), \overline{\alpha}^l(\Delta)]$ of upper and lower arrival curve provides an abstract event stream model that represents the maximum and minimum number of events that are seen in a time interval.

Definition 2 (Arrival Curve [55])**.** *Denote $R[s,t)$ as the number of events that arrive on an event stream in the time interval $[s,t)$. Then, R, $\overline{\alpha}^u$ and $\overline{\alpha}^l$ represents the upper and lower bound on the number of event in any interval $t-s$, that is,*

$$\overline{\alpha}^l(t-s) \leq R[s,t) \leq \overline{\alpha}^u(t-s), \forall t \geq s \geq 0,$$

with $\overline{\alpha}^l(\Delta) \geq 0$, $\overline{\alpha}^u(\Delta) \geq 0$ for $\forall \Delta \in \mathbb{R}^{\geq 0}$.

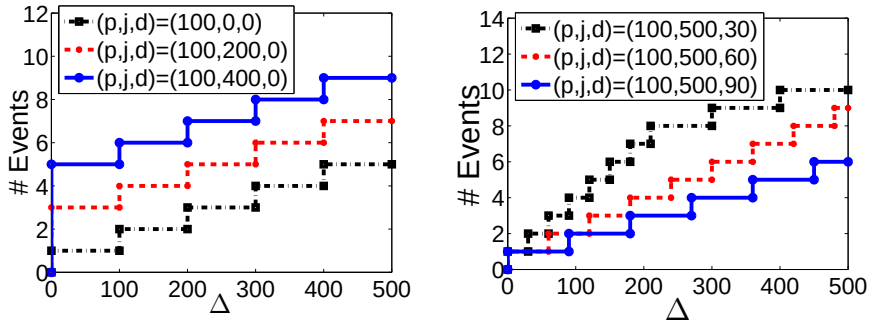
A similar concept corresponding to the upper arrival curve is the minimum distance function [54].

Definition 3 (Minimum Distance Function [54])**.** *The minimum distance function $\delta(q)$ is a pseudo super-additive[1] function, which returns a lower bound on the time interval between the first and the last event of any sequence of $q+1$ event occurrences.*

The minimum distance function is an inverse description of upper arrival curve. For example, $\delta(k) = \Delta_k$ denotes that, the first and the last event of any sequence of $k+1$ events is at least Δ_k time units apart, i.e., $\overline{\alpha}(\delta(k)) = k+1$.

The concept of arrival curve or minimum distance function substationally generalizes conventional event stream models, such as sporadic, periodic, periodic with jitter, and arbitrary event streams. For instance, for the arbitrary events modeled with the period p, the jitter j,

[1]For pseudo super-additive we denote the property of a function δ that $\forall a,b \in \mathbb{N}^+ : \delta(a+b) \geq \delta(a) + \delta(b)$. It corresponds to the property of "good" arrival functions in [46].



(a) with varying j (b) with varying d

Figure 3.1: The upper arrival curve of pjd event streams

and the minimum inter arrival distance d between successive two events, its upper arrival curve is

$$\overline{\alpha}^u(\Delta) = \min\{\lceil\frac{\Delta + j}{p}\rceil, \lceil\frac{\Delta}{d}\rceil\}. \tag{3.1}$$

This arrival pattern is called pjd pattern that is often used as a type of the complex arrival pattern in many previous works [55, 64]. Some properties of pjd are shown in Fig. 3.2, from which we find that: when the jitter j increases, the initial burst increases; when d increases, the arrival interval between any two events increases; when $d = p$, the event arrival pattern is sporadic.

Analogous to the arrival curve that provides an abstract event stream model, a tuple $\beta(\Delta) = [\beta^u(\Delta), \beta^l(\Delta)]$ of upper and lower *service curve* provides an abstract resource model.

Definition 4 (Service Curve [55])**.** *Denote $C[s,t)$ as the available resource in the time interval $[s,t)$. Then, C, β^u and β^l represents the upper and lower bound on the resource available in any interval $t-s$, that is,*

$$\beta^l(t-s) \leq C[s,t) \leq \beta^u(t-s), \forall t \geq s \geq 0,$$

with $\beta^l(\Delta) \geq 0$, $\beta^u(\Delta) \geq 0$ for $\forall \Delta \in \mathbb{R}^{\geq 0}$.

As an arrival curve $\overline{\alpha}_i$ specifies the event and a service curve β specifies the available processing time, the event arrival curve $\overline{\alpha}_i(\Delta)$ has to be transformed to the *workload arrival curve* α_i to indicate the amount of computation time required for the arrived events in any time interval Δ. Suppose that the WCET of an event stream is c_i. Then, the transformation can be done by $\alpha_i^u = c_i\overline{\alpha}_i^u$, $\alpha_i^l = c_i\overline{\alpha}_i^l$ and back by $\overline{\alpha}_i^u = \alpha_i^u/c_i$, $\overline{\alpha}_i^l = \alpha_i^l/c_i$.

3.3.2 System Settings

Except the setting for task activations, our system settings are the same as the system settings assumed in Chapter 2. Instead of the simply periodic or sporadic event stream, the task activations in this chapter are modeled as an arbitrary event stream. The event arrival curve is used to model the upper bound of the arbitrary event stream.

In general, in our settings, a dual-criticality task set $\tau = \{\tau_1, ..., \tau_n\}$ is given to be scheduled on a uniprocessor. All tasks are independent. Each task, τ_i, is defined by its upper event arrival curve $\overline{\alpha}_i^u$, relative deadline D_i, WCET $\mathbf{C_i}$ and criticality $\mathcal{X}_i$, where $\mathbf{C_i} = (C_i^L, C_i^H)$. The meaning of denotations and system behavior in LO and HI modes are the same as the descriptions in Chapter 2.

In this paper, two classic scheduling policies, i.e., the preemptive fixed-priority schedule and the earliest-deadline-first schedule, are studied in MCSs with the aforementioned settings. For the ease of expression in the sequel, we provide some short notations. In FP-scheduled systems, $hp(i)$ denotes the subset of all tasks with priorities higher than that of the task τ_i. $hpH(i)$ denotes the subset of HI-critical tasks with priorities higher than that of the task τ_i. $hpL(i)$ denotes the subset of LO-critical tasks with priorities higher than that of the task τ_i. In EDF-scheduled systems, we denote the subset of all LO-critical tasks in τ as $\tau^L = \{\tau_i \in \tau | \mathcal{X}_i = LO\}$, and the subset of all HI-critical tasks in τ as $\tau^H = \{\tau_i \in \tau | \mathcal{X}_i = HI\}$.

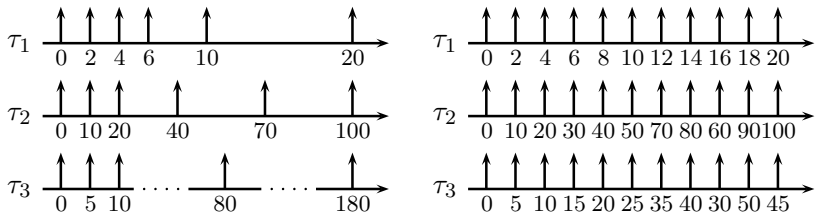
3.3.3 Motivation Example

An arbitrary task activation pattern that is modeled as the arrival curve can also be represented by the sporadic pattern. Since the sporadic pattern defines a minimum inter-activation interval, an arbitrary activation pattern can be represented by the sporadic pattern by defining a minimum inter-activation interval. However, this representation is pessimistic because it admits more events than the maximum number of events that will actually arrive.

Example 2. *In a uniprocessor system, there are three tasks, as shown in the following (task activations are set as pjd patterns, whose upper arrival curve $\overline{\alpha}_i^u$ is presented in Eq. 3.1):*

τ_i	L_i	C_i^L	C_i^H	D_i	$\overline{\alpha}_i^u\,(p, j, d)$
τ_1	LO	3	-	7	(10, 30, 2)
τ_2	HI	5	10	35	(30, 50, 10)
τ_3	HI	20	40	300	(100,220,5)

The as-early-as-possible event trace of $\overline{\alpha}_i^u$ is shown in Fig. 3.2(a). In order to apply the existing approaches to schedule this task set in MCS*s, those event traces should be modeled as*



(a) arrival curve model (b) sporadic model

Figure 3.2: The as-early-as-possible event trace of two different models

the sporadic tasks. Since sporadic model only defines a period, the minimum distance between two activations of a task is used as the period. The as-early-as-possible event traces of sporadic model is shown in Fig. 3.2(b). It is impossible to schedule this task set if events arrive as the assumption of sporadic model because the LO WCET *of* τ_1 *is larger than the period of its sporadic model. However, in real situations, the events will not arrive as frequently as the sporadic model assumes. By the sufficient busy-window schedulability test presented in Section 3.4.3.2, we find this task set is actually schedulable.*

3.4 Fixed Priority Schedulability Test

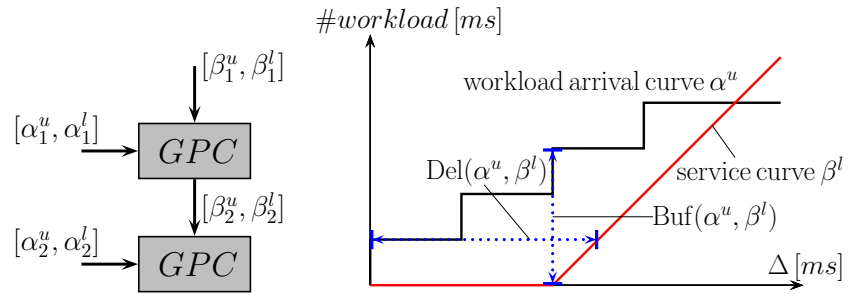
In this section, we present how to test the system schedulability with multi-criticality arbitrarily activated tasks in FP-scheduled systems.

3.4.1 Preliminaries

In this section, we introduce the well-established Modular Performance Analysis under the framework of Real-Time Calculus [44, 48] and the known Audsley's algorithm, which are the basis of the necessary and sufficient tests for verifying the schedulability of a given task set.

3.4.1.1 Modular Performance Analysis

In the framework of Real-Time Calculus, the task processing is often modeled by abstract performance component that acts as curve transformer in the domain of arrival and service curve, where the transferring function depends on the modeled processing semantics. The Greedy Processing Component (GPC) models a task that is triggered by the events which are queued up in the FIFO (first-in-first-out) buffer. A typical case is shown in Fig. 3.3(a), where $[\alpha_i^u, \alpha_i^l]$ and $[\beta_i^u, \beta_i^l]$ are respectively the workload arrival curve and resource service curve. The processing of two tasks in a preemptive fixed-priority scheduling system is abstracted as two



(a) System abstraction of two tasks

(b) Computing $\mathrm{Del}(\alpha^u, \beta^l)$ and $\mathrm{Buf}(\alpha^u, \beta^l)$

Figure 3.3: Modular performance analysis

GPCs. The lower service for the lower-priority task is the processing service left over after processing the higher-priority task:

$$\beta_2^l(\Delta) \stackrel{\text{def}}{=} \sup_{0 \le \lambda \le \Delta} \{\beta_1^l(\lambda) - \alpha_1^u(\lambda)\}. \tag{3.2}$$

As shown in Fig. 3.3(b), with the provided lower service β^l and upper arrival workload α^u, the WCRT and the maximum workload of backlogged events for an event stream processed at a GPC can be computed as follows [55]:

$$\mathrm{Del}(\alpha^u, \beta^l) \stackrel{\text{def}}{=} \sup_{\lambda \ge 0} \left\{ \inf\{\tau \ge 0 : \alpha^u(\lambda) \le \beta^l(\lambda + \tau)\} \right\}, \tag{3.3a}$$

$$\mathrm{Buf}(\alpha^u, \beta^l) \stackrel{\text{def}}{=} \sup_{\lambda \ge 0} \{\alpha^u(\lambda) - \beta^l(\lambda)\}. \tag{3.3b}$$

Note that, by applying Eqs. 3.3a, 3.3b, we get an upper bound of the WCRT, and the maximum backlogged workload. Then, to get the maximum number n_{max} of backlogged events, the WCET c of this task should be considered, i.e., $n_{max} = \lceil \mathrm{Buf}(\alpha^u, \beta^l)/c \rceil$.

Regarding the system of two tasks as shown in Fig. 3.3(a), assume the deadlines for the two tasks are D_1 and D_2, this task set can be schedulable if and only if $\mathrm{Del}(\alpha_1^u, \beta_1^l) \le D_1$ and $\mathrm{Del}(\alpha_2^u, \beta_2^l) \le D_2$.

3.4.1.2 Audsley's Algorithm

Audsley's approach was proven by Dorin et al. in [6] as an optimal algorithm to assign the task priorities in MCSs. Audsley's algorithm starts with no task being assigned a priority. Priorities are assigned from the lowest to the highest, so that, at each step, a task that can be assigned with the lowest priority is selected out. Once a task is selected out, it is removed from the unassigned priority tasks, and Audsley's algorithm continues to assign the priority to the next

task. Audsley's algorithm fails if there is no task that can be assigned with the lowest priority. The condition of using Audsley's algorithm to assign priorities [65] is that: the WCRT for a task τ_i can be determined by knowing which subset of tasks has higher priority than τ_i but without otherwise knowing what their specific priority assignments are.

Audsley's algorithm delivers an optimum priority assignment in a maximum of $n(n+1)/2$ steps. If Audsley's algorithm is not applicable, i.e., the above condition is not satisfied, the worst case for assigning priorities is to search over all $n!$ possible priority orderings.

3.4.2 A Necessary Test - NEC

This section presents the necessary test for verifying the schedulability of MCS scheduled by fixed priority. The necessary test is based on the fact that a schedulable system should be able to schedule any event traces that comply with the arrival curves. Hence, we set up two necessary conditions that should hold for the system being schedulable. Audsley's algorithm is applied to search the priority assignment that makes the two conditions hold.

3.4.2.1 Two Necessary Conditions

Suppose that there are n tasks in an MCS, as shown in Fig. 3.4(a). We suppose two special situations, and the system should be schedulable under the following two situations.

- Condition A: Suppose when the system is in LO mode, event traces might occur in the worst-case patterns.
- Condition B: Suppose the HI-critical event that appears the earliest runs over its LO WCET, at this time the system is switched to HI mode where LO-critical tasks are aborted; after the system enters into HI mode, HI-critical event traces might occur in the worst-case patterns.

When the MCS is in LO mode, no event will be dropped and the estimate of execution time will not change. The MCS in LO mode behaves as a non-MCS. Therefore, the Modular Performance Analysis in Section 3.4.1.1 can be used to verify the Condition A. For the Condition B, since the first HI-critical event triggers the mode-switch, there will be no backlogged HI-critical events when the system enters into HI mode. In this situation, the MCS in HI mode can also be considered as a non-MCS on which only HI-critical tasks run. The Modular Performance Analysis is also used for verifying the condition B.

3.4.2.2 Test by Applying Audsley's Algorithm

In many previous works [48,55,64], the Modular Performance Analyis is only used for analyzing the system with specific priority order. However, in order to apply Audsley's algorithm, the priority order of other tasks should have no effect on the WCRT of the lowest-priority task. Theorem 6 guarantees that the priority order has no effect on computing the WCRT of the lowest priority task, thus making Audsley's algorithm and the Modular Performance Analysis compatible for verifying the two necessary conditions.

In a system, suppose there are n tasks whose priorities need to be assigned so that all tasks can be schedulable, as shown in Fig. 3.4(a). For this system, we have the following theorem.

Theorem 6. *If the task τ_n is assigned the lowest priority without knowing the priority ordering of other tasks, the system can be abstracted as Fig. 3.4(b), where β_1^l is the lower service curve of the processor and other terms are the same as Section 3.3.2. When the system stays in LO mode, the lower service β_n^{lLO} for the task τ_n is bounded by*

$$\beta_n^{lLO}(\Delta) \overset{\text{def}}{=} \sup_{0\leq\lambda\leq\Delta}\left\{\beta_1^l(\lambda) - \sum_{j\in hp(n)} \overline{\alpha}_j^u(\lambda)\cdot C_j^L\right\}. \tag{3.4}$$

When the system is in HI mode, with solely HI-critical tasks executing C_i^H and no backlogged HI-critical events, the lower service β_n^{lHI} for the task τ_n is bounded by:

$$\beta_n^{lHI}(\Delta) \overset{\text{def}}{=} \sup_{0\leq\lambda\leq\Delta}\left\{\beta_1^l(\lambda) - \sum_{j\in hpH(n)} \overline{\alpha}_j^u(\lambda)\cdot C_j^H\right\}. \tag{3.5}$$

Proof. Without the loss of generality, the priority for the task τ_i is ordered in a descending order, i.e., the priority of τ_i is greater than the priority of τ_j if $i < j$.

When the system is in LO mode, by iteratively using Eq. 3.2, we have

$$\beta_{i+1}^l(\Delta) = \sup_{0\leq\lambda\leq\Delta}\{\beta_i^l(\lambda) - \overline{\alpha}_i^u(\lambda)\cdot C_i^L\}, \forall\ i \leq n-1,$$

where β_i^l is the lower service provided to the task τ_i. As

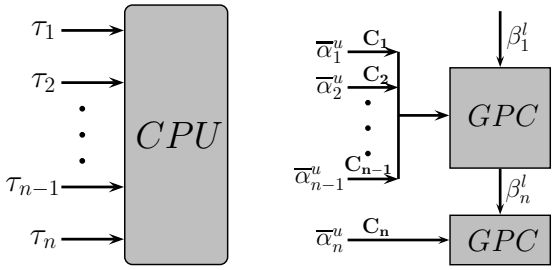
$$\beta_i^l(\Delta) = \sup_{0\leq\lambda\leq\Delta}\{\beta_{i-1}^l(\lambda) - \overline{\alpha}_{i-1}^u(\lambda)\cdot C_{i-1}^L\}, \forall\ i \leq n-1,$$

we have

$$\beta_{i+1}^l(\Delta) = \sup_{0\leq\lambda\leq\Delta}\{\beta_i^l(\lambda) - \overline{\alpha}_i^u(\lambda)\cdot C_i^L\} = \sup_{0\leq\lambda\leq\Delta}\left\{\sup_{0\leq\lambda'\leq\lambda}\{\beta_{i-1}^l(\lambda') - \overline{\alpha}_{i-1}^u(\lambda')\cdot C_{i-1}^L\} - \overline{\alpha}_i^u(\lambda)\cdot C_i^L\right\}.$$

As $\overline{\alpha}_i^u(\lambda) \geq \overline{\alpha}_i^u(\lambda')$, we have

$$\begin{aligned}
\beta_{i+1}^l(\Delta) &\leq \sup_{0\leq\lambda\leq\Delta}\left\{\sup_{0\leq\lambda'\leq\lambda}\{\beta_{i-1}^l(\lambda') - \overline{\alpha}_{i-1}^u(\lambda')\cdot C_{i-1}^L\} - \overline{\alpha}_i^u(\lambda')\cdot C_i^L\right\} \\
&= \sup_{0\leq\lambda\leq\Delta}\left\{\sup_{0\leq\lambda'\leq\lambda}\left\{\beta_{i-1}^l(\lambda') - \sum_{j=i-1}^{i}\overline{\alpha}_j^u(\lambda')\cdot C_j^L\right\}\right\} \\
&= \sup_{0\leq\lambda\leq\Delta}\left\{\beta_{i-1}^l(\lambda) - \sum_{j=i-1}^{i}\overline{\alpha}_j^u(\lambda)\cdot C_j^L\right\}, \forall\ i \leq n-1.
\end{aligned}$$



(a) System model (b) System abstraction

Figure 3.4: A mixed-criticality system with n tasks

Besides, as $\sup_{0\le\lambda'\le\lambda}\{f(\lambda')\} \ge f(\lambda)$, we have

$$\begin{aligned}\beta_{i+1}^l(\Delta) &\ge \sup_{0\le\lambda\le\Delta}\left\{\beta_{i-1}^l(\lambda) - \overline{\alpha}_{i-1}^u(\lambda)\cdot C_{i-1}^L - \overline{\alpha}_i^u(\lambda)\cdot C_i^L\right\} \\ &= \sup_{0\le\lambda\le\Delta}\left\{\beta_{i-1}^l(\lambda) - \sum_{j=i-1}^{i}\overline{\alpha}_j^u(\lambda)\cdot C_j^L\right\}, \forall\ i \le n-1.\end{aligned}$$

Therefore, we have

$$\beta_{i+1}^l(\Delta) = \sup_{0\le\lambda\le\Delta}\left\{\beta_{i-1}^l(\lambda) - \sum_{j=i-1}^{i}\overline{\alpha}_j^u(\lambda)\cdot C_j^L\right\}. \tag{3.6}$$

Similarly, the index i in Eq. 3.6 can be extended from $n-1$ to 1, thus

$$\beta_n^{lLO}(\Delta) \stackrel{\text{def}}{=} \sup_{0\le\lambda\le\Delta}\left\{\beta_1^l(\lambda) - \sum_{j\in hp(n)}\overline{\alpha}_j^u(\lambda)\cdot C_j^L\right\}. \tag{3.7}$$

For the other priority settings, the lower service curve of Eq. 3.7 is unchanged, as long as the task τ_n is set with the lowest priority. Thus, the theorem holds.

Eq. 3.5 can also be proved in the same steps as proving Eq. 3.4. □

With β_n^{lLO} and β_n^{lHI}, Condition A and Condition B can be verified by checking

$$\mathrm{Del}(\alpha_n^u(LO), \beta_n^{lLO}) \le D_n \tag{3.8a}$$

$$\mathrm{Del}(\alpha_n^u(HI), \beta_n^{lHI}) \le D_n, \tag{3.8b}$$

where $\alpha_n^u(LO) = \overline{\alpha}_n^u \cdot C_n^L$ and $\alpha_n^u(HI) = \overline{\alpha}_n^u \cdot C_n^H$. If the task that is assigned the lowest priority is a LO-critical task, only Eq. 3.8a needs to be verified because the WCRT of this task does not need to be certified in HI mode. For the HI-critical task, Eqs. 3.8a, 3.8b need to be verified.

Audsley's algorithm searches the available task that can be assigned the lowest priority by checking Eqs. 3.8a, 3.8b. If Eqs. 3.8a, 3.8b hold, this task is selected out with the assigned priority and Audsley's algorithm continues to assign priorities to the left tasks. If not, Audsley's algorithm will check Eqs. 3.8a, 3.8b by assigning the lowest priority to another task. If no task can be assigned the lowest priority, the necessary test fails and this task set is not schedulable.

Example 3. *Returning to our motivation example, by using the necessary test, the task* τ_1 *can only be assigned with the highest priority. Its* WCRT *is 6. The task* τ_2 *can only be assigned with the second priority. Its* WCRT *is 20 in LO mode and 10 in HI mode. The task* τ_3 *is assigned with the lowest priority. Its* WCRT *is 139 in LO mode and 200 in HI mode. This task set passes the necessary test.*

3.4.3 Two Sufficient Tests

The necessary test can only guarantee that a task set is not schedulable if the necessary test fails, and cannot guarantee that a task set is schedulable if the necessary test does not fail. In this section, we present two sufficient tests towards the arbitrary activated tasks. The task set that succeeds with the sufficient test is schedulable.

The idea of sufficient tests is to verify whether the upper bound of a task response time is smaller than this task's relative deadline. If so, this task is deemed schedulable. The task is classified to be the LO-critical task or the HI-critical task. Since the LO-critical task only runs in LO mode and the MCS in LO mode can be considered as a non-MCS, the response time is bounded by $\mathrm{Del}(\alpha_n^u, \beta_n^{lLO})$ in Eq. 3.8a. Hence, the Eq. 3.8a is also the sufficient verification for LO-critical tasks. For the HI-critical task, however, computing the upper bound of response time is not so straightforward.

In the following, we present two approaches to compute the upper bound of response time of HI-critical tasks. The first approach is called the workload arrival curve approach. Suppose a task is set with the lowest priority. By deriving the workload arrival curve of all higher-priority tasks in both LO and HI modes, the lower bound of provided service to the lowest-priority task is derived. Then, the response time of the lowest-priority task can be bounded by applying the Modular Performance Analysis. The second approach is to apply the busy-window analysis to compute the upper bound of response time. In non-MCS systems, the busy-window analysis allows to calculate an upper bound on the time interval the processor is busy processing a task τ_i and its interferences $hp(i)$. We extend the busy-window analysis to MCSs to analyze the upper bound of response time of a HI-critical task.

3.4.3.1 Workload Arrival Curve Analysis - WAC

The idea of this approach is to derive a workload arrival curve $\alpha_{hp(i)}^u$ that upper bounds the workload of all tasks with higher priorities than the HI-critical task τ_i in both modes, including a mode switch. Hence, the remaining service for the task τ_i can be safely bounded by using Eq. 3.2, and the WCRT can be computed by using Eq. 3.3a.

The system starts in LO mode. Before the mode switch, the WCETs of all tasks are assumed to be C_j^L. Then, the workload arrival curve of tasks with higher priorities than τ_i is that

$$\alpha_{hp(i)}^{LO} \stackrel{\text{def}}{=} \sum_{j \in hp(i)} \overline{\alpha}_j^u \cdot C_j^L.$$

Assume for the task τ_j, there are Buf_j events that are backlogged when the mode switch is triggered. Then, in HI mode, as the LO-critical tasks are not executed and the WCETs for HI-critical tasks are assumed to be C_j^H, we have

$$\alpha_{hp(i)}^{HI} \stackrel{\text{def}}{=} \sum_{j \in hpH(i)} (\overline{\alpha}_j^u + \text{Buf}_j) \cdot C_j^H. \tag{3.9}$$

To safely bound $\alpha_{hp(i)}^{HI}$, the maximum number of events $\text{Buf}_j^{\max}$ that can be backlogged in LO mode is used. The computation of $\text{Buf}_j^{\max}$ indicates the case that the task τ_j receives the interference from all the other tasks in $hp(i)$, i.e., the task τ_j is set with the lowest priority in $hp(i)$. For each task in $hp(i)$, by setting its priority as the lowest in $hp(i)$, $\text{Buf}_j^{\max}$ can be computed by using Eqs. 3.3b, 3.4, i.e.,

$$\text{Buf}_j^{\max} = \left\lceil \frac{\text{Buf}\big(\alpha_j^u(LO), \beta_j^{lLO}\big)}{C_j^L} \right\rceil. \tag{3.10}$$

Since $\text{Buf}_j^{\max}$ is rounded up, the released but not finished event at the mode switch is also included in $\text{Buf}_j^{\max}$. As the backlogged events cannot be over $\text{Buf}_j^{\max}$ and the released workload in HI mode cannot be over $\overline{\alpha}_j^u \cdot C_j^H$, $\alpha_{hp(i)}^{HI}$ in Eq. 3.9 is an upper bound of the workload in HI mode.

To get a workload arrival curve that upper bounds the workload of $hp(i)$ in both modes, the following theorem can be used.

Theorem 7. *For an* MCS *with a setting described in Section 3.3.2. The workload arrival curve that upper bounds the workload of $hp(i)$ in both modes can be computed by the following equation:*

$$\alpha_{hp(i)}^u(\Delta) \stackrel{\text{def}}{=} \sup_{0 \le \lambda \le \Delta} \Big\{ \sum_{j \in hp(i)} \overline{\alpha}_j^u(\Delta - \lambda) \cdot C_j^L + \sum_{j \in hpH(i)} (\overline{\alpha}_j^u(\lambda) + \text{Buf}_j^{\max}) \cdot C_j^H \Big\}. \tag{3.11}$$

Proof. We consider a time interval $[s, t)$ with $t - s = \Delta$, and set t_c as the time of mode switch. There are three possibilities, i.e.,

1. $t \le t_c$, the system stays in LO mode.
2. $s \ge t_c$, the system stays in HI mode.
3. $s < t_c < t$, the system travels from LO mode to HI mode.

Since $\alpha^{LO}_{hp(i)} \leq \alpha^{u}_{hp(i)}$ and $\alpha^{HI}_{hp(i)} \leq \alpha^{u}_{hp(i)}$, Eq. 3.11 holds for the case 1 and the case 2.

For the case 3, we set $aw[s,t)$ the arrival workload in the interval of $[s,t)$. We have

$$\begin{aligned} aw[s,t) &= aw[s,t_c) + aw[t_c,t) \\ &\quad \text{Subst. } \lambda = t - t_c \\ &= aw[s, s+\Delta-\lambda) + aw[s+\Delta-\lambda, s+\Delta) \\ &\leq \alpha^{LO}_{hp(i)}(\Delta-\lambda) + \alpha^{HI}_{hp(i)}(\lambda) \\ &\leq \alpha^{u}_{hp(i)}(\Delta) \end{aligned}$$

Hence, the bound computed by Eq. 3.11 safely bounds the workload arrival curve in both modes. □

With the $\alpha^{u}_{hp(i)}$, the lower bound of remaining service for the HI-critical task τ_i can be computed with Eq. 3.2, i.e.,

$$\beta^{l}_{i}(\Delta) \overset{\text{def}}{=} \sup_{0 \leq \lambda \leq \Delta} \left\{ \beta^{l}_{1}(\lambda) - \alpha^{u}_{hp(i)}(\lambda) \right\}.$$

Then, to compute the WCRT of τ_i, every job of the HI-critical task τ_i is assumed to be executed with the C^H_i. This assumption sufficiently bounds the workload of this task. Thus, the workload arrival curve of τ_i is $\alpha^{u}_{i} = \overline{\alpha}^{u}_{i} \cdot C^H_i$. This HI-critical task τ_i can be scheduled if $\text{Del}(\alpha^{u}_{i}, \beta^{l}_{i}) \leq D_i$.

Since there is no assumption in the priority ordering of the task set $hp(i)$ in this approach, the condition of applying Audsley's algorithm is satisfied. Audsley's algorithm is applicable for this approach.

Example 4. *Returning to the motivation example, by deriving the workload arrival curve of* $hp(3)$ *of applying Eq. 3.11, the* WCRT *of* τ_3 *is 338. By deriving the workload arrival curve of* $hp(1)$*, which is* $\overline{\alpha}_1 \cdot C^L_1$*, the* WCRT *of* τ_2 *is 37. This task set cannot be scheduled.*

3.4.3.2 Busy-Window Analysis - BW

In the non-MCS, the busy-window analysis allows to calculate an upper bound on the time interval that the processor is busy processing a task τ_i and its interferences from $hp(i)$ [66, 67]. Based on the busy-window, one can calculate an upper bound on a task's WCRT. In this section, we present a method that can sufficiently bound the response time of a HI-critical task in the MCS by applying the busy-window analysis. It first computes the maximum time to process any q HI-critical events of a task, based on which the upper bound of this task's WCRT is calculated.

Similar to the busy-window formulation of equation 3 in [67], the multi-event busy-window $B_i(q, \delta_i)$ [1] is defined.

[1] δ_i is the minimum distance function of the HI-critical task τ_i

Definition 5 (Multi-Event Busy-Window)**.** *Assuming the processor is initially idle, the multi-event busy-window $B_i(q, \delta_i)$ describes an upper bound on the amount of time that a resource requires to serve q activations of the HI-critical task τ_i in* MCS*s.*

During processing q activations of the HI-critical task τ_i, there can be two cases:

- case 1: the MCS always stays in LO mode.
- case 2: the MCS transits from LO mode to HI mode, or completely stays in HI mode.

For the case 1, the multi-event busy-window is denoted as $B_i^{LO}(q, \delta_i)$, which can be obtained by calculating the following formula until convergence [67, 68].

$$B_i^{LO}(q, \delta_i) \stackrel{\text{def}}{=} q \cdot C_i^L + \sum_{j \in hp(i)} \overline{\alpha}_j(B_i^{LO}(q, \delta_i)) \cdot C_j^L. \tag{3.12}$$

For the case 2, we define s as the time that the mode switch is triggered. s is restricted in the interval $[0,\ B_i^{LO}(q, \delta_i))$, because all q events would have been finished before the mode switch if $s \geq B_i^{LO}(q, \delta_i)$. If $s = 0$, it means that the MCS completely stays in HI mode. Denote $B_i^s(q, \delta_i)$ as the multi-event busy-window that the mode switch is triggered at s. Compared with $B_i^{LO}(q, \delta_i)$ in Eq. 3.12, the computation of $B_i^s(q, \delta_i)$ should separately consider LO-critical tasks and HI-critical tasks because LO-critical tasks can interfere τ_i only in LO mode. Hence, we formulate $B_i^s(q, \delta_i)$ as follows:

$$B_i^s(q, \delta_i) \stackrel{\text{def}}{=} q \cdot C_i^H + I_L(s) + I_H\big(s, B_i^s(q, \delta_i)\big), \tag{3.13}$$

where $I_L(s)$ refers to the maximum interference from $hpL(i)$ in the interval $[0, s)$, and $I_H(s, B_i^s(q, \delta_i))$ refers to the maximum interference from $hpH(i)$ in the interval $[0, B_i^s(q, \delta_i))$.

As LO-critical tasks are prevented from executing after s, the maximum interference from $hpL(i)$ is bounded by:

$$I_L(s) \stackrel{\text{def}}{=} \sum_{j \in hpL(i)} \overline{\alpha}_j(s) \cdot C_j^L. \tag{3.14}$$

Regarding to the computation of $I_H(s, B_i^s(q, \delta_i))$, we first compute the maximum interference of every task in $hpH(i)$, and accumulate them together. Consider a specific task $\tau_k \in hpH(i)$ and use $I_k(s, t)$ to denote the maximum interference of this task. The maximum number of events within $[0, t)$ is $\overline{\alpha}_k(t)$. Suppose C_k^H is the workload due to the release of m events in $[s, t)$ and $\overline{\alpha}_k(t) - m$ events are executed by C_k^L. Hence,

$$I_k(s, t) = m \cdot C_k^H + (\overline{\alpha}_k(t) - m) \cdot C_k^L.$$

To maximize $I_k(s, t)$, m should be as large as possible because $C_k^H \geq C_k^L$. There are two constraints on m. First, m should be less than the maximum number of arrival events during

$[0, t)$, i.e., $m \leq \overline{\alpha}_k(t)$. Second, m should be also less than the sum of backlogged events at time s and arrival events during $[s, t)$, i.e., $m \leq \overline{\alpha}_k(t-s) + \mathrm{Buf}_k(s)$, where $\mathrm{Buf}_k(s)$ is the maximum backlogged events at time s. We use $X_k(s,t)$ to denote the maximum m and $Y_k(s,t)$ to denote the number of events that are executed by C_k^L. Therefore,

$$X_k(s,t) \stackrel{\text{def}}{=} \min\left\{\mathrm{Buf}_k(s) + \overline{\alpha}_k(t-s),\ \overline{\alpha}_k(t)\right\}, \tag{3.15a}$$

$$Y_k(s,t) \stackrel{\text{def}}{=} \overline{\alpha}_k(t) - X_k(s,t). \tag{3.15b}$$

Exactly computing the $\mathrm{Buf}_k(s)$ is difficult as $\mathrm{Buf}_k(s)$ depends on the specific event arrivals of τ_k and $hp(k)$. Here, we provide an upper bound, i.e.,

$$\mathrm{Buf}_k(s) \stackrel{\text{def}}{=} \min\{\overline{\alpha}_k(s), \mathrm{Buf}_k^{max}\}, \tag{3.16}$$

where $\mathrm{Buf}_k^{\max}$ is computed with Eqs. 3.3b, 3.4 by setting τ_k with the lowest priority in $hp(i)$, which is the same as the Eq. 3.10 in deriving the workload arrival curve. Note that $X_k(s,t)$ is a valid upper bound on the events executed in $[s,t)$, but $Y_k(s,t)$ is neither an upper bound nor a lower bound. Nevertheless, the computed $I_k(s,t)$ with $X_k(s,t)$ and $Y_k(s,t)$ is an upper bound. Then, the maximum interference $I_H(s,t)$ is

$$I_H(s,t) \stackrel{\text{def}}{=} \sum_{k \in hpH(i)} \left\{X_k(s,t) \cdot C_k^H + Y_k(s,t) \cdot C_k^L\right\}. \tag{3.17}$$

With $I_L(s)$ and $I_H(s, B_i^s(q,\delta_i))$, $B_i^s(q,\delta_i)$ can be computed by iteration. Then, the $B_i(q,\delta_i)$ is the maximum $B_i^s(q,\delta_i)$ over all possible s, i.e.,

$$B_i(q,\delta_i) \stackrel{\text{def}}{=} \max(B_i^s(q,\delta_i)) \forall s,\ s \in [0, B_i^{LO}(q,\delta_i)).$$

To compute $B_i(q,\delta_i)$, s should be scanned. But it is not necessary to scan every s within $[0, B_i^{LO}(q,\delta_i))$. After an examination over $I_L(s)$ and $I_H(s,t)$, we find that only the points at which $\overline{\alpha}_j(s),\ \forall j \in hp(i)$ changes need to be checked.

Proposition 4. *$B_i^s(q,\delta_i)$ can only increase at the points where $\overline{\alpha}_j(s)$ changes $\forall j \in hp(i)$.*

Proof. Suppose s_1 and s_2 are two successive points that $\overline{\alpha}_j(s)$ changes $\forall j \in hp(i)$, and $s_1 < s_2$. Since $\overline{\alpha}_j(s)$ does not change within (s_1, s_2), $I_L(s)$ will not change within (s_1, s_2). When s increases from s_1 to s_2, from Eqs. 3.15a, 3.16, it can be known that $X_k(s,t)$ may decrease. Hence, $I_k(s,t)$ may also decrease, which leads to the decrease of $I_H(s,t)$. Therefore, $B_i^s(q,\delta_i)$ will also decrease when s increases from s_1 to s_2. We have $B_i^s(q,\delta_i) \leq B_i^{s_1}(q,\delta_i),\ \forall\ s \in (s_1, s_2)$. This means that $B_i^s(q,\delta_i)$ can only increase at the points where $\overline{\alpha}_j(s)$ changes $\forall j \in hp(i)$. □

Since within any two successive points, $B_i^s(q,\delta_i)$ will become smaller. To get $B_i(q,\delta_i)$, only the points at which $\overline{\alpha}_j(s),\ \forall j \in hp(i)$ changes need to be checked. [1]

[1] If $hp(i)$ is empty, i.e., task τ_i is set as the highest priority, s should be 0. This is because the WCRT will be the largest if every event of this task is executed with a large WCET estimation.

With $B_i(q, \delta_i)$, we know the WCRT $R_i(q)$ of the q-th job is bounded by

$$R_i(q) \stackrel{\text{def}}{=} B_i(q, \delta_i) - \delta_i(q-1), \tag{3.18}$$

where $\delta_i(0)$ is set to be 0.

The computation of multi-event busy-window $B_i(q, \delta_i)$ assumes that all q events arrive earlier than the completion of their prior jobs (the $(q-1)$-event busy-time), i.e.,

$$\delta_i(q-1) \leq B_i(q-1, \delta_i).$$

We denote the maximum number of events that can be in a multi-event busy-window as Q_i, where Q_i is the last event that arrives earlier than the completion of its prior job, i.e.,

$$Q_i \stackrel{\text{def}}{=} \max\big(n : \forall q \in \mathbb{N}^+, q \leq n : \delta_i(q) \leq B_i(q, \delta_i)\big).$$

Then, the WCRT R_i of the task τ_i can be found among the Q_i events, i.e.,

$$R_i \stackrel{\text{def}}{=} \max_{q \in [1, Q_i]} (R_i(q)).$$

The task τ_i can be scheduled if and only if $R_i \leq D_i$. As R_i can be determined by knowing $hp(i)$ and without knowing their specific priority assignments, Audsley's algorithm can be used to check the schedulability of a task set by this approach.

Example 5. *We now use the task set in the motivation example as a running example to explain the procedures of using busy-window analysis. It can be observed that only τ_3 is possible to be assigned with the lowest priority. Suppose τ_3 is set with the lowest priority, the other higher priority and HI-critical task is τ_2. By Eq. 3.10, we get that the maximum number of backlogged events of τ_2 in LO mode is 2, i.e.,* $\text{Buf}_2^{max} = 2$. *The* WCRT *of τ_3 is computed by applying the following rounds.*

In the first round, only one event of τ_3 is considered, i.e., $q = 1$. We first compute the busy-window $B_3(1, \delta_3)$ in LO mode. By Eq. 3.12, $B_3(1, \delta_3) = 78$. According to Proposition 4, only the points in $[0, 78)$ where $\overline{\alpha}_i^u$, $i \in 1, 2$ change need to be checked. We use s^* to denote those points. Then, for every s^*, $I_L(s^*)$ of Eq. 3.14 is computed. With $\text{Buf}_2^{max} = 2$ *and by successively applying Eqs. 3.16, 3.15a, 3.15b, 3.17, 3.13, $B_3^{s^*}(1, \delta_3)$ can be computed. The maximum $B_3^{s^*}(1, \delta_3)$ among all s^* is picked out. We get that $B_3^s(1, \delta_3) = \max(B_3^{s^*}(1, \delta_3)) = 140$. Since $\delta_3(0) = 0$, the* WCRT *$R_3(1)$ of one event is 140.*

In the second round, two events of of τ_3 are considered, i.e., $q = 2$. By the same computing steps as the first round, we get that $B_3^s(2, \delta_3) = 207$. As $\delta_3(1) = 5$, the WCRT *$R_3(2)$ is that $R_3(2) = B_3^s(2, \delta_3) - \delta_3(1) = 202$.*

We continue to increase q by one in every round and compute $R_3(q)$ in every round, until we find that $\delta_i(q-1) > B_i(q-1, \delta_i)$. In the motivation example, we find that, when $q = 10$, $B_3^s(10, \delta_3) = 747$. The earliest arrival time of the 11-th event is 780, which is greater than 747. It indicates that, the workload of first 10 events has no effect on the WCRT *of 11-th event.*

The 11-th event can be reconsidered as $q = 1$. Therefore, the maximum $R_3(q)$ where $q \leq 10$ represents the WCRT *of the task τ_3. In the motivation example, the* WCRT *of τ_3 is 261. Therefore, we testify that τ_3 is schedulable by setting τ_3 with the lowest priority.*

The task τ_3 with the lowest priority can now be removed from this taskset. We continue to assign priorities to τ_1 and τ_2. For the HI-critical task τ_2, we follow the same computation procedures as computing the WCRT *of τ_3 to obtain the* WCRT *of τ_2. For the LO-critical task, we only need to apply Eq. 3.8a to get its* WCRT *because LO-critical tasks are not processed in HI mode. Overall, we find that, τ_1 with the highest priority and τ_2 with the second priority are schedulable.*

3.4.3.3 Comparing WAC and BW

Complexity The computational overhead related to our schedulability tests can be attributed to two parts, i.e., the expense for searching feasible priority assignment, and the expense for verifying the schedulability of the task being assigned the lowest priority. Since WAC and BW apply Audsley's algorithm to search the feasible priority assignment, the overhead of the first part is the same for both approaches.

The computational overhead mainly depends on the second part, i.e., verifying whether a task can be assigned the lowest priority. If this task is a LO-critical task, WAC and BW use the same method as verifying Condition A of the necessary test. If this task is a HI-critical task, the complexities of WAC and BW are different. There are four steps of applying WAC, which are, computing the maximum backlogged events $\text{Buf}_j^{\max}$ for every task in $hp(i)$, computing the workload arrival curve $\alpha^u_{hp(i)}$ that upper bounds the workload of $hp(i)$, computing the lower bound of provided service β^l_i, and computing the WCRT by $\text{Del}(\alpha^u_i, \beta^l_i)$. For applying BW, there are two steps, which are, computing $\text{Buf}_j^{\max}$ of every task in $hp(i)$, and computing the response time of every event within a maximum busy-window. Both WAC and BW are the same in first step, but are different in other steps.

Among the different steps, the computational expense of WAC mainly depends on the operations of curves, for instance, the max-convolution used in deriving the $\alpha^u_{hp(i)}$. If the two curves are periodic, the computation expense depends on the least common multiple of the two periods. If they are aperiodic, the computational expense depends on the number of aperiodic segments. The computational expense of BW mainly depends on computing the WCRT of every event within a busy-window. Specifically, it depends on how large a busy-window is, how many events could be in the busy-window, and how many changes are there of $\overline{\alpha}_j(s)$, $\forall j \in hp(i)$.

In our simulations, the WAC, with the support of RTC/S tool [69], is often faster than the BW.

Tightness Comparing with WAC, BW sets more constraints in deriving the interference from $hp(i)$, which results in that BW is tighter than WAC on the schedulability test.

In BW, X_k of Eq. 3.15a and Y_k of Eq. 3.15b set a constraint that the maximum events within $[0, t)$ cannot exceed $\overline{\alpha}_k(t)$. Buf_k in Eq. 3.16 sets a constraint that the maximum backlogged events cannot exceed the arrival events before the mode switch and the worst-case backlogged events. Since s is constrained by Eq. 3.12, the interference from LO-critical tasks is also constrained. While for WAC, in order to integrate the framework of Real-Time Calculus to do the sufficient test, it does not explore so much constraints in deriving $\alpha^u_{hp(i)}(\Delta)$.

3.5 Earliest Deadline First Schedulability Test

In the EDF scheduling analysis, the demand bound function (DBF) is used to determine whether current platform is able to schedule the taskset. By artificially shortening the relative LO mode deadlines, the DBFs in two modes can be tuned to comply with the provided supply function. In the following, we first provide the schedulable conditions. Second, a hidden feature is presented, based on which we present the derivation of the DBF. Then, we show how to search feasible deadlines for a given taskset. At last, the effectiveness of our approach is proven to be more applicable than the state-of-the-art demand-based scheduling approach in [18].

3.5.1 Schedulable Conditions

In Chapter 2, we have presented the task demand model, based on which the schedulability analysis is presented. Analogously, we rely on the task demand to analyze the schedulability of a system scheduled by EDF algorithm. Once again, the demand bound function $\text{dbf}(\tau_i, \Delta)$ gives an upper bound on the maximum possible execution demand of the task τ_i in any time interval of length Δ, where demand is calculated as the total amount of required execution time of events with their whole scheduling windows within the time interval [18].

As presented in Proposition 1 of Section 2.5.1.2, in mixed-criticality systems, the taskset is schedulable if the DBFs in LO and HI modes are smaller than the lower bound of the provided

service of this platform [18], i.e.,

$$\text{Condition } EDF-LO: \forall \Delta \geq 0: \quad \sum_{\tau_i \in \tau} \text{dbf}_{\text{LO}}(\tau_i, \Delta) \leq \Delta,$$
$$\text{Condition } EDF-HI: \forall \Delta \geq 0: \quad \sum_{\tau_i \in HI(\tau)} \text{dbf}_{\text{HI}}(\tau_i, \Delta) \leq \Delta.$$

Therefore, to check whether a taskset is scheduable, one only needs to derive the DBFs in both modes.

3.5.2 A Hidden Feature

The timing constraints require that the finishing time of any event must not exceed its absolute deadline.

Definition 6 (Absolute Deadline)**.** *The absolute deadline of an event is the time point that is the sum of its arrival time and its relative deadline.*

For some event traces, the finishing times of some events must be smaller than their absolute deadlines in order to leave enough processing slack for the following events. The effective deadline is defined to show this timing constraint.

Definition 7 (Effective Deadline)**.** *The effective deadline of an event is referred to its allowable largest finishing time that guarantees other events of this task meet their absolute deadlines.*

We use $AD(e)$ and $ED(e)$ to respectively denote the absolute deadline and the effective deadline of an event e.

Theorem 8. *For a schedulable task, the finishing times of all its events cannot exceed their effective deadlines, and the minimum distance between any two successive effective deadlines should not be smaller than its* WCET*.*

Proof. Based on the definition of effective deadlines, for a schedulable task, the finishing times cannot exceed the effective deadlines. If a task is guaranteed to be schedulable, any its event should be sufficiently given at least a processing interval of its WCET. Therefore, the minimum distance between any two effective deadlines should not be smaller than its WCET. Otherwise, some events may miss their absolute deadlines. □

The task τ_1 in the motivation example is used to show the difference between absolute deadlines and effective deadlines. The as-early-as-possible events are shown in Fig. 3.5. The absolute deadlines of e_1, e_2, e_3, e_4 are $7, 9, 11, 13$. The effective deadlines of them are $4, 7, 10, 13$, as every released event should be given 3 processing time units. e_1, e_2, e_3 must be finished no

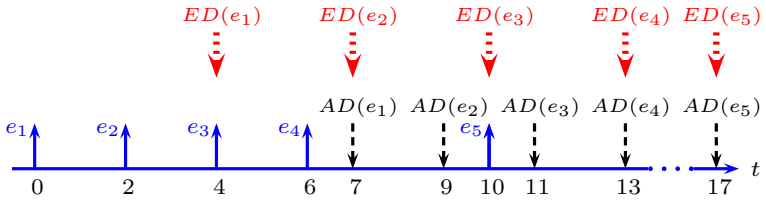


Figure 3.5: Event trace, absolute deadlines, and effective deadlines of the task τ_1 in the motivation example

later than $ED(e_1), ED(e_2), ED(e_3)$. Otherwise, e_4 would have no enough time to be finished before $AD(e_4)$.

Theorem 8 presents a very important hidden feature in deriving the DBF because finishing times of events are constrained by the effective deadlines, instead of absolute deadlines. In the following, we respectively derive the DBFs of LO and HI modes.

3.5.3 Demand Bound Function of LO mode

If the system is in LO mode, system starts from an idle state and every task behaves as a normal task with parameters $(\delta_i(q), C_i^L, D_i^L)$. The derivation of DBF needs to know the maximum number of events that can be released and must be finished within an interval of Δ. If there are at most n such events in this interval, the DBF is n folds of WCET. In order to obtain this maximum number, the as-early-as-possible activations are assumed. The effective deadlines corresponding to the as-early-as-possible activations are derived as follows.

First, the minimum distance function (MDF) of event trace can be generally categorized to be two types. One type fulfils that $C_i^L \leq \delta_i(1)$, which means that the WCET is not greater than the minimum inter-activation interval. The effective deadlines are the same as the absolute deadlines for this type. The other type fulfils that $C_i^L > \delta_i(1)$, which means that the WCET is greater than the minimum inter-activation interval. The effective deadlines will be different with absolute deadlines for this type. The MDF of event trace is generalized as follows

$$h \stackrel{\text{def}}{=} \min\{q | \delta_i(q+1) - \delta_i(q) > C_i^L\}. \tag{3.20}$$

where $\delta_i(0) = 0$, $q \in \mathbb{N}$. If $h = 0$, this is the type of $C_i^L > \delta_i(1)$. Otherwise, it is the type of $C_i^L \leq \delta_i(1)$.

The absolute deadlines corresponding to the as-early-as-possible events are shown in Fig. 3.6. We denote e_j as the j-th arrival event. If the arrival time of the first event e_1 is s, the earliest arrival time of the event e_{j+1} is $s + \delta_i(j)$, and its absolute deadline is $s + \delta_i(j) + D_i^L$. We

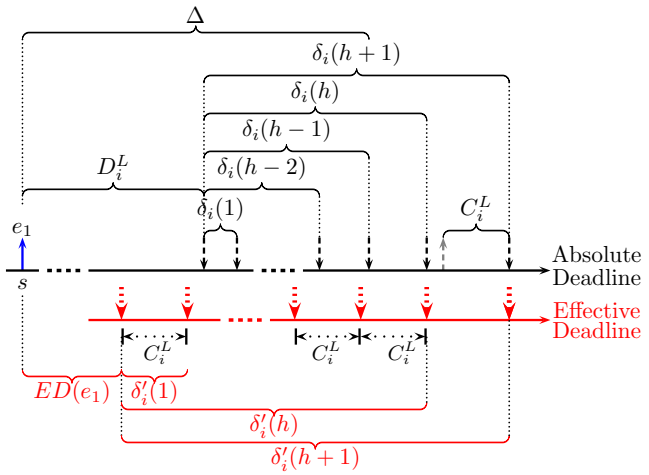


Figure 3.6: The absolute deadlines and effective deadlines corresponding to the as-early-as-possible event trace

consider the event e_{h+1} and the event e_{h+2}. Their absolute deadlines are $s+\delta_i(h)+D_i^L$ and $s+\delta_i(h+1)+D_i^L$. As $\delta_i(h+1)-\delta_i(h) > C_i^L$, their effective deadlines are the same as their absolute deadlines. For the event e_h, its absolute deadline is $s+\delta_i(h-1)+D_i^L$. As $\delta_i(h)-\delta_i(h-1) \leq C_i^L$, its effective deadline is $s+\delta_i(h)+D_i^L-C_i^L$, which is smaller than its absolute deadline. As every event should be given at least C_i^L, the minimum distance between two effective deadlines is C_i^L. Therefore, for the event e_{h-1}, its effective deadline is $s+\delta_i(h)+D_i^L-2C_i^L$. For the event e_{h-j+1}, its effective deadline is $s+\delta_i(h)+D_i^L-j\cdot C_i^L$.

In Fig. 3.6, $\delta_i'(k)$ represents the MDF of effective deadlines. Since the effective deadlines of first h events are separated by C_i^L and the effective deadlines of later events are the same as the absolute deadlines, the $\delta_i'(k)$ can be generalized as follows:

$$\delta_i'(k) \stackrel{\text{def}}{=} \begin{cases} k\cdot C_i^L, & k \leq h \\ h\cdot C_i^L + \delta_i(k) - \delta_i(h), & k > h, \end{cases} \tag{3.21}$$

where $\delta_i(h+1)-\delta_i(h) > C_i^L$, $k \in \mathbb{N}^+$.

Then, within an interval of Δ in Fig. 3.6, the maximum number of events that are released and finished is known. The demand bound function of LO mode is thus concluded as that:

$$\text{dbf}_{\text{LO}}(\tau_i, \Delta) \stackrel{\text{def}}{=} \begin{cases} 0, & \Delta < ED(e_1) \\ q\cdot C_i^L, \delta_i'(q-1) \leq \Delta - ED(e_1) < \delta_i'(q), \end{cases} \tag{3.22}$$

where $ED(e_1) = D_i^L + \delta_i(h) - \delta_i'(h)$. Note that, if $ED(e_1) < 0$, it is impossible to schedule this task.

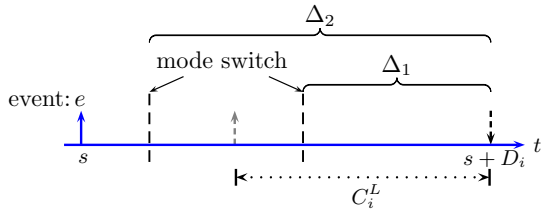


Figure 3.7: Bounding the demand of an event

3.5.4 Demand Bound Function of HI mode

The idea of deriving the DBF is based on the condition that the taskset can be schedulable in LO mode (i.e., Condition 1 holds). When we derive the DBF of HI mode, we therefore assume that all effective deadlines are met in LO mode.

We first consider the demand of a single event. In particular, suppose an event e of the task τ_i arrives at time s. As shown in Fig. 3.7, its absolute deadline $AD(e)$ is $s + D_i$. Assume its effective deadline $ED(e) = AD(e)$. If the mode switch is invoked at the time $s + D_i - \Delta_1$, the execution time length of this event before the mode switch is at least $C_i^L - \Delta_1$ because the deadline is met in LO mode. Hence, $\mathrm{dbf_{HI}}(e, \Delta_1) = C_i^H - (C_i^L - \Delta_1)$, where $\mathrm{dbf_{HI}}(\mathrm{e}, \Delta_1)$ denotes the DBF of the event e within Δ_1 in HI mode. If the mode switch is invoked at the time $s + D_i - \Delta_2$, as $\Delta_2 > C_i^L$, this event may not be executed before the mode switch. Hence, $\mathrm{dbf_{HI}}(\mathrm{e}, \Delta_2) = \mathrm{C_i^H}$. In general, we have

$$\mathrm{dbf_{HI}}(e, \Delta) \stackrel{\mathrm{def}}{=} C_i^H - [\![C_i^L - \Delta]\!]_0,$$

where $[\![A]\!]_k = \max(A, k)$.

Since events are finished before their effective deadlines in LO mode, such information should be used in deriving the DBF of HI mode, as shown in the following example.

Example 6. *Consider an event trace with $\delta_i(1) < C_i^L < \delta_i(2) - \delta_i(1)$. As shown in Fig. 3.8, the first three as-early-as-possible events are e_1, e_2, e_3. Denote the arrival time of e_1 is s. We assume, when the system stays in LO mode, those events can be schedulable, i.e., their effective deadlines are met. In this case, the effective deadline $ED(e_1)$ of e_1 is $s + D_i^L + \delta_i(1) - C_i^L$. Similarly, for the event e_2 and the event e_3, their effective deadlines are the same as their absolute deadlines because $\delta_i(2) - \delta_i(1) > C_i^L$.*

From the definition of the effective deadline, we deduce that, if the mode switch is invoked at the time $s + D_i^L + \delta_i(2) - \Delta$ as shown in Fig. 3.8, the remaining time for the event e_1 to meet its effective deadline is $RM(e_1) = ED(e_1) - \big(s + D_i^L + \delta_i(2) - \Delta\big)$. Therefore, the execution time length of the event e_1 before the mode switch is $C_i^L - RM(e_1)$. In Fig. 3.8, as there are at most 3 effective deadlines within Δ, the DBF *is at most $3 \cdot C_i^H$. Since the execution*

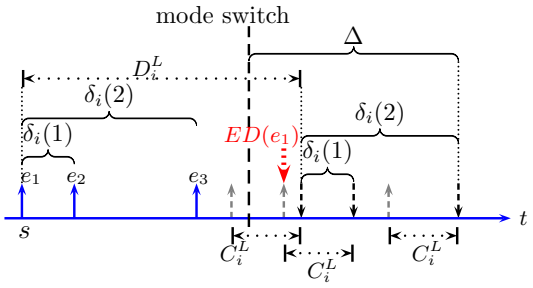


Figure 3.8: Bounding the demand of an event trace

time of the event e_1 *before the mode switch is at least* $[\![C_i^L - RM(e_1)]\!]_0$*, the tight* DBF *is* $\mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta) = 3 \cdot C_i^H - [\![C_i^L - RM(e_1)]\!]_0$.

The maximum number of effective deadlines within an interval of Δ depends on the MDF $\delta_i'(k)$ of effective deadlines. We set $D_i^H = D_i^L$ to keep the relative deadlines unchanged after the mode switch. Hence, the effective deadlines are also not changed after mode switch. As shown in Fig. 3.8, within an interval Δ that $\delta_i'(k) \leq \Delta < \delta_i'(k+1)$, the DBF is at most $(k+1) \cdot C_i^L$. In this interval, there might be an event that has already been executed before the mode switch.

Lemma 9. *In the interval of* Δ *that is* $\delta_i'(k) \leq \Delta < \delta_i'(k+1)$ *in HI mode, if the system needs to schedule* $k+1$ *events, the execution time of these events before the mode switch is not smaller than* $[\![C_i^L - (\Delta - \delta_i'(k))]\!]_0$.

Proof. For the time interval of length Δ, there are at most $x = \Delta - \delta_i'(k)$ time units left for the "first" event (the first event in HI mode) if the system schedules $k+1$ events in HI mode. If $x \geq C_i^L$, the "first" event may be not executed before the mode switch because there is enough interval for scheduling the "first" event. If $0 \leq x < C_i^L$, the execution time of the "first" event is at least $C_i^L - x$ because the "first" event is supposed to meet its effective deadline in LO mode. In general, $[\![C_i^L - (\Delta - \delta_i'(k))]\!]_0$ represents the least execution time. □

According to Lem. 9, the DBF of HI mode is concluded as follows:

$$\mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta) \stackrel{\mathrm{def}}{=} (k+1) \cdot C_i^H - [\![C_i^L - (\Delta - \delta_i'(k))]\!]_0,$$

where $\delta_i'(k) \leq \Delta < \delta_i'(k+1)$ and $\delta_i'(0) = 0$.

3.5.5 Demand Bound Function Tuning

In the above analysis, we set $D_i^H = D_i^L$. If D_i^L is decreased, i.e., $D_i^L < D_i^H$, the DBF of HI mode is different as the deadline is postponed $D_i^H - D_i^L$ after the mode switch. If the system

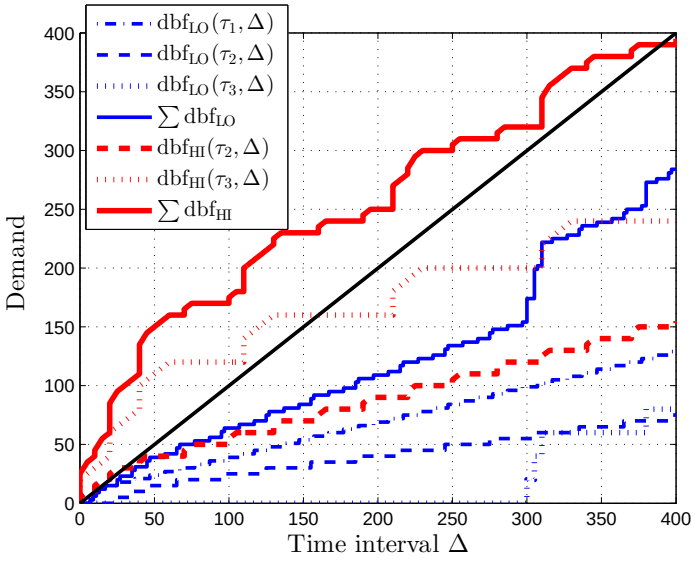


Figure 3.9: Demand bound functions of motivation example before the tuning

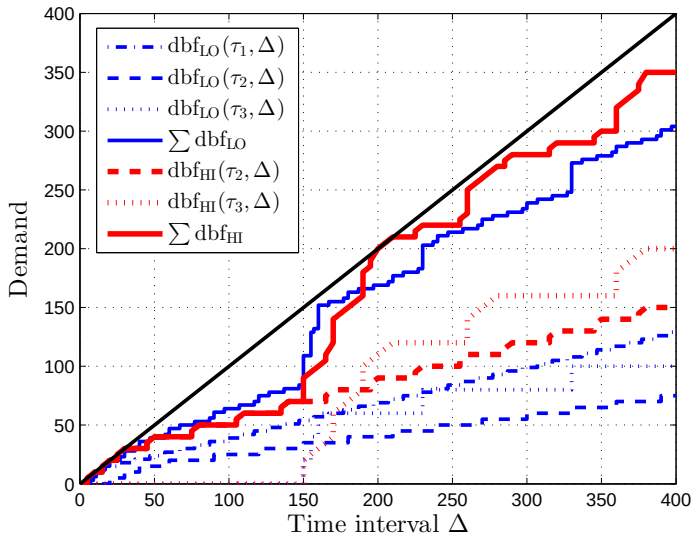


Figure 3.10: Demand bound functions of motivation example after the tuning

is schedulable in LO mode with the relative deadline of D_i^L, there will be extra $D_i^H - D_i^L$ time units for events to meet their absolute deadlines after the mode switch.

Theorem 9. *If a HI-critical task τ_i is schedulable in LO mode, its* DBF *of HI mode is that:*

$$\mathrm{dbf_{HI}}(\tau_\mathrm{i}, \Delta) \stackrel{\mathrm{def}}{=} \begin{cases} 0, & \text{if } 0 \le \Delta < D_i^H - D_i^L \\ (k+1) \cdot C_i^H - [\![C_i^L - x]\!]_0, & \\ \text{where } x = \Delta - \delta_i'(k) - \left(D_i^H - D_i^L\right) & \\ \text{and } \delta_i'(k) \le x + \delta_i'(k) < \delta_i'(k+1). & \end{cases} \tag{3.23}$$

Proof. After the mode switch, there is at least an interval of $D_i^H - D_i^L$ within which there is no deadline. Hence, if $0 \le \Delta < D_i^H - D_i^L$, $\mathrm{dbf_{HI}}(\tau_\mathrm{i}, \Delta) = 0$. If $\Delta \ge D_i^H - D_i^L$, within an interval of Δ that $\delta_i'(k) \le \Delta - \left(D_i^H - D_i^L\right) < \delta_i'(k+1)$, there are at most $k+1$ effective deadlines. Hence, $\mathrm{dbf_{HI}}(\tau_\mathrm{i}, \Delta) \le (\mathrm{k}+1) \cdot \mathrm{C_i^H}$. If the system needs to schedule $k+1$ events during this interval, the "first" event will probably have been executed before the mode switch. According to Lem. 9, the execution time is no smaller than $[\![C_i^L - (\Delta - \delta_i(k))]\!]_0$ in the case that $D_i^H = D_i^L$. In the case that $D_i^L < D_i^H$, $x = \Delta - \delta_i'(k) - \left(D_i^H - D_i^L\right)$ is the time interval for scheduling the "first" event to meet the D_i^L. As D_i^L is met, the execution time is $[\![C_i^L - x]\!]_0$. Hence, $\mathrm{dbf_{HI}}(\tau_\mathrm{i}, \Delta) = (\mathrm{k}+1) \cdot \mathrm{C_i^H} - [\![\mathrm{C_i^L} - \mathrm{x}]\!]_0$ where $\delta_i'(k) \le x + \delta_i'(k) < \delta_i'(k+1)$. □

Based on Them. 9, $\mathrm{dbf_{HI}}(\tau_\mathrm{i}, \Delta)$ can be tuned by changing the D_i^L. In order to make this dual-criticality system schedulable, conditions 1, 2 should hold. A heuristic approach is shown in Algo. 2 to search the feasible D_i^L. First, all HI-critical tasks are the candidates whose relative deadlines can be used to tune the DBFs, as shown in line 1. In the while loop, the D_i^L gradually decreases from D_i^H to C_i^L until conditions 1, 2 are met (lines 6-14). If no candidate is left to tune the DBFs after the exhaustive search, this algorithm fails to find the feasible D_i^L (lines 3-5). Note that, $\Delta_{\max}$ in line 7 is the maximum busy interval that the system starts processing tasks and backs to the idle state. This interval can be computed by using the method of computing busy interval size bound in [70].

Example 7. *Returning to the motivation example, suppose the three tasks are scheduled by* EDF*. If the LO deadlines are not reduced, their demand bound functions are shown in Fig. 3.9. It is not schedulable as the $\sum \mathrm{dbf_{HI}}$ is greater than the provided lower service. After the LO deadlines are reduced to that $D_2^L = 20$ and $D_3^L = 150$, their demand bound functions are tuned by Algorihtm 2, as shown in Fig. 3.10. After the tuning, this task set are schedulable.*

3.5.6 Effectiveness

In [18], the demand-based approach is demonstrated by extensive experiments to have greater schedulability over other existing scheduling approaches. The approach proposed in this paper is more applicable than the approach in [18] in that, any taskset that is tested schedulable by

Algorithm 2 Heuristic of searching feasible D_i^L

```
candidates ← {i|τ_i ∈ HI(τ)}
while true do
    if candidates = ∅ then
        return failure
    end if
    for D_i^L = D_i^H to C_i^L do
        for Δ = 0 to Δ_max do
            if Σ_{τ_i∈τ} dbf_LO(τ_i, Δ) > β^l(Δ) then
                remove τ_i from candidates; break
            else if Σ_{τ_i∈HI(τ)} dbf_HI(τ_i, Δ) < β^l(Δ) then
                return success
            end if
        end for
    end for
end while
```

the approach in [18] can also be tested schedulable by ours, even some tasksets that are tested not schedulable by the one in [18] may also be tested schedulable by ours.

Lemma 10. *For a sporadic task τ_i defined in [18], the* $\mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta)$ *and* $\mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta)$ *derived by the approach in this paper are the same as* DBF *derived by the approach in [18].*

Proof. In [18], the sporadic task is defined by a tuple $\left(C_i^L, C_i^H, D_i^L, D_i^H, T_i\right)$, where T_i is the minimum inter-activation interval, and for the other notations, please recall them from this paper. The relationship among them is that $C_i^L \leq D_i^L \leq D_i^H \leq T_i$.

Since $C_i^L \leq T_i$, the effective deadlines are the same as the absolute deadlines. The MDF of effective deadlines is that $\delta_i'(k) = k \cdot T_i$, and $ED(e_1) = D_i^L$. Therefore, the $\mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta)$ of Eq. 3.22 can be transformed to be as follows:

$$\mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta) \stackrel{\mathrm{def}}{=} [\![(\lfloor(\Delta - D_i^L)/T_i\rfloor + 1) \cdot C_i^L]\!]_0,$$

which is the same as DBF of LO mode in [18].

In a similar way, if τ_i is a HI-critical task, the $\mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta)$ of Eq. 3.23 can also be transformed to be the same as HI mode DBF in [18]. □

This lemma shows that, our approach can achieve the same schedulability as the approach in [18] if a task is modeled as the sporadic task. However, the sporadic task model is often a pessimistic representation.

Corollary 1. *For a task τ_i with* MDF $\delta_i(k)$*, denote* $\mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta)$ *and* $\mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta)$ *as its* DBF *of LO and HI modes, and denote* $\mathrm{dbf}^{\mathrm{s}}_{\mathrm{LO}}(\tau_i, \Delta)$ *and* $\mathrm{dbf}^{\mathrm{s}}_{\mathrm{HI}}(\tau_i, \Delta)$ *as the* DBF *of its pessimistic representation. It can be deduced that* $\mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta) \leq \mathrm{dbf}^{\mathrm{s}}_{\mathrm{LO}}(\tau_i, \Delta)$ *and* $\mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta) \leq \mathrm{dbf}^{\mathrm{s}}_{\mathrm{HI}}(\tau_i, \Delta)$.

Proof. If the task τ_i is represented by its pessimistic representation, the MDF of its pessimistic representation is that $\delta_i^s(k) = k \cdot \delta_i(1)$. Since $\delta_i^s(k) = k \cdot \delta_i(1) \leq \delta_i(k)$, we have $\mathrm{dbf}_{\mathrm{LO}}(\tau_\mathrm{i}, \Delta) \leq \mathrm{dbf}^{\mathrm{s}}_{\mathrm{LO}}(\tau_\mathrm{i}, \Delta)$ and $\mathrm{dbf}_{\mathrm{HI}}(\tau_\mathrm{i}, \Delta) \leq \mathrm{dbf}^{\mathrm{s}}_{\mathrm{HI}}(\tau_\mathrm{i}, \Delta)$. □

This corollary shows that, if the pessimistic representation of a task is tested schedulable by the approach in [18], it must also be tested schedulable via our approach by modeling it as MDF because

$$\begin{array}{l} \mathrm{dbf}^{\mathrm{s}}_{\mathrm{LO}}(\tau_\mathrm{i}, \Delta) \leq \beta^{\mathrm{l}}(\Delta, \mathrm{t}) \\ \mathrm{dbf}^{\mathrm{s}}_{\mathrm{HI}}(\tau_\mathrm{i}, \Delta) \leq \beta^{\mathrm{l}}(\Delta, \mathrm{t}) \end{array} \Longrightarrow \begin{array}{l} \mathrm{dbf}_{\mathrm{LO}}(\tau_\mathrm{i}, \Delta) \leq \beta^{\mathrm{l}}(\Delta, \mathrm{t}) \\ \mathrm{dbf}_{\mathrm{HI}}(\tau_\mathrm{i}, \Delta) \leq \beta^{\mathrm{l}}(\Delta, \mathrm{t}) \end{array} \tag{3.24}$$

However, if a taskset is tested schedulable by our approach, it may not be tested schedulable by the approach in [18].

3.6 Schedulability Evaluation

In this section, towards the arbitrary event streams, we present the schedulability evaluation on our proposed analyzing approaches. In specific, the proposed approaches are:

- NEC: The necessary test in Section 3.4.2, showing the necessary conditions that tasks should meet in order to be scheduled by fixed priority. NEC provides an upper bound for the sufficient tests, because task sets that are tested schedulable by any sufficient test are deemed to pass this test.
- WAC: The sufficient test by deriving the workload arrival curve in Section 3.4.3.1.
- BW: The sufficient test by the busy-window analysis in Section 3.4.3.2.
- Tuning: In EDF-scheduled systems, LO deadlines of HI-critical tasks are tuned to conform the demand bound function to below the provided supply bound.

Besides, for the sporadic tasks, an existing approach is used to compare with our proposed approaches.

- AMC-max: The sufficient test by the most powerful response-time calculation for fixed-priority scheduling from [8]. This method is only valid for sporadic tasks whose relative deadlines are smaller than periods.

The RTC/S tool is used to the NEC and WAC test. It is also used to compute the maximum number of backlogged events in Eq. 3.10 for BW test. All the results are obtained from a simulation host with Intel i7-4770 processor and 16GB RAM.

3.6.1 Task Set Generation

The task set is generated in the same way as [19]. A random task set is generated by starting with an empty task set $\tau = \emptyset$, where random tasks are successively added. Although the four approaches can handle the task with any activation pattern, for the easiness, the task is set as pjd pattern in our simulations. By artificially varying the parameters, the effects of jitter and burst on the system schedulability are evaluated.

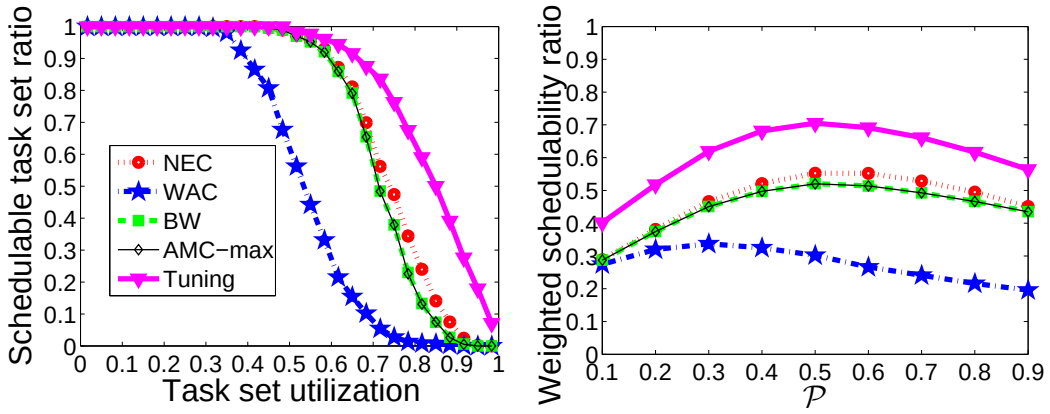
The random task set is generated as follows:

- The task utilization is a value of $(x + 0.5)/30$, where $x \in \{0, 1, ..., 29\}$.
- The probability of a random task being a HI-critical task is $\mathcal{P}$.
- $C_i(LO)$ is drawn from the uniform distribution over $\{1, 2, ..., C_L^{\max}\}$. There are two settings for $C_L^{\max}$. The first setting is to set $C_L^{\max} = 10$, in order to make the generated tasks mostly be light tasks (low utilization). The second setting is to set $C_L^{\max} = 40$, in order to produce a task set mixed with light and heavy tasks.
- $C_i(HI)$ is drawn from the uniform distribution over $\{C_i(LO), C_i(LO) + 1, ..., 4 \cdot C_i(LO)\}$ if $L_i = HI$.
- The period p_i is drawn from the uniform distribution over $\{C_i(L_i), C_i(L_i) + 1, ..., 200\}$.
- The jitter j_i is set as $\mathcal{X} \cdot p_i$, where $\mathcal{X} \in [0, 5)$.
- The minimum inter distance d_i is set as $\mathcal{Y} \cdot p_i$, where $\mathcal{Y} \in [0, 1)$
- The relative deadline is set as that $D_i(LO) = D_i(HI) = \mathcal{Z} \cdot p_i$, where $\mathcal{Z} \in [0, 5)$.

We introduce that

$$U_{LO}(\tau) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} \big(C_i(LO)/p_i\big),\ U_{HI}(\tau) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau^H} \big(C_i(HI)/p_i\big), \tag{3.25}$$

where τ^H is the set of all HI-critical tasks in task set τ. The task set utilization is defined as $U(\tau) \stackrel{\text{def}}{=} (U_{HI} + U_{LO})/2$. For every task set generation, the utilization is allowed to be located in $[U^* - 0.005, U^* + 0.005]$, where U^* is a targeted utilization. If the generated utilization is smaller than $U^* - 0.005$, a new random task is added. If the generated utilization is greater than $U^* + 0.005$, this task set is discarded, and a new empty task set is started, until a task set with the allowed utilization is found.



(a) $(\mathcal{P}, \mathcal{X}, \mathcal{Y}, \mathcal{Z}) = (0.5, 0, 0, 1)$ (b) $(\mathcal{X}, \mathcal{Y}, \mathcal{Z}) = (0, 0, 1)$

Figure 3.11: Schedulability results towards the sporadic light task sets (all subfigures share the same color scheme)

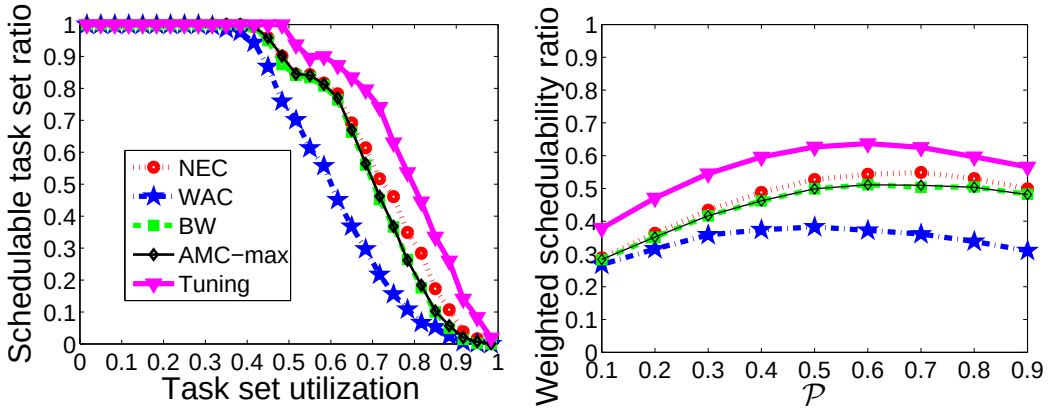
3.6.2 Evaluation Results

In our experiments, for each target utilization, 1000 task sets were generated, and the schedulability of those task sets was determined by the four analyzing approaches. The graphs are best viewed online in colour.

3.6.2.1 Schedulability Test on Sporadic Task Sets

First, we evaluate the four schedulability test approaches towards the sporadic task sets with implicit deadlines.

Fig. 3.11(a) and Fig. 3.12(a) show the percentage of light and mixed task sets that are tested schedulable. By setting $(\mathcal{P}, \mathcal{X}, \mathcal{Y}, \mathcal{Z}) = (0.5, 0, 0, 1)$, tasks are generated as implicit-deadline sporadic tasks with 50% being HI-critial. In both figures, we observe that BW achieves exactly the same schedulable percentage as AMC-max, while outperforms the WAC by a large margin. This is expected as BW explores the same constraints as AMC-max to compute the WCRT. WAC pessimistically derives the workload arrival curve of higher-priority tasks, thus resulting in an overestimate of WCRT. Besides, we also observe that, the schedulable percentages of BW and AMC-max are slightly less than the limit illustrated by NEC upper bound, which indicates that schedulability tests of BW and AMC-max are very tight. Regarding to the Tuning approach, we find that it is the best among all approaches as this approach is applied in the EDF schedule.



(a) $(\mathcal{P}, \mathcal{X}, \mathcal{Y}, \mathcal{Z}) = (0.5, 0, 0, 1)$ (b) $(\mathcal{X}, \mathcal{Y}, \mathcal{Z}) = (0, 0, 1)$

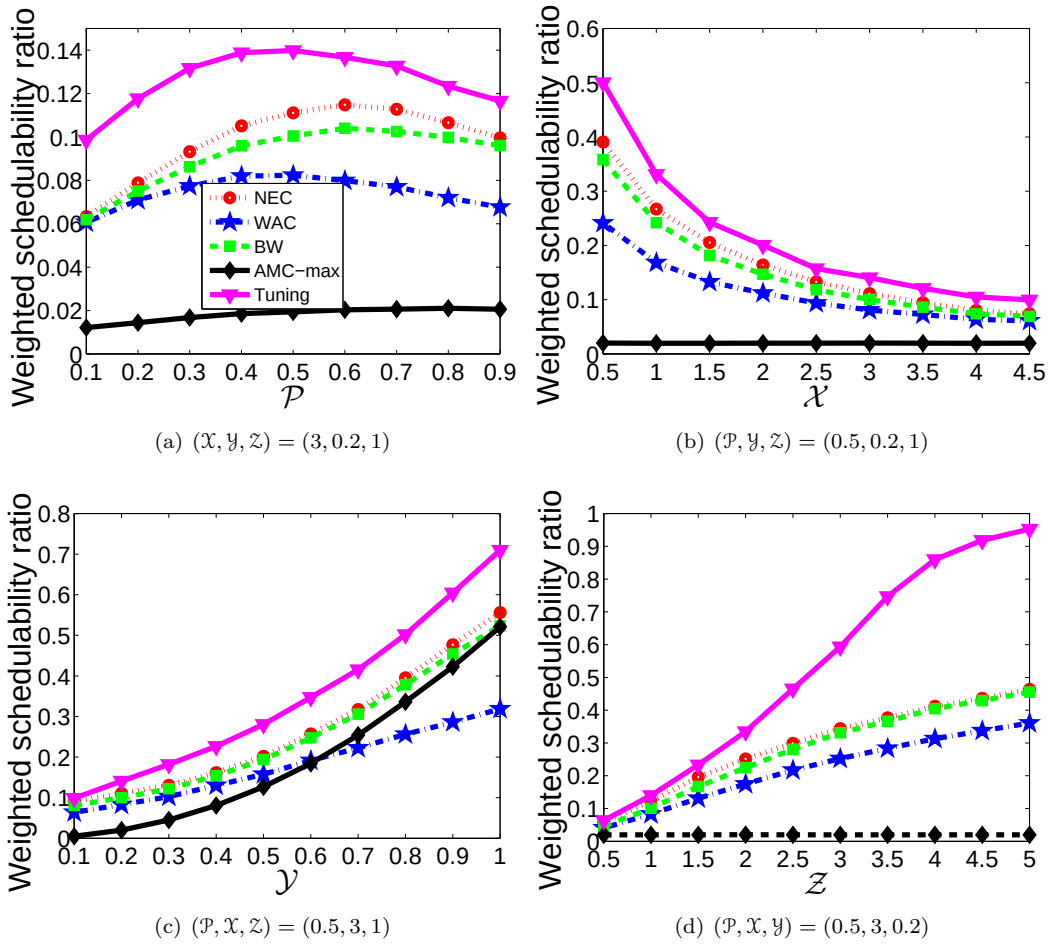
Figure 3.12: Schedulability results towards the sporadic mixed task sets (all subfigures share the same color scheme)

Next, we study the effects of the parameters $(\mathcal{P}, \mathcal{X}, \mathcal{Y}, \mathcal{Z})$ on the system schedulability by our proposed tests. To evaluate those parameters, the *weighted schedulability ratio* $W(\mathcal{P}, \mathcal{X}, \mathcal{Y}, \mathcal{Z})$ [71] is used, which is defined as follows:

$$W(\mathcal{P}, \mathcal{X}, \mathcal{Y}, \mathcal{Z}) \stackrel{\text{def}}{=} \Big(\sum_{\forall \tau} U(\tau) \cdot S(\tau, \mathcal{P}, \mathcal{X}, \mathcal{Y}, \mathcal{Z})\Big) / \sum_{\forall \tau} U(\tau),$$

where $S(\tau, \mathcal{P}, \mathcal{X}, \mathcal{Y}, \mathcal{Z})$ is the schedulability ratio of the task set whose utilization and parameters are $U(\tau)$ and $(\mathcal{P}, \mathcal{X}, \mathcal{Y}, \mathcal{Z})$.

In order to generate implicit-deadline sporadic tasks, $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ must be (0, 0, 1). Hence, only $\mathcal{P}$ can be investigated. The weighted schedulability ratio w.r.t., $\mathcal{P}$ is shown in Fig. 3.11(b) and Fig. 3.12(b). It shows that the achieved weighted schedulability ratios of BW and AMC-max are exactly the same and are slightly lower than the upper bound of NEC. The performance gap between BW and NEC is small when $\mathcal{P}$ is small and becomes large when $\mathcal{P}$ increases. This is because, when $\mathcal{P}$ is small, most tasks are LO-critical. Since the schedulability test towards the LO-critical tasks are the same in the four approaches, i.e., Eq. 3.8a is used as the sufficient and necessary test, the schedulability ratio of the four approaches are expected to be close if $\mathcal{P}$ is small. From the four figures, it is observed that the superiority of BW/AMC-max over WAC is greater in scheduling light task sets than in scheduling mixed task sets and the Tuning approach in EDF schedule is always the best.



(a) $(\mathcal{X}, \mathcal{Y}, \mathcal{Z}) = (3, 0.2, 1)$

(b) $(\mathcal{P}, \mathcal{Y}, \mathcal{Z}) = (0.5, 0.2, 1)$

(c) $(\mathcal{P}, \mathcal{X}, \mathcal{Z}) = (0.5, 3, 1)$

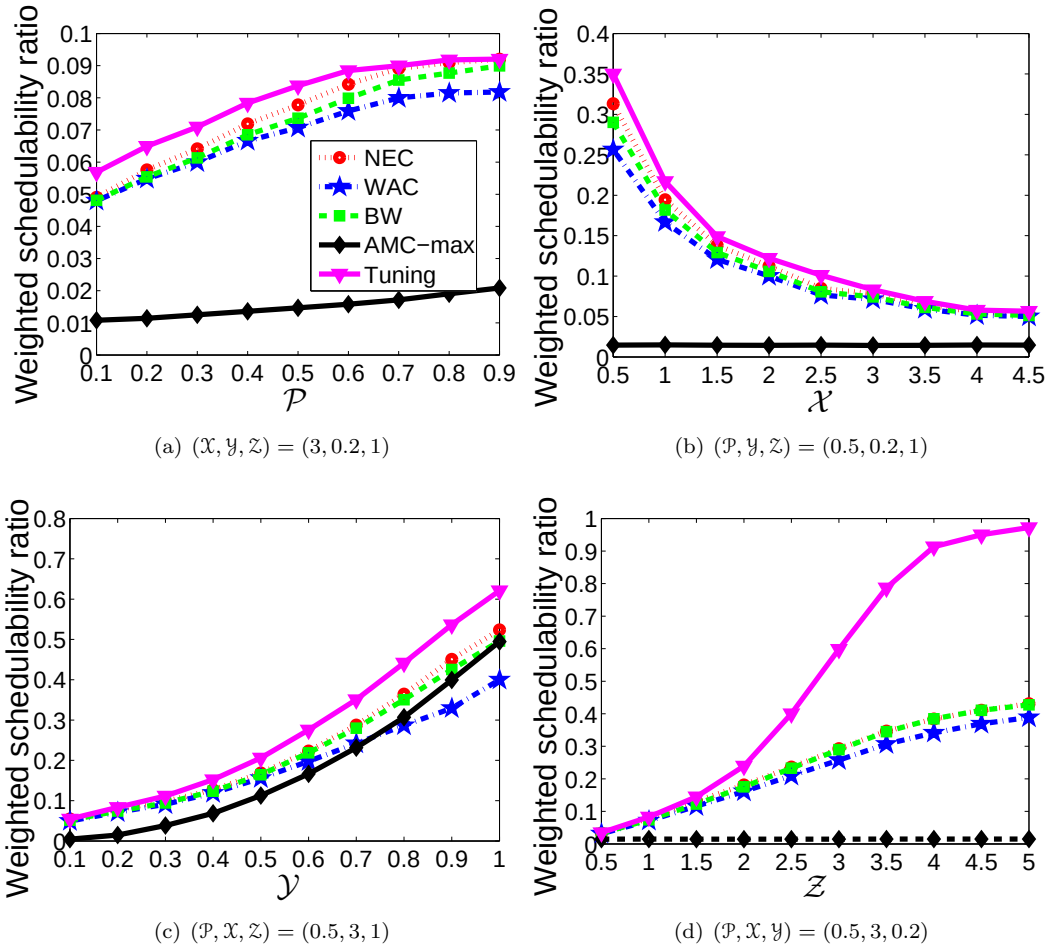
(d) $(\mathcal{P}, \mathcal{X}, \mathcal{Y}) = (0.5, 3, 0.2)$

Figure 3.13: The effects of parameters $(\mathcal{P}, \mathcal{X}, \mathcal{Y}, \mathcal{Z})$ on the system schedulability towards light task sets (all subfigures share the same color scheme as the first figure)

3.6.2.2 Schedulability Test on Arbitrarily Activated Task Sets

Except the schedulability evaluations on arbitrarily activated task sets, we also evaluate how the results are changed towards the arbitrarily activated task sets by varying the parameters $(\mathcal{P}, \mathcal{X}, \mathcal{Y}, \mathcal{Z})$ (one parameter each time), whose meanings have been presented in Section 3.6.1.

Since the activation patterns of tested tasks are not sporadic and the relative deadlines are arbitrary, AMC-max cannot be directly used in testing the schedulability of those tasks. However, in a pessimistic way, sporadic model with implicit deadlines can still be used to model the tasks with arbitrary activations and arbitrary relative deadlines. Suppose for a task τ_i, the minimum interval between any two task activations is T_i and its relative deadline is D_i. This



(a) $(\mathcal{X}, \mathcal{Y}, \mathcal{Z}) = (3, 0.2, 1)$

(b) $(\mathcal{P}, \mathcal{Y}, \mathcal{Z}) = (0.5, 0.2, 1)$

(c) $(\mathcal{P}, \mathcal{X}, \mathcal{Z}) = (0.5, 3, 1)$

(d) $(\mathcal{P}, \mathcal{X}, \mathcal{Y}) = (0.5, 3, 0.2)$

Figure 3.14: The effects of parameters $(\mathcal{P}, \mathcal{X}, \mathcal{Y}, \mathcal{Z})$ on the system schedulability (all subfigures share the same color scheme as the first figure)

task can be modeled as a sporadic task whose minimum inter distance and relative deadline is $\min(T_i, D_i)$. In this way, AMC-max can be used to test the schedulability of those tasks[1].

Fig. 3.13 and Fig. 3.14 show the effects of $(\mathcal{P}, \mathcal{X}, \mathcal{Y}, \mathcal{Z})$ on the system schedulability, respectively. From those figures, we can conclude that,

- Fig. 3.13(a) shows that, for the light task set, it is good to improve the system schedulability with the same portion of LO-critical and HI-critical tasks. Fig. 3.14(a) shows that, for the mixed task set, the system schedulability is better with a higher portion of HI-critical tasks.

[1]If there are burst events for activating a task, i.e., $T_i = 0$, this task is unschedulabled by AMC-max test.

- Fig. 3.13(b) and Fig. 3.14(b) show that, the increase of jitter decreases the system schedulability. This is expected as the increase of jitter will also increase the number of task activations within a certain interval.
- Fig. 3.13(c) and Fig. 3.14(c) show the increase of minimum inter-arrival distance between two events improves the system schedulability, because this parameter decreases the number of task activations within a certain interval. When $\mathcal{Y} = 1$, BW and AMC-max perform the same. This is expected because tasks become sporadic tasks with implicit deadlines when $\mathcal{Y} = 1$.
- Fig. 3.13(d) and Fig. 3.14(d) show that, the increase of relative deadline is helpful to increase the system schedulability. In addition, it also shows that the superiority of Tuning increases with the increase of relative deadline.

Overall, in all those figures, we observe that: the BW constantly outperforms the WAC, and keeps close results to NEC. This shows that BW is a very effective approach of analyzing the system schedulability towards the arbitrary activated tasks. Besides, from the performance of AMC-max, it can be observed that using sporadic task model to model arbitrary activated tasks leads to very pessimistic schedulability test results. Furthermore, the Tuning approach is shown to be the best on all figures, which confirms that the Tuning approach in EDF schedule is a very effective approach to increase the system schedulability.

3.7 Summary

In this chapter, we proposed to use the arrival curve and minimum distance function to more accurately represent the upper bound of task activations; based on this representation, the schedulability test of FP and EDF algorithms are proposed. From the experimental results, we found that, the decrease of the jitter, the increase of minimum interval between any two task activations, and the increase of the relative deadlines will increase the system schedulability. Using sporadic tasks to model arbitrary tasks with arbitrary deadlines will result in very pessimistic schedulability test results, especially for scheduling tasks whose minimum task activation interval and relative deadlines are small.

Chapter 4

Adaptive Workload Management

We have presented how to adaptively postpone the mode-switch by making use of the system static and runtime slacks in Chapter 2, where LO-critical tasks are only guaranteed in LO and Border modes. Indeed, the dual-criticality system can also be interpreted as a system mixed with soft and hard tasks, where deadlines of hard tasks must be guaranteed and soft tasks are then given the best possible service. This interpretation is important in the case that the worst-case activation patterns or WCETs of soft tasks are unknown because it is then impossible to guarantee LO-critical tasks scheduability in LO mode. In this chapter, we treat the soft task as the LO-critical task and the hard task as the HI-critical task and address the problem of how to give the best service to soft tasks while sufficiently guarantee the timing requirements of HI-critical tasks.

4.1 Overview

In Chapter 3, we have presented the mixed-criticality schedulability analysis towards the arbitrarily activated tasks. The interpretation of mixed-criticality system in Chapter 3 is that LO-critical tasks have deadlines and their deadlines should be met under the condition that HI-critical tasks will not overrun their LO WCETs. However, in reality, there also exists some cases that LO-critical tasks may not have strict response time requirement. In those cases their response times should be as small as possible to guarantee the system good performance. This brings a new question that how the response time of LO-critical tasks can be minimized.

To address this concern, a recent design in fixed-priority system is to set LO-critical tasks as a group sharing the highest priority because in this way LO-critical tasks can always be served

in the first place. But to guarantee the timing requirements of the HI-critical tasks, the execution of high-priority LO-critical tasks should be constrained within a certain bound [72–74]. When their execution demand exceeds this bound, either their execution is delayed by a regulator [72] or the execution priorities are exchanged with the HI-critical tasks [74]. Nevertheless, computing such a bound for the LO-critical tasks is nontrivial as the bound on the one hand should be sufficient to guarantee the execution of the HI-critical tasks and on the other hand should not be too pessimistic to improve the QoS of LO-critical tasks.

There is related work in the literature to offline compute a workload bound by which the coming workload of the low-criticality is shaped at runtime. Wandeler *et al.* [41, 72] proposed to use the real-time interface to obtain the shaping bound and suggested to delay the non-real-time events when they exceed the pre-computed bound. With the computed shaping bound, the work in [61, 75] discussed in more details on how to monitor the LO-critical events by switching the workload bound distribution among LO-critical tasks, or by using the workload arrival curve to monitor the whole group of LO-critical events. Tobuschat *et al.* [74] presented a scheme that exploits the throughput and latency slack of critical applications, by prioritizing non-critical accesses over critical accesses and exchanging the priorities when the non-critical workload exceeds the pre-computed bound.

The feasible bounding parameters in the aforementioned related work are however obtained offline, i.e., the bound used to shape the workload of the LO-critical tasks is computed offline and unchanged during the runtime. This bound is computed based on the assumptions of the worst-case event arrival patterns of all HI-critical tasks. While using the worst-case assumption during the runtime can sufficiently guarantee the safeness of the workload bound, it also makes the offline bound pessimistic due to the differences between the actual demand and the assumed worst-case demand of the HI-critical tasks. Such pessimism will significantly hamper the QoS of LO-critical tasks and reduce the overall system utilization. To reduce the pessimism, the shaping bound should be adaptively computed at runtime based on the actual demand from the HI-critical tasks. Such online adaption is however not so easy. On the one hand, the actual demand should be a valid upper bound that guarantees no HI-critical tasks miss their deadlines. On the other hand, the adaption decisions should be lightweight. While most of the previous work relies on the heavy numerical computation, to the best of our knowledge, no work so far has considered adaptively refining the feasible workload bound at runtime.

Being aware of this, in this chapter we propose to adaptively refine the bound of constraining the LO-critical tasks at runtime. Inspired by the online task procrastination technique in

dynamic energy management [47], the feasible bound for the LO-critical tasks is dynamically refined according to historical knowledge of the arrival workload in the past. Based on the refined bound, the priority-adjustment policy and workload-shaping policy are proposed to manage the LO-critical workload in order to provide the service for LO-critical tasks as much as possible, while guaranteeing sufficient service for meeting the hard real-time constraints of HI-critical tasks. The detail contributions of this chapter are as follows:

- We present an adaptive scheme for the online bound refinement in MCSs. By monitoring the arrival events of HI-critical tasks, the actual demand is adaptively computed to shape the workload of the LO-critical tasks at runtime. The computed actual demand can guarantee that the HI-critical tasks meet all their deadlines, while at the same time increase the QoS of LO-critical tasks.

- Two online workload management policies, namely, priority-adjustment policy and workload-shaping policy, are investigated. The priority-adjustment policy reduces the interference on HI-critical tasks through decreasing the priority of LO-critical tasks when the LO-critical workload exceeds the refined workload bound. The workload-shaping policy shapes the incoming LO-critical workload to comply with the refined bound by keeping the priority of LO-critical tasks always the highest.

- To update the refined bound for LO-critical tasks, real-time interface is traditionally used [41,72] online to compute the provided service and demand bound functions for lower priority tasks. To eliminate the complex real-time interface computation that requires intensive numerical calculation, a lightweight computation method with the complexity of $O(n \cdot log(n))$ is developed, making the online adaptations applicable at runtime.

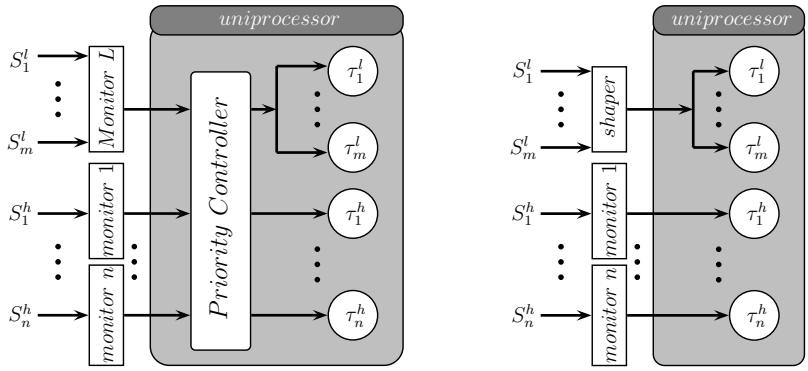
- Extensive experiments are presented to show the efficiency and effectiveness of our two proposed workload management policies. The adaptive shaping approaches are demonstrated to significantly improve the system utilization and QoS of LO-critical tasks. Comparing with the method that applies the exact real-time interface computation, the timing overhead of our proposed lightweight method is one and two orders of magnitude lower in the priority-adjustment policy and in the workload-shaping policy, respectively. The workload management effect of using the lightweight method has minor difference with that of using the exact Real-Time Interface method.

The rest of this chapter is organized as follows: Section 4.2 reviews related work. Section 4.3 provides our system model and settings. Section 4.4 presents the schedulability analysis by applying the real-time interface and the future workload bound by applying dynamic counters. Section 4.6 presents the priority-adjustment policy and workload-shaping policy to do the LO-critical workload management. Section 4.7 proposes a lightweight method for refining the runtime workload bound. Simulation results are presented in Section 4.8 and Section 4.9 concludes this chapter.

4.2 Related Work

In the related work section of Chapter 2 and Chapter 3, we have reviewed the relevant works based on the standard mixed-criticality model. In contrast with this standard model, the task model in this chapter is assumed to be without mode-switch at runtime. This model has been studied in recent work, such as [61, 62, 74]. In the following, we review the existing workload management approaches under this task model.

In the network calculus [46], the greedy shapers were proposed to strictly conform the overloaded packets to the arrival constraints. An important property of the greedy shaper is "greedy shaper comes for free" [46], which means that the shaping does not increase delay or buffer requirements. The use of greedy shaper was introduced to schedule the real-time events in [72], and it was found that the property of "greedy shaper comes for free" is also applicable for shaping real-time events. In [76], an admission server was proposed to enforce arbitrary real-time demand-curve interfaces. However, its implementation overhead is not low. In [77], by implementing a dual-bucket mechanism (similar to dynamic counters [64]) into FPGA, the runtime inputs are conformed to the designed arrival constraints. The implementation showed that the resource and timing overheads of this shaping scheme are very low in FPGA. Another monitoring method based on the minimum distance functions (i.e., an inverse representation of arrival curves) was proposed to do the event model verification [75]. This method is able to monitor the periodic burst events. Its runtime overhead can be reduced by using an l-repetitive function to represent the minimum distance function. The monitoring difference between using the dynamic counters and l-repetitive function was explored in [78, 79]. Although both dynamic counters and l-repetitive functions are effective in monitoring some certain-pattern event traces, only dynamic counters approach was presented how to use the past monitoring results to predict the future event traces [64].



(a) System setup for priority-adjustment policy

(b) System setup for workload-shaping policy

Figure 4.1: Mixed-criticality systems scheduled by the preemptive FP policy

The multi-mode monitoring and workload monitoring were proposed in [61] to monitor the LO-critical workload by switching the workload bound distribution of specific LO-critical tasks. A similar approach is to monitor the LO-critical events as a group [62] by using workload arrival curve, in which way the system utilization increases while the timing constraints of HI-critical tasks are met. By monitoring the runtime workload, a scheme named workload-aware shaping was proposed to improve the resource utilization by prioritizing the LO-critical accesses over HI-critical accesses, and exchange the priorities among them only when the coming LO-critical workload exceeds the designed bound [74]. Phan *et al.* [73] introduced another technique to use an optimal greedy shaper for shaping periodic tasks with jitter to improve the schedulability of real-time systems.

None of the preceding monitoring or shaping scheme allows to refine the shaping bound. Thus, their shaping schemes are pessimistic because of the differences between the actual demand and the worst-case demand. In this chapter, we aim to fill this gap by providing an adaptive bound refinement scheme.

4.3 System Settings

In this chapter, we consider a uniprocessor scheduled by the preemptive fixed-priority (FP) scheduling policy. Towards the two workload management polices, there are two system settings, respectively as shown in Fig. 4.1. In the two systems, there are a set of LO-critical tasks $\tau^l = \{\tau_1^l, \tau_2^l, ..., \tau_m^l\}$ and a set of HI-critical tasks $\tau^h = \{\tau_1^h, \tau_2^h, ..., \tau_n^h\}$. Every task is activated

by an event stream and the corresponding event streams for those tasks are denoted as $S^l = \{S_1^l, S_2^l, ..., S_m^l\}$ and $S^h = \{S_1^h, S_2^h, ..., S_n^h\}$. For every HI-critical event stream, a monitor is applied to monitor its arrival events. For the HI-critical tasks, in order to strictly meet their deadlines, their activation patterns are often fixed and its upper bound can be represented by an arrival curve. For LO-critical tasks, such as multi-media tasks in a car, their workload may sometimes be overloaded. Hence, their workload needs to be regulated, in order to constrain their interferences on HI-critical tasks. Note that, recent works of [61, 62, 74] have the same assumptions on system as ours.

In the system for priority-adjustment policy, as shown in Fig. 4.1(a), a priority controller is applied in the system to dynamically assign priorities to the arriving events. The LO-critical events are grouped together sharing the same priority, and are monitored by the Monitor L during the runtime. The priority of all LO-critical tasks is named LO-critical priority in the following. The priorities order among HI-critical tasks is fixed and unchanged during the runtime, while the LO-critical priority can be changed to lower level or higher level, subjected to the actual demand of HI-critical tasks. This means that the priority controller can only change the priority of τ^l from 1 to $n+1$. Without the loss of generality, the tasks $\tau_1^h, ..., \tau_n^h$ in τ^h are prioritized in a descending order, that is, the priority of the task τ_i^h is higher than that of the task τ_j^h when $i < j$.

In the system for workload-shaping policy, as shown in Fig. 4.1(b), a shaper is used to manage the inflowing workload from LO-critical event streams. In this case, the priorities of all tasks cannot be changed during the runtime. The priorities for HI-critical tasks $\tau_1^h, ..., \tau_n^h$ are also set as a descending order. All LO-critical tasks are grouped together to share the highest priority among all tasks, with the aim of maximizing the service for LO-critical tasks. In contrast to the priority controller in priority-adjustment policy, the shaper is only responsible for the release of LO-critical events. The principle of releasing an event is that, the released event should not result in a deadline miss of any HI-critical task.

In all monitors or shapers, buffers are used to store the backlogged events for every event stream, because there may be some events that have arrived but not released. The buffer obeys the principle of first-come-first-serve. The size of the buffer is assumed to be large enough. Note that, although the WCET of every event can be different, the LO-critical events can be simply considered from one stream whose workload is bounded by a workload arrival curve, which is similar to the workload bound of a group tasks in [62].

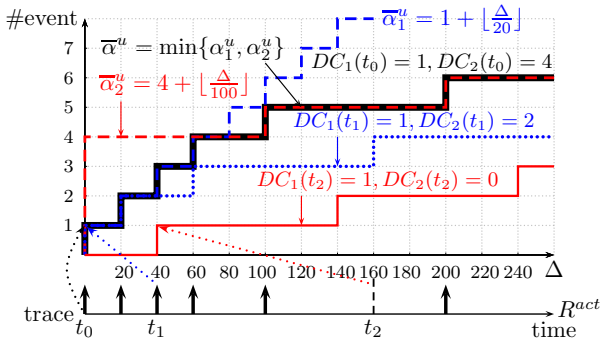


Figure 4.2: An example for using dynamic counters to predict the future events

4.4 Real-time calculus routines and interface analysis

In this section, we present the basic routines to construct the actual arrival curve and demand bound function at runtime, on top of which we present the interface analysis to guarantee no deadline misses for HI-critical tasks.

4.4.1 Arrival Curves and Service Demand with Historical Information

Before presenting the schedulability analysis, we first introduce how to derive the actual arrival curve and demand bound functions with historical information. The actual arrival curve consists of the workload of future events, the backlogged events, and the carry-on event (active and not finished), while the demand bound function also consists of the demand of the future events, the backlogged events, and the carry-on event.

4.4.1.1 Future Events and their Demand Bound

The arrival pattern of future events can be predicted by using the past information. We adopt the dynamic counters approach to monitor the runtime events as this approach was presented how to fast obtain a tight bound on the number of future events.

In principle, any (discrete) complex arrival pattern can be bounded by a set of upper and lower staircase functions [80]. Therefore, suppose the arrival curve $\overline{\alpha}^u(\tau_i, \Delta)$ of task τ_i is composed of n' staircase functions, i.e.,

$$\forall \Delta \in \mathbb{R}_{\geq 0} : \overline{\alpha}^u(\tau_i, \Delta) \leq \min_{j=1..n'} \{N_j^u + \lfloor \frac{\Delta}{\delta_j^u} \rfloor\},$$

where N_j^u is the initial value of a staircase function and δ_j^u is the stair length.

Algorithm 3 Implement a dynamic counter to track a staircase function

Input: signal s, $\triangleright tuple < DC_j, CLK_j >$;
1: **if** s = eventArrival **then**
2: **if** $DC_j = N_j^u$ **then**
3: reset_timer(CLK_j, δ_j^u)
4: $k_j = 0$
5: **end if**
6: $DC_j \leftarrow DC_j - 1$
7: **end if**
8: **if** $s = CLK_j Timeout$ **then**
9: $DC_j \leftarrow \min(DC_j{+}1, N_j^u)$
10: reset_timer(CLK_j, δ_j^u)
11: $k_j = k_j + 1$
12: **end if**
13: **if** $DC_j < 0$ **then**
14: report_exception
15: **end if**

A dynamic counter (DC_j) can be used to track a single upper staircase function ($\overline{\alpha}_j^u$) and predict its future workload. The detailed tracking algorithm can be seen in Algo. 3 [64]. DC_j tracks the potential burst capacity, and the auxiliary variable k_j in Algo. 3 tracks the offset between the current time t and the last δ_j^u. Below shows an example of using dynamic counters for event prediction.

Example 8. *As shown in Fig. 4.2, for the PJD task with (P, J, D)=(100, 300, 20), two staircase functions, $\overline{\alpha}_1$ and $\overline{\alpha}_2$, are used to approximate the arrival curve of this PJD task. Every staircase function is tracked by a counter. Assume the real arrival event trace is shown in the event trace R^{act}. By applying Algo. 3, DC_1 tracks the arrival event trace based on $\overline{\alpha}_1$, and DC_2 tracks the arrival event trace based on $\overline{\alpha}_2$. The minimum of $DC_1(t)$ and $DC_2(t)$ is the potential activations of this task at time t.*

As shown in Fig. 4.2, the bold line shows the worst-case arrival pattern at the beginning, $DC_1(t_0) = N_1^u = 1$, $DC_2(t_0) = N_2^u = 4$. At time t_1, two events have been recorded, and dynamic counters are updated to that $DC_1(t_1) = 1$, $DC_2(t_1) = 2$. The future event prediction is shown in the dotted line, which is less than the worst-case assumption. At time t_2, dynamic counters are updated to that $DC_1(t_2) = 1$, $DC_2(t_2) = 0$ since five events arrived during $[t_0, t_2]$. The prediction shown in the solid line is further less than the one at time t_1.

The potential burst capacity $DC_j(t)$, together with staircase function, yields the following future events prediction [64]:

$$\mathcal{U}_j(\tau_i, \Delta, t) = DC_j(t) + \begin{cases} \lfloor \frac{\Delta + (t - k_j \delta_j^u)}{\delta_j^u} \rfloor & \text{if } DC_j(t) < N_j^u \\ \lfloor \frac{\Delta}{\delta_j^u} \rfloor & \text{if } DC_j(t) = N_j^u \end{cases} \tag{4.1}$$

The above function bounds future event arrivals at time t. By using the monitor to track the event trace for tasks, DC_j and k_j are known during the runtime. Therefore, $\mathcal{U}_j$ is also known. Regarding to bound the number of future event arrivals w.r.t. a complex activations pattern of task τ_i, one can simply take the minimum over all the $\mathcal{U}_j$ [64]:

$$\overline{\alpha}^u(\tau_i, \Delta, t) = \min_{j=1..n'}(\mathcal{U}_j(\tau_i, \Delta, t)). \tag{4.2}$$

Definition 8 (Deadline Service Demand [39]). *Suppose the arrival curve of future events of a task τ_i at time t is $\overline{\alpha}^u(\tau_i, \Delta, t)$. To satisfy the required relative deadline, the minimum service demand is*

$$\mathrm{dbf}^{\mathrm{F}}(\tau_i, \Delta, t) = \overline{\alpha}^u(\tau_i, \Delta - D_i, t) \cdot c_i, \tag{4.3}$$

where D_i and c_i are the relative deadline and WCET *of task τ_i.*

4.4.1.2 Backlogged Events and their Demand Bound

During the runtime, if events arrive more frequently than the rate that they can be processed, some events may be backlogged. We denote the set of unfinished events of τ_i in the backlog at time t as $\mathbf{E}(\tau_\mathbf{i}, \mathbf{t})$. Then, the number of backlogged events can be denoted as $|\mathbf{E}(\tau_\mathbf{i}, \mathbf{t})|$. For each event $e_{i,j} \in \mathbf{E}(\tau_\mathbf{i}, \mathbf{t})$, we use $D_{i,j}$ to denote its absolute deadline.

Definition 9 (Backlogged Demand [39]). *Suppose the set of unfinished events of a task τ_i in the buffer at time t are denoted as $\mathbf{E}(\tau_\mathbf{i}, \mathbf{t})$. Let $D_{i,j}$ denote the absolute deadline for event $e_{i,j} \in \mathbf{E}(\tau_\mathbf{i}, \mathbf{t})$. A backlogged demand for this task is defined as*

$$\mathrm{dbf}^{\mathrm{B}}(\tau_i, \Delta, t) = c_i \cdot \begin{cases} (j-1), & D_{i,j} - t < \Delta < D_{i,j+1} - t, \\ |\mathbf{E}(\tau_\mathbf{i}, \mathbf{t})|, & \Delta \geq D_{i,|\mathbf{E}(\tau_\mathbf{i}, \mathbf{t})|} - t, \end{cases}$$

in which, c_i is the WCET*, and $D_{i,0}$ is defined as t for brevity.*

4.4.1.3 Carry-On Event and its Demand Bound

Suppose at time t, an event of task τ_i has been released but its execution is not finished. This event is called a carry-on event. Similar to the demand of backlogged events, the demand of carry-on event is defined as follows.

Definition 10 (Carry-On Demand). *Suppose $C(\tau_i, t)$ is used to denote the left time for finishing a carry-on event of τ_i at time t, and the demand for the carry-on event is that*

$$\mathrm{dbf}^{\mathrm{C}}(\tau_i, \Delta, t) = c_i \cdot \begin{cases} 0, & \Delta < D_c - t, \\ C(\tau_i, t), & \Delta \geq D_c - t, \end{cases}$$

where D_c is the absolute deadline of this carry-on event.

With the event arrival bound and demand bound of future, backlogged and carry-on events, we can derive the task online workload and demand bound function (see Section 2.3 for their definitions).

Lemma 11. *The workload arrival curve $\alpha(\tau_i, \Delta, t)$ and demand bound function $\mathrm{dbf}(\tau_i, \Delta, t)$ of task τ_i are that,*

$$\begin{aligned} \alpha(\tau_i, \Delta, t) &= \max\big(c_i \cdot \overline{\alpha}^u(\tau_i, \Delta),\ c_i \cdot \overline{\alpha}^u(\tau_i, \Delta, t) + c_i \cdot |\mathbf{E}(\tau_\mathbf{i}, \mathbf{t})| + C(\tau_i, t)\big), \\ \mathrm{dbf}(\tau_i, \Delta, t) &= \max\big(c_i \cdot \overline{\alpha}^u(\tau_i, \Delta - D_i),\ \mathrm{dbf}^{\mathrm{F}}(\tau_i, \Delta, t) + \mathrm{dbf}^{\mathrm{B}}(\tau_i, \Delta, t) + \mathrm{dbf}^{\mathrm{C}}(\tau_i, \Delta, t)\big), \end{aligned} \tag{4.4}$$

where the denotations of the right hand side are shown in above definitions.

Proof. This proof follows the same procedure as proving Lemma 2. We omit it as it is easy to verify. □

Note that the workload arrival curve is the same as the workload bound function defined in Section 3.3. The aim of using workload arrival curve $\alpha(\tau_i, \Delta, t)$ in this chapter is to keep the consistency with the default denotations in the framework of Real-Time Calculus.

4.4.2 Schedulability Analysis Based on Real-Time Interface

To sufficiently guarantee that the task τ_i can meet its deadline from time t, the provided lower service should be greater than its demand bound function, i.e.,

$$\beta^l(\tau_i, \Delta, t) \geq \text{dbf}(\tau_i, \Delta, t). \tag{4.5}$$

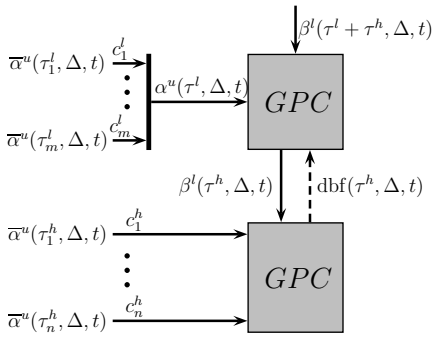


Figure 4.3: Real-Time Interface analysis

In the following, two approaches are presented to analyze the system schedulability. One is to derive the whole demand bound function of all HI-critical tasks, and guarantee that the provided lower service is greater than the whole demand bound function. This approach is often used in deriving the bound offline [61, 62, 72, 74]. The other approach is to guarantee the task schedulability individually, in which there are n inequalities to be guaranteed.

4.4.2.1 Schedulability Analsyis by Considering HI-Critical Tasks as a Group

In the initial setting, as the priority of LO-critical tasks is set as the highest, the system can be abstracted as Fig. 4.3, where

- $\alpha^u(\tau^l, \Delta, t)$ denotes the workload arrival curve of the set of all LO-critical tasks,
- $\beta^l(\tau^l + \tau^h, \Delta, t)$ denotes the provided lower service curve for all tasks,

- $\beta^l(\tau^h, \Delta, t)$ denotes the provided service for the set of all HI-critical tasks,
- and $\mathrm{dbf}(\tau^h, \Delta, t)$ denotes the demand bound function to meet deadlines of all HI-critical tasks.

Furthermore, for the easiness of the analysis in the following, we denote that

- $\tau_i^{h,n}$ represents the set of HI-critical tasks of $\tau_i^h, \tau_{i+1}^h, ..., \tau_n^h$. Then, we have $\tau_1^{h,n} = \tau^h$. By this way, $\beta^l(\tau_i^{h,n}, \Delta, t)$ and $\mathrm{dbf}(\tau_i^{h,n}, \Delta, t)$ respectively represent the lower service curve and the demand bound function for HI-critical tasks from τ_i^h to τ_n^h.

Theorem 10. *In order to guarantee that all* HI*-critical tasks are schedulable, the maximum feasible* $\alpha^u(\tau^l, \Delta, t)$ *is that,*

$$\alpha^u(\tau^l, \Delta, t) = \mathrm{RT}^{-\alpha}\Big(\mathrm{dbf}(\tau^h, \Delta, t),\ \beta^l(\tau^l + \tau^h, \Delta, t)\Big)^1. \tag{4.6}$$

Proof. To sufficiently guarantee that all HI-critical tasks can be scheduled, the provided service for the set of HI-critical tasks should be greater than their demand bound function, i.e.,

$$\beta^l(\tau^h, \Delta, t) \geq \mathrm{dbf}(\tau^h, \Delta, t). \tag{4.7}$$

According to the processing model of fixed-priority, we have

$$\beta^l(\tau^h, \Delta, t) = \mathrm{RT}\Big(\beta^l(\tau^l + \tau^h, \Delta, t), \alpha^u(\tau^l, \Delta, t)\Big). \tag{4.8}$$

By inverting Eq. 4.8, we can get that,

$$\alpha^u(\tau^l, \Delta, t) = \mathrm{RT}^{-\alpha}\Big(\beta^l(\tau^h, \Delta, t),\ \beta^l(\tau^l + \tau^h, \Delta, t)\Big).$$

Since $\beta^l(\tau^h, \Delta, t) \geq \mathrm{dbf}(\tau^h, \Delta, t)$, the maximum feasible $\alpha^u(\tau^l, \Delta, t)$ is obtained by using $\mathrm{dbf}(\tau^h, \Delta, t)$ to replace $\beta^l(\tau^h, \Delta, t)$. Hence, Eq. 4.6 holds. □

In Eq. 4.6, $\beta^l(\tau^l + \tau^h, \Delta, t)$ is also the full processing ability of the platform. As the platform is assumed to be a uniprocessor with constant processing ability, $\beta^l(\tau^l + \tau^h, \Delta, t) = \Delta$ during the runtime. Then, to get $\alpha^u(\tau^l, \Delta, t)$, one has to know $\mathrm{dbf}(\tau^h, \Delta, t)$. Based on the Real-Time Interface, $\mathrm{dbf}(\tau^h, \Delta, t)$ can be known by the backward derivation [41, 42]. The backward derivation step to get $\mathrm{dbf}(\tau^h, \Delta, t)$ is briefly introduced as follows.

As shown in Fig. 4.4, for achieving the schedulability of tasks from τ_i^h to τ_n^h, $\mathrm{dbf}(\tau_i^{h,n}, \Delta, t)$ should be greater than the demand bound function of task τ_i^h, and the provided service for tasks from τ_{i+1}^h to τ_n^h should be greater than $\mathrm{dbf}(\tau_{i+1}^{h,n}, \Delta, t)$, i.e.,

[1] $\mathrm{RT}^{-\alpha}(\beta', \beta)(\Delta) = \beta(\Delta + \lambda) - \beta'(\Delta + \lambda)$ for $\lambda = \sup\{\tau : \beta'(\Delta + \tau) = \beta'(\Delta)\}$, from the real-time interface in [41].

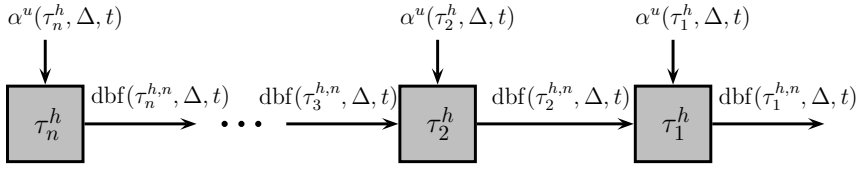


Figure 4.4: The flow of backward derivation

$$\begin{aligned} \mathrm{dbf}(\tau_i^{h,n}, \Delta, t) &\geq \mathrm{dbf}(\tau_i^h, \Delta, t), \\ \mathrm{RT}\Big(\mathrm{dbf}(\tau_i^{h,n}, \Delta, t), \alpha^u(\tau_i^h, \Delta, t)\Big) &= \beta(\tau_{i+1}^{h,n}, \Delta, t) \geq \mathrm{dbf}(\tau_{i+1}^{h,n}, \Delta, t). \end{aligned} \tag{4.9}$$

Therefore, by inverting Eq. 4.9, we can get that

$$\mathrm{dbf}(\tau_i^{h,n}, \Delta, t) = \max\Big\{\mathrm{dbf}(\tau_i^h, \Delta, t), \mathrm{RT}^{-\beta}\Big(\mathrm{dbf}(\tau_{i+1}^{h,n}, \Delta, t), \alpha^u(\tau_i^h, \Delta, t)\Big)\Big\}^1. \tag{4.10}$$

By applying Eq. 4.10 for $i = n-1, n-2, ..., 1$, the demand bound function $\mathrm{dbf}(\tau_1^{h,n}, \Delta, t)$ for the set of all HI-critical tasks is derived. Then, by applying Eq. 4.6, the maximum workload arrival curve $\alpha^u(\tau^l, \Delta, t)$ for LO-critical tasks can be computed.

4.4.2.2 Schedulability Analysis by Considering HI-Critical Tasks Separately

By considering the schedulability of each HI-critical task, we have the following theorem.

Theorem 11. *All* HI*-critical tasks are schedulable if and only if*

$$\beta^l(\tau_i^h, \Delta, t) \geq \mathrm{dbf}(\tau_i^h, \Delta, t), \ \forall\ i \in [1..n]. \tag{4.11}$$

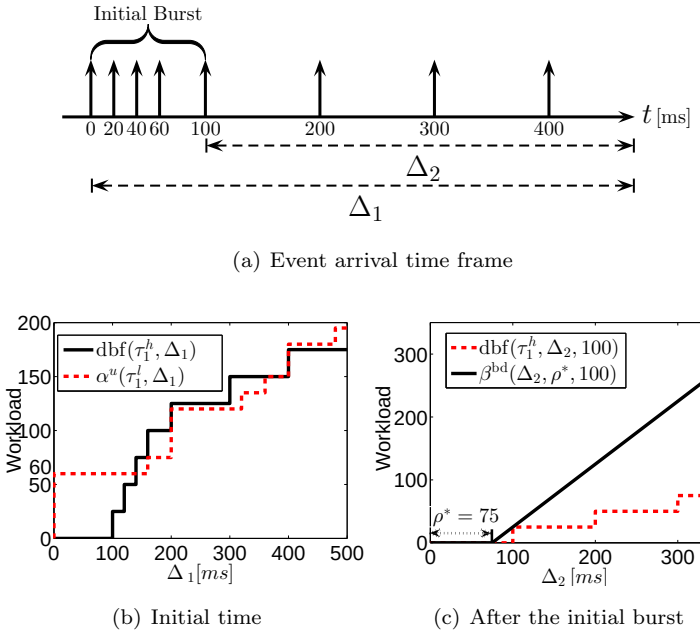
Proof. This theorem directly stems from the schedulable condition of Eq. 4.5. □

4.5 Motivation

In this section, we derive the offline bound for constraining the LO-critical workload, based on which we present an example to explain the problem.

For HI-critical tasks, as their event trace patterns are known, their workload arrival curves and demand bound functions are thus known. Then, by applying the backward derivation method, the demand bound function for the set of HI-critical tasks is thus derived. Suppose $\mathrm{dbf}(\tau^h, \Delta)$ is the offline demand bound function for scheduling the set of HI-critical tasks, and $\beta^l(\Delta)$ is the provided service of the platform. The offline bound $\alpha^u(\tau^l, \Delta)$ of LO-critical tasks is that

$$\alpha^u(\tau^l, \Delta) = \mathrm{RT}^{-\alpha}\Big(\beta^l(\Delta), \mathrm{dbf}(\tau^h, \Delta)\Big). \tag{4.12}$$



(a) Event arrival time frame

(b) Initial time

(c) After the initial burst

Figure 4.5: Motivating example

The problem of constraining the LO-critical workload by $\alpha^u(\tau^l, \Delta)$ is that, the bound is independent with the actual executions of HI-critical tasks, thus the constrained bound is sometimes too pessimistic as it ignores the fact that the actual demand of HI-critical tasks may deviate from the offline demand. We use a simple example to illustrate this problem.

Example 9. *In a mixed-criticality system, a* LO*-critical task* τ_1^l *with a high priority and a* HI*-critical task* τ_1^h *with a low priority are scheduled in a uniprocessor. The* HI*-critical task is a pjd model, which is characterized with a period* $p = 100\,ms$*, a jitter* $j = 300\,ms$*, and a minimum distance* $d = 20\,ms$*. The* WCET *is* $25\,ms$*. As shown in Fig. 4.5(a), for the* HI*-critical task, the worst-case activation pattern is an initial burst during the first* $100\,ms$*, and after that, the task is activated every* $100\,ms$*.*

Following the steps of the aforementioned analysis, the offline workload bound $\alpha^u(\tau_1^l, \Delta)$ *can be computed by applying backward derivation. The demand bound function* $\mathrm{dbf}(\tau_1^h, \Delta)$ *of the* HI*-critical task* τ^h *and workload arrival curve* $\alpha^u(\tau_1^l, \Delta)$ *of the* LO*-critical task* τ^l *(bold line) are shown in Fig. 4.5(b). It can be seen that within an interval of* $150\,ms$*, only* $60\,ms$ LO*-critical workload is allowed. Now, we consider another case. We assume that the initial burst of the* HI*-critical task happens within* $100\,ms$*, and there is no* LO*-critical workload within this* $100\,ms$*. If dynamic counters are used to predict its future events, we know the actual demand of* HI*-critical task at time* $t = 100\,ms$ *is bounded by* $\mathrm{dbf}(\tau_1^h, \Delta_2, t)$*, as shown in Fig. 4.5(c). Suppose a bounded delay function* $\beta^{\mathrm{bd}}(\Delta_2, \rho^*, 100)$ *at* $t = 100\,ms$ *is used to serve this* HI*-critical task, where* ρ^* *denotes the delayed interval. There will be no deadline miss as long as* $\beta^{\mathrm{bd}}(\Delta_2, \rho^*, 100)$

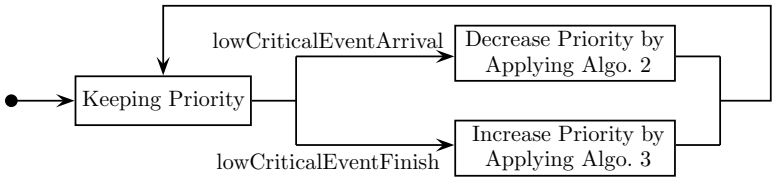


Figure 4.6: The flow of priority-adjustment policy

is greater than $\mathrm{dbf}(\tau_1^h, \Delta_2, t)$*. The largest delay is* $75\,ms$*, which indicates that the service for* HI*-critical task can be delayed for* $75\,ms$*. It also means that, within this delay interval, the* LO*-critical task can be processed. Therefore, the largest* LO*-critical workload at* $t = 100\,ms$ *can be* $75\,ms$*, instead of bounded by* $60\,ms$*.*

As this example illustrates, if the workload of LO-critical task is constrained based on the offline bound, the QoS for LO-critical tasks is not as high as the bound based on the largest delay of a bounded delay function at runtime. The approach using the largest delay to manage the LO-critical workload is similar to the energy management in [39], where the longest sleeping interval that the processor can have is updated to manage the power mode to save static energy while guaranteeing no tasks miss their deadlines. In this chapter, based on the schedulability analysis, we propose to update the actual demand bound function and workload arrival curve to manage the LO-critical workload.

4.6 LO-Critical Workload Management

By constantly updating the upper bound of constraint on LO-critical tasks, one can prevent the LO-critical interferences on HI-critical tasks either by decreasing their execution priorities or by using a shaper to shape the overloaded workload when the LO-critical workload violates the bound. In this section, we present how to apply the priority-adjustment policy and workload-shaping policy in managing the LO-critical workload at runtime.

4.6.1 Priority-Adjustment Policy

For the priority-adjustment policy, since the relative priority order of HI-critical tasks is fixed, we have to decide when to decrease or increase the priority of LO-critical tasks. All LO-critical tasks share one priority, which indicates that they are scheduled as a group. In the initial setting, the priority of LO-critical tasks is set as the highest. During the runtime, their priority is dynamically adjusted based on the actual demand bound functions of HI-critical tasks. The workflow of the priority-adjustment policy is illustrated in Fig. 4.6. Whenever a LO-critical

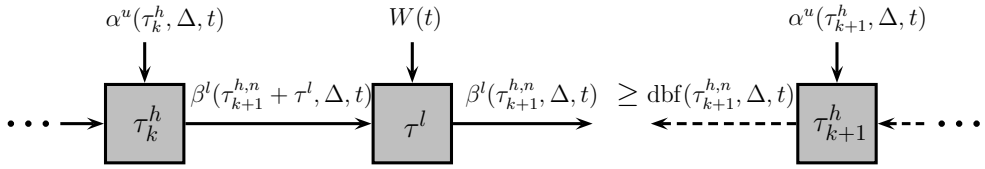


Figure 4.7: The diagram showing the verification of system scheduliability by priority-adjustment policy

event arrives or a LO-critical event is finished, the LO-critical priority will be decreased or increased, in order to not miss any deadline of HI-critical tasks.

4.6.1.1 Decreasing Priority

For the decrease of the priority of all LO-critical tasks, we have to verify whether current LO-critical workload has already violated the bound. Since the LO-critical workload is unpredictable, its future bound is unknown. Such verification only relies on the LO-critical workload that has already arrived. Therefore, every time when a new LO-critical event arrives, the schedulability verification should be done to guarantee that all HI-critical tasks can be scheduled. If the new arrival event results in other HI-critical tasks missing their deadlines, the LO-critical priority should be decreased to avoid the deadline miss. Now we formulate a general statement for the problem of how to find a feasible priority for LO-critical tasks.

Problem. *Let $P(\tau^l)$ denote the priority of the set of* LO*-critical tasks and $P(\tau_i^h)$ denote the priority of the* HI*-critical task τ_i^h. Suppose before time t, all* HI*-critical tasks are schedualable with the priorities order that $P(\tau_1^h) > ... > P(\tau_k^h) > P(\tau^l) > P(\tau_{k+1}^h) > ... > P(\tau_n^h)$. If a new* LO*-critical event arrives at time t, how to assign the priorities so that all* HI*-critical tasks can still keep schedulable.*

The principle for reassigning the priorities is that, the schedulability of HI-critical tasks should be guaranteed. The schedulability verification diagram is shown in Fig. 4.7. Suppose the LO-critical workload becomes $W(t)$ with the arrival of a new event. Then, by using the backward derivation, the demand bound function $\mathrm{dbf}(\tau_{k+1}^{h,n}, \Delta, t)$ for serving tasks from τ_{k+1}^h to τ_n^h is derived. In order to meet the deadlines of tasks $\tau_{k+1}^{h,n}$, the provided service $\beta^l(\tau_{k+1}^{h,n}, \Delta, t)$ for them should be greater than their demand bound function, that is,

$$\beta^l(\tau_{k+1}^{h,n}, \Delta, t) \geq \mathrm{dbf}(\tau_{k+1}^{h,n}, \Delta, t). \tag{4.13}$$

Algorithm 4 Procedure of priority reassignment with a new arrival LO-critical event

1: **for** $k^\flat = k+1 \to n$ **do**
2: Compute $\beta^l(\tau^{h,n}_{k^\flat+1}, \Delta, t)$ and $\mathrm{dbf}(\tau^{h,n}_{k^\flat+1}, \Delta, t)$ by Eqs. 4.10, 4.14, 4.15;
3: **if** $\beta^l(\tau^{h,n}_{k^\flat+1}, \Delta, t) \geq \mathrm{dbf}(\tau^{h,n}_{k^\flat+1}, \Delta, t)$ **then**
4: Change the priorities order to be that $P(\tau^h_1) > ... > P(\tau^h_{k^\flat}) > P(\tau^l) > P(\tau^h_{k^\flat+1}) > ... > P(\tau^h_n)$;
5: break;
6: **end if**
7: **end for**

By applying the forward computation of RT, we get the provided service for tasks $\tau^{h,n}_{k+1}$ and τ^l, which is,

$$\beta^l(\tau^{h,n}_i + \tau^l, \Delta, t) = \mathrm{RT}\Big(\beta^l(\tau^{h,n}_{i-1} + \tau^l, \Delta, t), \alpha^u(\tau^h_{i-1}, \Delta, t)\Big), \tag{4.14}$$

by applying $i = 2, ..., k+1$ and $\beta^l(\tau^h + \tau^l, \Delta, t) = \Delta$. To get $\beta^l(\tau^{h,n}_{k+1}, \Delta, t)$, the LO-critical workload should also be considered because the priority of LO-critical tasks is greater than tasks $\tau^{h,n}_{k+1}$. That is,

$$\beta^l(\tau^{h,n}_{k+1}, \Delta, t) = \beta^l(\tau^{h,n}_{k+1} + \tau^l, \Delta, t) - W(t). \tag{4.15}$$

If Eq. 4.13 does not hold, the LO-critical priority has to be decreased. In general, the Algo. 4 can be applied to find the highest feasible priority for LO-critical tasks. The procedure is that, every priority that is equal to or lower than the original LO-critical priority is a possible priority for LO-critical tasks. Therefore, Algo. 4 checks every possible priority from high to low until a feasible priority is found out. This algorithm leads to the following theorem.

Theorem 12. *Algo. 4 can always find a schedule, i.e., a priority assigned to* LO*-critical tasks, to sufficient guarantee all* HI*-critical tasks schedulable.*

Proof. Denote t^- as the time instant just before the arrival of new event, and denote t^+ as the time instant just after the arrival of new event. The demand bound functions of all HI-critical tasks are the same at t^- and at t^+. Since all HI-critical tasks are schedulable at time t^-, we have

$$\begin{aligned} \beta^l(\tau^h_i, \Delta, t^-) \geq \mathrm{dbf}(\tau^h_i, \Delta, t^-) = \mathrm{dbf}(\tau^h_i, \Delta, t^+),\ \forall\ i \in [1..k], \\ \beta^l(\tau^{h,n}_{k+1}, \Delta, t^-) \geq \mathrm{dbf}(\tau^{h,n}_{k+1}, \Delta, t^-) = \mathrm{dbf}(\tau^{h,n}_{k+1}, \Delta, t^+). \end{aligned} \tag{4.16}$$

Since the priorities of tasks from τ^h_1 to τ^h_k is greater than that of τ^l, the services for tasks from τ^h_1 to τ^h_k are not changed by the arrival of new LO-critical event, i.e., $\beta^l(\tau^h_i, \Delta, t^-) = \beta^l(\tau^h_i, \Delta, t^+),\ \forall\ i \in [1..k]$. Hence, tasks from τ^h_1 to τ^h_k are still schedulable.

With the new arrival of LO-critical event, the LO-critical workload changes from $W(t^-)$ to be $W(t^+)$. The worst case of applying Algo. 4 is to assign the lowest priority to LO-critical tasks. We now prove that the lowest priority is always a feasible priority.

Suppose the priority of LO-critical tasks is the lowest at t^+, then the processing resource $\beta^l(\tau^{h,n}_{k+1} + \tau^l, \Delta, t^+)$ will be fully provided to the task set $\tau^{h,n}_{k+1}$, and only the remaining service

after processing $\tau_{k+1}^{h,n}$ can be provided to LO-critical tasks. Therefore, we have

$$\beta^l(\tau_{k+1}^{h,n}, \Delta, t^+) = \beta^l(\tau_{k+1}^{h,n} + \tau^l, \Delta, t^+) = \beta^l(\tau_{k+1}^{h,n} + \tau^l, \Delta, t^-).$$

Because $\beta^l(\tau_{k+1}^{h,n}, \Delta, t^-) = \beta^l(\tau_{k+1}^{h,n} + \tau^l, \Delta, t^-) - W(t^-)$, we have

$$\beta^l(\tau_{k+1}^{h,n}, \Delta, t^+) = \beta^l(\tau_{k+1}^{h,n} + \tau^l, \Delta, t^-) \geq \beta^l(\tau_{k+1}^{h,n} + \tau^l, \Delta, t^-) - W(t^-) = \beta^l(\tau_{k+1}^{h,n}, \Delta, t^-)$$

Because of Eq. 4.16, we can get that $\beta^l(\tau_{k+1}^{h,n}, \Delta, t^+) \geq \mathrm{dbf}(\tau_{k+1}^{h,n}, \Delta, t^+)$. Hence, we have proved that the lowest priority is always a feasible priority to be assigned to LO-critical tasks and all tasks in $\tau_{k+1}^{h,n}$ are also schedulable. □

4.6.1.2 Increasing Priority

In our policy, we suppose that the LO-critical priority can be increased when the LO-critical workload is decreased. It indicates that, whenever a LO-critical event has been finished, the LO-critical priority can be increased. This is because, the interference imposed on HI-critical tasks with priorities lower than the LO-critical priority also decreases. The algorithm of increasing the LO-critical priority is similar to the algorithm of decreasing the LO-critical priority. As shown in Algo. 5, Eq. 4.13 is used to guarantee that the increased priority will not result in a deadline miss of all lower-priority tasks. The chosen priority gradually increases until an infeasible priority is found out. The reason for gradually increasing the chosen priority in Algo. 5 or gradually decreasing the chosen priority in Algo. 4, instead of searching the priority from the highest to the lowest in Algo. 5, or the lowest to the highest in Algo. 4, is that the LO-critical priority is supposed to keep unchanged with the finish or the arrival of the LO-critical event.

Theorem 13. *Algo. 5 can always find a schedule, i.e., a priority assigned to* LO*-critical tasks, to sufficient guarantee all* HI*-critical tasks schedulable.*

We omit the proof due to the similarity to theorem 12. The worst case of applying Algo. 5 is to keep the LO-critical priority unchanged.

4.6.1.3 Runtime Behavior

The monitors in this system setting are only used to monitor the arrival events without any interference on their releases. The priority controller is only responsible for reassigning the priorities. All events are scheduled based on their assigned priorities. We assume, for every priorities reassignment, there is an instant interrupt to force the processor to process events based on the new priorities order. As there are multiple streams of LO-critical events and all of them share one priority, they are served based on the principle of first-come-first-serve.

Algorithm 5 Procedure of priority reassignment after finishing a LO-critical event

1: **for** $k^\flat = k \to 1$ **do**
2: Compute $\beta^l(\tau_{k^\flat}^{h,n}, \Delta, t)$ and $\mathrm{dbf}(\tau_{k^\flat}^{h,n}, \Delta, t)$ by Eqs. 4.10, 4.14, 4.15;
3: **if** $\beta^l(\tau_{k^\flat}^{h,n}, \Delta, t) < \mathrm{dbf}(\tau_{k^\flat}^{h,n}, \Delta, t)$ **then**
4: Change the priorities order to be that $P(\tau_1^h) > ... > P(\tau_{k^\flat+1}^h) > P(\tau^l) > P(\tau_{k^\flat}^h) > ... > P(\tau_n^h)$;
5: break;
6: **end if**
7: **end for**

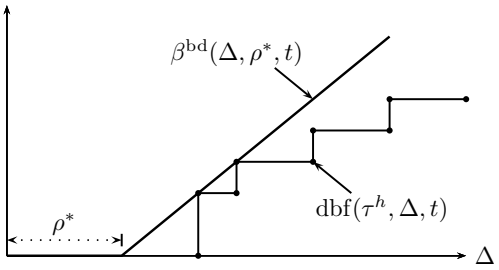


Figure 4.8: An illustration for the LFII

4.6.2 Workload-Shaping Policy

In contrast to the workload management by changing the execution priorities, the workload-shaping policy manages the LO-critical workload by using a shaper to regulate the inflow of LO-critical events. The priority of LO-critical tasks is constantly set as the highest.

4.6.2.1 The Release of an Event

To ensure that the released event should keep the schedulability of all HI-critical tasks, the WCET of a released event should not be larger than the longest feasible interference interval (LFII).

Definition 11 (Longest Feasible Interference Interval)**.** *The longest feasible interference interval* $\rho^*(t)$ *with respect to a given demand bound function* $\mathrm{dbf}(\tau^h, \Delta, t)$ *is defined as:*

$$\rho^*(t) = \max\{\rho^* : \beta^{\mathrm{bd}}(\Delta, \rho^*, t) \geq \mathrm{dbf}(\tau^h, \Delta, t),\ \forall \Delta \geq 0\}. \tag{4.17}$$

where $\beta^{\mathrm{bd}}(\Delta, \rho^*, t)$ *is a bounded-delay service curve that* $\beta^{\mathrm{bd}}(\Delta, \rho^*, t) = \max\{0, (\Delta - \rho^*)\},\ \forall \Delta \geq 0$.

The LFII is illustrated in Fig. 4.8, where $\mathrm{dbf}(\tau^h, \Delta, t)$ can be computed by using the backward derivation. $\beta^{\mathrm{bd}}(\Delta, \rho^*, t)$ means that the processing service is delayed for ρ^*. In this case, if the WCET of released event is smaller than the LFII, all HI-critical tasks can still

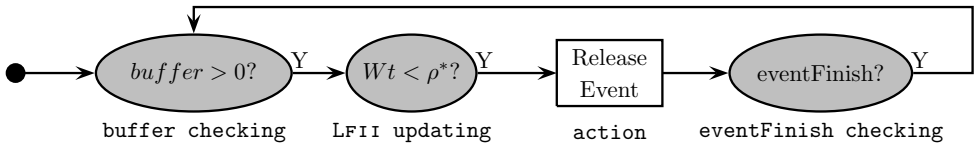


Figure 4.9: The flow of workload-shaping policy

be schedulable as the provided service for them is larger than $\beta^{\mathrm{bd}}(\Delta, \rho^*, t)$. Therefore, during the runtime, by constantly computing the LFII, the shaper decides how to release the arrival LO-critical events. For every update of the LFII, the backward derivation has to be applied in deriving the $\mathrm{dbf}(\tau^h, \Delta, t)$, then the binary searching is used to search the maximum feasible LFII.

4.6.2.2 The Adaptive Shaping Flow

The flow of workload-shaping policy is seen in Fig. 4.9. The shaper has three states and one action, which are states of `buffer checking`, `LFII updating`, `eventFinish checking`, and the action of `release event`.

At the beginning, the shaper stays in `buffer checking`, in which the shaper constantly checks the emergence of an event in the buffer. Whenever there is an event in the buffer, the shaper transits to the `LFII updating`, in which the shaper updates the LFII, and compares it with the WCET of the event that is to be released. Once the LFII is greater than the WCET, this event is released, and the shaper transits to the `eventFinish checking`. The shaper stays in this state until the released event is finished, then the shaper transits back to the `buffer checking`.

When the shaper is in the `LFII updating`, the time for updating the LFII is based on the execution of HI-critical events. The shaper is designed to update the LFII at the time when a HI-critical event is finished. If the updated LFII is still less than the WCET of the event to be released, the LFII will be updated again when the next HI-critical event is finished. The time setting for updating LFII is based on the fact that the demand of HI-critical tasks will decrease if any HI-critical event is finished, thus making the LFII greater.

4.7 A Lightweight Method

Both of schedulability analyzing approaches in Section 4.4 rely on the heavy computation, which prohibits their applications in the online cases. The computational overhead originates from

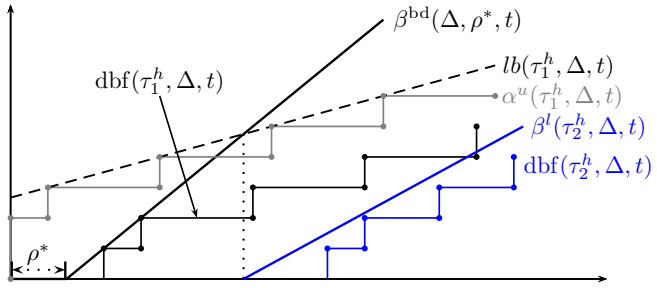


Figure 4.10: The scheme for illustrating the schedulability analysis of two tasks

three parts, i.e., RT computation, backward derivation, and the binary search for the LFII. To eliminate the heavy computation, a lightweight method is proposed in this section.

4.7.1 The Scenario of Setting the LO-Critical Priority as the Highest

Our mixed-criticality settings include a set of LO-critical tasks and a set of HI-critical tasks. To simplify the problem, we discuss how to compute a bound of LO-critical workload in the scenario where the LO-critical priority is higher than the priorities of all HI-critical tasks. Suppose such a bound is ρ^*. A feasible ρ^* should guarantee that all HI-critical tasks can be scheduled. Hence, the problem of obtaining the maximum ρ^* is that,

$$\begin{aligned} &\textbf{maximize} \quad \rho^*(t), \\ &\textbf{s.t.} \quad \beta^l(\tau_i^h, \Delta, t) \geq \mathrm{dbf}(\tau_i^h, \Delta, t), \ \forall\ i \in [1..n], && (4.18) \\ \textbf{or } &\textbf{s.t.} \quad \beta^l(\tau^h, \Delta, t) \geq \mathrm{dbf}(\tau^h, \Delta, t). && (4.19) \end{aligned}$$

This is a maximization problem with the constraint Eq. 4.18 or the constraint Eq. 4.19. The constraint Eq. 4.18 relies on solving n inequalities with forward RT computations, and the constraint Eq. 4.19 relies on backward deriving the group demand bound function. Both methods need to compute the complex (de-)convolution many times. To eliminate such complex computations, a leaky bucket [46] is proposed to represent the workload arrival curve and a closed-form equation is derived for representing the provided service. In the following, we introduce the closed-form equation by analyzing a system with only two HI-critical tasks.

4.7.1.1 Case for a System with Only Two HI-Critical Tasks

Given two HI-critical tasks τ_1^h and τ_2^h in a uniprocessor. Supposing at time t, their workload arrival curves are $\alpha^u(\tau_1^h, \Delta, t)$ and $\alpha^u(\tau_2^h, \Delta, t)$, and their demand bound functions are

$\mathrm{dbf}(\tau_1^h, \Delta, t)$ and $\mathrm{dbf}(\tau_2^h, \Delta, t)$. Suppose the provided service is a bounded delay function $\beta^{\mathrm{bd}}(\Delta, \rho^*, t) = \max\{0, \Delta - \rho^*\}$. As shown in Fig. 4.10, to meet the deadlines of the tasks τ_1^h and τ_2^h, the following inequalities should hold:

$$\beta^l(\tau_1^h, \Delta, t) \geq \mathrm{dbf}(\tau_1^h, \Delta, t), \quad \beta^l(\tau_2^h, \Delta, t) \geq \mathrm{dbf}(\tau_2^h, \Delta, t).$$

where

$$\beta^l(\tau_1^h, \Delta, t) = \beta^{\mathrm{bd}}(\Delta, \rho^*, t), \quad \beta^l(\tau_2^h, \Delta, t) = \mathrm{RT}\Big(\beta^l(\tau_1^h, \Delta, t), \alpha^u(\tau_1^h, \Delta, t)\Big).$$

If a leaky bucket $\mathrm{lb}(\tau_1^h, \Delta, t) = b_1(t) + r_1(t) \cdot \Delta$ is used to represent its workload arrival curve $\alpha^u(\tau_1^h, \Delta, t)$, then,

$$\beta^l(\tau_2^h, \Delta, t) = \mathrm{RT}\Big(\beta^l(\tau_1^h, \Delta, t), \mathrm{lb}(\tau_1^h, \Delta, t)\Big) = \sup_{0 \leq \lambda \leq \Delta} \Big\{ \max\{0, \lambda - \rho^*\} - b_1(t) - r_1(t) \cdot \lambda \Big\}.$$

As $\beta^l(\tau_2^h, \Delta, t) \geq 0$, we abbreviate $\beta^l(\tau_2^h, \Delta, t)$ as follows

$$\beta^l(\tau_2^h, \Delta, t) = \max\big\{0, (1 - r_1(t)) \cdot \Delta - \rho^* - b_1(t)\big\}.$$

Then, to keep the schedulability of both tasks, one only needs to guarantee the following two inequalities be true,

$$\max\{0, \Delta - \rho^*\} \geq \mathrm{dbf}(\tau_1^h, \Delta, t), \quad \max\big\{0, (1 - r_1(t)) \cdot \Delta - \rho^* - b_1(t)\big\} \geq \mathrm{dbf}(\tau_2^h, \Delta, t).$$

In this case, by using the leaky bucket to represent the original workload arrival curve, both $\beta^l(\tau_1^h, \Delta, t)$ and $\beta^l(\tau_2^h, \Delta, t)$ are derived to be rate-latency functions [46]. The rates w.r.t. τ_1^h and τ_2^h are 1 and $1 - r_1(t)$, and the latencies w.r.t. τ_1^h and τ_2^h are ρ^* and $(\rho^* + b_1(t))/(1 - r_1(t))$. Actually, for a system with more than two HI-critical tasks, if all workload arrival curves are represented by leaky buckets, the provided service for every HI-critical task can be derived to be a closed-form equation. This closed-form euqation is also a rate-latency function.

4.7.1.2 Closed-Form Equation for the Provided Service

Analogous to the schedulability analysis with only two HI-critical tasks, a similar procedure can be taken for analyzing a system with more than two HI-critical tasks. As analyzed in Section 4.4, the following inequality sufficiently guarantees that all HI-critical tasks can meet their deadlines.

$$\beta^l(\tau_i^h, \Delta, t) \geq \mathrm{dbf}(\tau_i^h, \Delta, t), \ \forall\, i \in [1..n].$$

To get $\beta^l(\tau_i^h, \Delta, t)$, a step-by-step forward RT computation should be used, which would block the computation speed. To remove this step-by-step computation, a closed-form equation is derived to represent the $\beta^l(\tau_i^h, \Delta, t)$.

Theorem 14. *In a system with n* HI*-critical tasks, the demand bound function* $\mathrm{dbf}(\tau_i^h, \Delta, t)$ *and workload arrival curve* $\alpha^u(\tau_i^h, \Delta, t)$ *are known at runtime. If a leaky bucket with the form of* $\mathrm{lb}(\tau_i^h, \Delta, t) = r_i(t) \cdot \Delta + b_i(t)$ *is used to conservatively represent* $\alpha^u(\tau_i^h, \Delta, t)$ *, i.e.,* $\mathrm{lb}(\tau_i^h, \Delta, t) \geq \alpha^u(\tau_i^h, \Delta, t)$*, the provided service* $\beta^l(\tau_i^h, \Delta, t)$ *for each task is as follows:*

$$\beta^l(\tau_i^h, \Delta, t) = \max\{0, (1 - R_i(t)) \cdot \Delta - \rho^* - B_i(t)\}, \tag{4.20}$$

where $R_i(t) = \sum_{j=1}^{i-1} r_j(t)$*,* $B_i(t) = \sum_{j=1}^{i-1} b_j(t)$*, and* $r_0(t) = 0$*,* $b_0(t) = 0$ *for brevity.*

Proof. We prove this by induction.

If $n = 1$,

$$\beta^l(\tau_1^h, \Delta, t) = \max\{0, (1 - R_1(t)) \cdot \Delta - \rho^* - B_1(t)\} = \max\{0, \Delta - \rho^*\} = \beta^{\mathrm{bd}}(\Delta, \rho^*),$$

which is true.

We assume that Eq. 4.20 is true for the task τ_n^h ($n \neq 1$). Then, for the task τ_{n+1}^h, by using the Real-Time Interface analysis,

$$\beta^l(\tau_{n+1}^h, \Delta, t) = \sup_{0 \leq \lambda \leq \Delta} \Big\{ \max\{0, (1 - R_n(t)) \cdot \Delta - \rho^* - B_n(t)\} - b_n(t) - r_n(t) \cdot \lambda \Big\}.$$

As $\beta^l(\tau_{n+1}^h, \Delta, t) \geq 0$, $\beta^l(\tau_{n+1}^h, \Delta, t)$ can be rewritten as follows for brevity,

$$\beta^l(\tau_{n+1}^h, \Delta, t) = \max\big\{0, (1 - R_{n+1}(t)) \cdot \Delta - \rho^* - B_{n+1}(t)\big\}$$

□

Corollary 2. *By using the conservative representation* $\mathrm{lb}(\tau_i^h, \Delta, t) \geq \alpha^u(\tau_i^h, \Delta, t)$ *to compute* $\beta^l(\tau_i^h, \Delta, t)$*, under the constraint of Eq. 4.18, all* HI*-critical tasks can meet their deadlines.*

Proof. Let $\beta^{ACT}(\tau_i^h, \Delta, t)$ denote the actual provided service for task τ_i^h. As $\beta^l(\tau_i^h, \Delta, t) \geq \mathrm{dbf}(\tau_i^h, \Delta, t)$ is true for all HI-critical tasks, one only needs to prove that the actual service $\beta^{ACT}(\tau_i^h, \Delta, t)$ is equal to or greater than $\beta^l(\tau_i^h, \Delta, t)$. We prove this also by induction.

When $n = 1$,

$$\beta^{ACT}(\tau_1^h, \Delta, t) = \beta^{\mathrm{bd}}(\tau_1^h, \Delta, t) = \beta^l(\tau_1^h, \Delta, t).$$

We assume $\beta^{ACT}(\tau_i^h, \Delta, t) \geq \beta^l(\tau_i^h, \Delta, t)$ is true for the task τ_n^h ($n \neq 1$). Then, for the task τ_{n+1}^h, we have

$$\beta^{ACT}(\tau_{n+1}^h, \Delta, t) = \mathrm{RT}\Big(\beta^{ACT}(\tau_n^h, \Delta, t), \alpha^u(\tau_n^h, \Delta, t)\Big) = \sup_{0 \leq \lambda \leq \Delta} \Big\{\beta^{ACT}(\tau_n^h, \lambda, t) - \alpha^u(\tau_n^h, \lambda, t)\Big\}.$$

As $\beta^{ACT}(\tau_n^h, \lambda, t) \geq \beta^l(\tau_n^h, \lambda, t)$ and $\mathrm{lb}(\tau_n^h, \lambda, t) \geq \alpha^u(\tau_n^h, \lambda, t)$, we have

$$\beta^{ACT}(\tau_{n+1}^h, \Delta, t) \geq \sup_{0 \leq \lambda \leq \Delta} \Big\{\beta^l(\tau_n^h, \lambda, t) - \mathrm{lb}(\tau_n^h, \lambda, t)\Big\} = \beta^l(\tau_{n+1}^h, \Delta, t).$$

Therefore, as $\beta^{ACT}(\tau_i^h, \Delta, t) \geq \beta^l(\tau_i^h, \Delta, t) \geq \mathrm{dbf}(\tau_i^h, \Delta, t)$ is true for all tasks, all deadlines can be met. □

4.7.1.3 Leaky Bucket Representation

From the Eqs. 4.4,4.1,4.2, we know the composition of $\alpha^u(\tau_i^h, \Delta, t)$ is the minimum of a set of staircase functions, and every staircase function is tracked by a dynamic counter. In principle, any leaky bucket can be used as long as this leaky bucket is equal to or greater than $\alpha^u(\tau_i^h, \Delta, t)$. But, in order to make our computation more tight, the leaky bucket should be as close to $\alpha^u(\tau_i^h, \Delta, t)$ as possible. Since the leaky bucket corresponding to the staircase function with the largest stair length in $\mathcal{U}(\tau_i^h, \Delta, t)$ is close to $\alpha^u(\tau_i^h, \Delta, t)$ in the long term, the staircase function with the largest stair length is thus used to compose a leaky bucket to represent the workload arrival curve.

Since the largest stair length in a workload arrival curve and the WCET of each task is known offline and unchanged during the runtime. Therefore, the leaky rate $r_i(t)$ is fixed to be $c_i/\delta_i^{\#}$, where $\delta_i^{\#}$ is the largest stair length. Hence, $1 - R_i(t)$ in Eq. 4.20 is also known offline and fixed during the runtime. Then, to get $\beta^l(\tau_i^h, \Delta, t)$, one only needs to know the bucket size $b_i(t)$. Suppose for the task τ_i^h, $DC_i^{\#}$ is the counter for tracking the chosen staircase function in Algo. 3. At time t, it can be derived from Eqs. 4.4, 4.1, 4.2 that

$$b_i(t) = C(\tau_i, t) + c_i \cdot |\mathbf{E}(\tau_\mathbf{i}, \mathbf{t})| + c_i \cdot \begin{cases} DC_i^\sharp + \frac{t - k_i^\sharp \cdot \delta_i^\#}{\delta_i^\#} & \text{if } DC_i^\sharp < N_i^{\sharp u} \\ DC_i^\sharp & \text{if } DC_i^\sharp = N_i^{\sharp u} \end{cases} \tag{4.21}$$

where $k_i^\sharp$ is the auxiliary variable corresponding with $DC_i^\sharp$ in Algo. 3. Then, $b_i(t)$ is easy to get by just applying Eq. 4.21. $\beta^l(\tau_i^h, \Delta, t)$ can also be conveniently obtained as $R_i(t)$ is fixed and $B_i(t)$ is easy to obtain with the support of Eq. 4.21.

4.7.1.4 Computing $\rho^*(t)$

To solve the maximization problem with the constraint of Eq. 4.18, the first step is to use current parameters in monitors to update $\beta^l(\tau_i^h, \Delta, t)$ and $\mathrm{dbf}(\tau_i^h, \Delta, t)$. In this comparison, only a limited segment of Δ needs to be compared. If $\beta^l(\tau_i^h, \Delta, t) \geq \mathrm{dbf}(\tau_i^h, \Delta, t)$ in this limited segment, $\beta^l(\tau_i^h, \Delta, t) \geq \mathrm{dbf}(\tau_i^h, \Delta, t)$ in any interval Δ.

In a schedulable system, suppose for a HI-critical task τ_i^h, at time t, there are $|\mathbf{E}(\tau_\mathbf{i}, \mathbf{t})|$ events that are backlogged in the buffer, and the absolute deadline trace is $D_{i,j}$, where j indicates the j-th event, as shown in Fig. 4.11. c_i is the WCET for this HI-critical task τ_i^h.

Lemma 12. *If $D_{i,x+1} - D_{i,x} = \delta_i^{max}$, where δ_i^{max} is the maximum stair length and $x > |\mathbf{E}(\tau_\mathbf{i}, \mathbf{t})|$, then for the absolute deadline of k-th event ($k \geq x$), we have*

$$D_{i,k+1} - D_{i,k} = \delta_i^{max}. \tag{4.22}$$

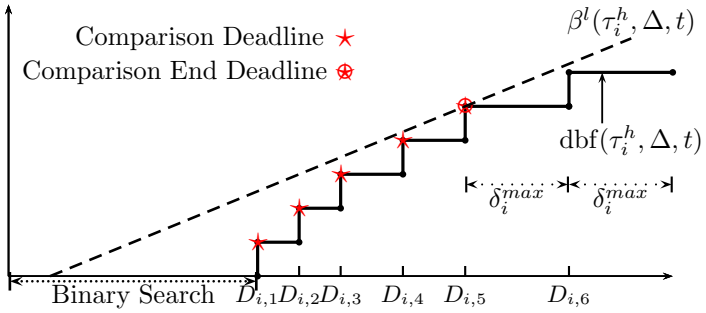


Figure 4.11: An illustration how to do the comparison

Proof. From Eqs. 4.4, 4.2, we know the deadline trace is decided by $\min_{j=1..n'}(\mathcal{U}_j(\tau_i^h, \Delta, t))$ and $B(\tau_i^h, \Delta, t)$. As $x > |\mathbf{E}(\tau_\mathbf{i}, \mathbf{t})|$, the absolute deadline for the x-th event only depends on $\min_{j=1..n'}(\mathcal{U}_j(\tau_i^h, \Delta, t))$. $\min_{j=1..n'}(\mathcal{U}_j(\tau_i^h, \Delta, t))$ is convex and is the minimum over all staircase functions. For the x-th and $(x+1)$-th events, if $D_{i,x+1} - D_{i,x} = \delta_i^{max}$, it indicates that $\min_{j=1..n'}(\mathcal{U}_j(\tau_i^h, \Delta, t))$ only depends on the staircase function with the largest stair length. For the k-th event $(k \geq x)$, $\min_{j=1..n'}(\mathcal{U}_j(\tau_i^h, \Delta, t))$ also depends on the staircase function with the largest stair length. Hence, $D_{i,k+1} - D_{i,k} = \delta_i^{max}$. □

Theorem 15. *Suppose the task τ_i^h is schedulable with a rate-latency function $\beta^l(\tau_i^h, \Delta, t)$. For the x-th and k-th event in lemma 12, if $\beta^l(\tau_i^h, D_{i,x}, t) \geq \mathrm{dbf}(\tau_i^h, D_{i,x}, t) \geq 0$, we have $\beta^l(\tau_i^h, D_{i,k}, t) \geq \mathrm{dbf}(\tau_i^h, D_{i,k}, t)$.*

Proof. Assume $\beta^l(\tau_i^h, \Delta, t) = \max\{0, r \cdot \Delta + b\}$, as $\beta^l(\tau_i^h, D_{i,x}, t) \geq \mathrm{dbf}(\tau_i^h, D_{i,x}, t) \geq 0$, that is,

$$r \cdot D_{i,x} + b \geq \mathrm{dbf}(\tau_i^h, D_{i,x}, t),$$
$$r \cdot D_{i,x} + b + (k-x) \cdot c_i \geq \mathrm{dbf}(\tau_i^h, D_{i,x}, t) + (k-x) \cdot c_i.$$

From lemma 12, as $D_{i,k+1} - D_{i,k} = \delta_i^{max}$, $\mathrm{dbf}(\tau_i^h, D_{i,k+1}, t) - \mathrm{dbf}(\tau_i^h, D_{i,k}, t) = c_i$. We have

$$\mathrm{dbf}(\tau_i^h, D_{i,k}, t) = \mathrm{dbf}(\tau_i^h, D_{i,x}, t) + (k-x) \cdot c_i.$$

As $r > \frac{c_i}{\delta_i^{max}}$ for a schedulable system, we have

$$r \cdot D_{i,x} + b + (k-x) \cdot c_i \leq r \cdot D_{i,x} + b + r \cdot (k-x) \cdot \delta_i^{max} \leq r \cdot D_{i,k} + b$$

Therefore, $\beta^l(\tau_i^h, D_{i,k}, t) = r \cdot D_{i,k} + b \geq \mathrm{dbf}(\tau_i^h, D_{i,k}, t)$. □

Theorem 15 indicates that, if a rate-latency function $\beta^l(\tau_i^h, \Delta, t) \geq \mathrm{dbf}(\tau_i^h, \Delta, t)$ within an interval of $[0, D_{i,x}]$, $\beta^l(\tau_i^h, \Delta, t)$ will be greater than $\mathrm{dbf}(\tau_i^h, \Delta, t)$ in any interval. In this paper, we define the earliest deadline that satisfies Eq. 4.22 as the comparison end deadline, as shown in Fig. 4.11. With the theorem 15, to do the comparison of Eq. 4.18, one only needs to compare $\beta^l(\tau_i^h, \Delta, t)$ with $\mathrm{dbf}(\tau_i^h, \Delta, t)$ before the comparison end deadline. For example, as shown in

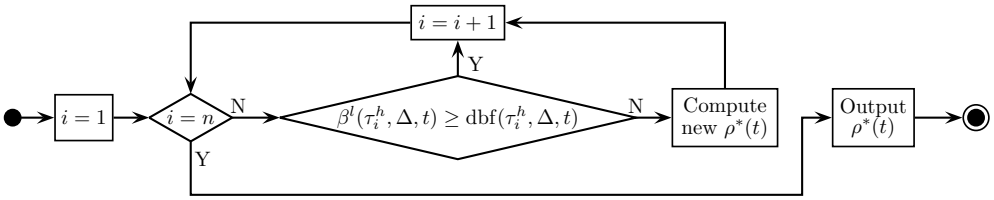


Figure 4.12: The program diagram to compute the maximum $\rho^*(t)$ with the constraint of n inequalities

Fig. 4.11, there are only 5 deadlines before the comparison end deadline. If $\beta^l(\tau_i^h, D_{i,j}, t)$ is greater than $\mathrm{dbf}(\tau_i^h, D_{i,j}, t)$ in these 5 deadlines, this rate-latency function in other deadlines is also greater than the $\mathrm{dbf}(\tau_i^h, D_{i,j}, t)$. As $0 \leq \rho^*(t) \leq D_{i,1}$, we use the binary search to get the maximum $\rho^*(t)$. The computing complexity for one HI-critical task is $O(log(n))$.

In short, for our proposed lightweight scheme, n inequalities need to be solved. As we only need to get the maximum $\rho^*(t)$ that makes all inequalities hold, it is not necessary to compute $\rho^*(t)$ for every inequality. The bubble sorting with one iteration can be used to pick out the maximum $\rho^*(t)$. It works like this, as shown in Fig. 4.12, we start the searching with a very large $\rho^*(t)$. As this large $\rho^*(t)$ will make $\beta^l(\tau_1^h, \Delta, t) < \mathrm{dbf}(\tau_1^h, \Delta, t)$, a new $\rho^*(t)$ will be computed by solving $\beta^l(\tau_1^h, \Delta, t) \geq \mathrm{dbf}(\tau_1^h, \Delta, t)$. This $\rho^*(t)$ is used in the second inequality. If the rate-latency function $\beta^l(\tau_2^h, \Delta, t)$ with $\rho^*(t)$ is greater than $\mathrm{dbf}(\tau_2^h, \Delta, t)$, this $\rho^*(t)$ is used in the third inequality. If not, we compute a new $\rho^*(t)$ of the second inequality, and use this new $\rho^*(t)$ in the third inequality. $\rho^*(t)$ is computed in this way until the last inequality. The computed $\rho^*(t)$ is the maximum LFII that satisfies all inequalities. The whole computing complexity is $O(n \cdot log(n))$.

4.7.2 The Lightweight Method in Workload Management Policies

We have introduced a lightweight method to compute the LO-critical workload bound in the scenario where the priority of LO-critical tasks is set as the highest. In this section, we analyze how to apply such lightweight method to our two proposed workload management policies.

4.7.2.1 The Lightweight Method in the Priority-Adjustment Policy

The problem of priority-adjustment policy is how to search a feasible priority for LO-critical tasks, that is, to verify the system schedulability for every possible priorities assignment. Suppose the priorities assignment is that $P(\tau_1^h) > ... > P(\tau_k^h) > P(\tau^l) > P(\tau_{k+1}^h) > ... > P(\tau_n^h)$.

Under this setting, we present how the proposed lightweight method can be used to verify the system schedulability.

Based on this priorities assignment, it can be derived that, by using the leaky bucket to represent the workload arrival curve for every task, the provided services for HI-critical tasks are that,

$$\beta^l(\tau_i^h, \Delta, t) = \begin{cases} \max\{0, (1 - R_i(t)) \cdot \Delta - B_i(t)\} & \text{if } i \leq k \\ \max\{0, (1 - R_i(t)) \cdot \Delta - B_i(t) - W(t)\} & \text{if } i > k \end{cases}, \tag{4.23}$$

where $R_i(t)$, $B_i(t)$ are the same as Eq. 4.20, and $W(t)$ is the workload of LO-critical tasks at time t. Except the priorities setting, the derivation of Eq. 4.23 is the same as Eq. 4.20. For tasks τ_i^h where $i \leq k$, there is no interference from LO-critical tasks, thus $\rho^* = 0$. For tasks τ_i^h where $i > k$, there is the interference of $W(t)$ from LO-critical tasks. Such interference $W(t)$ can be considered as a leaky bucket $\text{lb}(\tau^l, \Delta, t) = W(t) + 0 \cdot \Delta$. Therefore, the closed-form equation is still applicable.

With $\beta^l(\tau_i^h, \Delta, t)$ of Eq. 4.23, if $\beta^l(\tau_i^h, \Delta, t) \geq \text{dbf}(\tau_i^h, \Delta, t),\ \forall\ i \in [1..n]$ does not hold, the HI-critical tasks are not schedulable, and the LO-critical priority should be decreased, until $\beta^l(\tau_i^h, \Delta, t) \geq \text{dbf}(\tau_i^h, \Delta, t),\ \forall\ i \in [1..n]$ holds.

4.7.2.2 The Lightweight Method in the Workload-Shaping Policy

The priorities setting in the workload-shaping policy is the same as the scenario in Section 4.7.1. Since the maximum $\rho^*(t)$ can be computed with a low overhead in the scenario, such $\rho^*(t)$ is used in managing the LO-critical workload by the shaper.

4.8 Implementation and evaluation

In this section, we evaluate the two proposed adaptive workload management policies and compare their performances with the offline approaches. The simulator is implemented in MATLAB by applying MPA and RTC/S tools [69] on a simulation host with Intel i7-4770 processor and 16 GB RAM.

4.8.1 Evaluation Setup

We use the system shown in Fig. 4.1 for our experiments. The model contains a set of LO-critical tasks and a set of HI-critical tasks, where each task set contains 5 independent tasks.

The activation pattern of HI-critical tasks is a pjd model whose event arrival curve is shown in Eq. 3.1. For any HI-critical task τ_i^h, the period p_i is a random integer from $[100, 300]\,ms$.

The jitter j_i is set to be equal to period. The distance d_i is set to be an random integer from $[0, p]\, ms$. The relative deadline D_i is set to be equal to the period p_i. Each task utilization $U(\tau_i^h)$ and a task set utilization $U(\tau^h)$ are defined as $U(\tau_i^h) = c_i/p_i,\ U(\tau^h) = \sum_{i=1}^{n} c_i/p_i$, where c_i is the task WCET and n is the task number in a task set. Task utilizations are generated using the UUnifast [51], giving an unbiased distribution of utilization values. The WCET is set based on the utilization and selected period, i.e., $c_i = p_i \cdot U(\tau_i^h)$. The priorities assignment among HI-critical tasks follows the principle of ensuring no HI-critical task misses in the case that no LO-critical interference exists. The Audsley's algorithm [7] is applied to assign feasible priorities to all HI-critical tasks. In the case that no feasible priorities assignment is found for a task set, this task set is dropped and a new task set is generated until a feasible priorities assignment is found. In the simulation, there are four types of HI-critical task set. The first type is named Type 1 whose utilization is 0.2. Successively, the second type is Type 2 with utilization 0.3, the third type is Type 3 with utilization 0.4, and the fourth type is Type 4 with utilization 0.5.

The LO-critical tasks are activated sporadically, while their mean inter-arrival rates were chosen such that together they impose an additional utilization. Their mean inter-arrival intervals are in the interval $[50, 100]$. Denote the $U(\tau^l)$ as the utilization of the LO-critical task set. All LO-critical events follow the principle of first-come-first-service, so there is no individual priority for the LO-critical task. We don't set the deadlines for LO-critical events, and don't drop any LO-critical events at runtime. For a uniprocessor system, the utilization cannot exceed 1. Since the utilizations of the four HI-critical task sets are $0.2, 0.3, 0.4, 0.5$, the ranges of LO-critical utilization $U(\tau^l)$ w.r.t. Type 1, Type 2, Type 3, Type 4 are set to be $[0.1, 0.8], [0.1, 0.7], [0.1, 0.6], [0.1, 0.5]$, in order to constrain the utilization of all tasks within 1.

In this work, six workload management approaches are evaluated, as shown in the following:

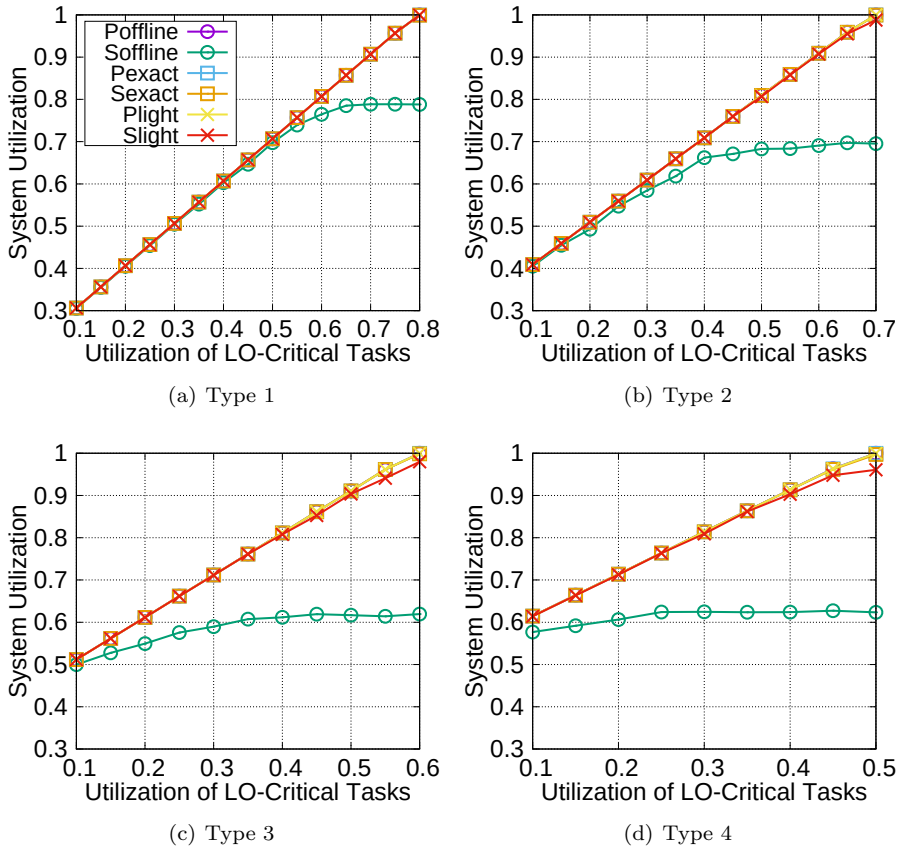
- Poffline: By setting the LO-critical priority as the lowest, there is no interference on HI-critical tasks. In this approach, the priority controller or shaper is not necessary.
- Soffline: By setting the LO-critical priority as the highest, the LO-critical workload is shaped to comply with the bound that is computed offline.
- Pexact: Applying the priority-adjustment policy proposed in Section 4.6.1 to manage the LO-critical workload. For searching the feasible priorities assignment in this policy, the exact RT and backward computation are applied.

- Sexact: By setting the LO-critical priority as the highest, the shaping policy proposed in Section 4.6.2 is used to adaptively shape the LO-critical workload so that no HI-critical tasks will miss their deadlines. The shaping bound is updated by applying the exact backward computation.
- Plight: In contrast to the Pexact method by applying the exact RT and backward computation, the lightweight method proposed in Section 4.7 is used to do the priority adjustment in this approach.
- Slight: In contrast to the Sexact method that applies the exact backward computation, the lightweight method proposed in Section 4.7 is used to update the shaping bound in this approach.

For every simulation, we simulate HI-critical event traces with a 10 sec time span. In order to evaluate the performance of different workload management approaches, the following four metrics are used.

- System Utilization U_s: Suppose t_u is the cpu time within 10 sec that is used to process tasks. System utilization U_s is referred to $t_u/10$. U_s represents how much extent that the system processing capacity can be exploited.
- Average Response Time of LO-Critical Tasks R_L: This metric is referred to the average time span between the time that a LO-critical event arrives and the time that this event is finished. R_L reflects the QoS of LO-critical tasks.
- Latency Ratio of HI-Critical Task Set L_H: Suppose D_i is the relative deadline of a HI-critical task and R_i is its average response time, then the task set latency ratio is referred to be $\sum_{i=1}^{n} \Big(R_i/D_i\Big)/n$, where n is the task number in this task set. L_H reflects the influence of workload management approaches on the latency of HI-critical tasks.
- Timing Overhead of Decision Making T_o: For both the exact and the lightweight methods, T_o is referred to the computation time needed to update the priority assignments or to release a LO-critical event.

Note that U_s may not be equal to $U(\tau^h) + U(\tau^l)$ because LO-critical tasks may not be fully served.



(a) Type 1

(b) Type 2

(c) Type 3

(d) Type 4

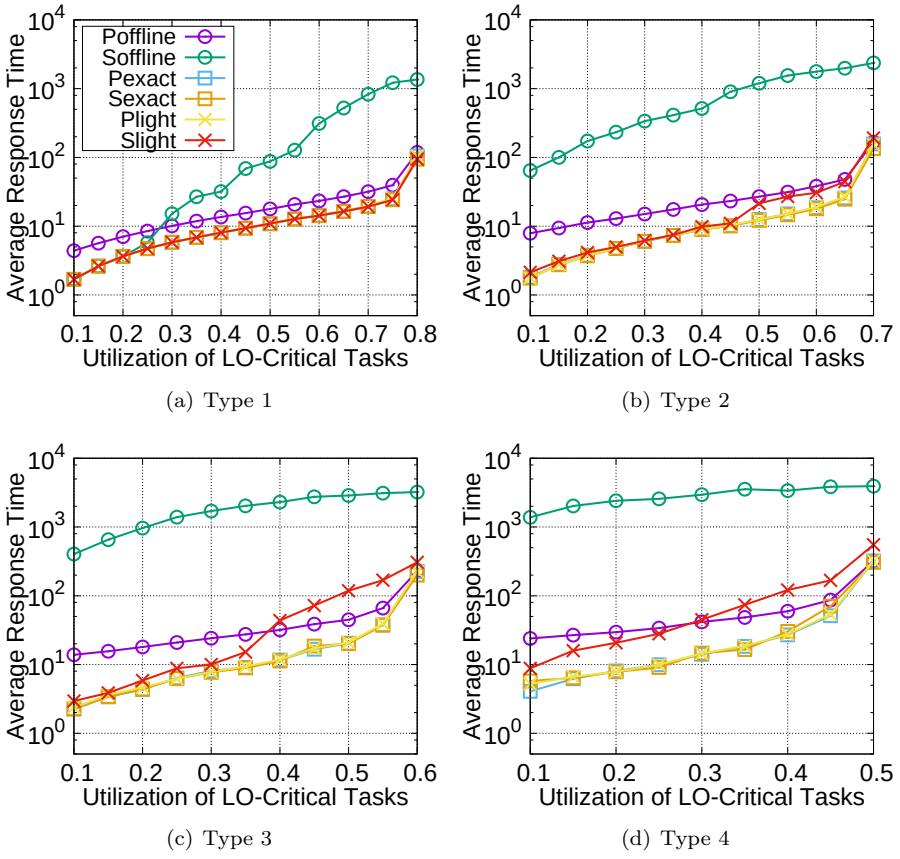
Figure 4.13: The system utilization w.r.t the utilization of LO-critical tasks

4.8.2 Simulation Results

In the following, we report simulation results for the listed four types, and the computation expenses by applying the exact computation and by applying the lightweight method in our proposed workload management policies. Under every specific setting of $U(\tau^h)$ and $U(\tau^l)$, 1000 test cases are generated to evaluate the performance of different workload management approaches. All the result figures are best seen in color online.

4.8.2.1 System Utilizations

The system utilizations of different workload management approaches working in the four types are shown in Fig. 4.13. From it, we find that, except Soffline, the system utilizations of using other approaches increase linearly with the utilization of LO-critical tasks, and the highest system utilizations can almost reach 1. This shows that, all approaches except the Soffline,



(a) Type 1

(b) Type 2

(c) Type 3

(d) Type 4

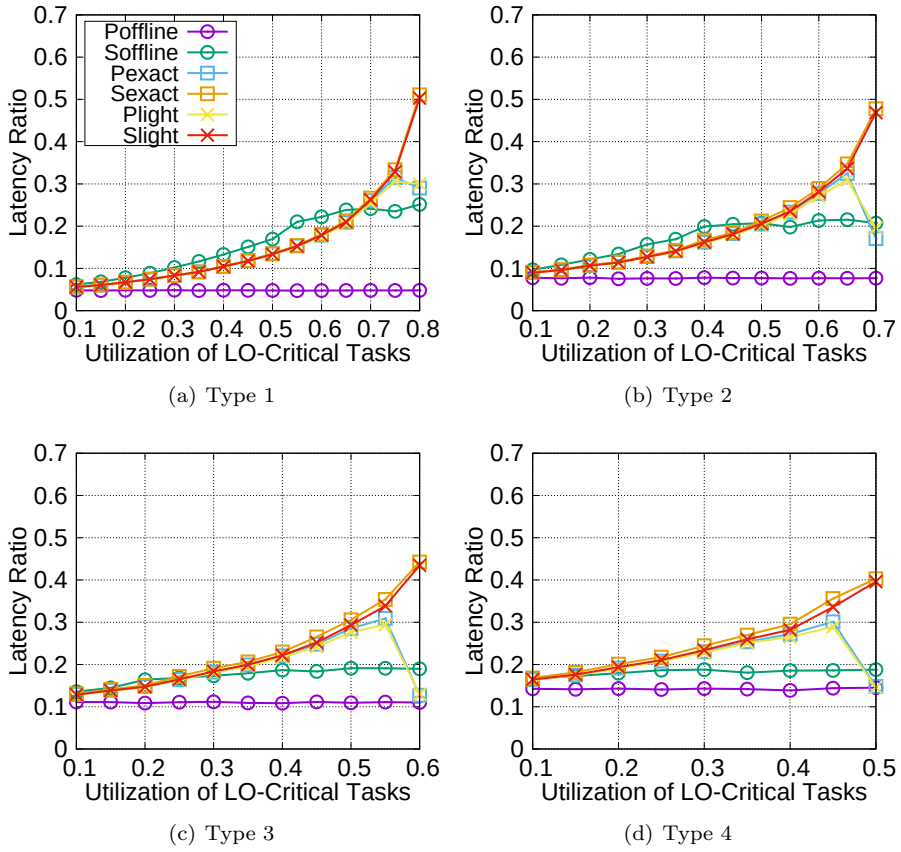
Figure 4.14: The average response time of LO-critical events

can fully make use of the system resources to process LO-critical and HI-critical tasks. The reason for the low system utilization of using Soffline is that the offline shaping bound is quite pessimistic, which results in a lot waste of processing resource.

4.8.2.2 Average Response Time of LO-Critical Tasks

The average response times of LO-critical tasks w.r.t. four types of HI-critical task sets are seen in Fig. 4.14. In general, we make three main observations.

- First, the shaping offline performs the worst among the six approaches. Fig. 4.14 shows that the average response time of using Soffline can be hundred times larger than that of using other methods. This is because the offline shaping bound is very pessimistic. If the utilization of LO-critical events exceeds this bound, many LO-critical events will be delayed for a long time.



(a) Type 1 (b) Type 2

(c) Type 3 (d) Type 4

Figure 4.15: The latency ratio of HI-critical events

- Second, the average response times of using Poffline are larger than those of using four online methods with the exception that Slight performs worse than Poffline when utilization of LO-critical tasks is large in Types 3, 4. By setting the priority of all LO-critical tasks as the lowest, LO-critical events cannot be processed ahead of HI-critical events, so the service that LO-critical tasks receive is lower than that of applying the priority-adjustment policy. By using the online shaping, the LO-critical event can be processed ahead of the HI-critical event only when the WCET of LO-critical event is smaller than the computed LFII. In the four types, the WCET of LO-critical event is smaller than the computed LFII in most cases, except the case of applying Slight when both $U(\tau^l)$ and $U(\tau^h)$ are greater than 0.3.

- Third, Sexact, Pexact, and Plight achieve almost the same average response time in the four types, while Slight performs badly in Type 3 and Type 4. This demonstrates that

the lightweight computing method achieves the same results as using the exact computing methods by priority-adjustment policy. The lightweight method of workload-shaping policy is the same as the exact method only when $U(\tau^h)$ is small.

4.8.2.3 HI-Critical Task Set Latency Ratio

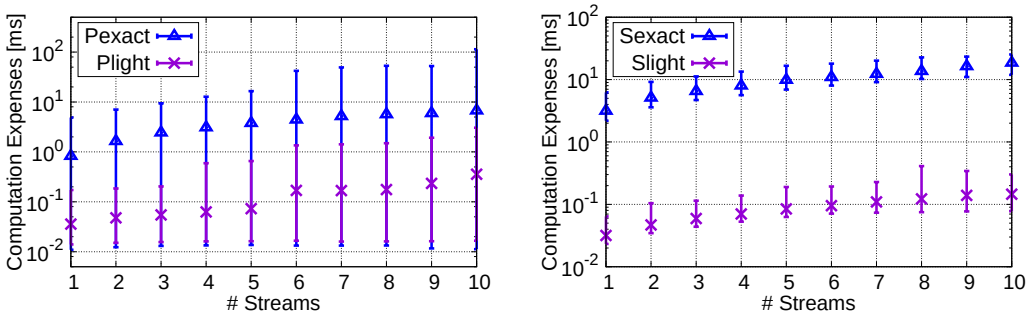
The HI-critical task set latency ratio w.r.t. four types of sets are seen in Fig. 4.15. In general, we make three main observations.

- Poffline has the lowest latency ratio, because the HI-critical tasks are served without the interference from LO-critical tasks.
- In the four types, the latency ratio of Soffline first increases, then keeps constant. The latency ratio increases because the LO-critical interference increases. However, the Soffline imposes a threshold on the LO-critical interference. Once the LO-critical interference exceeds this threshold, LO-critical interference will be throttled, and the latency ratio of HI-critical tasks will not increase. Besides, from Type 1 to Type 4 of Fig. 4.15, it can be seen that this threshold decreases with the increase of $U(\tau^h)$.
- With the increasing of $U(\tau^l)$, the latency ratios of Ponline and Plight first increase and then decrease. In the priority-adjustment policy, the latency ratio will be increased if HI-critical tasks receive an increasing LO-critical interference. However, as $U(\tau^l)$ increases, the LO-critical workload also increases, which will result in that priorities of HI-critical tasks will be set higher than LO-critical priority after LO-critical workload exceeds a certain threshold.

From Fig. 4.15, we also observe that the offline approaches have smaller latency ratio than the online approaches. This indicates that the online approaches sacrifice some QoS for HI-critical tasks to improve the QoS of LO-critical tasks. However, since the timing requirements of all HI-critical tasks are sufficiently met, such sacrifice is worthwhile to improve the QoS of LO-critical tasks.

4.8.2.4 Timing Overheads of Decision Making

For the priority-adjustment policy, the LO-critical priority has to be adjusted when a LO-critical event arrives or the execution of a LO-critical event finishes, by using the Algo. 4 or Algo. 5 to decrease or increase the LO-critical priority. We report the computation expenses



(a) Priority-adjustment policy

(b) Workload-shaping policy

Figure 4.16: Computation expense of the two adaptive workload management policies

of applying Algos. 4, 5 to adjust the priority. Fig. 4.16(a) shows the worst, best and average case computational expenses of using the exact and lightweight methods w.r.t. the number of HI-critical event streams. In Fig. 4.16(a), the average computational expense of one stream is 0.83 ms by using the exact method, and 0.034 ms by using the lightweight method. From Algos. 4, 5, we know the complexity of searching feasible priority increases with the increase of the streams, which can be found that the computational expense increases with the increase of streams in this figure. In general, the computational expense of lightweight method is one order of magnitude lower than that of the exact method by priority-adjustment policy.

The workload-management policy depends on the LFII to shape the LO-critical workload. The LFII is updated when the shaper is in `LFII updating state` and a HI-critical event is finished. Fig. 4.16(b) shows the worst, best, and average case computational expenses of updating the LFII w.r.t. the number of HI-critical event streams. The average computational expense of one stream is 3.15 ms by using the exact method, and 0.031 ms by using the lightweight method. For the ten streams, the average computational expenses of the two methods are 18.5 ms and 0.14 ms. In general, the computational expense of lightweight method is two orders of magnitude lower than that of the exact method.

From the above results, we find that the workload-shaping policy is effective when LO-critical events have low WCETs but may become ineffective when their WCETs are high. The workload-shaping policy is generally effective in regulating different kinds of events, while suffering the problem of frequent priority changes that will incur some extra runtime overheads. Therefore, from the perspective of implementations in a real platform, combining the two policies could be a possible solution that can overcome their own drawbacks and thus become more

effective than using the two policies individually. In this chapter, since we focus on the evaluations of proof-of-concept simulations and the effectiveness of such practical implementations rely on the specific hardware platforms, the combination of the two approaches is thus not discussed.

4.9 Summary

In this chapter, we develop the adaptive workload management in MCSs to improve the QoS of LO-critical tasks, while sufficiently guaranteeing the hard real-time constraints of HI-critical tasks. The priority-adjustment policy and the workload-shaping policy have been presented. In order to make the two polices applicable in the online cases, the lightweight method was proposed to replace the complex RT computation and backward derivation in the schedulability verification during the runtime. Simulation results demonstrate the effectiveness of the two adaptive workload management policies, and show the low timing overhead of the lightweight method.

Chapter 5

A Case Study of Applying Mixed-Criticality Scheduling to an Autonomous Driving System

System functionalities are often abstracted as independent tasks with specific parameters, based on which a lot of scheduling algorithms were proposed [4]. Although most of the proposed algorithms were presented to be effective in the theoretic analysis or by simulations, their performances need to be evaluated more from the perspective of practical implementation due to the following reasons. First, the abstractions on many functionalities are sometimes over simplified as in this way the scheduling analysis on them will become much easier. This however ignores many important functionality features like precedence constraints that will directly result in the failure of some scheduling algorithms. Second, most scheduling algorithms are based on the ideal assumptions that the implementation will not incur any overhead. Indeed, in many cases the implementation overhead has a large impact that should not be ignored in the evaluation of scheduling algorithm effectiveness [16]. Last, to the best of our knowledge, all previous systems are either simulated or artificially generated, which omits many potential difficulties for real-life use. Those potential difficulties need to be discovered and analyzed as soon as possible because they may help to divert the research to a more correct direction.

An autonomous driving system is claimed to be a typical mixed-criticality system because the functionalities in this system have different importance or criticalities. In this chapter, we present a case study of applying some basic mixed-criticality system concepts to schedule a real-life autonomous driving system. We investigate the authenticity of the current mixed-criticality

Figure 5.1: The overview of the model car. Its size is 120 cm×70 cm×35 cm and its weight is around 20 kg

model and the scheduling overhead of using mixed-criticality scheduling theory to schedule a real-life system.

5.1 Overview

Within the past decade, the autonomous driving has become a hot issue in the intelligent vehicle research domain. Many large companies such as Tesla and BMW are now devoting increasing efforts to develop the autonomous driving car [81, 82]. However, constructing and designing an autonomous driving car need a tremendous of investment that often blocks many small research groups involving in the autonomous driving research.

In this chapter, instead of constructing a real car, we designed an autonomous driving 'model' car, an overview of which is shown in Fig. 5.1. This car was originally a customized remote controlled car of the type SY-BLS5 equipped with additional metal bars on top. The motor is powered by a 11.7 V Li Battery that can drive the car to reach more than 80 km/h. In contrast with the real car, this model car is smaller but its movement is as flexible as the real car. Therefore, testing its autonomous driving is cheaper and testing results provide us some insights on how to achieve the autonomous driving on a real car. After some efforts, we successfully implemented some functionalities such as navigation and lane detection on this car. Those functionalities need a scheduler to manage their executions.

The mixed-criticality scheduling theory is based on the assumption that more and more components with different levels of criticality are integrated onto a common hardware platform,

where high criticality tasks need to be given more timing guarantee than low criticality tasks. The standard mixed-criticality scheduling also has some assumptions that tasks are independent and low criticality tasks can be aborted whenever a high criticality task overruns. Those assumptions are now receiving more and more doubts because they go against a lot of practical facts and industry standards [26, 27]. To sweep those doubts and establish the fundamental principles for designing a practical mixed-criticality scheduler, a real-life system should be used to examine the authenticity of those assumptions and evaluate the practical effectiveness of scheduling algorithms.

The autonomous driving system is a typical mixed-criticality system because this system contains at least two criticality levels, i.e., some safety-critical tasks (high criticality) and some mission-critical tasks (low criticality). In this chapter, we develop an autonomous driving system, based on which we develop two scheduling algorithms that are implemented in the Raspberry Pi 3 board. The detail contributions of this chapter are as follows:

- We present a hardware/software co-design on developing an autonomous driving car.
- We develop a task graph to present the precedence constraints among tasks. Tasks are classified into two levels – LO criticality and HI criticality. We develop the multi-core mixed-criticality scheduling algorithms called "TTS-MS" and "ETS-MS" to support the precedence-constrained task set of this autonomous driving system.
- We implement the two proposed scheduling approaches with the hierarchical scheduling framework in a Raspberry Pi board system and evaluate the overhead and effectiveness of the presented schedulers.

The rest of this chapter is organized as follows. Section 5.2 briefly presents the hardware/sofware co-design on developing the autonomous driving functionalities. Section 5.3 provides the specific task scheduling analysis and present two mixed-criticality scheduling approaches. Section 5.4 evaluates the two developed scheduling approaches and last section presents the conclusion and discussion.

5.2 Hardware/Software Co-Design

In order to achieve the autonomous driving, we designed a hardware structure as shown in Fig. 5.2. In the designed autonomous driving system a four-core hardware platform Raspberry

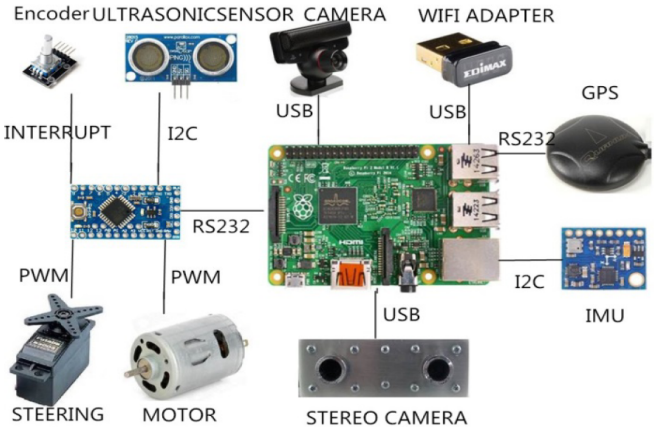


Figure 5.2: System hardware structure

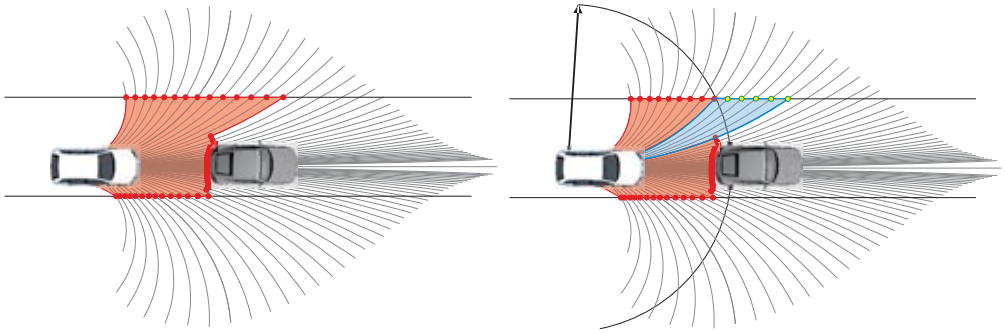
Pi 3 was used as a main processor. A global positioning system (GPS) and an inertial measurement unit (IMU) were used for navigation. One stereo camera and one webcam were used for local navigation. An Atmega 328 micro-controller and an encoder sensor were used to control the motor movement. A wifi adapter was used to receive the wifi signal so that we can remotely log in the linux system in the Raspberry Pi board. Based on those components, we develop the corresponding navigation applications and some traffic signs recognition applications on this car, as shown below.

5.2.1 Navigation

For the car global navigation, we relied on the GPS and the IMU. The limitations of GPS and IMU are that they cannot recognize the obstacles and traffic signs in front of the car. To allow the car to recognize objects such as pedestrians, we introduced the stereo cameras. In the following, we briefly describe our navigating approaches of using GPS+IMU and stereo cameras.

5.2.1.1 Global Navigation with GPS+IMU

NEO-6M GPS module and GY-85 IMU were chosen because they were commonly used in some similar projects. They also have a variety of documentation and examples for their usage. For the function development, we chose the Graphhopper service [83] that provides web API and generates the route in JSON. All functions are recast to support the web API requests. Some of them are completely rewritten.



(a) All tentacles should be discarded (b) Safe tentacles can be selected

Figure 5.3: Illustration that all tentacles may collide with objects. (a) a case that all colliding tentacles are discarded. (b) a case that the car can choose blue tentacles, where the black semicircle represents the crash distance to avoid a collision (figure from [2])

5.2.1.2 Local Navigation

For local navigation it was planned to implement a tentacles driving algorithm derived from [2]. This approach is to precompute a number of possible paths the vehicle can take, depending on the speed of the car, as shown in Fig. 5.3. This has advantages if computational resources are limited. Creating the precomputed paths (tentacles) has to be done only once. The path which comes closest to the desired target can be selected and executed. This target can be the point at which the global navigation gives the next instruction. As the paths are evaluated several times per second, the car never fully executes a path.

To avoid the obstacle, the tentacles algorithm only requires the car to check each collision path it has detected by its sensors. An array is often used to store the feasible tentacles so that the search on a feasible tentacle will be fast. In addition, we can also compute a crash distance for each tentacle, the distance within which the collision with an obstacle might occur. The crash distance allows the car to drive on a narrow road with another vehicle in front of it and avoid the collision with it by selecting the tentacle with the longest crash distance. It should be noted that this crash distance should be longer than a certain extent below which the car would be unable to stop, as shown in Fig. 5.3.

5.2.2 Traffic Light Detection

Since the size of our car is small, the front cameras may miss elevated signs or traffic lights. So we added another webcam on top of the car, pointing slightly upwards. To detect traffic lights

a number of simple image processing steps were run on the image. First, it was converted to HSV color space, in which hue was a separate value and the image can be filtered by color. This allowed to distinguish traffic light colors. Morphologic and blur operations closed holes created by noise in the image. A circle detection algorithm was run and the results were verified in regard to their circular shape and continuous color to obtain the final results. To exclude tail lights from leading vehicles and similar candidates for false positives outside of a Region Of Interest (ROI) were discarded.

5.2.3 Traffic Sign Recognition

Traffic sign recognition was implemented by using a similar approach to traffic light detection. The image was first converted to HSV color space. After filtering by color and applying Canny Edge Detection [84], shapes were detected via a function of OpenCV and possible signs were selected by checking them for certain characteristics regarding shape and color. In case of a speed sign the algorithm attempted to match the number against stored template images to determine the number inside the sign. Only the speed that limits up to the car's maximum speed of 80 km/h were recognized. While detection of some signs, like STOP or yield signs, also worked very good. Differentiation of the speed signs was currently unreliable. This may be improvable by using an optical character recognition library [85] or by utilizing a better template matching algorithm.

5.2.4 Lane Detection

The processing steps of lane detection were kept as simple as possible to increase throughput. One camera of the stereo camera setup was used as input because it had the ground in view. The ROI was restricted to areas below the horizon. At first the image was converted into grayscale and Canny Edge Detection [84] was run. The integrated algorithm to find contours in OpenCV was called, returning edges which were connected. Results were passed to the Ramer-Douglas-Peucker algorithm [86]. This simplified detected contours and allowed to filter out small lines, reducing noise. The remaining lines were compared against lines from the last iterations in a verification step and returned if they matched.

5.3 Task Scheduling

In the above section we list some functionalities that can help the car to achieve autonomously driving. Those functionalities are not completely independent. For example, the lane detection relies on the video stream from the stereo camera, while the stereo camera is also used for local navigation by tentacles algorithm. In order to present such relationship among those tasks, every functionality is divided into some tasks and a task graph is developed to show their dependencies as Fig. 5.4. This task graph is a directed acyclic graph (DAG) with 10 nodes, where each node represents a task. The task functionality is described in Tab. 5.1.

Table 5.1: Task Properties

No.	Name	Description
T1	Capture2	Get a frame from the top camera
T2	SignsProc	Detect traffic signs in a frame
T3	LightsProc	detect traffic lights in the frame
T4	Capture0	Get the left frame of the stereo camera
T5	Capture1	Get the right frame of the stereo camera
T6	LanesProc	Detect lanes and implement steering
T7	DepthMapProc	Generate Depth map and implement collision avoidance
T8	GPSProc	Get location from GPS and IMU
T9	SensorFusionSpeed	Sensor fusion of different sensors and send proper speed
T10	SensorFusionSteering	Sensor fusion of different sensors and send proper steering

Our processing platform is a four-core ARM cpu that is able to simultaneously run 4 tasks. By allocating tasks to each core, the parallel execution will help the platform to minimize the processing expense on running this task graph. In the following, we present a concrete task allocation approach, based on which we present a mixed-criticality scheduling approach to this task graph.

5.3.1 Task Allocation

Before we present the mixed-criticality scheduling approach, we first discuss how to allocate tasks for a system only with the same criticality tasks. The discussed results will be used to decide the allocated core for a task in the system with different criticality tasks.

A task can be allocated to a core by the global scheduling or by the partitioned scheduling. The global scheduling allocates the task dynamically at runtime, where a task can be allocated to any core. By allocating tasks dynamically, the global scheduling has the advantage of balancing

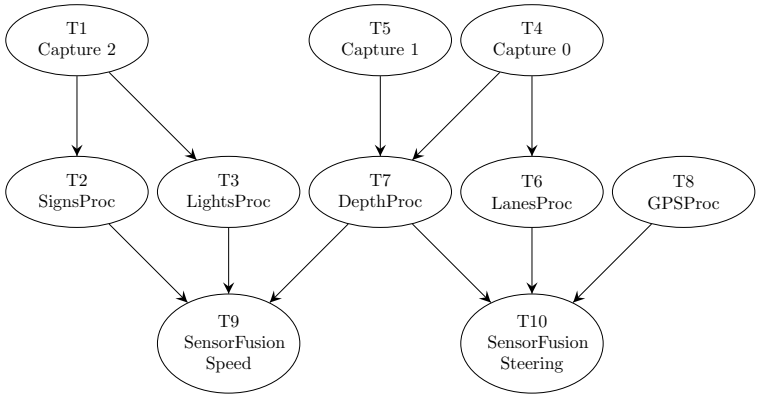


Figure 5.4: Task graph of the autonomous driving

the load automatically, lowering the average response time and increasing the efficiency of reclaiming the available time slack. However, it has the disadvantage of the high task migrations, high implementation difficulty, and low system reliability. Compared to the global scheduling, the partitioned scheduling allocates tasks to cores statically, which means that a task will be attached to a core permanently after the system starts. Thus there is no task migration in the partitioned scheduling. The advantages of partitioned scheduling are high system reliability. The automotive industry prefers partitioned scheduling because partitioned scheduling is more reliable and much easier than the global scheduling to be implemented. In this thesis, we only present the partitioned scheduling analysis.

In partitioned scheduling, each task in the task graph needs to be allocated to a core at the beginning. Suppose the task is time-triggered, then the triggering time of each task also needs to be decided. The principle of allocating tasks and setting triggering time is to minimize the makespan within which every task in this task graph can be completed, because in this way the execution of this task graph will be more frequent so that the car can react to the environment much faster.

The makespan minimization under the constraint of a task graph has been studied in a lot of paper [87–90]. It has been proved that the makespan minimization on a multi-core system is an NP-hard problem [87], which means an optimal solution is not able to be obtained in polynomial time and the solution also cannot be verified in polynomial time, either. However, heuristic approaches such as list scheduling or polynomial time approximation scheme can be applied to obtain a suboptimal solution.

A matlab toolbox named Torsche (Time Optimization of Resources, Scheduling) [91] offers

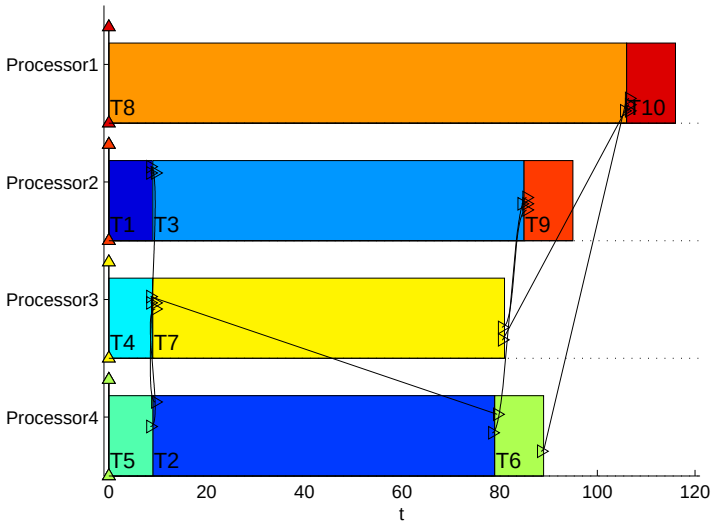


Figure 5.5: Task scheduling illustration

a number of solutions for various off-line and online scheduling problems. This toolbox also fits for the task allocation problem in our case. First, a matrix P is used to represent the task precedences. We define that $P(i,j)$ represents the dependency between task τ_i and task τ_j, where $P(i,j) = 1$ implies that τ_j cannot be started before τ_i and $P(i,j) = 0$ implies that there is no constraint between τ_i and τ_j. Then, based on the task graph of Fig. 5.4, the precedence matrix is that

$$P = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Second, in order to sufficiently isolate task execution, the worst-case execution time (WCET) should be obtained. The real-time system developer often adopts two approaches [92–94] to obtain the WCET of a task, which are

- dynamic timing analysis: Most parts of industry estimate the execution time bounds by measuring the end-to-end execution time of a task for a subset of the possible executions. This determines the minimal and maximal observed execution time.

- static timing analysis: The static methods do not rely on executing code on a simulator or on a real hardware. They analyze the possible control paths in the task and combine it with other abstract models of hardware architecture, to obtain the execution time bounds.

In our platform, we rely on the dynamic timing analysis because the unpredictable interference in our multi-core design makes it impossible to apply the static timing analysis. The maximal execution time in 2 hours test is used as a task WCET. The average and maximum execution times are listed in Tab. 5.2. With the task precedence matrix P and task WCET, Torsche toobox can be applied to allocate tasks and determine the triggering time for each task, also as shown in Tab. 5.2. The task schedule plot is presented in Fig. 5.5, where the makespan is 116 ms. Then the period of this task graph can be set as 116 ms.

Table 5.2: Task execution times and allocations

No.	Name	AverExecu (ms)	WCET (ms)	Core Index	Triggering Time
T1	Capture2	3	9	2	0
T2	SignsProc	54	70	2	9
T3	LightsProc	43	76	4	9
T4	Capture0	5	9	3	0
T5	Capture1	6	9	4	0
T6	LanesProc	4	10	3	81
T7	DepthMapProc	50	72	1	9
T8	GPSProc	67	106	1	0
T9	SensorFusionSpeed	3	10	1	85
T10	SensorFusionSteering	4	10	2	106

In the above scheduling the WCET is our measured maximum execution time in experiments. This measured WCET in general underestimates the WCET and is not safe for hard real-time systems. In order to make the WCET safer, the measured WCET can be multiplied by a coefficient $\lambda\,(\lambda > 1)$ and its product can be considered as a safer WCET. Although a large λ increases the system safety, it will result in a lot available processing time being wasted because the real execution time will be much less likely to reach our assumed WCET. Therefore, in order to keep the system high utilization and also keep its high safety, we propose to apply the mixed-criticality scheduling to the system.

5.3.2 Mixed-Criticality Scheduling

The mixed-criticality scheduling adopts the mode-switch to guarantee all tasks' functionalities in normal mode and HI-critical tasks' functionalities in critical mode. In normal mode, tasks'

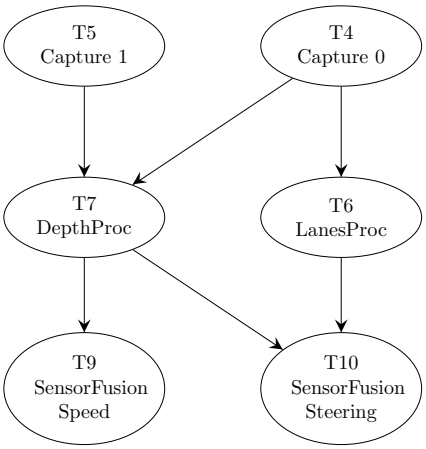


Figure 5.6: HI mode task graph of the autonomous driving

execution times are assumed to be within a measured WCET. If a HI-critical task overruns its measured WCET, the system switches to critical mode in which LO-critical tasks are aborted in order to provide larger execution time budget to HI-critical tasks.

In the following, we first divide our tasks to LO- and HI-critical tasks. Then we present a scheduler named time-triggered scheduler with mode switch (TTS-MS) and a scheduler named event-triggered scheduler with mode switch (ETS-MS) to provide the timing guarantee in LO and HI modes.

5.3.2.1 Task Criticality Classification

Since the system is divided into LO and HI modes, the task is also categorized into two classes, LO criticality and HI criticality. The HI-critical tasks are often referred to the safety-critical tasks. We assume the safety-critical tasks are path tracking, collision avoidance, steering and speed control. The remaining tasks are set as LO-critical tasks. As a result, the task graph in LO mode is the same as Fig. 5.4 and the task graph in HI mode will be like Fig. 5.6.

5.3.2.2 Time-Triggered Scheduler with Mode Switch

The system mode now has been divided into LO mode with a LO mode task graph and HI mode with a HI mode task graph. The system should be in LO mode in most of the time because aborting some tasks in HI mode makes the autonomous car become unstable. To achieve this, we on the one hand should avoid the mode switch from LO to HI as much as we

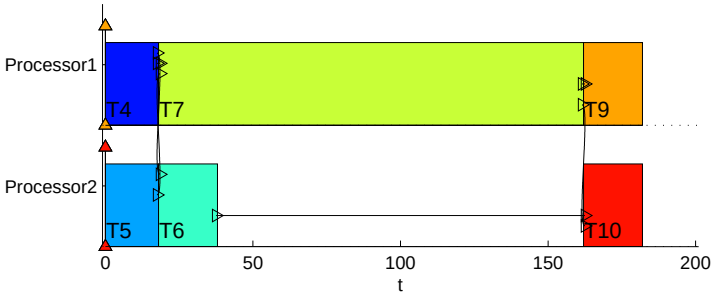


Figure 5.7: Task scheduling of HI mode task graph

can, and on the other hand should switch the system back to LO mode as soon as possible when the system is already in HI mode.

Since the LO mode is triggered to HI mode by a HI-critical task overrunning its LO WCET, the LO WCET should be set high enough. We thus set the task measured maximum execution time as its LO WCET. For LO-critical tasks, their executions will be stopped immediately after they overrun their LO WCETs. To guarantee that the LO-critical tasks can be finished, their LO WCETs are also set as the measured maximum execution times. When the system is switched to HI mode, only HI-critical tasks will remain. The HI WCET should be an upper bound on the execution times. Since this upper bound is not able to be obtained by the static timing analysis, the HI WCET can only be given an artificially pessimistic value. We set the HI WCET of a HI-critical task as twice of its LO WCET. The LO and HI WCET of a task T_i is still denoted as C_i^L and C_i^H. We thus have $C_i^H = 2 \cdot C_i^L$. In addition, when the system is in HI mode, an effective scheme is to switch the system mode back to LO mode immediately after completing all remaining HI-critical tasks.

As the LO WCET is the same as the measured maximum execution time, the system schedule in LO mode can be the same as Fig. 5.5. For the task graph of Fig. 5.6 in HI mode, if this task graph is independently scheduled without any connection to its LO mode schedule, the Torsche toolbox can also be applied and the scheduling result will be like Fig. 5.7. We know that only two cores are needed for this task graph. Since the partitioned scheduling is applied to schedule this task graph, task is not allowed to migrate at runtime, which means that the schedule of Fig. 5.7 is in fact not possible because HI-critical tasks are distributed to four cores in LO mode.

Although the task allocation in HI mode cannot be the same as Fig. 5.7, Fig. 5.7 provides us the reference time for setting the task triggering time in HI mode. For example, the triggering

time of task T10 in Fig. 5.7 is 162 (ms). This release time can be used as the HI mode release time because all predecessors of T10 must have been completed before 162 (ms). Denote this release time as r_i^H. For other HI-critical tasks, their released times of Fig. 5.7 also guarantee that all their predecessors can be completed before those released times. Hence, in HI mode we keep the allocated core of a task unchanged and its triggering time is set as r_i^H. Note that, this approach of determining the triggering time is not a general approach, which may not be applicable for other task graphes.

Based on the above approach, the specific mixed-criticality scheduling result is shown in Tab. 5.3, where '−' represents that there is no value.

Table 5.3: Task parameters of mixed-criticality scheduling

No.	Name	Criticality	Core Index	WCET (C_i^L, C_i^H)	Triggering (r_i^L, r_i^H)
T1	Capture2	LO	2	(9,-)	(0,-)
T2	SignsProc	LO	4	(70,-)	(10,-)
T3	LightsProc	LO	2	(76,-)	(10,-)
T4	Capture0	HI	3	(9,18)	(0,0)
T5	Capture1	HI	4	(9,18)	(0,0)
T6	LanesProc	HI	4	(10,80)	(79,18)
T7	DepthMapProc	HI	3	(72,80)	(9,18)
T8	GPSProc	LO	1	(106,-)	(0,-)
T9	SensorFusionSpeed	HI	2	(10,20)	(85,162)
T10	SensorFusionSteering	HI	1	(10,20)	(106,162)

5.3.2.3 Event Scheduler

Another effective scheduling approach is to rely on the event to trigger a task. Suppose a task will release an event once this task has been finished. This event can be used as a signal to trigger the following task, in this way the task dependency constraints are maintained. For the event scheduler, the task allocations in LO mode still rely on the Torsche toolbox. LO-critical tasks that overrun their LO WCETs will be directly aborted. This scheduler is called event-triggered scheduler with mode switch, abbreviated as ETS-MS.

5.4 Implementation Evaluation

We developed the time-triggered scheduler with mode switch (TTS-MS) and event-triggered scheduler with mode-switch (ETS-MS) by extending a framework called hierarchical scheduling framework (HSF) [95]. The TTS-MS and ETS-MS were implemented in Raspberry Pi 3 model

B with a 1.2 GHz 64-bit quad-core ARMv8CPU and the operating system was raspbian 4.4.13 linux.

We mainly investigated the runtime overhead of our developed scheduler. The timing expenses of the following actions can be used to evaluate the scheduler runtime overhead. The larger those timing expenses are, the worse the runtime scheduling performance is.

- Job handling: When a job arrives or finishes, the scheduler should spend some time in signaling and registering in a bitmap data structure. When a job finishes, the scheduler also needs to deregister it from a bitmap.
- Overrun handling: When a LO-critical task overruns, it is immediately aborted. When a HI-critical task overruns, the system mode will be immediately switched to HI mode, where all LO-critical tasks are aborted.
- Overrun checking: A monitor is used to constantly check the task execution. This enables the system to detect the task overrunning whenever it happens.
- Others: There are some other scheduling expenses like task execution preemption handling, reactivating LO-critical tasks once the system is switched to LO mode from HI mode.

In addition to the above metrics, we also investigate the job overrun ratio and job drop ratio. The job overrun ratio is the ratio that jobs overrun their given LO WCETs among all released jobs and the job drop ratio is the ratio that jobs are dropped among all released jobs.

5.4.1 Results of TTS-MS Implementation

The scheduler was run for one hour and there was no deadline miss of HI-critical tasks. We record the above timing expenses, task overrun rate and task drop rate.

Tab. 5.4 shows our recorded timing expenses in the one hour running. Note that the absolute overhead is the sum of all scheduling timing expenses of each core and the relative overhead is its time ratio of those timing expenses in the one hour. The total timing overhead only accounts for 0.27%, which demonstrates that our scheduler is very lightweight.

The job overrun ratio and job drop ratio are shown in Fig. 5.8. The job overrun ratio of each core is around 1% to 2%. This confirms that the measured maximum WCET is not an upper bound on the execution time, but it is very close to this upper bound. Compared to the job overrun ratio, there is a significant difference of job drop ratio among the four cores.

Table 5.4: Timing expense of TTS-MS, where unit is millisecond

Metrics	Core 1	Core 2	Core 3	Core 4	sum
Job handling	338.328	510.768	356.076	498.6	1703.722
Overrun handling	224.628	253.932	200.844	305.82	985.224
Overrun checking	887.424	1200.036	970.188	1303.296	4360.944
Others	538.128	783.864	530.532	781.116	2633.64
Absolute overhead	1988.508	2748.6	2057.64	2888.832	9683.58
Relative overhead	0.055%	0.076%	0.057%	0.08%	0.27%

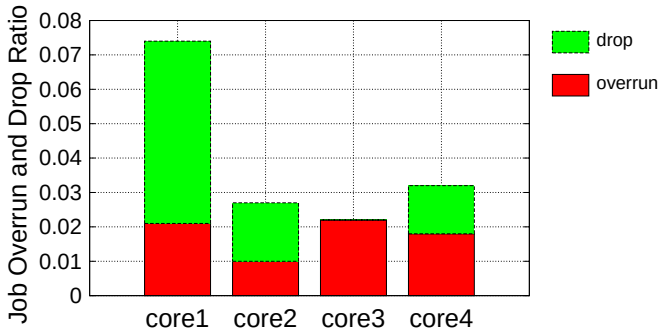


Figure 5.8: Job overrun rate and job drop rate in TTS-MS

The job drop ratio on core 1 is more than 5%, while there is no job dropped in core 3. The difference comes from the task allocation. In core 1, the LO-critical task "GPSProc" has the longest execution time. This increases the probability of dropping job "GPSProc" because the overrun of any HI-critical task during the execution of "GPSProc" will make "GPSProc" be immediately dropped. While in core 3, there is only HI-critical task. Since any HI-critical task is not allowed to be dropped, the core 3 has no dropped job.

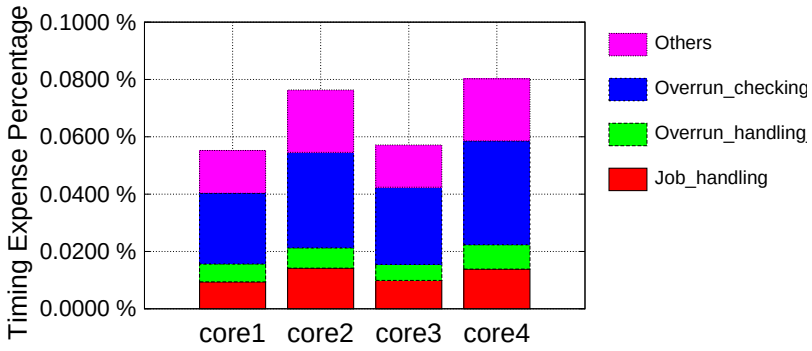


Figure 5.9: Scheduling overhead distribution in TTS-MS

Table 5.5: Timing expense of ETS-MS, where unit is millisecond

Metrics	Core 1	Core 2	Core 3	Core 4	sum
Job handling	208.216	289.183	211.644	424.35	1133.394
Overrun handling	270.6117	314.53	135.845	232.293	1773.5
Overrun checking	790.167	1320.409	690.171	1139.203	3939.951
Others	486.954	644.88	369.862	764.452	2266.125
Absolute overhead	1755.924	2569.005	1407.527	2560.302	8292.758
Relative overhead	0.049%	0.071%	0.039%	0.071%	0.23%

The distribution of scheduling timing expenses on each core is shown in Fig. 5.9, which shows that the timing expense proportion among the four cores are roughly 2:3:2:3. This proportion just fits the allocated task number on each core, which implies that the scheduling overhead depends on the allocated task number. Besides, the timing expense of the overrun checking is the most among the four metrics. The execution of every task is monitored by an overrun checking thread. The checking towards task execution has to be very frequent because otherwise the task overrun is not able to be detected in time. The higher the overrun checking frequency is, the larger the timing overhead will be. The timing expense of job handling also occupies a large portion in the scheduling expense. The job handling includes the job arrival handling and job finish handling. Since jobs arrive and finish frequently, the job handling will also be frequent and thus needs a lot time. Compared to job handling, the overrun handling takes less time because tasks have relatively high LO WCETs and are not likely to overrun them. Thus the overrun handling is not frequently activated and its timing expense is relatively small.

5.4.2 Results of ETS-MS Implementation

This scheduler was also run for one hour and there was also no HI-critical task deadline miss. The timing expenses of the four metrics are listed in Tab. 5.5. The total scheduling expense only accounts for 0.23% of whole processing time, which demonstrates that ETS-MS is a very lightweight scheduler. The ETS-MS has less scheduling overhead than TTS-MS whose total scheduling expense accounts for 0.27% of whole processing time. One reason for the better performance of ETS-MS is that only one event dispatcher is needed in ETS-MS, while every task needs one event dispatcher in TTS-MS.

The job overrun ratio and job drop ratio are presented in Fig. 5.10. From this figure we find that core 1 has the highest job drop ratio and core 3 has no dropped job, which is the same as

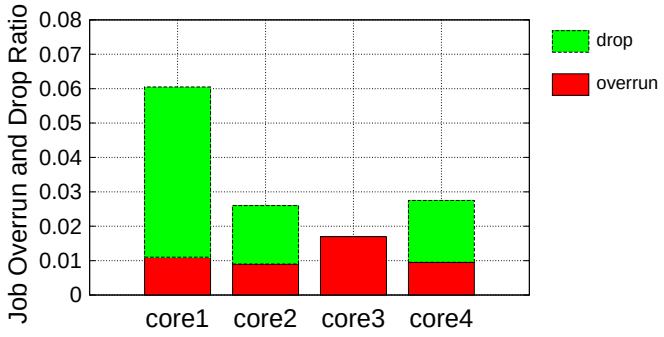


Figure 5.10: Job overrun rate and job drop rate in ETS-MS

TTS-MS. Besides, the distributions of the two ratios are also similar to those of TTS-MS. This is because both TTS-MS and ETS-MS have the same task allocations on each core.

The distributions of four scheduling overhead metrics are shown in Fig. 5.11, where observations similar to TTS-MS are made. The scheduling overhead proportion among the four cores is consistent with the number of allocated tasks on each core and the overrun checking still occupies a large portion on the whole scheduling overhead. Reasons for them are the same as our analysis to TTS-MS.

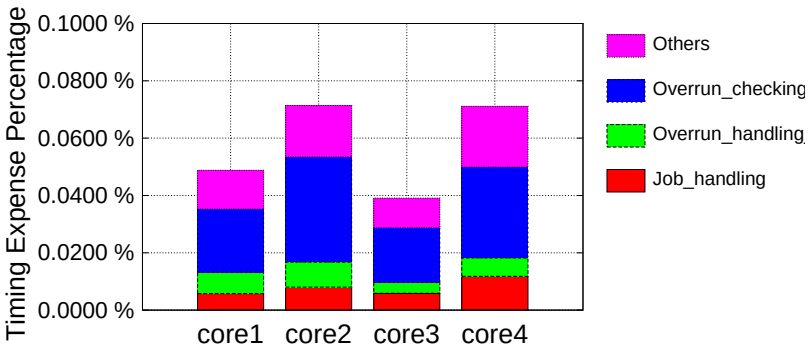


Figure 5.11: Scheduling overhead distribution in ETS-MS

5.5 Summary

In this chapter, we have presented a case study of applying the mixed-criticality scheduling concept to schedule a real autonomous driving system. We first present the hardware/software co-design and introduce the developed applications that are used to achieve autonomous driving. Since the applications are not independent from each other, a task graph is developed to denote the dependencies among those applications.

5. A CASE STUDY OF APPLYING MIXED-CRITICALITY SCHEDULING TO AN AUTONOMOUS DRIVING SYSTEM

In order to apply the mixed-criticality scheduling approach, the task and the system are modeled based on the assumptions in mixed-criticality scheduling. Specifically, the system is divided into two modes and tasks are categorized into two categories, i.e., LO criticality and HI criticality. To address the different timing requirements of the two criticalities, the system is divided into two modes, i.e., LO mode and HI mode. The system starts from LO mode and will switch to HI mode once a HI-critical task overruns our given execution time budget (LO WCET). The time-triggered scheduler and event-triggered scheduler, both with mode switch, are developed and implemented. Evaluation results demonstrate that the overhead of mixed-criticality scheduling can be very small.

This chapter also implies some challenges on the mixed-criticality scheduling. First, in a real-life system, tasks may not be independent. This fact is ignored by most of the state-of-the-art mixed-criticality scheduling research, which makes many research results difficult to be applicable for a system whose tasks are constrained by a task graph. Second, the WCETs are assumed to be known by most research, which however is not true in a real case. The integration of tasks and components makes the system's behaviors unpredictable. The exact WCETs are indeed very difficult to be obtained in a multi-core system full of unpredictable behaviors. Based on these two points, the scheduling approaches proposed in the previous chapters are also not applicable to this autonomous driving system because those approaches are built on the assumptions of independent tasks and known WCETs,

Chapter 6

Conclusions

6.1 Main Results

The aim of this thesis is to address problems intrinsically existing in the current mixed-criticality scheduling theory. The problems are categorized into a few aspects and the corresponding solutions are presented. Our results enrich the mixed-criticality scheduling theory and provide some foresight for its practical implementation. The main contributions are summarized as follows:

- Regarding to the problem that the standard mode-switch scheme in mixed-criticality scheduling is abrupt and pessimistic, we propose an on-the-fly overrun budgeting (FFOB) scheme. This scheme is able to make use of the system static and dynamic slacks to postpone the mode-switch for both FP- and EDF-scheduled systems. Thus, the system can stay in LO mode as long as possible, where LO- and HI-critical tasks are guaranteed. A feature called automatic schedulability guarantee is explored to transfer the dual-criticality guarantee to the conventional system schedulability guarantee, thus reducing the online computation overhead. Results of simulations and implementations in a real embedded system demonstrate that, compared to the state-of-the-art scheduling approaches, the proposed FFOB scheme significantly improves the QoS of LO-critical tasks, while maintaining the same guarantee to HI-critical tasks.

- Regarding to the problem that the sporadic task model is quite limited in representing the complex task activations, we extend the sporadic task model to the arbitrarily activated task model. Specifically, by using the arrival curve to represent the upper bound of task activations, we integrate the well-established results from Real-Time Calculus to analyze

the schedulability of a mixed-criticality system composed of arbitrarily activated tasks. Schedulability evaluations demonstrate that, towards the sporadic tasks, our proposed test can achieve the same effectiveness as two state-of-the-art approaches, AMC-max in FP system and EDF-VD in EDF system. However, towards some tasks whose activation events can be blocked or whose deadlines can be arbitrary, our tests have much better performance.

- Regarding to the problem that the system is not adaptive in providing the service to LO-critical tasks, we propose two online adaptive workload management policies, i.e., priority-adjustment policy and workload-shaping policy, to improve the system runtime adaptability. By using monitors to monitor the arrival events, the demand of HI-critical tasks can be updated online, based on which a safe service bound for the LO-critical tasks is computed to regulate the coming LO-critical workload. The runtime computation of this safe service bound is heavy. To eliminate this complex computation, a lightweight method with the complexity of $O(n \cdot log(n))$ is developed, making the online adaptations applicable at runtime. Experimental results confirm that the two adaptive workload management approaches significantly improve the system utilization and shortening the response time of LO-critical tasks. The timing overhead of the proposed lightweight method is demonstrated to be one and two orders of magnitude lower in the two workload management policies, respectively.

- Regarding to the problem that there is no real-life system that is ever scheduled by the mixed-criticality approach, we develop an autonomous driving system and apply the mixed-criticality scheduling to it. In detail, we present a hardware/software co-design on developing an autonomous driving car, based on which a task graph is plotted to represent the task precedence constraint. We model the autonomous driving system as a dual-criticality system and classify tasks into LO-critical and HI-critical tasks. We develop the time-triggered and event-triggered schedules, both with mode-switch, to schedule this system. Scheduling performance in terms of the overhead and dropped jobs is evaluated. The results show that the scheduling is lightweight and the ratio of dropped jobs is low. Our implementation also implies some challenges that need to be addressed for applying mixed-criticality scheduling to a real-life system.

6.2 Future Perspectives

The contributions presented in this thesis provide partial solutions for a few problems in the mixed-criticality scheduling. There remains a lot of challenges in developing the mixed-criticality scheduling theory and applying it into real-life systems. A few important future perspectives are listed in the following:

- The motivation of developing mixed-criticality scheduling is to reduce the system size, weight, and power by integrating different criticality tasks into a same platform. The real-time system community models this system from the timing perspective, with the ignorance on the system predictability and reliability. Indeed, task integration will incur more system unpredictable behaviors, which makes it more difficult to obtain a reliable worst-case execution time and thus makes the timing analysis unreliable. To overcome this problem, the hardware architecture has to be involved into the consideration when designing a mixed-criticality system. For example, we may apply the cache partition or deterministic bus access to avoid task execution interferences, as in this way the system predictability will be improved.

- The industry standards that classify tasks to several criticality levels are often used as arguments to justify the mixed-criticality scheduling research. However, a common assumption in the mixed-criticality system model that low criticality tasks can be dropped actually goes against many industry standards. Recent research also shows the increasing concern that the current mixed-criticality system model deviates a lot from real-life industry systems and thus has no practical value. To overcome this problem, developing a more representative system model is necessary to guide the mixed-criticality system research to a correct direction.

- Towards a specific task graph, we present a specific scheduling solution in this thesis. But towards a general task graph, there is still neither an optimal solution nor a heuristic solution of mixed-criticality scheduling.

- The experiments and tests of applying mixed-criticality scheduling to real-life systems are still few. Such work is significant because hidden problems or challenges will be exposed and our misunderstandings towards the mixed-criticality scheduling will also be corrected.

References

[1] C. Buckl, A. Camek, G. Kainz, and C. Simon. **The software car: Building ICT architectures for future electric vehicles**. In *Electric Vehicle Conference (EVC)*, pages 1–8, 2012. ix, 1, 2

[2] Felix Von Hundelshausen, Michael Himmelsbach, Falk Hecker, Andre Mueller, and Hans-Joachim Wuensche. **Driving with tentacles: Integral structures for sensing and motion**. *Journal of Field Robotics*, **25**(9):640–673, 2008. xi, 119

[3] Manfred Broy, Ingolf H Kruger, Alexander Pretschner, and Christian Salzmann. **Engineering Automotive Software**. *Proceedings of the IEEE*, **95**(2):356–373, 2007. 1

[4] Alan Burns and Rob Davis. **Mixed criticality systems: A review**. Technical report, Department of Computer Science, University of York, 2015. 2, 4, 7, 10, 12, 13, 14, 115

[5] Steve Vestal. **Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance**. In *Real-Time Systems Symposium (RTSS)*, pages 239–243, 2007. 3, 12, 48

[6] François Dorin, Pascal Richard, Michaël Richard, and Joël Goossens. **Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities**. *Real-Time Systems*, **46**(3):305–331, 2010. 3, 46, 48, 49, 54

[7] Neil C Audsley. **On priority assignment in fixed priority scheduling**. *Information Processing Letters*, **79**(1):39–44, 2001. 3, 13, 48, 107

[8] Sanjoy K Baruah, Alan Burns, and Robert I Davis. **Response-time analysis for mixed criticality systems**. In *Real-Time Systems Symposium (RTSS)*, pages 34–43, 2011. 3, 10, 13, 14, 37, 46, 47, 48, 74

[9] Sanjoy Baruah, Vincenzo Bonifaci, Gianlorenzo D'Angelo, Haohan Li, Alberto Marchetti-Spaccamela, Suzanne Van Der Ster, and Leen Stougie. **The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems.** In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 145–154, 2012. 3, 13, 18, 49

[10] Pengcheng Huang, Georgia Giannopoulou, Rehan Ahmed, Davide B Bartolini, and Lothar Thiele. **An Isolation Scheduling Model for Multicores**. In *Real-Time Systems Symposium (RTSS)*, pages 141–152, 2015. 3

[11] James H. Anderson, Sanjoy K. Baruah, and B. Brandenburg. **Multicore Operating-System Support for Mixed Criticality**. In *Proceedings of the Workshop on Mixed Criticality: Roadmap to Evolving UAV Certification*, 2009. 4, 7

[12] M. S. Mollison, J. P. Erickson, J. H. Anderson, S. K. Baruah, and J. A. Scoredos. **Mixed-Criticality Real-Time Scheduling for Multicore Systems.** In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1864–1871, 2010. 4, 7

[13] J. L. Herman, C. J. Kenna, M. S. Mollison, J. H. Anderson, and D. M. Johnson. **RTOS Support for Multicore Mixed-Criticality Systems**. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 197–208, 2012. 4, 7

[14] Micaiah Chisholm, Bryan C. Ward, Namhoon Kim, and James H. Anderson. **Cache Sharing and Isolation Tradeoffs in Multicore Mixed-Criticality Systems**. In *IEEE Real-time Systems Symposium (RTSS)*, pages 305–316, 2015. 4, 7

[15] Huang Ming Huang, Christopher Gill, and Chenyang Lu. **Implementation and Evaluation of Mixed-Criticality Scheduling Approaches for Periodic Tasks**. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 23–32, 2012. 4, 7

[16] Lukas. Sigrist, Georgia. Giannopoulou, Pengcheng Huang, and Lothar Thiele Gomez, Andres. **Mixed-criticality runtime mechanisms and evaluation on multicores**. In *Real-Time and Embedded Technology and Applications Symposium*, pages 194–206, 2015. 4, 7, 115

[17] Sanjoy K Baruah, Vincenzo Bonifaci, Gianlorenzo D'Angelo, Alberto Marchetti-Spaccamela, Suzanne Van Der Ster, and Leen Stougie. **Mixed-criticality scheduling of sporadic task systems**. In *Algorithms–ESA 2011*, pages 555–566. Springer, 2011. 4, 13, 49

[18] Pontus Ekberg and Wang Yi. **Outstanding paper award: Bounding and shaping the demand of mixed-criticality sporadic tasks**. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 135–144, 2012. 4, 10, 13, 14, 18, 19, 26, 34, 48, 49, 65, 66, 72, 73, 74

[19] Pontus Ekberg and Wang Yi. **Bounding and shaping the demand of generalized mixed-criticality sporadic task systems**. *Real-time systems*, **50**(1):48–86, 2014. 4, 10, 13, 14, 75

[20] Nan Guan, Pontus Ekberg, Martin Stigge, and Wang Yi. **Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems**. In *Real-Time Systems Symposium (RTSS)*, pages 13–23, 2011. 4

[21] Pengcheng Huang, Pratyush Kumar, Georgia Giannopoulou, and Lothar Thiele. **Energy efficient dvfs scheduling for mixed-criticality systems**. In *Conference on Embedded Software (EMSOFT)*, pages 1–10, 2014. 4

[22] Bjorn B Brandenburg, Hennadiy Leontyev, and James H Anderson. **An overview of interrupt accounting techniques for multiprocessor real-time systems**. *Journal of Systems Architecture*, **57**(6):638–654, 2011. 6

[23] Ken Tindell, Alan Burns, and Andy Wellings. **An extendible approach for analyzing fixed priority hard real-time tasks**. *Real-time Systems*, 1994. 6

[24] Sergio Santos, Jose Rufino, Tobias Schoofs, Cassia Tatibana, and James Windsor. **A portable ARINC 653 standard interface**. In *2008 IEEE/AIAA 27th Digital Avionics Systems Conference*, pages 1–E, 2008. 6

[25] Traub Matthias. *Durchgängige timing-bewertung von vernetzungsarchitekturen und gateway-systemen im kraftfahrzeug.* PhD thesis, Karlsruher Institut für Technologie (KIT), 2010. 6, 46

[26] Alexandre Esper, Geoffrey Nelissen, Vincent Nélis, and Eduardo Tovar. **How realistic is the mixed-criticality real-time system model?** In *Proceedings of the 23rd International Conference on Real Time and Networks Systems (RTNS)*, pages 139–148, 2015. 7, 117

[27] Rolf Ernst and Marco Di Natale. **Mixed Criticality Systems - A History of Misconceptions?** In *IEEE Design Test*, 2016. 7, 117

[28] Ariel Gomez, Lars Schor, Pranaw Kumar, and Lothar Thiele. **SF3P: a framework to explore and prototype hierarchical compositions of real-time schedulers**. In *Rapid System Prototyping (RSP)*, pages 2–8, 2014. 8, 40

[29] Alan Burns and Sanjoy Baruah. *Timing faults and mixed criticality systems*, **6875**. Springer Berlin Heidelberg, 2011. 10, 13, 14, 48

[30] Arvind Easwaran. **Demand-based scheduling of mixed-criticality sporadic tasks on one processor**. In *Real-Time Systems Symposium (RTSS)*, pages 78–87, 2013. 10, 13, 14, 49

[31] I. Bate, A. Burns, and R. I. Davis. **A Bailout Protocol for Mixed Criticality Systems**. In *Real-Time Systems (RTSS)*, pages 259–268, 2015. 10, 13, 37

[32] Hang Su and Dakai Zhu. **An elastic mixed-criticality task model and its scheduling algorithm**. In *Conference on Design, Automation and Test in Europe (DATE)*, pages 147–152, 2013. 10

[33] Pengcheng Huang, Georgia Giannopoulou, Nikolay Stoimenov, and Lothar Thiele. **Service adaptions for mixed-criticality systems**. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 125–130, 2014. 10

[34] Francois Santy, Laurent George, Philippe Thierry, and Joël Goossens. **Relaxing mixed-criticality scheduling strictness for task sets scheduled with FP**. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 155–165, 2012. 10, 13

[35] Enrico Bini, Marco Di Natale, and Giorgio Buttazzo. **Sensitivity analysis for fixed-priority real-time systems**. *Real-Time Systems*, **39**(1-3):5–30, 2008. 10

[36] Alan Burns and Sanjoy Baruah. **Towards a more practical model for mixed criticality systems**. *Real-Time Systems Symposium (RTSS)*, pages 1–6, 2013. 10, 13

[37] Sanjoy Baruah, Vincenzo Bonifaci, Gianlorenzo Dangelo, Haohan Li, Alberto Marchettispaccamela, Nicole Megow, and Leen Stougie. **Scheduling Real-Time Mixed-Criticality Jobs**. *IEEE Transactions on Computers*, **61**(8):1140–1152, 2012. 11

[38] Yann-Hang Lee, Krishna P Reddy, and C Mani Krishna. **Scheduling techniques for reducing leakage power in hard real-time systems**. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 105–105, 2003. 11

[39] Kai Huang, Luca Santinelli, Jian-Jia Chen, Lothar Thiele, and Giorgio C Buttazzo. **Applying real-time interface and calculus for dynamic power management in hard real-time systems**. *Real-Time Systems*, **47**(2):163–193, 2011. 11, 21, 89, 94

[40] Muhammad Ali Awan and Stefan M Petters. **Enhanced race-to-halt: A leakage-aware energy management approach for dynamic priority systems**. In *Real-Time Systems (ECRTS)*, pages 92–101, 2011. 11

[41] Ernesto Wandeler and Lothar Thiele. **Real-time interfaces for interface-based design of real-time systems with fixed priority scheduling**. In *Conference on Embedded Software (EMSOFT)*, pages 80–89, 2005. 12, 24, 25, 82, 83, 91

[42] Lothar Thiele, Ernesto Wandeler, and Nikolay Stoimenov. **Real-time interfaces for composing real-time systems**. In *Conference on Embedded Software (EMSOFT)*, pages 34–43, 2006. 12, 24, 91

[43] James Barhorst, Todd Belote, Pam Binns, Jon Hoffman, James Paunicka, Prakash Sarathy, John Scoredos, Peter Stanfill, Douglas Stuart, and Russell Urzi. **A research agenda for mixed-criticality systems**. *Cyber-Physical Systems Week*, 2009. 12

[44] Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. **Real-time calculus for scheduling hard real-time systems**. In *International Symposium on Circuits and Systems (ISCAS)*, pages 101–104, 2000. 14, 24, 47, 53

[45] Sanjoy K Baruah, Aloysius K Mok, and Louis E Rosier. **Preemptively scheduling hard-real-time sporadic tasks on one processor**. In *Real-Time Systems Symposium (RTSS)*, pages 182–190, 1990. 14, 19

[46] Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*. Springer, 2001. 20, 50, 84, 100, 101

[47] Kai Huang, Luca Santinelli, Jian-Jia Chen, Lothar Thiele, and Giorgio C Buttazzo. **Adaptive dynamic power management for hard real-time systems**. In *Real-Time Systems Symposium (RTSS)*, pages 23–32, 2009. 21, 83

[48] Samarjit Chakraborty, Simon Künzli, and Lothar Thiele. **A General Framework for Analysing System Properties in Platform-Based Embedded System Designs.** In *Conference on Design, Automation and Test in Europe (DATE)*, pages 190–195, 2003. 24, 25, 47, 53, 56

[49] Neil Audsley, Alan Burns, Mike Richardson, Ken Tindell, and Andy J Wellings. **Applying new scheduling theory to static priority pre-emptive scheduling**. *Software Engineering Journal*, **8**(5):284–292, 1993. 27

[50] Iain Bate and Alan Burns. **An integrated approach to scheduling in safety-critical embedded control systems**. *Real-Time Systems*, **25**(1):5–37, 2003. 37

[51] Enrico Bini and Giorgio C Buttazzo. **Measuring the performance of schedulability tests**. *Real-Time Systems*, **30**(1-2):129–154, 2005. 38, 107

[52] **Raspberry Pi 3 Board**. `https://www.raspberrypi.org/products/\raspberry-pi-3-model-b/`. Accessed: 2016-07-20. 40

[53] Marek Jersak. *Compositional scheduling analysis using standard event models*. PhD thesis, University of Braunschweig-Institute of Technology, 2005. 45, 46, 50

[54] Rafik Henia, Arne Hamann, Marek Jersak, Razvan Racu, Kai Richter, and Rolf Ernst. **System level performance analysis–the SymTA/S approach**. *IEE Proceedings-Computers and Digital Techniques*, **152**(2):148–166, 2005. 46, 47, 50

[55] Ernesto Wandeler. *Modular performance analysis and interface-based design for embedded real-time systems*. PhD thesis, ETH Zurich, 2006. 46, 47, 50, 51, 54, 56

[56] Björn B Brandenburg, Hennadiy Leontyev, and James H Anderson. **An overview of interrupt accounting techniques for multiprocessor real-time systems**. *Journal of Systems Architecture*, **57**(6):638–654, 2011. 46

[57] Ken W Tindell, Alan Burns, and Andy J. Wellings. **An extendible approach for analyzing fixed priority hard real-time tasks**. *Real-Time Systems*, **6**(2):133–151, 1994. 46

[58] ARINC Specification. **653-2: Avionics Application Software Standard Interface: Part 1-Required Services**. Technical report, Avionics Electronic Engineering Committee, 2006. 46

[59] Sanjoy Baruah and Alan Burns. **Implementing mixed criticality systems in Ada**. In *Reliable Software Technologies-Ada-Europe 2011*, pages 174–188. Springer, 2011. 48

[60] Nan Guan, Pontus Ekberg, Martin Stigge, and Wang Yi. **Improving the scheduling of certifiable mixed-criticality sporadic task systems**. Technical report, Department of Information Technology, Uppsala University, 2013. 49

[61] Moritz Neukirchner, Kai Lampka, Sophie Quinton, and Rolf Ernst. **Multi-mode monitoring for mixed-criticality real-time systems**. In *Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 1–10, 2013. 49, 82, 84, 85, 86, 90

[62] Moritz Neukirchner, Philip Axer, Tobias Michaels, and Rolf Ernst. **Monitoring of workload arrival functions for mixed-criticality systems**. In *Real-Time Systems Symposium (RTSS)*, pages 88–96, 2013. 49, 84, 85, 86, 90

[63] Nikolay Nikolaev Stoimenov. *Compositional Design and Analysis of Distributed, Cyclic, and Adaptive Embedded Real-Time Systems*. PhD thesis, The University of Adelaide, Australia, 2011. 49

[64] Kai Lampka, Kai Huang, and Jian-Jia Chen. **Dynamic counters and the efficient and effective online power management of embedded real-time systems**. In

Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), pages 267–276, 2011. 51, 56, 84, 88

[65] Neil C Audsley. *Optimal priority assignment and feasibility of static priority tasks with arbitrary start times.* Citeseer, 1991. 55

[66] John P Lehoczky. **Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines.** In *Real-Time Systems Symposium (RTSS)*, pages 201–209, 1990. 60

[67] Simon Schliecker, Jonas Rox, Matthias Ivers, and Rolf Ernst. **Providing accurate event models for the analysis of heterogeneous multiprocessor systems.** In *Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 185–190, 2008. 60, 61

[68] Moritz Neukirchner, Sophie Quinton, Tobias Michaels, Philip Axer, and Rolf Ernst. **Sensitivity analysis for arbitrary activation patterns in real-time systems**. In *Conference on Design, Automation and Test in Europe (DATE)*, pages 135–140, 2013. 61

[69] Ernesto Wandeler and Lothar Thiele. **Real-Time Calculus (RTC) Toolbox**. `http://www.mpa.ethz.ch/Rtctoolbox`, 2006. 64, 106

[70] Haohan Li and Sanjoy Baruah. **An algorithm for scheduling certifiable mixed-criticality sporadic task systems**. In *Real-Time Systems Symposium (RTSS)*, pages 183–192, 2010. 72

[71] Andrea Bastoni, Björn Brandenburg, and James Anderson. **Cache-related preemption and migration delays: Empirical approximation and impact on schedulability**. *Proceedings of Operating Systems Platforms for Embedded Real-Time (OSPERT)*, 2010. 77

[72] Ernesto Wandeler, Alexander Maxiaguine, and Lothar Thiele. **On the use of greedy shapers in real-time embedded systems**. *ACM Trans. Embed. Comput. Syst.*, **11**(1):1–22, 2012. 82, 83, 84, 90

[73] Linh TX Phan and Insup Lee. **Improving schedulability of fixed-priority real-time systems using shapers**. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 217–226, 2013. 82, 85

[74] Sebastian Tobuschat, Moritz Neukirchner, Leonardo Ecco, and Rolf Ernst. **Workload-aware shaping of shared resource accesses in mixed-criticality systems**. In *Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 1–10, 2014. 82, 84, 85, 86, 90

[75] Moritz Neukirchner, Tobias Michaels, Philip Axer, Sophie Quinton, and Rolf Ernst. **Monitoring arbitrary activation patterns in real-time systems**. In *Real-Time Systems Symposium (RTSS)*, pages 293–302, 2012. 82, 84

[76] Farhana Dewan and Nathan Fisher. **Efficient admission control for enforcing arbitrary real-time demand-curve interfaces**. In *Real-Time Systems Symposium (RTSS)*, pages 127–136, 2012. 84

[77] Kai Huang, Gang Chen, Christian Buckl, and Alois Knoll. **Conforming the runtime inputs for hard real-time embedded systems**. In *Design Automation Conference (DAC)*, pages 430–436, 2012. 84

[78] Biao Hu, Kai Huang, Gang Chen, and Alois Knoll. **Evaluation of runtime monitoring methods for real-time event streams**. In *Design Automation Conference (ASP-DAC)*, pages 582–587, 2015. 84

[79] Biao Hu, Kai Huang, Gang Chen, Long Cheng, and Alois Knoll. **Evaluation and Improvements of Runtime Monitoring Methods for Real-Time Event Streams**. *ACM Transactions on Embedded Computing Systems*, **15**(3):1–26, 2016. 84

[80] Kai Lampka, Simon Perathoner, and Lothar Thiele. **Analytic real-time analysis and timed automata: a hybrid method for analyzing embedded real-time systems**. In *Conference on Embedded Software (EMSOFT)*, pages 107–116, 2009. 87

[81] **Model S Software Version 7.0**. `https://www.tesla.com/presskit/autopilot`. Accessed: 2016-09-07. 116

[82] **BMW autonomous car plan**. `http://www.bmwblog.com/tag/bmw-autonomous-car/`. Accessed: 2016-09-07. 116

[83] **Graphhopper**. `https://graphhopper.com/`. Accessed: 2016-09-07. 118

[84] John Canny. **A computational approach to edge detection**. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986. 120

[85] **Optical character recognition**. https://en.wikipedia.org/wiki/Comparison_of_optical_character_recognition_software. Accessed: 2016-09-07. 120

[86] David H Douglas and Thomas K Peucker. **Algorithms for the reduction of the number of points required to represent a digitized line or its caricature**. *Cartographica: The International Journal for Geographic Information and Geovisualization*, **10**(2):112–122, 1973. 120

[87] Ronald L Graham. **Bounds for certain multiprocessing anomalies**. *Bell System Technical Journal*, **45**(9):1563–1581, 1966. 122

[88] Lai Tsung-Chyan, Yu N Sotskov, N Yu Sotskova, and F Werner. **Optimal makespan scheduling with given bounds of processing times**. *Mathematical and Computer Modelling*, **26**(3):67–86, 1997. 122

[89] Sergey V Sevastianov and Gerhard J Woeginger. **Makespan minimization in open shops: A polynomial time approximation scheme**. *Mathematical Programming*, **82**(1-2):191–198, 1998. 122

[90] Johnny C Ho and Johnny S Wong. **Makespan minimization for m parallel identical processors**. *Naval Research Logistics (NRL)*, **42**(6):935–948, 1995. 122

[91] Premysl Sucha, Michal Kutil, Michal Sojka, and Zdenek Hanzálek. **Torsche scheduling toolbox for matlab**. In *Computer Aided Control System Design (CACSD)*, pages 1181–1186, 2006. 122

[92] Guillem Bernat, Antoine Colin, and Stefan M Petters. **WCET analysis of probabilistic hard real-time systems**. In *Real-Time Systems Symposium (RTSS)*. IEEE, 2002. 123

[93] Reinhold Heckmann and Christian Ferdinand. **Worst-case execution time prediction by static program analysis**. In *Parallel and Distributed Processing Symposium (PDPS)*, 2004. 123

[94] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, et al. **The worst-case execution-time problem-overview of methods and survey of tools**. *ACM Transactions on Embedded Computing Systems (TECS)*, **7**(3):36, 2008. 123

[95] Lukas Sigrist, Georgia Giannopoulou, Pengcheng Huang, Andres Gomez, and Lothar Thiele. **Mixed-criticality runtime mechanisms and evaluation on multicores**. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 194–206, 2015. 127