UNIVERSITY OF HERTFORDSHIRE

Ultra-survivable Mixed-Criticality Systems based on Empiric Worst-Case Execution Time and Criticality Arithmetic

Sajid Fadlelseed

This thesis is submitted to the University of Hertfordshire for partial fulfilment of the requirements for the degree

Doctor of Philosophy

December 2023

Abstract

Most of the existing mixed-criticality schedulers don't take into account the fact that low-criticality tasks operate as a group to implement one or more high-criticality services. Consequently, arbitrary dropping of any of them may compromise the safety of the high-criticality service they collectively implement. In addition, it is hard to estimate a safe upper-bound of service execution time, which implies disruptive degradation in case of execution time overrun by any of the high or low-criticality services. This research introduces novel approaches for mixed-criticality systems by building dependable services from many less dependable services and proportionate adaptation to the empiric execution time overrun problem for systems services during the system mission.

The presented approaches are mid and short-term mixed-criticality schedulers, Critical Arithmetic Adaptive Tolerance-based Mixed-criticality Protocol (ATMP-CA) and Criticality Arithmetic Lazy Bailout Protocol (LBP-CA), based on Criticality Arithmetic (CA), and the framework E-ATMP based on the Empiric Worst Case Execution Time (EWCET). Criticality Arithmetic schedulers change the system configuration in case of core failures or systems transitions between normal and criticality runtime modes. EWCET is initially the determined optimistic EWCET estimate but gets updated during runtime to a higher value whenever a Worst-Case Execution Time (WCET) overrun occurs, and dynamically re-allocates schedules of mixed-criticality tasks using the E-ATMP framework. Both approaches deliver smoother degradation than reference schedulers in the literature.

Build and architect, systems, and criticality, from many less dependable components, and represent criticality by the architecture of these components, respectively, with incorporating adaptive responses based on empiric information during the system mission, and foreseen future, guarantees smooth degradation to the total system utility when transient or permanent resource shortages occur.

Acknowledgements

I wish to express my gratitude for the support and guidance provided by my supervisor, Dr. Raimund Kirner, throughout my PhD journey. His wealth of knowledge, expertise, and experience has proven to be invaluable in shaping my research. Additionally, I extend my thanks to my second supervisor Dr. Xianhui Che, for her valuable contributions to my work. I also appreciate the guidance and encouraging feedback from Dr. Catherine Menon, which has consistently motivated me to enhance the quality of my research. I extend my thanks to the National University - Sudan, for sponsoring me to pursue a Ph.D. in Computer Science.

Special appreciation goes to my father, Prof. Qurashi Ali, for his wisdom, motivation, and support. I am also grateful to my loving and kind mother, Mrs. Amira Makki, and my siblings, Dr. Ahmed, Dr. Asim, Areej, Arwa, and Thabit, whose guidance has consistently been my greatest source of strength. The utmost heartfelt appreciation goes to my life's irreplaceable treasure, Amani, for her unwavering support, love, patience, and kindness during the most challenging times. And to my beloved souls in my life, my daughter Amira and my son Qurashi.

I extend my thanks to my colleagues at STRI for sharing in our struggles, laughter, and camaraderie throughout this Ph.D. journey. Gratitude is also owed to all the students I have encountered, whose contributions have enriched my knowledge.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter presents the context of this thesis and explains the main research question, derived sub-questions, and contributions. Then, the chapter outlines the success criteria for evaluating the research's answers. Following that it presents relevant publications and provides an overview of the thesis structure.

Section 1.1 presents the research context and significance. Section 1.2 addresses the main research question, including sub-questions. Section 1.3 demonstrates the overview of the research's answers in response to the research question. Section 1.4 outlines the success criteria for evaluating the research's answers. Section 1.5 presents relevant publications. Section 1.6 provides an overview of the thesis structure. Section 1.7 concludes this chapter.

1.1 Research Context

The failure of a safety-critical system, such as the Anti-lock Braking System (ABS), or auto-pilot in an aircraft, or spaceship landing system, can result in severe consequences, including loss of life, property damage, or environmental harm. Such systems perform critical functions with a high degree of reliability, safety, and security. They require strict adherence to safety standards e.g. International Electrotechnical Commission (IEC) 61508, to ensure their correct operation, especially in case of failures and resource shortages. Each program/service/component/task provided by one of these systems has a specific level of safety Criticality. In the past, these systems were often designed to operate on platforms where each processor executes services with the same level of Criticality. However, recent advancements in hardware technologies put back considerations for efficient usage of computing resources by integrating services with different criticality levels on the same processor, instead of the partitioned criticality systems, while ensuring the correctness of the system and running services.

The Safety Integrity Level (SIL) of system services is a critical concept in many domains. Each of these domains has standards that specify vocabulary for indicating a service's safety integrity, for example, the International Standardisation Organisation (ISO) 26262 standard, Automotive Safety Integrity Levels (ASIL), and the Design Assurance Levels DO-178C (DAL) in the Aerospace Recommended Practice 4654 (ARP). The ISO 26262 is an international standard for the functional safety of electrical and/or electronic systems that are installed in serial production road vehicles [3]. The DAL in ARP is a classification system that categorises the potentially catastrophic effects of software failure in airborne systems. The allocated SIL to a service indicates information about the safety levels needed when developing a component. Validating a service during the development with high SIL, requires higher strictness than developing the same service to a low SIL level. Integrating services with different criticalities on the same platform requires considering the safety integrity levels required for system services and components.

Mixed Criticality systems are high-integrity safety-critical systems that integrate services of different criticalities on a single or common platform while preserving the same safety integrity levels of the systems services and components. The theory for modelling, scheduling, and verification of mixed-criticality systems was introduced in a seminal paper by Steve Vestal in 2007 [4]. Modelling mixed-criticality systems considers defining the safety integrity and criticality levels of system services. It also includes specifying runtime modes activated whenever a resource shortage occurs, along with constraints on system services based on their criticality levels during each activated runtime criticality mode in the system. Scheduling mixed-criticality systems involves ensuring that all services meet their timing constraints, while also preventing services with Low Criticality (LO) from interfering with services with High Criticality (HI) during critical/non-critical modes throughout the system operation. The verification of mixed criticality is a pre-runtime analysis for the safety Integrity levels of the mixed-criticality system services and components, during normal and critical runtime criticality modes.

Scheduling mixed-criticality services on uni and multi-core processors extended the Real-time Scheduling Theory (RTSh) scheduling for dependable computing. In partitioned safety-critical systems, RTSh is applied to coordinate the execution of these services based on the concept of priority, where services are scheduled in order of priority to ensure they meet their timing constraints, regardless of their criticality or safety integrity levels. In a priority-based scheduling approach, services with the same criticality are prioritised based on a defined model or priority scheme established by the system designer. This model could be based on *Fixed* or *Dynamic* priority assignment. An example of Fixed-priority is prioritising services based on their arrival frequency, this approach often leads to the use of Fixed-priority Rate Monotonic (RM) scheduling [5], as it is the optimal Fixed-priority scheduling algorithm when priorities are assigned based on the rate of service execution. An example of dynamic priority is prioritising services based on the earliest deadline at any scheduling point or decision, this approach leads to the use of Earliest Deadline First (EDF) scheduling [5], as it is the optimal dynamic-priority scheduling algorithm when priorities are assigned based on the earliest deadline. Vestal found that RM scheduling algorithm is not optimal for mixed-criticality services [4]. In addition, Baruah et al. found that neither EDF is optimal [6].

Mixed-criticality scheduling theory provides the framework for scheduling mixed-criticality service sets using augmented models for real-time scheduling algorithms [7]. The framework provides pre-runtime verification analysis that assures the safety integrity of the system services during normal execution and resource shortages, before the system deployment. A priori assurance for the system safety integrity, despite the runtime mode of the system, is of utmost consideration for dependable systems. It also provides runtime monitoring that provisioning the consumption of CPU time by system services and applies scheduling decisions by delaying, compromising, or even aborting the execution of services that exceed their allocated budget for execution time [8, 9].

Determining the execution time of service is an open problem [10] [11, 1]. The complexity of possible execution paths, and the size of possible input, complicates determining an exact execution time for service, despite its safety integrity level or criticality. Therefore, only *upper* bounds for *best* and *worst* case scenarios can be derived to determine the Best-Case Execution Time (BCET) and Worst-Case Execution Time (WCET) respectively. Different methods are used to measure services WCET, which are categorised into *static* and *dynamic* methods. The static methods are based on analysing the program's code along with the target micro-architecture that the code will execute on. The dynamic methods are simpler than the static methods because they are based on running the program's code with different input sizes and different processor states. Each run is then profiled, and the one with the largest execution time is considered to be the worst-case execution time of the service [12, 11, 1]. Schedulability analysis for the integrity of system services is then performed based on this information for each service.

Underestimating the WCET of a service may cause the service to overrun the allocated WCET, which results in violating the assumptions made during the pre-runtime verification analysis and the interference from lower criticality services may violate the safety integrity of higher criticality services. Also, overestimating the WCET results in inefficient usage for CPU time by allocating more time to services who already completed their needed computation. In both cases, underestimate or overestimate, other services with lower priority than running service may fail to meet their deadlines. If the failed services are of criticality lower than the criticality of the service overrun its WCET, it can be acceptable to delay, compromise, or abort the lower criticality services to avoid interrupting service of higher criticality. However, If the service overruns its WCET is of criticality lower than services currently released and waiting for execution, a delay, compromise, or aborting to the higher criticality service, violates the basic rule of mixed-criticality systems for integrating the services on the same platform, without compromising the safety integrity levels of each service [4, 13, 9].

The causes of resource shortages can take different forms, such as WCET overrun, core failures, and reduced computing capacity a.k.a energy-saving modes. [14, 15]. The impact of resource shortage is defined by its type and duration, and the effect is a form of degradation in the overall system safety and *utility* [16, 17, 18]. This effect of degradation in the system, in the context of mixed-criticality systems, is bounded according to the concrete *assumptions* and *expectations* during the pre-runtime analysis conducted before the system deployment [7]. Mixed-criticality systems ensure that services in such situations are prioritised based on criticality, to give available remaining resources - resource is remaining cores in case of core failure or Central Processing Unit (CPU) time in case ofWCEToverrun - to services with high criticality. Mixed-criticality systems also minimise the effect of reduced priority on low-criticality services in response to resource shortages.

Existing mixed-criticality scheduling models, schemes, and frameworks differ in bounding the level of resulting degradation on the system, and in the level of survivability when assumptions and expectations, established during pre-runtime verification, are violated during the system mission. The degradation in most existing research is often considered disruptive, lacking quantified responses to bind the impact of a resource shortage based on its type and duration during the system mission. Indeed, there are different interpretations for smooth or *graceful degradation* in real-time and mixed-criticality literature, and few sources that provide concrete models express smooth/graceful degradation.

In the context of real-time systems, an abstract definition by Avivzienis [19] mentioned that *fail-operational*, *fail-soft*, and *graceful degradation* are different names pointing to the same concept. This concept is the deviation from the *correct-execution*, which models the normal behaviour of the system [19]. Avivzienis stated that "The criteria for correct execution of a set of [services]" [19] are:

a. The [services] and their data are not altered or halted by faults.

b. The results of [services] do not contain fault-caused errors.

c. The execution time of each [service] does not exceed [WCET].

d. The storage capacity that is available for each [service] remains above a specified minimum value.

The system degrades gracefully in one of the following two cases:

1. Some of all services fails to meet the (a) and (b).
2. Some or all services fail to satisfy (c) and (d).

The following Truth-table translates Avivzienis definition 1.1 for graceful/smooth degradation.

Services		Criteria				Graceful-degradation
some	all	a	b	c	d	
F	F	N/A				F
F	T	T	T	F	F	T
T	F	F	F	T	T	T
T	T	N/A				F

Table 1.1: Avivzienis Truth-table for smooth/graceful degradation

The downside of this definition is that the abstract criteria named "d" cannot ascend to a concrete implementation, as there is no guarantee that services won't exceed their allocated WCET estimate .

Burns et al. provided a concrete definition for graceful degradation in the context of survivability for mixed-criticality systems scheduling [20]. It is based on tolerating a bounded number of high and low-criticality services' WCET overruns before aborting or compromising lower criticality services. The authors stated that "The key to graceful degradation is that the response to such failures must be commensurate with the magnitude of the failure" [20]. They classified the response, per failure magnitude, to the following criteria:

a. Full-operational: all [services] execute correctly (i.e. meet their deadlines).

b. Fail-robust: [some high-criticality services are allowed to skip number ofWCEToverruns].

c. Fail-resilient: [some low-criticality services are delayed or aborted].

d. Fail-safe/restart: best-effort and restart techniques that have no guarantees.

The system degrades gracefully when it evaluates to the criteria "d," making it Fail-resilient. Hence, under their definition, graceful degradation starts after defined magnitude of tolerable WCET overruns by high-criticality services is exceeded, during Fail-robust, and compromising low-criticality tasks is a must. The following Truth-table translates Burns et al. definition 1.3 for graceful/smooth degradation.

Table 1.2: Add caption

WCET overrun		Graceful-degradation
Robustness	Resilience	
F	F	F
F	T	T
T	F	F
T	T	T

Table 1.3: Buns Truth-table for smooth/graceful degradation

The downside of this definition is that it does not incorporate the global utility of the system, defined as the sum of the utility per service for all services [18, 21]. In this context, dropping a low-criticality service with low utility is preferred over dropping a low-criticality service with high utility. In addition, arbitrary dropping or aborting for low-criticality services may compromise the safety of high-criticality services implemented by a many low-criticality services, a known practice in industry for realising service of high integrity level by a many services with low integrity levels [22].

This thesis discusses answers to questions about defining and evaluating smooth degradation in mixed-criticality protocols on both uni and multi-core processor platforms in case of transient and permanent resource shortages, adaptivity in response to WCET overrun, and building highly dependable components from many less dependable components. The following section presents the research questions that motivated and guided the findings in this thesis.

1.2 Research Questions

The aim of this thesis is to answer questions about defining and evaluating smooth degradation in mixed-criticality protocols on both uni and multi-core processor platforms in case of transient and permanent resource shortages, adaptivity in response to WCET overrun, and building highly dependable components from less dependable components. Five objectives influenced the contributions and answers, which involved investigating the literature to understand smooth degradation definitions and techniques, and implied the implementation for different solutions for handling resource shortages, including the evaluation between existing and novel ones developed during the thesis.

The first objective is to form a definition for smooth degradation for mixed-criticality systems during resource shortages. During critical behaviour of the system, due to resource shortages, the criticality of the service is incorporated in granting resources to mixed-criticality services. The second objective is to investigate existing policies and protocols under Fixed-priority based scheduling for granting resources to mixed-criticality services, and how each protocol provides different adaptation techniques, and levels of degradation in response to resource shortages. The third objective is to determine evaluation metrics and develop a simulator that facilitates flexibility in defining the permanent and transient resource shortages, mixed-criticality, levels of degradation, and the evaluation between two or more protocols. In real-time systems, dependability is enabled by the use of Software (SW)/Hardware (HW) redundancy. The fourth objective is to develop mixed-criticality protocols that provide smoother degradation, considers systems with high criticality services constituted out of many redundant low criticality services. The fifth objective is to exploit monitoring facilities for execution time or watchdog timers, which are essential components in mixed-criticality systems, to enable the detection of WCET overrun, and trigger adequate responses for recovery.

RQ: *How to provide smooth degradation (also called graceful degradation) for integrated systems with services of mixed-criticality in cases of resource shortages?*

This the main research question. In this research project, six sub-questions have been identified as critical and important for the success of the thesis.

RQ1: *What is a suitable definition of smooth degradation?*

Investigating the types and durations of resource shortages using the Tolerance-based Real-Time Computing Model (TRTCM) model [23], which establishes a relationship between resource shortages and the overall system utility. This is answered in Chapter 3.

RQ2: *What protocols are possible to grant resources to services of different criticality in case of resource shortage?*

Research existing approaches and adaptation techniques to tolerate transient and permanent faults in mixed-criticality systems, and identify methods for allocating resources to mixed-criticality services. This is answered in Chapter 7.

RQ3: *How can we evaluate and compare the smooth degradation of one protocol against another?*

Evaluating the adaptivity of a certain mixed-criticality scheduling protocol can be realised by terms of achieved system utility, and number of successful and compromised services. Designing and implementing a simulator to simulate resource shortages on mixed-criticality systems is important for experimental evaluation by implementing the found protocols during the research. This is answered in Chapter 7.

RQ4: *How can we provide higher levels of dependability from less dependable components?*

Exploring of existing fault-tolerance techniques to realise dependability, and how these techniques are implemented in both the literature and real-world standards for industrial safety-critical systems. This is answered in Chapter 4.

RQ5: *How can the system adapt based on occurrences of services' overruns ofWCETestimates?*

Exploring the available adaptation techniques for theWCEToverrun in the literature, and whether adopting the sameWCETafter overrun, under any technique, is better, or adapting the system schedule based on the recently observed new, longer WCET. This is answered in Chapter 5.

RQ6: *What is a novel mixed-criticality scheduling protocol that provides an improvement on smooth degradation of service in case of resource shortage?*

Developing protocols and frameworks that enable building dependable mixed circularity systems, that can tolerate permanent and transient resource shortages while providing smooth degradation to the system. This is answered in Chapter 5.

1.3 Contributions

This research has focused on three main directions. Building highly dependable components, from many less dependable components. Evaluating and improving the smooth degradation for Mixed-criticality scheduling protocols on uni and multi-core. Developing a framework that provides adaptive responses toWCET overrun, while improving the smooth degradation. Figure 1.1 shows the relation between the existing approaches in the literature and the novel protocols developed during the research of this thesis, including the evaluations between existing ones.

- A concrete model for Criticality Arithmetic (CA). that enables realising high criticality components from many low criticality components. The research found that the awareness of CA by mixed-criticality protocols enhances the system dependability, and smoother degradation for free. The limitation is that it is limited to two or three criticality levels. This contribution contributes in answering research question RQ4, it is discussed in Chapter 4 and evaluated in Chapter 7 Section 7.1.

- Mid-term Criticality Arithmetic (CA) protocol. ATMP-CA uses information about CA to enable building highly dependable systems from less dependable components. ATMP-CA shows smoother degradation than compared protocols, Standard Adaptive Mixed-Criticality Protocol (SAMP), and Adaptive Tolerance-based Mixed-criticality Protocol (ATMP). The limitation of the study is that it's based on two criticality levels. This contribution contributes in answering research question RQ4, it is discussed in Chapter 4 Section 4.1 and evaluated in Chapter 7 Section 7.1.

- Short-term Criticality Arithmetic (CA) protocol. The Criticality Arithmetic Lazy Bailout Protocol (LBP-CA) uses CA to enable building dependable systems for a quicker return to normal or low criticality mode, which improves the smooth degradation compared to the reference protocols, Bailout Protocol (BP), and Lazy Bailout Protocol (LBP). The limitation of this study is that it allows the drop of high criticality task replica in case another replica of the service is completed priorly. This contribution contributes in answering research question RQ4, it is discussed in Chapter 4 Section 4.2 and evaluated in Chapter 7 Section 7.1.

- Empiric Worst Case Execution Time (EWCET) model for online adaptation toWCEToverrun. EWCET for the first job of each service gets initialised with a givenWCETestimate of that service, and for each succeeding job, the EWCET is the maximum of the previous job EWCET and the current job's execution time. Therefore, any update of the EWCET is based on the maximum with the previous job's EWCET. This contribution contributes in answering research question RQ5, it is discussed in Chapter 5 Section 5.1 and evaluated in Chapter 7 Section 7.4.

- Empiric Adaptive Tolerance-based Mixed-criticality Protocol (E-ATMP) framework, which is a reconfiguration method for mixed-criticality systems to provide smooth degradation for services in case of aWCEToverrun occurs. E-ATMP uses the existing Adaptive Tolerance-based Mixed-criticality Protocol (ATMP) framework and adds additional mechanisms to facilitate the deployment of the EWCET updates during run-time. E-ATMP provides a powerful line of defence against the threat ofWCETunderestimations in a world where is increasingly difficult to provide safe and accurateWCETestimates due to the ever-increasing processor HW complexity. This contribution contributes in answering research question RQ5, it is discussed in Chapter 5 Section 5.2 and evaluated in Chapter 7 Section 7.4.

- Evaluation for three sets of analysis for smooth degradation of the achieved system utility between SAMP and ATMP protocols, under computing capacity shortage. It shows the absolute utility achieved and the number of dropped tasks when running the task set over three cases 8, 4, and 2 cores under full and half frequency speed. Then, I compared the three cases between the two protocols, SAMP and ATMP. This contribution is discussed and evaluated in Chapter 7 Section 7.2, which answers research question RQ2.

- An evaluation for the ATMP utility function, as defined by the TRTCM model, and integrated it with the real-time model, the E-MC. The evaluation is a comparison of smooth degradation between TRTCM-ILP and TRTCM-Elastic. TRTCM-Elastic is an integration of the TRTCM utility function and the real-time model, RT Elastic. RT Elastic provides a heuristic function for finding an acceptable configuration for the system services but lacks the utility metrics assigned to each service as featured in TRTCM. This contribution is discussed and evaluated in Chapter 7 Section 7.3, which answers research question RQ2.

- A simulator that allows experimenting with different configurations for system resources and running services by defining different types of resource shortages as mentioned above and reports the achieved overall system utility by each protocol, the number of successfully scheduled services, and the number of dropped or compromised services. This contribution is discussed in Chapter 7 Section 7.3, which answers research question RQ2.

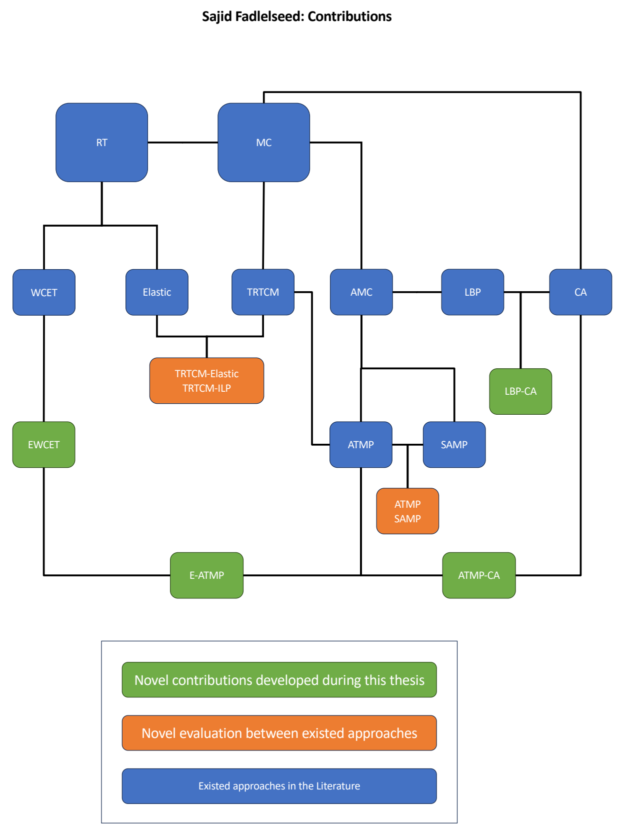

Sajid Fadlelseed: Contributions
RT
MC
WCET
Elastic
TRTCM
AMC
LBP
CA
TRTCM-Elastic
TRTCM-ILP
LBP-CA
EWCET
ATMP
SAMP
ATMP
SAMP
E-ATMP
ATMP-CA
Novel contributions developed during this thesis
Novel evaluation between existed approaches
Existed approaches in the Literature

Figure 1.1: Contributions/Solutions Answer Research Question

1.4 Success Criteria

The success of this thesis in addressing both main and sub-questions is measured through concrete evaluation metrics classified into two categories, and applied in four experiments. The first evaluates smooth degradation based on the achieved system and service absolute utility. The second evaluates smooth degradation based on the number of allocated services to cores, the number of dropped or degraded services during allocation, and the number of executed and aborted services.

The first experiment assesses mid-term CA protocols (SAMP-CA, and ATMP-CA), and CA agnostic protocols (SAMP and ATMP) on a multi-core platform. The second experiment compares the achieved system utility between ATMP and SAMP protocols for service allocation on a multi-core system. The third experiment assesses smooth degradation on the uni-core system between two variants of the TRTCM model: TRTCM-ILP and TRTCM-Elastic. The fourth experiment assesses smooth degradation on the uni-core system between EWCET-aware and EWCET-agnostic protocols.

Evaluaion Metrics: Evaluating smooth transitions between protocols studied in this thesis to answer RQ3 has been accomplished using: Offline and Online Feasibility Analysis, Number of successful services per protocol, Number of failed services per protocol, Number of compromised services per protocol, Number of postponed services per protocol, Ratio of achieved system/service utility between the protocols.

- Offline and Online Feasibility Analysis: This analysis checks whether the system can ensure task feasibility both before deployment (offline) and during operation (online). This analysis is essential in all experiements in this thesis. In the evaluation chapter, Chpater 7, before conducting an experiment, an offline schedulability test is applied to the task set to ensure that they can share the same processor without interference during normal operation of the system or in low-criticality mode. During the experiement, whenever the system experience a resource shortage of core-failure or WCET overrun, an onine test is applied to ensure the schedulability of the task set.

- Number of successful services per protocol: This counts services successfully executed according to each protocol's requirements or constraints.

- Number of failed services per protocol: This counts services that could not be completed within their expected parameters or deadlines for each protocol.

- Number of compromised services per protocol: This counts services that were partially completed with reduced quality or functionality due to a resource shortage.
- Number of postponed services per protocol: This counts services that were delayed beyond their scheduled or desired execution times under each protocol due to a resource shortage.
- Ratio of achieved system/service utility between the protocols: This Compares the overall utility achieved by the system or individual services by each protocol.

1.5 Publications

The work related to CA has been published in two papers, a journal paper, and a conference paper. The work related to the EWCET and E-ATMP framework has been submitted to a journal.

- Sajid Fadlelseed, Raimund Kirner, and Catherine Menon. "ATMP-CA: Optimising Mixed-Criticality Systems Considering Criticality Arithmetic." Electronics 10.11 (2021): 1352.
- Sajid Fadlelseed, Raimund Kirner, and Catherine Menon. "LBP-CA: A Short-term Scheduler with Criticality Arithmetic." Proceedings of Abstracts School of Physics Engineering and Computer Science (2022).
- Sajid Fadlelseed, Che Xianhui, and Raimund Kirner. "E-ATMP: Using Empiric Worst-Case Execution Time to Build Ultra-Survivable Mixed-Criticality Systems." IEEE Transactions on Dependable and Secure Computing (2023) [Submitted]

1.6 Thesis Structure

This section presents the thesis structure

- Chapter 2 presents background on Real-time and Mixed-criticality systems and scheduling. It explains basic concepts and terminologies utilised in this research, it also presents the mixed-criticality use case introduced by Vestal, which triggered the research on mixed-criticality systems scheduling.

- Chapter 3 reviews the related work in the literature. The chapter presents significant contributions in real-time systems research 3.1,WCET analysis 3.2, dependability, 3.3 and mixed-criticality systems scheduling 3.4.
- Chapter 4 presents the CA model, and the developed protocols ATMP-CA and LBP-CA. The evaluation of this chapter is under the experimental evaluation chapter, Chapter 7 in Section 7.1. ATMP-CA is a CA aware allocation and Integer Linear Programming (ILP) formulation for the mapping tasks to cores are presented in Section 4.1.2 and Section 4.1.3 respectively. The chapter summary in Section 4.3, draws the chapter's conclusions.
- Chapter 5 presents the EWCET model and the development of the E-ATMP framework. The evaluation of this chapter is under the experimental evaluation chapter, Chapter 7 in Section 7.4. the EmpiricWCETand E-ATMP framework are presented in Section 5.2. The chapter summary in Section 5.4, states the chapter's conclusions.
- Chapter 6 presents the implementation of developed contributions: ATMP-CA, LBP-CA, EWCET, and E-ATMP. ATMP-CA implementation is presented in Section 6.1, LBP-CA Bailout in Section 6.2, EWCET and E-ATMP in Section 6.3. The presentation of this chapter is based on Unified Modeling Language (UML) diagrams. Each protocol's classes are presented, including the relations between super and derived classes.
- Chapter 7 presents an evaluation of existing protocols in the literature (TRTCM-ILP, TRTCM-Elastic, SAMP, ATMP, BP, and LBP), and the novel contributions developed during the research of this thesis (ATMP-CA, SAMP-CA), LBP-CA, and E-ATMP.
- Chapter 8 Concludes the thesis research findings and contributions in Section 8.1 and 8.2, respectively, including the outlook for future work in Section 8.3. Section 8.4 closes thesis.

1.7 Chapter Summary

In this chapter, I presented the context of this research in Section 1.1, including main and sub-questions in Section 1.2, contributions in Section 1.3, success criteria in Section 1.4, publications in Section 1.5, and finally, the thesis structure in Section 1.6.

Chapter 2

Background

This chapter, Chapter 2, presents background on Real-time and Mixed-criticality systems and scheduling. It explains basic concepts and terminologies utilised in this research, it also presents the mixed-criticality use case introduced by Vestal, which triggered the research on mixed-criticality systems scheduling. Section 2.1 explains the basics of Real-time scheduling. Section 2.2 explains the basics of Mixed-criticality scheduling. Section 2.3 concludes this chapter.

Since Vestal paper in 2007 [4], different approaches and mechanisms [24] have been proposed to address the problem of efficient allocation for computing resources to serve services with different levels of assurance and criticalities. However, the research community has realised the need to extend Real-time scheduling theory to consider the importance of system components based on their priority and criticality levels. Efficient design and implementation for complex mixed safety and mission-critical systems under the constraints of non-functional requirements Sizw, Weight, and Power (SWAP) is a challenge. The correctness of the implementation and the feasibility of the design are verified under the highest assurance for best and worst-case scenarios. Therefore, the integration of these systems with different levels of criticalities has been investigated in depth in the literature with a focus on defining the possible policies to grant resources for services in case of resource shortage. Granting must assure the continuity of critical services with high assurance while affording non-critical services an acceptable degraded level.

The following sections present the basics of Real-time systems and scheduling. Characteristics of real-time systems and tasks are discussed, including the schedulability analysis applied before the system deployment. In addition, Vestal's example that revealed the limitation of the existing priority-assignment algorithm is explained, including the modified real-time priority-assignment al-

```
void realTimeTask(){
  int x;
  while(1){
    x = readSensor();
    x += 2;
  sleep(1000);
  }
}
```

Figure 2.1: Implementation of a real-time task in the C programming language

gorithms that Vestal suggested to consider when developing a mixed-criticality system.

2.1 Real-time Systems and Scheduling

This section explains real-time scheduling. It presents the timing characteristics in real-time services and policies for optimal priority assignment to services. Next, we provide a comprehensive overview of real-time service scheduling and the required schedulability tests. Finally, we discuss existing fault-tolerance methods designed to mitigate the impact of transient, intermittent, and permanent faults.

A real-time system is one in which the correctness of the system depends not only on the logical results of computation, but also on the time in which results are conducted. The output of the system and the time when outputs were generated, determine whether an execution is successful or not. The study of Real-time systems includes the planning and scheduling of workload on the processor cores so that the *timeline* guarantees for workload are never violated. In Real-time systems, workload is quantified into discrete pieces called tasks, which represents the unit for computation in the processor. To illustrate at what point in time such a task can be executed we use the concept of *timeline*, which goes from zero to a larger value that represents the real-world clock time of the system execution, and when the task is scheduled it is placed on the timeline based on the decision of the scheduler to execute its functionality. When a task is scheduled, usually there is a basic code block, a loop, some functionality to loop over, and a delay at the end to determine the period of the task. Here is an implementation of Real-time task τ_i, written in programming language C, that reads sensor data about every 1000 milliseconds:

A real-time task is characterised by the following tuple:

$$\tau = \langle id, p, d, c, ut, wr \rangle$$

- $\tau.id$ task identifier.
- $\tau.p$ task period.
- $\tau.d$ task relative deadline.
- $\tau.c$ task worst-case execution time.
- $\tau.ut$ task load or utilization of CPU time.
- $\tau.wr$ task worst-case response time, depends on other tasks in the task set.

  Arrival of a task $\tau_{i,j}$ is called an *instance* of the task or *job*. The functionality of an instance of a task $\tau_{i,j}$, a job, is defined in the task code. This means that jobs are repeated executions of tasks' functionality.

Real-time tasks and services are classified according to the rate of arrivals. A *periodic* task arrives with constant time intervals. A *sporadic* task arrives in a bounded time interval. A *Aperiodic* task has no guaranteed minimum time between two subsequent arrivals. Real-time tasks and services are also classified according to deadline. *implicit* deadlines equal to periods. *constrained* deadlines less than or equal period. *arbitrary* deadlines can be equal, less, or higher than periods. In case no deadline is given, it's assumed that the relative deadline is equal to the task period. Note that the longer the execution time, the more CPU resources are needed, and if the execution time becomes too large, the system gets overloaded. Real-time tasks can be *independent* or *dependent.* Independent tasks do not share a resource other than the processor time, whereas dependent tasks share a resource other than the processor time, e.g. memory, or rely on the completion of another task to get selected for execution.

Periodic task sets are classified as synchronous, or asynchronous task sets. Tasks that arrive at the same time, simultaneously, are synchronous, but they are asynchronous if their arrivals are sequential. **Worst-case Execution Time (WCET)** The WCET is the longest number of CPU cycles or the longest path in the possible execution paths required by a service. It is obtained either by dynamic or static analysis. Dynamic analysis runs the same task many times with different input types and sizes, as well as different processor states, whereas static analysis analyses the code paths given a certain processor architecture. Different situations may render a task to overrun its WCET. Figure 2.2 from

Wilhelm [1] shows a task with variations in execution times based on input data or different factors related to the environment. The shortest execution time is called the BCET, and the longest time is the WCET.

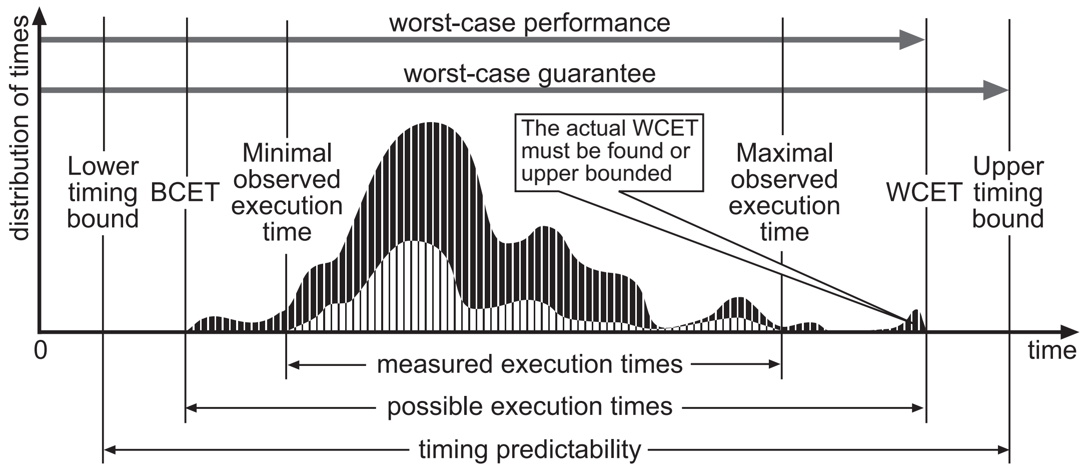


Figure 2.2: Minimum and maximum are the best-case BCET and worst-case WCET [1]

Run-time task model The basic run-time mechanism takes the first task in the ready state *queue* and executes it on the processor.

1. *ready*: in this state, the task is not executing but can execute when the scheduling point is reached.
2. *running*: in this state, the task has been scheduled to the CPU and currently executing.
3. *blocked*: in this state, a task is not scheduled until another task releases a resource.
4. *suspended* completely remove the current of the task from the schedule.

Priority Assignment: Assigning priorities to tasks is obtained either using *fixed* or *dynamic* priority assignment algorithms. A scheduling algorithm generates a schedule for a given set of tasks and a certain type of run-time system. The priority assignment algorithm is implemented by a scheduler that decides what order tasks should be executed. A schedule is *feasible* if all tasks in a task set meet their deadlines according to certain priority assignments. A set of tasks are *schedulable* if there is a scheduling algorithm that can generate a feasible schedule.

Schedulability Test: Schedulability tests are classified to *sufficient*, *necessary*, and *exact*. The schedulability test is *sufficient* if it shows that a set of tasks is schedulable. The schedulability test is *necessary* if it shows that a set of tasks is not not-schedulable. The schedulability test is *exact* (sufficient and necessary) if it shows that a set of tasks is schedulable.

Fixed-priority Assignment: In Rate Monotonic (RM) [5], the priority of a task $\tau_i.pr$ is determined by the rate or period of the request for the task execution. The rate of requests per task is the reciprocal of the task's period $\frac{1}{\tau_i.p}$. Tasks with lower periods (higher rates) have the highest priorities. RM is optimal for all task sets with implicit deadlines over all Fixed-priority assignment algorithms. The schedulability test is sufficient. In Deadline Monotonic (DM) [25], the priority of a task $\tau_i.pr$ is determined by the deadline of the task. Tasks with shorter relative deadlines are assigned the highest priorities. DM is optimal for all task sets with constrained deadlines $\tau_i.d$ over all Fixed-priority assignment algorithms.

Dynamic-priority assignment: In Earliest Deadline First (EDF) [5], The priority of a task for execution is dynamically determined by the absolute deadline of a task. Tasks with the closest absolute deadline get the highest priority. The schedulability test is exact. EDF is optimal for any task-sets with implicit deadlines over all dynamic priorities schedulers. [26].

Response Time Analysis (RTA): RTA is an exact schedulability test for Fixed-priority assignment schedulers. Started by Harter and extended by Pandya and Audsley [27, 28][29][30][31]. RTA is a fixed point iteration that calculates the Worst-case Response Time (WCRT) $\tau_i.wr$ for each task and then compares this value with the task's deadline $\tau_i.d$, and if the response time $\tau_i.wr$ converges to a value less than or equal to the task deadline $\tau_i.d$ then it is schedulable, otherwise, it is not [32]. The RTA test is applied to each task τ_i against tasks with a priority higher than the task in question. The response time value is obtained from the following, where HP is a set of tasks with priority higher than task τ_i priority $\tau_i.pr$:

$$\tau_i.wr = \tau_i.c + \sum_{\tau_j.pr \in HP(\tau_i.pr)} \left\lceil \frac{\tau_i.wr}{\tau_j.p} \right\rceil \cdot \tau_j.c \tag{2.1}$$

The RTA is valid under the following assumptions:

- uni-core processor system

- synchronous tasks
- independent tasks
- periodic tasks
- tasks with implicit or constrained deadlines.
- context switch and all overheads are assumed to be 0.
- a task can't abandon or drop itself.
- tasks WCET is less or equal to task deadlines.

Least Upper Bound (LUB) Another feasibility test for the Static and Dynamic Priorities technique enables calculating the accumulated utilization U_{bounds} of all tasks in the system. Started by Liu et al. [5] and Fineberg et al. [33] [29]. If the resulted U value does not exceed a guarantee bound, all timing constraints will be met [5]. The utilization is the fraction of processor time that is used for executing the task set. The utilization of a task $\tau.ut$ is equal to $\tau_i.ut = \frac{\tau_i.c}{\tau_i.p}$. The total utilization of the processor time for all tasks is $\sum_{i=1}^{n} \tau_i.ut$. LUB provides a sufficient schedulability test. This bound converges to 69% as the number of tasks approaches infinity. [5]:

$$\lim_{n \to \infty} U_{RMbounds} = n(2^{1/n} - 1) \leq 0.68 \tag{2.2}$$

Under the EDF algorithm, tasks can use 100% utilization for the processor time.

$$U_{EDFBound} = \sum_{n=1} \frac{\tau_i.c}{\tau_i.p} \leq 1 \tag{2.3}$$

The LUB is valid under the following assumptions:

- uni-core processor system
- synchronous tasks
- independent tasks
- periodic tasks
- tasks with implicit deadlines.
- context switch and all overheads are assumed to be 0.
- a task can't abandon or drop itself.

- tasks WCET is less or equal to task deadlines.

Server-Based Scheduling In server-based scheduling, a designated task with the highest priority acts as a *server*[34]. This server collects any surplus unused slack time during system execution and then allocates this collected WCET to specific tasks based on the server algorithm. This allows tasks to utilise surplus execution time efficiently. It is mainly used for scheduling aperiodic tasks. Aperiodic servers for Fixed-priority scheduling are: Polling Server, Deferrable Server, Sporadic Server, Slack Stealer [15]. Aperiodic servers for Dynamic-priority scheduling are Dynamic Polling Servers, Dynamic Sporadic Servers, Total Bandwidth Servers, Tunable Bandwidth Servers, and Constant Bandwidth Servers [15].

Classification of Multi-processor Architectures: Different types of multi-processors and multi-core systems can be classified based on their capabilities and speed. This includes *Homogeneous*, *Uniform Heterogeneous*, and *Non-uniform Heterogeneous*. By processor capabilities, we mean the size of cache memory, energy efficiency, etc., where processor speed refers to the clock speed of the processor; a higher clock speed indicates a faster processor. Table 2.1 presents the capabilities and speed by each classification. The speed between processors is equal only in Homogenous systems, where capabilities are different in the Non-uniform Heterogeneous but equal in the Uniform Heterogeneous systems [14].

Asymmetric and Symmetric Multi-processing The process of defining which core the task executing is called mapping of a task which requires the study of overhead and communication delay for the efficient usage of each core. There are two types, Symmetric Multiprocessing (SMP) and Asymmetric Multiprocessing (AMP). In SMP, all tasks across all cores share the same view of the operating system. In AMP, each core operates with its own independent operating system [35].

Multi-core Processor Scheduling Algorithms Multi-core and multi-processor systems solve the problem of assigning tasks to cores. The allowed migration of a task or job from one core/processor to another, determines the scheduler class, either *No-migration* or *Migration*. The No-migration class, or partitioned, prohibits tasks or jobs (instances of a task) from migrating to complete execution on a core or processor different from the one to which they are initially allocated. The Migration class, or global, allows such a feature where schedulers can allow tasks and jobs to continue execution on different processor.

Architectural Configurations	Capabilities	Speed
Homogeneous	=	=
Uniform Heterogeneous	=	!=
Non-uniform Heterogeneous	!=	!=

Table 2.1: Different Architectural Configurations

2.2 Mixed-criticality Systems

Mixed-criticality system runs services with different criticalities in a common computing platform and provides a bounded degradation for low-criticality services and services in case of resource shortage. The following tuple characterises the real-time mixed-criticality task model:

$$\tau = \langle id, p, d, c_{opt}, c_{pes}, l, wr_{mode} \rangle$$

- $\tau.id$ task's identifier.
- $\tau.p$ task's period.
- $\tau.d$ task's relative deadline.
- $\tau.c_{opt}$ task's optimistic WCET.
- $\tau.c_{pes}$ task's pessimistic WCET.
- $\tau.ut_{opt}$ task's load or utilization of CPU time using the optimistic WCET and calculated by $\tau.ut_{opt} = \frac{\tau.c_{opt}}{\tau.p}$. It represents the workload of the task during normal or LO criticality mode. $\tau.ut_{pes}$ task's load or utilization of CPU time using the pessimistic WCET and calculated by $\tau.ut_{pes} = \frac{\tau.c_{pes}}{\tau.p}$. It represents the workload of the task during the HI criticality mode.
- $\tau.wr_{mode}$ task's worst-case response time according to the activated mode, where $\tau.wr_{LO}$ denotes response time during the LO criticality mode and $\tau.wr_{HI}$ denotes the HI criticality mode response, and $\tau.wr_{CH}$ caps the interference by the LO criticality on HI criticality tasks before the activation of HI criticality mode.

Adaptive Mixed-Criticality (AMC) The AMC response time analysis for Fixed-priority mixed-criticality tasks, is the exact text for testing the schedulability of the MC task set. AMC defines the analysis for each mode in the system, LO and HI criticality mode, and the transition from LO to HI mode.

LO-criticality mode In this mode, Low Criticality (LO)criticality tasks $\tau_i.l = LO$ and High Criticality (HI)criticality tasks $\tau_i.l = HI$, are assumed to execute with in their optimistic WCET. The task is considered schedulable if its response time during Low Criticality (LO)criticality mod $\tau_i.wr_{LO}$, is less than or equal to its deadline. During this mode, tasks are prioritised according to their priority.

$$\tau_i.wr_{LO} = \tau_i.c_{opt} + \sum_{\tau_j.pr \in HP(\tau_i.pr)} \left\lceil \frac{\tau_i.wr_{LO}}{\tau_j.p} \right\rceil \cdot \tau_j.c_{opt} \tag{2.4}$$

Transtion from Low Criticality (LO)to High Criticality (HI)criticality mode

$$\begin{aligned}\tau.wr_{CH} = \tau_i.c_{pes} \\ + \sum_{\tau_j.pr \in HP(\tau_i.pr)} \left\lceil \frac{\tau_i.wr_{CH}}{\tau_j.p} \right\rceil \cdot \tau_j.c_{opt} \\ + \sum_{\tau_j.pr \in HPL(\tau_i.pr)} \left\lceil \frac{\tau_i.wr_{LO}}{\tau_j.p} \right\rceil \cdot \tau_j.c_{opt}\end{aligned} \tag{2.5}$$

HI-criticality mode MC_{HI} This test is for the schedulability for the set of HI-criticality tasks $HPH(i)$ with priority higher than critical task $\tau_i(hi)$. However, under any circumstances, this test may be used only in the worst critical situations where the system starts to drop critical tasks according to the available capacity.

$$\tau_i.wr_{HI} = \tau_i.c_{pes} + \sum_{\tau_j.pr \in HPH(\tau_i.pr)} \left\lceil \frac{\tau_i.wr_{HI}}{\tau_j.p} \right\rceil \cdot \tau_j.c_{HI} \tag{2.6}$$

A typical MC system is defined with a number of runtime modes that differentiate between normal mode and critical modes which the MC system may experience. The normal mode, or Low criticality mode (LO), represents the specified behaviour when no resource shortage, and the critical mode, High criticality mode (HI) is activated when the system experiences a resource shortage.

The MC system verifies the schedulability of the system services in normal and critical modes before the system deployment. In Low Criticality (LO)criticality mode, all system services and tasks despite their criticality levels are scheduled according to the underlying real-time scheduler and the priority assignment algorithm. If the system experiences core failure or WCET overrun,

Task ID	Period	Deadline	Criticality	Optimistic WCET	Pessimistic WCET
τ_1	2	2	LO	1	2
τ_2	4	4	HI	1	1

Table 2.2: Mixed-criticality Task Set

the system activates the High Criticality (HI)criticality mode, and provisions the prioritization of system services based on their priority and criticality, and High Criticality (HI)criticality services are assured to continue execution while Low Criticality (LO)criticality services are subject to abort or abandon for their execution during the High Criticality (HI)criticality mode. The reason for dropping the Low Criticality (LO)criticality services is to eliminate the interference caused by low criticality with higher priority on services with High Criticality (HI)criticality but with lower priority, which is a phenomenon known by *criticality inversion* [4, 36]. The criticality inversion problem was first presented by Vestal [4], but De Niz et al. [36]. [36] who coined this phenomenon with name criticality inversion. The presented problem by Vestal is that assume a task set of two tasks τ_1 and τ_2, defined by the timing and criticality configuration in Table 2.2.

If we use Deadline-monotonic scheduling - which is optimal for all task sets with deadlines less than or equal to their periods and utilisation bound less than 1.0 - task τ_1 is assigned higher priority and both tasks are schedulable during the Low Criticality (LO)criticality mode using their optimistic WCET $\tau_i.c_{LO}$ estimate. In the case of τ_1 overrun its optimistic WCET $\tau_i.c_{LO}$, τ_2 has lower priority but higher criticality, hence, τ_2 can't be scheduled until k τ_1 completes execution. However, the system is feasible if τ_2 assigned higher priority. These findings are found by Vestal and proposed dropping τ_1 to eliminate its interference on the High Criticality (HI)criticality task τ_2. Vestal suggested two priority assignment algorithms for assigning higher priority to the higher criticality task τ_2, to solve the problem. The first is *period transformation* [37] which is a technique originally developed for graceful degradation for critical tasks in partitioned criticality systems - where tasks with equal criticality execute on the same platform, and the second is Audsley optimal priority assignment algorithm (OPA) [38, 39].

Period Transformation The period transformation has been developed in partitioned criticality systems to guarantee the schedulability of critical tasks in case of a resource shortage in the form of processor overload [37]. The key concept is, to increase a task priority, is to transform the task into a group of tasks with smaller periods, and WCETis allocated equally between the tasks in the group. This is achieved by dividing the task period and WCET by a value that is sufficient to reduce the period to a value Δ at or below the period of every other task on the system.

$$\tau.p_{transformed} = \frac{\tau.p}{\Delta} \tag{2.7}$$

$$\tau.c_{transformed} = \frac{\tau.c}{\Delta} \tag{2.8}$$

Audsley optimal priority assignment algorithm (OPA) The ordering is partitioned for two queues: the first is ordered, consisting of the $\tau_i(LO)$ priority tasks, and the remaining unordered $\tau_i(HI)$ priority tasks. All tasks in the unordered partition are chosen and placed at the top of the ordered partition and verified for feasibility. If the chosen task can be scheduled, then the priority of the task will remain, and the ordered partition will increase by one position. If the task is not schedulable it is returned to its original priority. This continues until either all tasks in the unordered partition have been checked and found to be unschedulable, or else the ordered partition constitutes the final priority assignment [38, 39]. The worst-case response time for a task by equation 1 is extended by Vestal to [4] consider the criticality of the task, by knowing which subset of tasks has higher priority than $\tau(HI)$ but without knowing their specific priority assignments.

$$\tau_i.wr = \tau_i.c_{opt} + \sum_{\tau_j.pr \in HP(\tau_i.pr)} \left\lceil \frac{\tau_i.wr}{\tau_j.p} \right\rceil \cdot \tau_j.c_{opt} \tag{2.9}$$

Fault Tolerant Systems The number of spare computers or redundant peripheral devices to allocate is limited by the non-functional requirements of Size/Shape, Weight, and Power (SWaP) that constrain the system design. Two complementary approaches to obtain reliable computing. Fault-in-tolerance, to prevent faults from happening and Fault-tolerance to prevent faults becomes failures (protective and adaptive fault-tolerance) online [19]. Preventive fault-tolerance applied to ensure total quality assurance to deliver system services. Adaptive techniques are applied to smooth degrade service quality, but ensure the continuity of vital services desired by the system user. This degradation is realized by abandoning several non-critical services to ensure the continuity of critical ones. The safety and quality of delivered service justifies its reliability [21, 40].

Multiple task errors due to multiple faults, affect this task job's primary and backups and that job becomes erroneous. Multiple task errors can affect different jobs of the erroneous task or other jobs in different tasks may be affected as well. Redundancy techniques for fault-tolerance scheduling are either *temporal for re-execution* or *spatial for replication* of tasks among the available processors. The choice of redundancy technique should be determined by the scheduling algorithm [35].

Primary and Replica Tasks For static or dynamic scheduling, the basic idea is to use the available slack reserved during the schedulability test. The generated schedule contains the original task usually called *primary*, and the re-execution instance of the same task named *replica* or *backup* [30] task. The active-replicas are released in parallel on different processors if faults occur or not, where passive-replicas are only invoked in case its primary task has failed in its execution, note that activated passive-replicas must be de-allocated after the faults tolerated [30].

2.3 Chapter Summary

This chapter has explained the background on real-time systems and scheduling in Section 2.1, and the basics of mixed-criticality systems and scheduling in Section 2.2.

Chapter 3

Related Work

This chapter reviews the related work, it presents significant contributions in real-time systems research considered in this thesis and WCET analysis, dependability, and mixed-criticality systems scheduling. Section 3.1 reviews Real-time scheduling. Section 3.2 reviews WCET. Section 3.3 reviews Dependability. Section 3.4 reviews Mixed Criticality Systems. Section 3.5 concludes the chapter.

The most robust and compelling findings in mixed-criticality systems research since the start of the mixed-criticality research field by Vestal's seminal paper [4] in 2007 are as follows: Baruah et al. [6] proved that none of the existing priority assignments is optimal for mixed-criticality systems scheduling. The work by Burns in [20] introduced the importance of runtime survivability based on AMC, and an explicit invitation in [7] for mixed-criticality researchers to focus more on developing frameworks that achieve runtime survivability.

The establishment of the mixed-criticality field of research by Vestal highlighted that existing Fixed-priority real-time priority assignment algorithms are not optimal when criticality and safety integrity levels of services are introduced to the scheduling problem, nor are Dynamic-priority assignment algorithms, as shown in [6].

The development of the Adaptive Mixed-Criticality (AMC) by Baruah et al. [41] represents the foundation for priority assignment, response time analysis, and scheduling for mixed-criticality services in most of the existing research in [42, 20, 43, 44, 45, 46, 47, 24].

The introduction of survivability by Burns et al. [20] raised the need for a focus on runtime survivability and smoother degradation than AMC. The model provides runtime assurance against tolerable WCET overruns using robust and resilience metrics, which quantify the level of system degradation. An interesting idea is relaxing AMC's pessimistic runtime behaviour and integrating tolerability against several tolerable WCET overruns. The model is built on top of AMC and defines proportionate responses to WCET overrun.

Few sources have followed one of the most important sources, Baruah et al. [7], in my opinion, in mixed-criticality systems and scheduling research after Vestal's seminal paper [4]. The paper raised attention to the need for frameworks that put more consideration into runtime survivability, using existing models and protocols, most of which have been developed with a focus on providing pre-runtime verification with no consideration for runtime survivability. Less attention has been given to this shift, as observed in [48, 49, 50, 51], announced by Baruah et al [7], and this thesis provides a concrete response to this shift.

In the following sections, we review optimal fixed-priority assignment algorithms, WCET, and dependability before reviewing research on mixed-criticality systems. This is important as these three areas provide the foundations for the research on mixed-criticality systems from the perspective of this thesis.

Most of the existing approaches in mixed-criticality systems, are based on the Vestal model [4]. Vestal's model was later improved By Baruah et al in [8] by adding runtime monitoring and the analysis for schedulability of system services in low, high criticality modes, including the interference during the change from LO mode to HI mode. However, though the Vestal and Baruah models are safe, they lack the required *runtime-survivability* during the system mission. Burns augmented Baruah's model in et al [20], by adding the notion for *runtime survivability* followed by another publication by Baruah et al [7] where triggered the need for considering both: the pre-runtime verification support, and the runtime survivability facilities that handle failures and resource shortages during the system mission.

3.1 Real-time Scheduling

One significant finding in Vestal's seminal paper is that known real-time priority assignment algorithms, considered to be optimal, such as RM, DM [5] [25], and EDF algorithms [5], are not optimal when criticality is introduced [4, 6, 24]. Vestal highlighted the problem of existing priority-assignment algorithms using a use-case example for a mixed-criticality task set with multiple criticality levels. The problem presented in Vestal's paper, referred to as *criticality-inversion* by Di Niz et al. in [36], is discussed in the background chapter. However, Vestal found that Audsley's optimal priority assignment [38, 39], known as OPA, is optimal. In this section, we review the optimal priority assignment algorithms such as RM, EDF, and other real-time model alternatives to the Liu and Layland model.

Real-time systems prioritise the execution of system tasks using priority assignment algorithms. Priority assignment algorithm manages the allocation of processor time among tasks while ensuring that all tasks complete their computational requirements before their respective deadlines [29].

Liu and Layland (1973) [5] proposed two priority assignment algorithms and techniques for deriving schedulability tests [5]. The two priority assignment algorithms are RM and EDF [5]. RM ensures that tasks with deadlines equal to their periods meet their deadlines if they pass the LUB schedulability test. The safe LUB for the utilization of processor time for any task set scheduled under RM is 0.71. This means that if the total utilization of all tasks in a task set is less than 0.71, then the task set can be guaranteed to meet all deadlines under RM scheduling [5]. The EDF algorithm assigns priorities dynamically at runtime, where the task with the shortest deadline has the highest priority for execution. EDF guarantees that all tasks in a task set meet their deadlines if they pass the LUB schedulability test. The safe LUB of the processor utilization for tasks scheduled under EDF is 1.0. This means that if the total utilization of all tasks in a task set is less than or equal to 1.0, then the task set can be guaranteed to meet all deadlines under EDF scheduling [5, 52].

Though Liu and Layland's utilisation bounds are safe, their assumptions were too pessimistic and lacked facilities for adaptability to overloads e.g. when task set utilisation is greater than 1.0 due to resource shortage e.g. core failure. The elastic model introduced in Elastic Task model [15], enables tasks to adjust their rate of arrivals within predefined bounds whenever the system experiences an overload. The key idea is to treat task utilisation as if it were a spring. For example, if a task set's utilization exceeds 1.0 and the underlying scheduler is EDF, the compression equations for the springs are modified to bring the task utilization within the EDF bounds by relaxing the tasks' periodic rates of arrivals. Conversely, if the processor is underutilised, a decompression equation can be used to increase the task's rate of arrivals. Though the Elastic model enables different profiles of execution rate flexibility to keep the system within a certain load, it lacks the aspect of value or utility that associates each task with the overall system value. Considering importance-based or utility-based adaptability is crucial to avoid adapting tasks with less importance than more important ones.

The TU model is another perspective for real-time scheduling [16, 29]. The Time-driven model is an integration for the fixed-priority model and best-effort scheduling [53]. This model is based on the fact that the completion of each task within a set of tasks, has a value to the overall system that can be expressed as a function that varies with time. However, the model considers traditional real-time models, based on the Liu and Layland model, that are static and have no consideration for the dynamic nature of real-time systems and their environment. Also, periodic rates of task arrivals may change due to application dependencies, resulting in a constantly dynamic changing of task set configuration [16], limiting task arrivals by predefined bounds, as in the Elastic model, may not be adequate for practical cases. The model redefines the traditional concept of a deadline using the *critical-time* concept. critical time is equivalent to the deadline in traditional real-time models but assigns two completion values for each task: a high value before the critical time/deadline, and a low value after the critical time/deadline, to prove the point that task completion after the deadline can still contribute positively to the overall system utility.

3.2 WCET Analysis

Another finding by Vestal is that the greater the criticality of a service, the higher and more the execution time estimate tends to be. Therefore, the schedulability of services is verified against other services at the criticality level of the service in question [4]. The impact of the WCET estimate has received less attention from the mixed-criticality research. In this section, we review existing sources considering WCET measurement in real-time and mixed-criticality systems, and clarify the need for redundant WCET estimates for each task in the system including the reasons for such needs. Also, an assumption in these sources is that every pessimistic WCET estimate has to be larger than the optimistic WCET estimate, however, we show that this has little consensus with findings by Altmeyer in [13]. The implicit goal is to show that none of the following sources considered exploiting the WCET overrun, online or empirically, during the execution of the system and optimising the system configuration accordingly, as this is the novel approach followed in this thesis for handling WCET overrun.

Puschner et al. authored one of the very first research publications on how to calculate an upper bound for WCET estimates [10]. The study provides logic constructs that enable programmers to express and differentiate the behaviour of different implementations of an algorithm, using preconditions and rules to be met before the WCET measurement process. The control-flow information includes details about loops, conditional branches, function calls, and other program structures influencing the flow of execution, and it is not possible to extract all flow information directly from the source code [54]. Control flow is modelled using techniques such as integer linear programming and branch trace information to map assembly instructions to their origin in the source code [55]. The control flow graph is necessary for WCET analysis, and the complexity of the control flow can impact the precision of the analysis [56]. Kirner et al. have developed techniques that enable the mapping and transformation of this control-flow information from the source code to the corresponding machine code [57]. These techniques facilitate the analysis and optimisation of a program's behaviour at the machine code level. The work in [1] and [12, 11], provides comprehensive reviews of different methods, tools, and techniques used in the literature and industry to measure the WCET. Though measuring WCET in RT systems has been researched extensively, the problem is still open [14].

Most of the research based on Vestal's model [24] [58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68] which supplies more than a single WCET estimate for each task, researched WCET influence on the schedulability analysis of tasks. Typically, two or more WCET upper bounds are supplied to each task and the schedulability analysis is performed using the lowest estimate when verifying the schedulability of all tasks during normal behaviour of the system, whereas the largest WCET estimate is considered when verifying the system at critical behaviour, for example, resource shortages.

[13] investigated this source of the above findings by reviewing the static, dynamic, and hybrid methods for estimating the WCET of a task. Static methods may provide a higher level of confidence than dynamic methods but do not necessarily estimate larger WCET than the later estimations by dynamic methods or measurement-based methods [69, 70]. Static analysis methods are complex and limited to the micro-architectural model under investigation, therefore, therefore, the results can't be generalised to other micro-architectural models.

Dynamic or measurement-based methods are straight forward method and don't involve the same level of complexity (the need for the micro-architectural model) in obtaining WCET estimates. The problem with dynamic methods is the difficulty in determining a complete set of possible inputs and a set of initial processor states [13]. Hybrid analysis uses both methods, static and dynamic, in a single method for measuring WCET estimates but produces larger pessimistic WCET. Hybrid analysis measures program code using dynamic methods, then, considers the measurements during the static analysis [71, 62, 72].

[73] proposed a method for estimating WCET using existing approaches with considerations for the interference from other tasks in the system. The method

predicts the possible interference delay, which is the maximum overlap between tasks competing for resources, and computes the maximum interference time, then adds it to the task's WCET in question. The used method is based on Timing Composability (TC) [14], which estimates WCET by taking the sum of the execution time in memory-related computation and processing-related computations, to form the final WCET.

In a working group during the 2015 Dagstuhl seminar, a group led by Maiza et al [9] discussed questions regards WCET estimates, in the context of mixed-criticality systems. The discussion included a question about why mixed-criticality tasks have more than WCET. Also, once a task has more than a single WCET, then, this WCET estimate is not the *worst-case* execution time! The most interesting question is should mixed-criticality researchers integrate WCET analysis and schedulability analysis in a single form of verification method. The first question was answered but the latter two questions remained unanswered. The answer to the first question is that the many WCET values are due to the variety of WCET measuring methods. Therefore, system designers tend to build cumulative confidence by supplying tasks with more than a single WCET estimation, each to be subject to verification and use in a specific system mode. The E-ATMP model developed in this thesis monitors the WCET online during system execution and reconfigure the system accordingly. E-ATMP answers the two later questions. E-ATMP framework updates the WCET estimate of a task as it progresses during the system mission by the EWCET model developed in this thesis, but uses the WCET estimate generated by static or dynamic analysis as a starting point for the system pre-runtime verification analysis, and after each update of a HI criticality task WCET, a runtime verification and schedulability analysis is performed to ensure the schedulability of the whole system configuration.

A method that updates the scheduler decisions based on the execution path taken by high-criticality tasks was proposed in Cagnizi et al. [74]. The method predicts the finishing time and the resulting slack out of the taken path, a interrupt tool is introduced to allow low-criticality tasks to execute and to promote the low-criticality tasks to interrupt the running high-criticality task. However, no policy for overrun is stated in the method this may render the system unsafe if promoted low-criticality task to become high criticality overrun its WCET by taking long execution. A gap highlighted by Jiang et al. between different models from the literature and industrial standards [75]. The gap is that it is difficult to integrate theoretical models into the industry, primarily because these models are developed without taking into account considerations for industrial safety standards. They extended the AMC scheme [8] into an industrial architecture named (Z-MC), which accounts for industrial considerations for safety standards. Z-MC provides runtime safety analysis with sufficient isolation between critical and non-critical services using a flexible model to integrate with different safety standards.

The online reconfiguration and redundancy mechanisms provided by Kadeed et al. [76] use the concept of *foreseen* core failures to ensure sufficient isolation specified by safety standards between low-criticality and high-criticality services during the online-reconfiguration [76]. The proposed mechanism facilitate continuous monitoring for service failures , and determines the possible core failure if service degradation exceeds a defined level of criticality. Therefore, services replication to different cores is chosen instead than migrating them because the detection of the failure is conducted before the core failure. However, service replication depends on pre-defined solution plans before the system mission for the reconfiguration process. Santy et al. introduced the concept of *allowance* [77]. It is an offline model that measures the maximum time that can be added to the optimistic WCET of high-criticality tasks without breaking the schedulability. EWCET is an online model that updates maximum LO and HI task's WCET empirically and reconfigures the system accordingly without breaking the schedulability.

Draskovic et al. the authors proposed a model that considers task WCET as random variables; and introduced metrics that adhere to safety standards [78] for systems subject to certification. Dong et al. presented a slack-based monitoring approach, which the its a mechanism that monitors tasks completed with less than their estimated WCET budget [79], so the resulting slack can be utilised by both high and low-criticality tasks. Allocating a slack budget to a high-criticality task reduces the frequency of switching between low and high-criticality modes while allocating a slack budget to a low-criticality task minimises the number of times the system enters the high criticality mode.

3.3 Dependable Computing

The fundamental principle of MC systems involves a shift from a partitioned architecture, where each processor handles tasks with specific criticality levels, to an integrated mixed-criticality architecture. Therefore, there is a need for Fault Tolerance mechanisms to ensure the system's reliability and maintain its dependability, especially in situations where resource shortages could impact the safety integrity levels of the system [19]. One of the fault tolerance techniques [23], known as redundancy, can be employed to enhance overall system dependability. Redundancy, in the context of real-time fault-tolerance systems band scheduling, can be categorised into temporal and spatial-redundancy techniques [80, 81]. In the following, we review the concept of dependability, with a focus on redundancy and what redundancy techniques are used in the literature for achieving mixed-criticality systems dependability. The approach in this thesis for achieving dependability is composing highly dependable components from several redundant, less dependable components. Different approaches are followed in the literature, and here we review the significant studies on dependable mixed-criticality systems using redundancy. The implicit goal in this section is to show that none of the following sources considered the form of redundancy followed for dependability in this thesis, named Criticality Arithmetic, where redundant components/services have equal or lower criticality than the criticality of the component/services they implement.

Based on temporal redundancy, the work by Pathan in [61, 82, 42] applied task re-execution to tolerate WCET overruns. The study also tolerates a bounded number of WCET overruns and considers task replication, where each replica has different implementations and WCET estimates, but all replicas have the same level of priority and criticality as their primaries. Re-execution from recent checkpoints has been studied by [82, 44] where failed services jobs can execute from the last point the system considered to be safe.

Four criticality modes model was introduced by Albayati et al. [42, 42, 64]. The model derived schedulability based on AMC analysis that considers the number of re-execution times per service according to the service probability of failure. An approach that employs re-execution and checkpoints was followed by Von in [63], and [44], [63] provided the analysis to predict the duration of high criticality mode before the activation, where [44] included rules for accessing critical memory sections [15]. The work in [83] applied task re-execution and one of the alternative solutions proposed early by Vestal [4], period transformation [37], where high criticality jobs are decomposed to smaller jobs with shortest deadline. [84] employed EDF-VD under the the assumption that the inter-arrival time between two consecutive transient faults is at least equal to the hyper-period of the task set, to determine the wasted time caused by the transient faults. The mechanism for detecting detects transient faults, using bit flips, during job execution and restarts the job to re-execute the same computation proposed in [68]. The maximum number of allowed re-executions depends on the failure requirements for specific tasks, which are usually mapped to the criticality level and certification requirements.

Hardware redundancy techniques proposed in [85, 86, 87, 88]. Standby-sparing (SS) for energy saving in mixed-criticality systems proposed in [85], which is a technique that employs a dual-processor platform consisting of a primary processor and a spare processor, and the spare remains dormant until the primary processor fails, and each task in the system has a main copy on the primary processor and a backup copy on the spare processor. [86] considered integration of SS fault tolerance with energy management. Using DVFS, the study in [87] proposed varying utilisation levels of both processors, the primary and the standby, with different processing speeds, and reduces tasks replicas tasks as late as possible [88] defines two physical processor cores into a single logical core that executes the same service simultaneously, to improve the hardware-based adaptive redundancy in multi-core processors. The work in [89, 90, 67] assumed a Time-Triggered Architecture (TTA), where periods between tasks arrivals in constant, and provided a model to tolerate at most one fault over the hyper-period, although the specific fault-tolerant techniques employed are not explicitly mentioned. The work in [90] generates a schedule tree at design-time and selects the schedulable node at runtime to re/execute high criticality tasks, [67] uses a tree-based approach which selects nodes dependency between tasks and selects the node that satisfies certification requirements.

Under the spatial redundancy category, one of the first studies on dependable mixed-criticality scheduling that employs task migration was proposed by Saraswat et al. [58]. In their approach, high-criticality services have higher priority than low-criticality services for the allocation of recovered services to processing cores. An analysis for calculating the time a high-criticality task with low priority can wait for a low-criticality task with high priority to complete execution was defined in [60]. This analysis detects the WCET overrun by low-criticality tasks when the waiting high-criticality task slack becomes equal to zero. Approaches that considered optimisation for high critical replicas were proposed in [59, 91, 2], to determine low criticality tasks timing parameters, and high criticality tasks replicas executes on the same processor, but the work in [2] differ by that it optimises high and low criticality tasks using Integer linear programming for the task allocation, where [59, 91] heuristics. [92] applied N-Modular Redundancy (NMR), each high criticality task has more than two replicas execute on different cores, and at the event of a fault, a majority voting mechanism is used to determine the correct result. [93] discusses various fault tolerance techniques such as replication, and re-execution. However, it does not specify a concrete fault-tolerance model. [65] presented offline partitioned scheduling to provide timing guarantees per partition without the need for online adaptation.

Criticality Arithmetic (CA) augments the fault tolerance technique of redundancy by combining redundant components with lower criticalities to compose a component with higher criticality. A theoretical model for CA was established by Menon et al. [22]. A practical implementation of the CA model established by Menon, which includes ATMP-CA and LBP-CA, was developed during the research for this thesis and published in a journal [94] and conference [95] papers. This integration enhances ATMP with fault-tolerance techniques such as redundancy and criticality inheritance provided by CA. CA redundancy differs from traditional Fault-tolerance (FT) redundancy in that in FT the redundant replicas and the services they provide have equal criticality. However, in CA, the redundant replicas of tasks that implement a service have a lower criticality than the service they provide. This distinction opens up new perspectives for addressing resource shortages in mixed-criticality systems.

3.4 Mixed Criticality Scheduling

Tolerance-based Real-Time Computing Model (TRTCM) [18] is based on the time/utility Real-time model [17]. TRTCM allows the system designer to define a performance parameter (rate, throughput, jitter, etc.) to optimize in case of a resource shortage using Linear Programming (LP). TRTCM translates each task into several possible configurations, each with a certain utility to the overall system utility, and then selects the configurations with maximum utility overall services. ATMP-CA uses the TRTCM utility optimisation function to enhance the system after a core failure. It modifies the LP problem formulated by TRTCM, exploiting the information about criticality arithmetic. A task τ_i contains the period $\tau_i.p$, the utility function $\tau_i.f_{util}$, and the relative utility $\tau_i.u$. The basic principle of these utility functions has been described in [23] and their use to reconfigure systems in [2, 96]. These utility functions are inspired by a more generic variant of utility functions of Jensen et al.[16].

The utility function defines the relation between the task's period $\tau_i.p$ and the relative utility $\tau_i.u$. While the utility function could have an arbitrary shape, in most cases it is sufficient to describe f_{util} by two line equations, as shown in 3.1.

As shown in the figure, the shape of the utility function is defined by the two points $\langle p_{prim}, 1.0\rangle$ and $\langle p_{tol}, u_{tol}\rangle$. Thus, we specify the utility function by

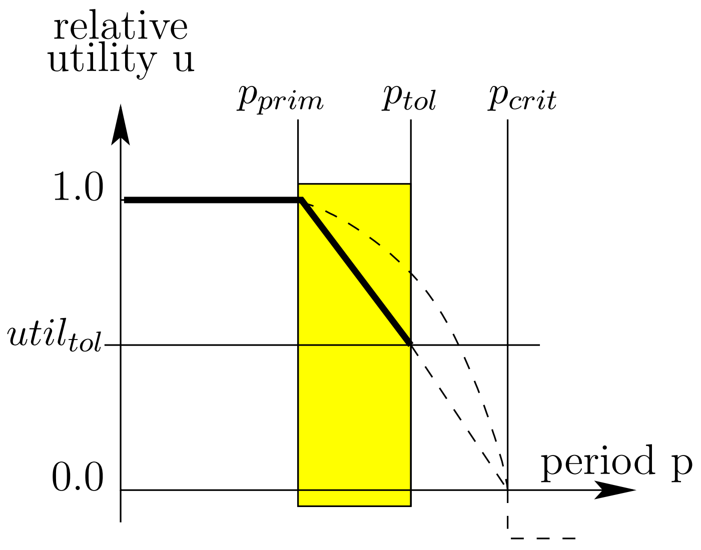


Figure 3.1: TRTCM System-utility function [2]

these three parameters: primary period p_{prim}, tolerance period p_{tol}, and tolerance utility u_{tol}. The primary period p_{prim} represents the optimal arrival rate and any rate higher than that does not increase the relative utility anymore. This is because, as shown in Figure 3.1, for all periods $\tau_i.p \leq p_{prim}$, the relative utility, $\tau_i.u$ is equal to one. The tolerance period p_{tol}, defines the highest period that is considered to be acceptable for the service implemented by this task. The tolerance utility u_{tol} is the relative utility according to the tolerance period.

When optimising the overall system utility, we do not use the relative utilities $\tau_i.u$ for all the tasks $\tau_i \in \Gamma$, but rather use the absolute utility $\tau_i.U$, which is the product of the relative utility $\tau_i.u$ and the criticality value $\tau_i.l$ of task τ_i:

$$\tau_i.U = \tau_i.u \cdot \tau_i.l \tag{3.1}$$

Using the absolute utility $\tau_i.U$ of each task $\tau_i \in \Gamma$ is necessary for the overall system optimisation, as this allows to make sure that tasks of higher criticality get in general less degraded than tasks of lower criticality.

The Adaptive Mixed-Criticality (AMC) is a uniprocessor fixed-priority scheduling scheme that offers a priority assignment algorithm based on Audsley's optimal priority algorithm [39] and a mixed-criticality aware schedulability test based on Response Time Analysis (RTA) [97] [32]. AMC enters the high criticality mode, whenever a HI or LO service consumes its optimistic-WCET budget without signaling completion. AMC abandons all low criticality services released during the high criticality mode. However, it allows the completion of low criticality services released before the trigger of high criticality mode. ATMP-CA reconfigure in case of a core failure. ATMP-CA optimises the system's throughput per core and uses the AMC scheduling test on each core to test the schedulability of the optimised reconfiguration according to the remaining number of cores.

Another MC Protocol is AMC, which [43] integrates the AMC [8] protocol with the TRTCM model [2]. The integration of ATMP is realized through four key functionalities: First, it maximizes system utility by optimizing each service's utility using the *utility function* provided by TRTCM, defining a solution space from the available safety margins within each service. Second, an online search algorithm is employed to narrow down the TRTCM solution space. Third, ATMP tests the schedulability of each solution and updates the search algorithm accordingly. Fourth, ATMP assigns priorities to services based on their safety margins to drop the ones with less criticality and adaptability. During the development of this thesis, I integrated ATMP and CA to create a CA-aware version of ATMP, named ATMP-CA [94]. As mentioned in the previous subsection 3.3, this integration enhances ATMP with fault-tolerance techniques such as redundancy and criticality inheritance provided by CA.

The MC Bailout-based BP scheduling [46, 47, 98] differs from traditional MC models by that it defines three criticality modes to schedule the execution of tasks: Normal, Bailout, and Recovery (Mixed-criticality models uses two criticality modes: HI and LOmodes). Normal mode represents the desired system behaviour or normal course of execution for system services. It corresponds to the typical low-criticality mode in mixed-criticality scheduling. Bailout mode represents the typical critical mode that mixed-criticality scheduling activates to ensure the schedulability of high-criticality tasks whenever a transient fault occurs, such as a task overrun. Recovery mode ensures that all high-criticality tasks are completed before returning to Normal mode.

The BP was originally developed in [46] to enable the application of Mixed-criticality models in the avionics and automotive industry. BP minimizes the negative impact on low criticality tasks during resource shortages (WCET overrun) by ensuring a quick return to normal operation after completing the recovery process. LBP-CA offers a quicker return than BP. The quicker return is achieved by exploiting the information about criticality arithmetic to reduce the BP recovery process time. The advantage of LBP-CA is the earlier return to normal mode compared to BP, allowing for the release of low criticality services that would otherwise be dropped during Bailout and Recovery modes under BP.

LBP [98] extends BP by, instead of abandoning the release of low services during critical modes such as Bailout-mode and Recovery-mode, inserting the low services into a low priority queue for future execution when the system returns to the Normal mode. In other words, BP allows the low services to execute during the critical modes if they were released before entering the critical modes, while LBP-CA inserts the ones released after the activation of the critical mode into a low priority for future execution. LBP-CA reduces the waiting time for low-criticality services. The key idea is that the relationship between waiting time and critical time is that the less time the system stays in critical mode before returning to normal mode, the more time low jobs have available to execute upon returning to normal mode.

[99] integrated the elastic model to the uni-core Earliest Release Earliest Deadline First algorithm (ER-EDF), and extended it to multi-core systems in [100] and provided a schedulability test based on Guaranteed Bound Analysis in [101]. [102] added as a survivable component to the elastic model in federated architecture to stretch LO-tasks as needed to support the underlying MC elastic facilities. [103, 104] exploits slack time generated by ER-EDF and introduces a mode change mechanism to ER-EDF. Orr et al. [105, 106, 107] Integrate elastic model to mixed-criticality but incorporate the WCET parameter to the elasticity of the task. The model allows the platform to determine tasks' elastic coefficients according to the context of the system execution. The work later was remodeled under the Liu and Layland model [5] for the analysis of real-time tasks, and the concept of criticality was neglected. [108, 109] and integrated it to the Fluid scheduling model [110].

3.5 Chapter Summary

This chapter reviewed the related work to this thesis, it presents significant contributions in real-time systems in Section 3.1, WCET analysis in Section 3.2, dependability in Section 3.3, and mixed-criticality systems scheduling in Section 3.4. The real-time systems review concludes that the research community realised the need to extend scheduling theory to include the concept of scheduling tasks with different criticality levels on the same processor. WCET review concludes that extending real-time scheduling theory by employing redundant WCET estimations can provide an enhancement for Real-time scheduling by changing the WCET according to the activated criticality mode of the mixed-criticality system. Dependability review concludes that expanding redun-

Table 3.1: Summary of related work strength and limitations

Ref.	Year	Sys Model	Task Model	Fault Model	Tasks Crit. Levels	System Crit. Levels	Schedule	Algorithm	Strengths and Limitations
[58]	2009	Multi-core	Periodic	Permanent	Dual	Dual	Dynamic	CBS	Checkpoint and Rollback Recovery, My work on E-ATMP neglected the use of these techniques.
[59]	2009	Multi-core	Periodic	Any	Multi	No	Fixed	ILP	ATMP-CA uses replication on different nodes and ILP optimisation but ATMP-CA replicas have lower criticlaity than their primaries.
[60]	2013	Multi-core	Periodic	Permanent	Dual	Dual	Dynamic	ZSRM	This work minimises number of needed replicas, ATMP-CA and LBP-CA decompose services based on ATMP-CA arithmetic for each criticality level.
[111]	2014	Single-core	Sporadic	Transient	Dual	Dual	Fixed	FTMC	This work uses re-execution and bounded number of faults. E-ATMP uses postponed executions and execution time updated.
[112]	2014	Single-core	Sporadic	Transient	Dual	Multi	EDF-VD	Any	This work includes energy-efficient strategies with Dynamic scheduling.
[91]	2014	Multi-core	Periodic	Any	Dual	Dual	Fixed	Any	Uses reallocation and reexecution for fixed priority schedulers.
[113]	2015	Multi-core	Periodic	Transient	Not specified	Not specified	Static	Heuristic	This is a heuristic with replication and Checkpoint.
[91]	2015	Single-core	Sporadic	Any	Dual	Dual	Dynamic	Schedulability test	Uses re-execution and focuses on dynamic scheduling with schedulability tests.
[114]	2016	Single-core	Sporadic	Transient	Dual	Four	Fixed	AMC	They used four criticlaity modes with reallocation and ILP for reallocation in the presence of transient faults.

Table 3.2: Summary of related work strength and limitations

Ref.	Year	Sys Model	Task Model	Fault Model	Tasks Crit. Levels	System Crit. Levels	Schedule	Algorithm	Strengths and Limitations
[42]	2016	Multi-core	Sporadic	Permenant	Dual	Dual	Fixed	Modified AMC	The authors used re-execution and provided and new schedulanility test based on AMC. Thw work in ATMP-CA, LBP-CA, E-ATMP uses AMC with no modification.
[115]	2016	Multi-core	Sporadic	Transient	Dual	Dual	Dynamic	EDF-VD	The authors used Re-execution, re-allocation, re-configuration techniques.
[63]	2016	Single-core	Sporadic	Transient	Dual	Multi	Fixed	Schedulability test	They used re-execution for failed trasks with more than two criticlaity levels.
[83]	2017	Single-core	Periodic	Transient	Dual	Dual	Fixed	EDF-VD	Dynamic scheduling that uses re-execution for failed tasks.
[20]	2018	Single-core	Periodic	Transient	Dual	Dual	Fixed	AMC	This work uses allows bounded skipping for tasks executions and allows priority adjustment. Citicality is adjusted/inherited in ATMP-CA and LBP-CA.
[64]	2018	Multi-core	Sporadic	Permenant	Dual	Multi	Fixed	Modified AMC	Integrated AMC to re-allocaion and re-execution techniques.
[85]	2019	Single-core	Periodic	Transient	Dual	Dual	Dynamic	EDF-VD	Uses Re-execution with dynamic priorities.
[66]	2019	Multi-core	Periodic	Transient	Multi	Dual	Dynamic	LETR-MC	Task Replication but replicas have similar criticality levels.
[65]	2019	Multi-core	Periodic	Any	Dual	Dual	Fixed	MSDRTGs	Uses re-execution and re-allocation but tasks utility to the overall system is not considered.
[93]	2019	Multi-core	Sporadic	intermittent	Dual	Dual	Fixed	WCRT	Can be integrated with TRTCM with their usage for the re-execution and replication techniques.

Table 3.3: Summary of related work strength and limitations

Ref.	Year	Sys Model	Task Model	Fault Model	Tasks Crit. Levels	System Crit. Levels	Schedule	Algorithm	Strengths and Limitations
[116]	2019	Multi-core	Sporadic	Transient	Multi	Many	Fixed	LETR-MC	Uses replication with more than two criticality levels.
[117]	2020	Single-core	Periodic	Transient	Dual	Dual	Dynamic	FANTOM	Framework for Re-execution and dynamic priority assignment.
[118]	2020	Single-core	Sporadic	Transient	Dual	Dual	Fixed	FTS-RFH	Aplies Checkpoint and Rollback Recovery but no consideration for the service and system utility.
[119]	2021	Multi-core	Periodic	Transient	Dual	Three	Dynamic	REALISM-scheme	Applies DVFS techniques with re-execution and replication.
[89]	2022	Single-core	Periodic	Transient	Dual	Dual	Fixed	Time Triggered	This theiss is based on even-driven systems, this work is based on time-triggered architecture which can integrate theTRTCM
[67]	2022	Multi-core	Periodic	Transient	Multi	Multi	Dynamic	EDF-VD	Uses re-execution with many criticality levels.
[87]	2022	Multi-core	Periodic	Any	Dual	Dual	Dynamic	POEVD	Hardware redundancy or Standby-Sparing and uses re-exeution re-allocation techniques.
[44]	2022	Multi-core	Sporadic	Any	Multi	Multi	Fixed	MSRP-FT	Uses DVFS, re-execution and replication but replicas have same criticality of their primaries.
[92]	2022	Multi-core	Any	Transient	Dual	Dual	Dynamic	MSSAC	Uses replication techniques for dynamic priorities.
[67]	2023	Single-core	Periodic	Transient	Multi	Dual	Dynamic	MC-IMW	Replication technique for more than two levels of criticlaity.

dancy techniques to include different forms of redundancy like task and hardware replication, can address the limitation of real-time scheduling theory. Mixed-criticality review concludes that none of the existing approaches update the system according to the empiric execution time of system services as in EWCET and E-ATMP framework introduced in this thesis, and constructing dependable mixed-criticality systems by realising high criticality services from redundant low criticality services with criticality less than or equal to the service they implement.

Chapter 4

Criticality Arithmetic: Building Dependable Mixed-criticality Systems

This chapter introduces novel CA protocols, ATMP-CA and LBP-CA. The protocols introduced in this chapter are distinct implementations that concretely manifest the theoretical model proposed in a seminal work by Menon [22]. Starting with an introduction to SIL and CA. Then, Mid-term ATMP-CA and Short-term LBP-CA system models and methodologies are presented. Finally, we conclude the chapter.

The organisation of this chapter is as follows Section 4.1 presents the ATMP-CA protocol. Section 4.2 presents the LBP-CA protocol. Section 4.3 concludes the chapter.

Criticality Arithmetic (CA) is the process of combining multiple redundant, lower-criticality jobs or components to implement a higher-criticality function. It is an engineering perspective that exploits the ability to compose a SW or HW component or service of a certain safety integrity level, by several components or services of a lower safety integrity level.

Safety Integrity Level (SIL) and Automotive Safety Integrity Levels (ASIL) are standards used to assess and specify the safety integrity of systems in different industries, such as process industries and automotive. The International Standardisation Organisation (ISO) 26262 [120] is the primary standard for functional safety in the automotive industry. It defines four levels: ASIL A, ASIL B, ASIL C, and ASIL D, where ASIL D represents the highest level of safety integrity. The ASIL levels in IISO 26262 are determined based on the assessment of the potential risk and the severity of the potential injury. The higher the

ASIL level, the more stringent the safety requirements for the system. Another standard, IEC 61508 [121] is a general standard that provides a framework for the functional safety of electrical, electronic, and programmable electronic systems. The Decomposition of high-integrity services into lower-integrity services is presented in table 4.1.

Despite the number of safety integrity levels stated by a standard, existing mixed-criticality schedulers may act incorrectly when they are subject to scheduling a set of mixed-criticality task sets that include criticality arithmetic aware tasks. The criticality arithmetic agnostic mixed-criticality schedulers, drop the allocation of low criticality services which lead to some level of degradation in the system utility, to ensure the safety and schedulability of high criticality services, in case of a resource shortage. In contrast, the criticality arithmetic aware mixed-criticality scheduler presented in this chapter exploits the information about criticality arithmetic, and enables finer graceful degradation for the system utility in case of resource shortages such as core-failures, compared to other mixed-criticality schedulers as shown by the experimental evaluation in Chapter 7 Section 7.1.

ASIL	Valid Decomposition
D	C(D) + A(D)
	B(D) + B(D)
	D(D) + QM(D)
C	B(C) + A(C)
	C(C) + QM(C)
B	A(B) + A(B)
	B(B) + QM(B)
A	A(A) + QM(A)

Table 4.1: Valid ASIL Decomposition

Example Application of Criticality/SIL Arithmetic: The ABS is an example of a safety-critical system where ASIL decomposition is applied to minimise cost while maintaining safety. The primary function of the ABS is to prevent the wheels from locking up during braking, thereby maintaining traction with the road surface and preventing skidding.

The ABS may include components such as wheel speed sensors and hydraulic actuators, each assigned a certain ASIL criticality level. For example, wheel speed sensors detect the speed of each wheel and provide data to the ABS controller. These sensors are assigned the highest criticality level, ASIL D.

Hydraulic actuators, which are responsible for adjusting the brake pressure in response to commands from the ABS controller, are assigned ASIL C.

By decomposing ASIL components such as the wheel speed sensors or hydraulic actuators, manufacturers can use a broader range of off-the-shelf products, which can be less expensive than a single high ASIL component.

As seen in Table **??**, a component with ASIL D, such as the wheel speed sensors, can be decomposed into two lower ASIL components, ASIL C(D) and A(D), or into two components of ASIL B(D). Similarly, the hydraulic actuators can be decomposed into two components of ASIL B(C) and A(C).

4.1 ATMP-CA: Adaptive-based Criticality Arithmetic aware Mid-term Scheduling

The ATMP [96], implements the TRTCM model [23, 2], which maximises each processing core by modifying the rate of arrival for running services in the system. The ATMP classifies system services and tasks by their flexibility to change their rate of arrival in case of resource shortage. The flexibility of a task to modify arrival rate, and its usefulness to the overall system mission, determine its allocation to computing cores in case of a core-failure. In such cases, ATMP sorts tasks by descending criticality. Then, ILP optimisation for tasks in the system is performed on the remaining processing-cores. The ILP optimises partitioned tasks on each core if are schedulable after AMC schedulability test, then it is processed by the processing core, and the binary search algorithm is performed to maximise the utilisation. In case that ILP solver finds no feasible solution to the whole task set, a single task with the least flexibility and criticality is dropped at each step according to its adaptation capability.

The ILP optimisation performed by ATMP-CA consists of two contributions:

1. The allocation of tasks to processing cores is criticality arithmetic aware.

2. The modification of the ILP goal function considers different forms of redundant tasks implementing a service based on criticality arithmetic.

4.1.1 ATMP-CA System Model

The following describes the system model. The system model consists of three components: the *platform*, the *service*, and the *task* models. The system consists of a multi-core platform that runs a set of mixed-criticality tasks. A criticality arithmetic agnostic service is a service that is implemented by a single task, whereas a criticality arithmetic aware service is implemented by two or more tasks, also known as *replicas*. In a criticality arithmetic agnostic service, the criticality of the service is equal to the criticality of the task that implements it, whereas, in a criticality arithmetic aware service, the criticality of the service is always higher than that each task that implements it.

The following describes the *platform*, *service*, and *task* models for criticality arithmetic multi-core mixed-criticality systems.

Platform model The platform model is constituted from many processing cores $cr \in Cores$.

The computational capability $Cap(cr)$ is modelled for each core $cr \in Cores$. The WCET $c(\tau)$ of task τ is estimated according to the platform computing capacity, because WCET is based on a defined computing capacity denoted as $Cap(cr) = 1.0$.

The total computing capacity of the whole system can be calculated as

$$\sum_{cr \in Cores} Cap(cr)$$

This platform model allows the precise modeling of platforms with homogeneous cores. In the case of non-homogeneous cores, it would be better to instead use a different WCET of a task for each core $cr \in Cores$.

Service model Here we describe the set of services and the service.

Γ is the set of services to be scheduled:

$$\Gamma = \{s_i, ..., s_n\} \mid i \geq 1$$

A service can be realised by a single task or through multiple tasks using criticality arithmetic. The system S provides a number of services s, each service is identified by the following tuple:

$$s = \langle id, l, \gamma \rangle$$

$s.id$ the service-identifier.

$s.l$ the criticality level of a the service, with $s.l > 0$. The $s.\vec{l}$ represents all criticality levels defined by the platform: $\vec{l} = (l_1, \ldots, l_k)$, where the minimum level is l_1, and the maximum criticality level is l_k.

$s.\gamma$ the task set $\tau \in \gamma$ collaborates to realise service s . in case single task realises service ($|s.\gamma| = 1$), the task in considered to be agnostic criticality-arithmetic, and in this case the criticality of the task has equal criticality to the service it implement. If more than single task collaborate in implementing the service ($|s.\gamma| > 1$), then service use criticality arithmetic: each task criticality in $s.\gamma$ has a criticality less or equal than the criticality of the service it implements but with redundant executions. In fault-tolerance task models, redundancy assumes the same criticality among service replicas. The final sentence highlights the novelty of CA

Task Model Here we describe the set of task/s to implement a service. The task τ is identified by the following tuple:

$$\tau = \langle id, s, d, c, l, uf, p, u\gamma \rangle$$

$\tau.id$ is the task's identifier.

$\tau.s$ service realised by task τ.

$\tau.d$ the task deadline d.. The deadline model is implicit-deadline, where the task's deadline and period are equal e.g. $\tau.d = \tau.uf.p_{prim}$. We assume that the schedulability analysis is included in ATMP-CA, however, it is not a part of the ILP optimiser.

$\tau.c$ the execution time c, of task τ. A task may have redundant c estimates doe each criticality level.

$\tau.l$ task's criticality level l with $l > 0$. The $\vec{s.l}$ represents all criticality levels defined by the platform: $\vec{l} = (l_1, \ldots, l_k)$, where the minimum level is l_1, and the maximum criticality level is l_k.

$\tau.uf$ the tasks utility function τ. The chosen performance metric such as period/rate, throughput, jitter, and energy. In this context, and in the evaluation 7 the focus is on the period. The following tuple characterises the utility function:

$$uf = \langle p_{prim}, p_{tol}, u_{tol} \rangle$$

where, p_{prim} represents the primary-period where the relative utility is 1.0. The tolerance-period is p_{tol}, minimum relative utility is u_{tol}, and the utility is ranged between the primary-period and tolerance bounds.

$\tau.p$ and $\tau.u$ denote the task's optimised period p and utility u. The optimised period p is selected from the range between the primary and tolerance bounds:

$$\tau.uf.p_{prim} \leq \tau.p \leq \tau.uf.p_{tol}$$

The task's utility function $\tau.u = \tau.uf(\tau.p)$ defines the resulting utility. The achieved *absolute utility* $\tau.U$, is multiplication of optimised utility by the tasks criticality $\tau.U = \tau.uf(p) \cdot \tau.l$. The $\tau.U$ is an evaluation metric for ATMP-CA and reference scheduler SAMP 7.

4.1.2 Criticality-Arithmetic-Aware Allocation (CAAA)

The ATMP-CA allocation of tasks to cores is different from the ATMP. The allocation avoids assigning a replicated task from a service to cores that another replica was allocated to previously. This guarantees fault tolerance for the replicated tasks, allowing a maximum of one disruption from each core failure. Furthermore, if a service has more task replicas than cores available, the ATMP-CA core allocation discards this replica.

The criticality arithmetic aware allocation is a multi-core scheduling algorithm, that assigns/allocates tasks to processor cores. criticality arithmetic aware allocation algorithm aims to avoid allocating replicated tasks to the same core. For example, assume we have two cores c_1, c_2 and two high-criticality criticality arithmetic aware services s_1, s_2, each service is constituted by two $s_1.\gamma = 2, s_2.\gamma = 2$ low-criticality services $s_1.\tau_A, s_1.\tau_B$ and $s_1.\tau_C, s_1.\tau_D$ respectively.

Criticality arithmetic agnostic schedulers may allocate both low criticality replicas of the same high criticality service $s_1.\tau_A$ and $s_1.\tau_B$, or $s_1.\tau_C$ and $s_1.\tau_D$ to the same core $c_1(s_1.\tau_A, s_1.\tau_B)$ and $c_2(s_2.\tau_A, s_2.\tau_B)$. Criticality arithmetic aware schedulers allocate each low criticality replica of the same high criticality service s_1 or s_2 to the different core $c_1(s_1.\tau_A, s_2.\tau_A)$ and $c_2(s_1.\tau_B, s_2.\tau_B)$.

The algorithm starts with two inputs. The first input is a list of sorted tasks by decreasing criticality. The second input is a list of cores available for allocating the given tasks. Criticality-based sorting is to ensure that the highest criticality services are allocated before the lower criticality ones. Though we assume similar cores in terms of speed and memory, ATMP-CA is not limited to cores with equal capabilities.

The criticality arithmetic aware allocation algorithm works by iteratively removing the task with the maximum criticality from the sorted task list, searching the processing core with the minimum utilisation, and assigning the task to the found core. The found core has to have no replica of the task in question. If the task already has a replica on the core, or the task load is greater than the available capacity on that core. Then, ATMP-CA searches for another core, If a suitable core is found, the task is allocated to the core. Otherwise, the task is dropped

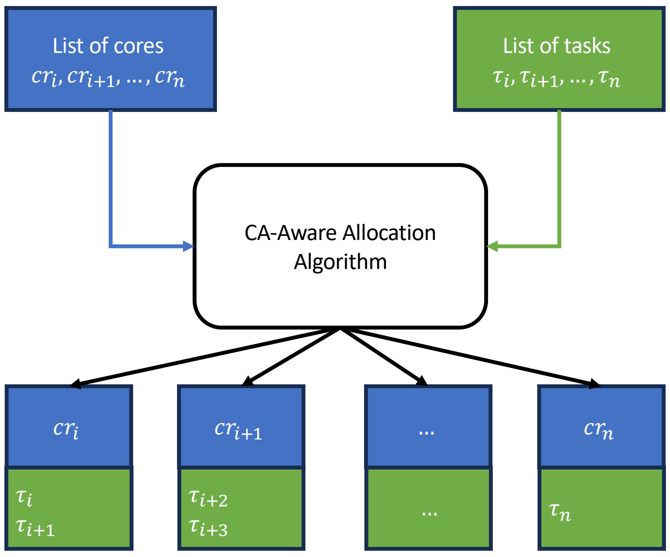


Figure 4.1: Conceptual Diagram for Criticality Arithmetic aware Tasks-to-Cores Allocation

Algorithm 1 shows the implementation of the Criticality Arithmetic aware tasks to cores under in ATMP-CA. The algorithm receives two input: Γ, tasks list sorted with descending criticality, and CS, the list of all available cores for allocation process. The outer while-loop from line 2-12 runs as long as there are tasks in Γ. In line 3 the function $getTaskWithMaxCrit(\Gamma)$ removes from Γ the task with maximum criticality. In line 4 the list CS with all core IDs is copied as CS'. This copy is needed in case of replicated tasks to make sure that no two replicated tasks of the same service end up on the same core. In line 5 the function $getCoreWithMinLoad(CS')$ removes from the core list CS' the core with currently the minimum task load assigned. Line 6-8 checks whether the task t_{id} already has a replica on the core c_{id}. If this is the case, then inside the loop a new core is extracted from CS' until either a core is found that has no replica of t_{id} allocated or all cores have been tried without success. Line 9-11 does register the allocation of the task t_{id} to core c_{id} only if the previous search for the core without a replica already allocated was successful. If this search was not successful, then task t_{id} is simply dropped and not allocated to a core. This search can only fail if there are fewer operational cores available than the number of tasks replicas services using criticality arithmetic is implemented with. When Algorithm 1

Algorithm 1: Criticality-Arithmetic-Aware Allocation of Tasks to Cores

```
Input : $\Gamma$: task list sorted by criticality;
        $CS$: list of computing elements ($Cores$);
begin
    while $\Gamma \neq \emptyset$ do
        $t_{id} \leftarrow getTaskWithMaxCrit(\Gamma)$;
        $CS' = CS$;
        $c_{id} \leftarrow getCoreWithMinLoad(CS')$;
        while $hasReplica(t_{id}, c_{id}) \wedge CS' \neq \emptyset$ do
            $c_{id} \leftarrow getCoreWithMinLoad(CS')$;
        end
        if $\neg hasReplica(t_{id}, c_{id})$ then
            $addTaskToCore(t_{id}, c_{id})$;
        end
    end
end
```

terminates, then each of the tasks in the task set has been either allocated to a core or has been dropped. The purpose of this allocation is to assign the tasks to a core. Later, within each core, as part of the utility optimisation, which is the same as in ATMP [96], some tasks might be removed again from a core to pass the schedulability test.

4.1.3 Formulation of Criticality Arithmetic-Aware ILP

Here we describe the ILP formulation to find the optimal task periods in case of resource shortages. We describe the constants and variables of that ILP problem, the goal function to optimise the system utility, and the different constraints that have to be considered.

Optimisation parameters (constants): In ATMP the units of scheduling are tasks. As presented in Section 4.1.1, every single task τ_i of a task set Γ consists of the following components:

$$\tau_i = \langle uf, s, d, c, WT, p, u \rangle$$

where the utility function *uf* is characterised by the following properties: $\tau_i.uf = \langle p_{prim}, p_{tol}, u_{tol} \rangle$. We model the utility function and the criticality of each task τ_i in the ILP problem with the following constants:

$\tau_i.c_i$	...	the WCET of τ_i
$\tau_i.uf.p_{prim}$	...	the primary period (with utility $\tau.uf.u_{prim} = 1.0$),
$\tau_i.uf.p_{tol}$	...	the tolerance period,
$\tau_i.u_{tol}$	...	the utility at the tolerance period $p_{tol,i}$
$\tau_i.WT$	...	the criticality weight of τ_i
$Cap(cr)$	...	the computing capacity of $cr \in Cores$

The parameters $\tau_i.p_{prim}$, $\tau_i.p_{tol}$, $\tau_i.u_{tol}$ characterise a task's utility function by two linear lines, as shown in Figure 3.1.

The horizontal line is a constant utility of 1.0, which can be directly expressed as an ILP constraint. The sloped line of each task's utility function can be also derived from $\tau_i.p_{prim}$, $\tau_i.p_{tol}$, $\tau_i.u_{tol}$, for which we have to calculate its slope $\tau_i.k_i$ and y-intercept q_i to express it as a line equation:

$$\text{line equation} \ldots \tau_i.u = \tau_i.p \cdot \tau_i.k + \tau_i.q \tag{4.1}$$

$$\text{slope} \ldots \tau_i.k = \frac{\tau_i.u_{tol} - 1}{\tau_i.p_{tol} - \tau_i.p_{prim}} \tag{4.2}$$

$$\text{y-intercept} \ldots \tau_i.q = \frac{\tau_i.p_{tol} - \tau_i.u_{tol} \cdot \tau.p_{prim}}{\tau_i.p_{tol} - \tau_i.p_{prim}} \tag{4.3}$$

Optimisation variables to optimise task configurations are task period $\tau.p$ and the relative utlity of the task $\tau_i.u$:

$\tau_i.p$	...	the selected rate of arrival of task τ_i,
$\tau_i.u$	...	the selected or optimised utility of task τ_i,

Objective function The optimisation ILP goal function maximises the system utility through maximising the utility variable $\tau_i.u$ of each task τ_i multiplied by its criticality weight $\tau_i.WT$:

$$SU_{S.tol} = \sum_{\tau_i \in TS} \tau_i.WT \cdot \tau_i.u \tag{4.4}$$

The criticality weight $\tau_i.WT$ is explained below at the optimisation constraints.

Optimisation constraints We express the piecewise affine approximations

$$\tau_i.u = \begin{cases} 1 & \text{if } \tau_i.uf.p_{prim} \geq \tau_i.p \\ \tau_i.p \cdot \tau_i.k + \tau_i.q & \text{if } \tau_i.uf.p_{prim} < \tau_i.p_i \leq \tau_i.uf.p_{tol} \end{cases}$$

of the utility functions to the following constraints:

$$\tau_i.u \leq 1 \tag{4.5}$$

$$\tau_i.u \leq \tau_i.p \cdot \tau_i.k + \tau_i.q_i \tag{4.6}$$

The *resource constraints* are used to limit the workload at each of the available cores $cr_i \in Cores$. The maximum workload a core cr can take is its computing capacity $Cap(cr)$:

$$\sum_{\tau_i \in \Gamma} \frac{\tau_i.c}{\tau_i.p} \leq \sum_{cr \in Cores} Cap(cr) \tag{4.7}$$

The *tolerance constraints* determine the maximal acceptable period of $\tau_i.p$

$$\tau_i.p \leq \tau_i.p_{tol} \tag{4.8}$$

In ATMP-CA, the weight $\tau_i.WT$ is set to the task criticality $\tau_i.l$, whereas in ATMP-CA, we calculate the weight $\tau_i.WT$ of a task τ_i according to the sate of replicas on all cores. The $\tau_i.WT$ is the criticality $\tau_i.l$ of the task $\tau_i$when other replicas implement the same service already been allocated with their maximum utility or previously processed core, and inherit the weight $\tau_i.WT$ for the criticality $\tau_i.s.l$ of the service $\tau_i.s$ it implements, which is greater than or equal to the criticality of the task $\tau_i.l$.

The algorithm of ILP formulation in ATMP-CA to calculate the weight WT_i is shown in Algorithm 2.

The algorithm input is an individual task, for which we want to calculate its weight $\tau_i.WT$ concerning the task criticality $\tau_i.$ or the criticality of the service $s.l$ the task τ_i implements. Then, the algorithm checks for two conditions: the first checks if replicas of the task in question have been allocated to another core. the second checks if previously allocated replicas of task in question have been allocated with maximum utility or primary period. The latter is dependent on the former. In case any one of the conditions fails, then we calculate the weight for the service criticality. If both are satisfied, then ATMP-CA calculates the weight based on the task's criticality.

The algorithm receives one input, the task τ, that we want to calculate its weight WT_i as presented in Equation 4.4. As shown In line 2, the function *CoresHaveReplicaWithMaxUtil*(τ) checks if a task's replica τ has been optimised with maximum utility in the previously processed cores, The function *CoresHaveReplica*(τ) assesses whether the yet-to-be processed cores will include an allocation of a replica of task τ. If any of the functions returned *True*, then we select, as presented in line 3, the task's criticality $\tau.l$. However, we select in line 4 the criticality $\tau.s.l$ of the service $\tau.s$ that task τ implement. Finally in line 7, the calculated weight WT for the LP optimiser objective function is set to the selected criticality l.

Algorithm 2: Calc-CA-Aware-ILP-Weight

```
Input : $\tau$: task for which to calculate its ILP weight $WT$;
1 begin
2     if $CoresHaveReplicaWithMaxUtil(\tau) \vee CoresHaveReplica(\tau)$ then
3         $l = \tau.l$
4     else
5         $l = \tau.s.l$
6     end
7     $WT = l$
8 end
```

4.2 LBP-CA: Criticality Arithmetic aware Short-term Scheduler

This section presents the Criticality Arithmetic Lazy Bailout Protocol (LBP-CA). LBP-CA is a short-term mixed-criticality scheduling protocol with smooth degradation based on criticality arithmetic. LBP-CA Improves the recovery from transient resource shortages by a quick return to normal behaviour or low criticality mode and enhances the schedulability of tasks by minimising the negative impact on low criticality services using the knowledge of criticality arithmetic. LBP-CA is built upon the LBP protocol, and LBP itself is based on the BP protocol. The following presents the system model, three criticality levels—Normal, Bailout, and Recovery—on which the three protocols operate are presented, and the quick return mechanism featured in LBP-CA is explained using Figure 4.2.

4.2.1 LBP-CA System Model

The following explains the *platform*, *service*, *task*, *job*, and *system modes* models.

Platform Model The platform model consists of a single computing element, called core cr. For this thesis, it does not matter whether this core is part of a multi-processor CPU or a single-core processor CPU.

Service Model The set of services, the set of tasks to implement a service, and the criticality of a service. are identical to the ATMP-CA service model, where Γ is the set of services to be scheduled, defined by the tuple following tuple $\langle id, l, \gamma \rangle$, and each service is realised by single or many instances with using the information of criticality-arithmetic. The platform S runs a group of mixed-criticality services s, where each service is identified by $\Gamma = \{s_i, ..., s_n\} \mid i \geq 1$. $s.id$ represents the service's name. $s.l$ is the service's criticality level, with $s.l > 0$. A higher value of $s.l_i$ means a higher level of criticality. The vector $\vec{s.l}$ is used to represent all possible criticality levels in a system: $\vec{l} = (l_1, \ldots, l_k)$, with l_1 being the minimum and l_k being the maximum possible criticality level. $s.\gamma$ is the set of tasks $\tau \in \gamma$ collaborates to implement the service s . If only one task implements the service ($|s.\gamma| = 1$), then no criticality arithmetic is used, and the task in this case has the same criticality as the service or criticality arithmetic agnostic task. If multiple tasks implement the service ($|s.\gamma| > 1$), then criticality arithmetic is used: the criticality of each task in $s.\gamma$ has a criticality less than the criticality of the service it implements but with redundant executions. In fault-tolerance task models, redundancy assumes the same criticality among service replicas. The final sentence highlights the novelty of criticality arithmetic

Task Model Each task τ of a task set γ is defined as follows:

$$\tau = \langle s, d, c, l, p \rangle$$

$\tau.s$ is the service that is implemented by task τ.

$\tau.d$ is the relative deadline d of task τ. We assume constrained deadlines, where the task's deadline is less than or equal to its period e.g. $\tau.d \leq \tau.p$.

$\tau.c$ is the WCET estimate c of task τ. CPU scheduler requires different WCET bounds for each criticality level. Therefore tasks have two WCWCETET, $\tau.c_{LO}$ and $\tau.c_{HI}$, where low criticality tasks lower WCET bound is equal to its higher WCET bound $\tau.c_{LO} == \tau.c_{HI}$, and high criticality tasks higher WCET bound is always greater than the lower WCET bound $\tau.c_{HI} \geq \tau.c_{LO}$.

$\tau.l$ is the criticality level l of task τ with $l > 0$. A higher value of l means a higher level of criticality. The vector $\vec{\tau.l}$ is used to represent all possible criticality levels in a system: $\vec{l} = (l_1, \ldots, l_k)$, with l_1 being the minimum and l_k being the maximum possible criticality level.

$\tau.p$ represents the task's period. Note that there is no use for the utility function function here.

Job Model The individual instances of a task τ_i at runtime are called jobs $\tau_{i,k} \mid k \geq 0$, where $\tau_{i,0}$ denotes the first job of task τ_i. A job $\tau_{i,k}$ has the following parameters:

$$\tau_{i,k} = \langle a, p, d, et, ec, l \rangle$$

where the parameters are defined as follows:

$\tau_{i,k}.a$ is the arrival time

$\tau_{i,k}.et$ actual execution time

$\tau_{i,k}.l$ is the criticality level, which is inherited from its task $\tau_{i,k}.l = \tau_i.l$.

System Modes We have three system modes, Normal, Bailout, and Recovery.

Normal-mode: Represents the low criticality mode where the system tasks and services are scheduled and executed within their assigned WCET estimates.

Bailout-mode: A high criticality mode that is activated due to a transition from normal mode in response to WCET overrun by one of the low or high criticality tasks. Only high criticality tasks $\tau.l = HI$ scheduled in this mode, and low criticality tasks $\tau.l = LO$ are abandoned from future releases, except the ones already released before the mode activation. The execution of high criticality tasks during this mode, is controlled by the variable, Bailout Fund BF. The BF funds high-criticality services with execution time according to their need and accumulates unused execution time from high-criticality tasks, where $\tau.l = HI$, that execute with less than their respective WCET bounds $\tau.c_{LO}$ and $\tau.c_{HI}$, and from low criticality tasks, where $\tau.l = LO$, execute with less than their WCET bounds, where $\tau.c_{LO} == \tau.c_{HI}$, and are released before the WCET overrun.

Recovery-mode: an additional criticality mode activated at the end of Bailout-mode when the BF fund is zero, and a task with high criticality and the lowest priority (longest deadline) is released during the most recent Normal or Bailout modes. Direct return from bailout mode to normal mode may cause interference in the schedule safety, because the released task with the highest criticality and lowest priority may be interfered with by a lower criticality task with higher priority. In Normal mode, tasks are prioritised according to their deadlines $\tau.d$, not criticality $\tau.l$.

4.2.2 Quick Return to Normal Behaviour using Criticality Arithmetic

Allocated tasks to cores by a criticality arithmetic mid-term scheduler as ATMP, are prioritised for execution on the underlying core. The underlying real-time scheduler in this section is the Fixed-priority Deadline Monotonic scheduling algorithm DM [122], where tasks are prioritised according to their deadlines. Tasks with shorter deadlines have higher priority than tasks with longer deadlines. Provisioning the prioritisation of tasks on each core scheduler is necessary since DM is not optimal for mixed-criticality tasks [4, 6]. Rate Monotonic RM and Earliest Deadline EDF First are not optimal as well [5].

AMC protocol provides a scheduling scheme that augments Fixed-priority schedulers to consider the notion of criticality, provides a pre-runtime verification, priorities assignment [8], and guaranteed bounded degradation for the system services whenever a resource shortage in form of WCET overrun occurs, by aborting all low criticality tasks and abandon their future releases. However, the guaranteed bound is verified offline and under certain assumptions. Short-term schedulers and frameworks are important in case AMC assumptions are violated to enable runtime survivability [7]. [5].

The Short-term schedulers, BP and LBP, ensure the schedulability of high-criticality tasks and minimise the return time from high-criticality modes to low-criticality modes. The criticality arithmetic aware short-term scheduler augments the criticality arithmetic agnostic short-term schedulers by the ability for a quicker return to normal execution mode using the criticality arithmetic information.

I integrated Criticality arithmetic to existing Short-term scheduling Bailout-based protocols, BP and LBP, the resulting protocol is LBP-CA. LBP-CA exploits information about CA) via task redundancy. Compared to BP and LBP, it shows that LBP-CA can return earlier than a normal schedule after a resource shortage. The two key ideas of Bailout-based protocols are that the available slack in an FP schedule is translated into a Bank of execution time funds and the two criticality modes developed by ACM, are extended by one additional mode, the Recovery-mode. [5].

The Bank of execution time, named *Bailout Funds* and denoted as BF, is a variable that is initialised and updated during different transitions between the system modes. It is restricted to be greater or equal to zero $BF \geq 0$. The initialisation and update of BF can be according to the need to withdraw extra execution time or *funds* by high criticality tasks, or deposit underspent execution time either from low or high criticality tasks, during certain system runtime modes. low criticality tasks, released before a resource shortage, deposit their execution times to BF but never withdraw an execution time. high criticality tasks deposit their execution times to BF whenever a resource shortage occurs and allowed to withdraw from BF whenever they exceed their $\tau.c_{LO}$ estimate. [5].

Bailout-based protocols, BP, LBP, and LBP-CA, use three criticality modes: Normal, Bailout, and Recovery modes. In the following, the system model, and definition for each mode are presented, followed by a figure4.2 that shows the three system criticality modes, including the transitions and activation from/to and of each mode. While explaining the figure, the novel feature of LBP-CA for a quick return to normal operation is highlighted in this section and evaluated in the experimental evaluation chapter, Chapter 7. [5].

Figure 4.2 shows the transitions between system modes. starting mode is normal mode. In this mode, tasks' execution is prioritised according to their deadlines; task with a shorter deadline has the highest priority. low criticality jobs exceed their optimistic $\tau.c_{LO}$ are aborted, and the transition from normal mode to bailout mode is activated when a high criticality job exceeds its optimistic execution τ_1. [5].

As shown in Figure 4.2, The return to the normal mode from the Bailout-mode, relies on two conditions. The first is that an idle instant has occurred - which indicates that no job is released for execution at the idle instant - and the bailout fund BF is reset to zero. [5].

When a high criticality task overruns its $\tau.c_{LO}$ execution time, transition to Bailout-mode is activated and the bailout fund BF is initialised to zero $BF = 0.0$. At the activation of the Bailout-mode, the high criticality task, which activated the mode, deposits the difference between its optimistic execution time $\tau.cLO$, and the pessimistic execution time $\tau.cHI$ into the bailout fund BF, where

$$BF = \tau.c_{HI} - \tau.c_{LO}$$

. Simultaneously, all future releases for low-criticality tasks are abandoned during this mode. Then, all high criticality tasks withdraw execution time whenever they overrun the optimistic execution time $\tau.c_{LO}$, including the one activated the Bailout-mode, and deposit the difference between their respective WCET estimates, the optimistic and pessimistic WCET, where

$$BF = BF + \tau.c_{HI} - \tau.c_{LO}$$

. [5].

During the Bailout, low criticality jobs released before the activation, and high criticality jobs, update the bailout fund BF by reducing the the underspent execution time $\tau.c$, if their actual execution time $\tau.c$ is less than the optimistic execution time $\tau.c_{LO}$, where

$$BF = BF - (\tau.c_{LO} - \tau.c)$$

. low criticality jobs released before the activation are aborted their execution times are used to reduce the bailout fund, where

$$BF = BF - \tau.c_{LO}$$

[5].

Funded high criticality jobs overrun optimistic WCET during Bailout-mode and complete before consuming the funded execution time budget, reduce the bailout fund BF, where

$$BF = BF - (\tau.c_{HI} - \tau.c)$$

. [5].

As seen in Figure 4.2, the transition from Bailout-mode to Normal or Recovery modes occurs according to the occurrence of an idle instant or when the bailout fund BF becomes zero. In the case of idle instant occurrence, a transition from bailout mode to normal mode is activated, and BF is reset to zero. In the case where the bailout fund BF becomes zero, a transition from Bailout-mode to Recovery-mode is activated, and the high criticality job with the lowest priority

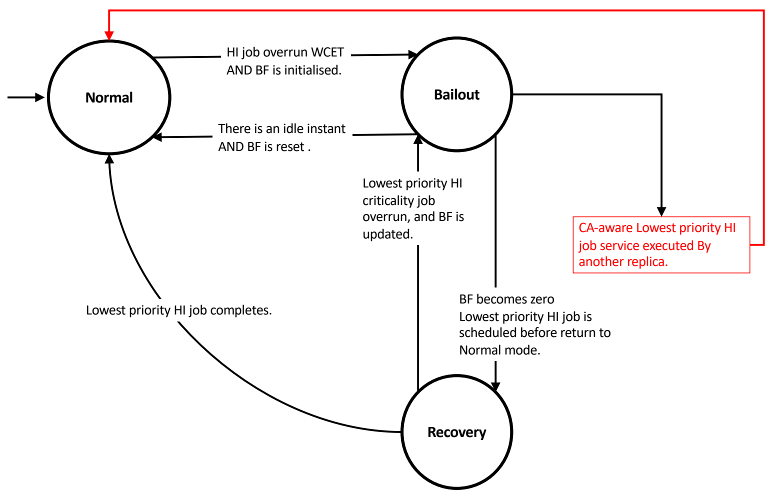

Normal
Bailout
Recovery
HI job overrun WCET AND BF is initialised.
There is an idle instant AND BF is reset .
Lowest priority HI criticality job overrun, and BF is updated.
Lowest priority HI job completes.
BF becomes zero Lowest priority HI job is scheduled before return to Normal mode.
CA-aware Lowest priority HI job service executed By another replica.

Figure 4.2: Criticality mode changes in Bailout-based protocols, LBP-CA quick return to Normal-mode is coloured in red.

is recorded to execute in the Recovery-mode. The return from Recovery-mode to Bailout occurs when the recorded high criticality job overruns its optimistic execution time $\tau.c_{LO}$. In this case, it initialises and deposits its budget to the bailout fund BF, and the Bailout-mode continues as mentioned above. The Recovery-mode is designed for the safety of the high-criticality task with the lowest priority (longest deadline). In existing Bailout-based protocols, BP and LBP, the transition from Recovery-mode to normal mode is only activated when the high criticality job completes its execution time. [5].

The LBP protocol differs from the classic BP protocol by allowing the release of low-criticality tasks during the Bailout and Recovery modes. This is applied by inserting the released low criticality tasks into a low priority queue to possible eventual execution during idle time in the future. LBP-CA differs from both reference protocols, BP and LBP, in facilitating a quicker return from Bailout to Normal mode. LBP-CA accelerates the activation of the transition from Bailout to Normal mode when the lowest-priority, high-criticality job service has already been executed by another replica. This acceleration prevents the transition from Bailout mode to Recovery mode. This mechanism is highlighted by the red colour in Figure 4.2. The quicker return to normal mode enables the system to resume the release for low criticality jobs. [5].

4.3 Chapter Summary

This chapter introduced novel Criticality Arithmetic (CA) protocols, ATMP-CA and LBP-CA. Started with an introduction to SIL and Criticality Arithmetic. Then, Mid-term ATMP-CA and Short-term LBP-CA system methodologies are presented in Section 4.1 and Section 4.2, respectively.

Chapter 5

EWCET and E-ATMP: WCET Overrun Treated as WCET Update

This chapter presents the motivation and methodology for the development of the Empiric Adaptive Tolerance-based Mixed-criticality Protocol (E-ATMP) framework. First, the used system model in the development of the E-ATMPframework is described. Then, the EWCET and E-ATMP framework are explained in detail. Finally, an example of E-ATMP handling WCET overrun is discussed. The organisation of this chapter is as follows: Section 5.1 presents the system model. Section 5.2 explains the E-ATMP framework. Section 5.3 presents an example of the E-ATMP framework. Section 5.4 concludes the chapter.

It is challenging to determine a safe upper bound for the WCET, especially in cases involving complex processor hardware. More readily available are methods to establish an optimistic WCET estimate, which may be close to the real WCET but lacks a guarantee of safety.

High-criticality tasks typically have a pessimistic or higher WCET estimate used for task schedulability analysis. This analysis takes into account the optimistic WCET when verifying the safety of execution during normal, best-case, or low-criticality system modes, while the pessimistic WCET is applied for the analysis of disruptive, worst-case scenarios in high-criticality system modes.

The concept of Empiric Worst Case Execution Time (EWCET) introduces a new adaptation model for responding to the execution time overrun in real-time and mixed-criticality systems. EWCET updates the execution time during the task's runtime and can adjust the execution schedule in response to variations observed in each execution. EWCET initially is determined by the optimistic

WCET estimate. WCET serves as an upper bound for the execution time of service code under the worst-case scenario. It can be determined through dynamic experiments or static analysis, where the latter involves analysing the worst longest execution path of the service code.

When a high-criticality task exceeds its WCET budget without completion, the measurement of empirical execution for the high-criticality task begins. This measurement can only be interrupted by tasks with equal or higher criticality, but it resumes once the high-criticality task resumes execution. Conversely, when a low-criticality task exhausts its WCET budget without completion, it is deferred for future completion.

EWCET cannot function independently; it depends on a reconfiguration protocol to adjust the schedule based on recently acquired EWCET data. The reconfiguration algorithm must be capable of utilising this new information to ensure smooth degradation. A framework that manages the integration of EWCET and reconfiguration protocol has been developed to answer the research question, coined by the E-ATMP framework.

The E-ATMP framework uses EWCET and reconfigures schedules of mixed-criticality tasks and services using the ATMP protocol. ATMP protocol exploits safety margins of systems tasks in the form of utility functions to enable a smooth degradation for the overall system utility.

Therefore, E-ATMP addresses this question with a novel approach that builds upon recent findings in the literature. Since E-ATMP consistently provides results that are at least as good as reference methods, it offers a concrete answer to the question of how the system can adapt based on occurrences of tasks' overruns of WCET estimates.

5.1 EWCET System Model

We assume a single processor system S, which schedules a task-set $\Gamma = \{\tau_i, ..., \tau_n\}$ using a FP algorithm. The system S consists of the following parameters:

$$S = (l, \Gamma, JQ)$$

$S.l$ is the criticality mode of the system: $S.l \in \{0, \mathit{Val}_1, \mathit{Val}_2, \dots\}$, where $\mathit{Val}_1, \mathit{Val}_2, \dots$ denote the different criticality levels of the tasks (see Section 5.1.1), which are all expressed as numbers > 0. By default, the system is in ground mode: $S.l = 0$. A criticality mode $S.l > 0$ denotes the case that an overrun of a task's WCET estimate has happened. The change of the criticality mode based on the runtime events is described in 5.2.

$S.\Gamma$ is the set of tasks to be scheduled:$\Gamma = \{\tau_i, ..., \tau_n\} \mid i \geq 1$.

$S.JQ$ a queue structure maintained by the FPPS algorithm, which contains an ordered list of jobs released for execution.

5.1.1 Task Model

Each task $\tau_i \in \Gamma$ consists of the following parameters:

$$\tau_i = \langle id, p, d, c, l, u, U, uf \rangle$$

where the parameters of task τ_i are defined as follows:

$\tau_i.id$ is the task identifier. $\tau_i.p$ is the task period. In our case, the period is an output of the system optimisation based on the utility function described in Section 5.2.2.

$\tau_i.d$ is the relative task deadline

$\tau_i.c$ is the WCET estimate (can be optimistic). In this model, each WCET estimate can be potentially optimistic, regardless of the task's criticality level. The system is supposed to find an overall smooth degradation of system services, even if the WCET estimate of the most critical task is overrun.

$\tau_i.l$ is the criticality level of the task. As explained in Section 5.2.2, we map utility values to numeral weights ($\tau_i.l > 0$), for the sake of the system optimisation. In our model the criticality levels not only serve as an ordering relation between different criticalities, but the value itself is also important for the system optimisation.

Other mixed-criticality research often uses only two symbolic criticality levels like $\{HI, LO\}$, which in our model would be mapped to numeral weights, for example, $HI = 2.0$ and $LO = 1.0$.

$\tau_i.u$ is the relative utility of the task.

$\tau_i.U$ is the absolute utility of the task, which is the product of its relative utility $\tau_i.u$ and its criticality $\tau_i.l$:

$$\tau_i.U = \tau_i.u \cdot \tau_i.l$$

f_{util} is the utility function, which describes the relationship between the task's period $\tau_i.p$ and its relative utility $\tau_i.u$. The utility function is further detailed in Section 5.2.2.

Furthermore, it is assumed that it is acceptable that a task is continued with its execution after its deadline has passed, as this is necessary to be able to observe the EWCET. This would also require that the real-time application is within a *temporal safe-state*, in order to tolerate the delay of the service due to the overrun and the system reconfiguration.

5.1.2 Job Model

The individual instances of a task τ_i at runtime are called jobs $\tau_{i,k} \mid k \geq 0$, where $\tau_{i,0}$ denotes the first job of task τ_i. A job $\tau_{i,k}$ has the following parameters:

$$\tau_{i,k} = \langle a, p, d, et, ec, l \rangle$$

where the parameters are defined as follows:

$\tau_{i,k}.a$ is the arrival time

$\tau_{i,k}.et$ execution time

$\tau_{i,k}.ec$ is the empiric EWCET. The empiric EWCET of a job $\tau_{i,k}.ec$ is the maximum of the tasks WCET estimates $\tau_i.c$ and all the execution times $\tau_{i,k}.et$ observed so far from all job instances so far of task τ_i:

$$\tau_{i,k}.ec = \max\left(\max_{n\in\{0\ldots k\}}\left(\tau_{i,n}.et\right), \tau_i.c\right) \tag{5.1}$$

Thus, the empiric EWCET $\tau_{i,k}.ec$ of a job is not defined only by the task τ_i and its current job $\tau_{i,k}$ itself, but also by the whole history of execution times $\tau_{i,k}.et$ of all previous job instances $\tau_{i,n \mid 0\leq n<k}$ of the same task.

$\tau_{i,k}.l$ is the criticality level, which is inherited from its task $\tau_{i,k}.l = \tau_i.l$.

Theorem 5.1.1 *The empiric WCET $\tau_{i,k}.ec$ is non-strict monotonically growing with each new job instance $\tau_{i,k}$ against the previous job instance $\tau_{i,k-1}$.*

Proof 5.1.1 The proof of this theorem follows directly from Equation 5.1, as the EWCET for the first job $\tau_{i,0}$ gets initialised with task's WCET estimate $\tau_i.c$ ($\tau_{i,0}.ec = \tau_i.c$), and for each succeeding jobs $\tau_{i,k}$ the EWCET is the maximum of the previous job $\tau_{i,k-1}$'s EWCET and the current job's execution time $\tau_{i,k}.et$ ($\tau_{i,k}.ec = \max(\tau_{i,k-1}.ec, \tau_{i,k}.et)$). So any update of the EWCET is based on the maximum with the previous job's EWCET, which proves the theorem. □

The reconfiguration framework for E-ATMP will be presented in the processing section.

5.2 The Reconfiguration Framework E-ATMP

This subsection offers a comprehensive explanation of the E-ATMP framework, divided into two parts. The first part covers the framework's construction, while the second covers the formulation of resource shortages as an ILP problem.

5.2.1 Construction of E-ATMP Framework

The E-ATMP framework is designed and constructed to ensure the smooth degradation of service in mixed-criticality systems when resource shortages occur. E-ATMP achieves this by coordinating various system components and effectively managing the WCET overruns of both low-criticality and high-criticality tasks. The framework's primary goal is to maintain system functionality and prioritise critical tasks based on importance and adaptability during resource shortages.

E-ATMP coordinates the following system components: the underlying real-time scheduler, responsible for task scheduling and execution; the mixed-criticality scheduler, which manages task scheduling based on their criticality levels; EWCET management, involving the considering of actual task execution times; and the reconfiguration protocol used to make system adjustments in the event of resource shortages.

The main goal of the framework is to handle WCET overrun while maximising system utility to take actions so the system continues to operate smoothly. When a task with low criticality overruns its execution time, the framework's response is to postpone the execution of the task. In the case of high-criticality task overruns, the framework allows the task to continue its execution beyond the allocated WCET.

The flowchart presented in Figure 5.1 provides a visual representation of the sequence of steps and decisions involved in performing the E-ATMP process. It comprises 14 sequential steps (Step 1 . . . 14), five decision points (A, B, C, D, E), and four loops (Loop 1 . . . 4), guiding the process from the start through its continued progress.

To gain a better understanding of the flowchart, we will now describe the basic functionality of the four loops labelled Loop 1, Loop 2, Loop 3, and Loop 4, along with the decision stages labeled A, B, C, and D. The transition between different steps depends on the decisions made at the decision stages, which, in turn, dictate whether loops continue or terminate. The legend shows the colour codes for the different components of the framework: blue are actions by the short-term real-time schedulers, green are the actions by the mixed-criticality scheduler, pink denotes the actions for the EWCET handling, and orange denotes the call of the existing ATMP framework for smooth degradation.

The four loops that E-ATMP uses for the flow of its decisions are Loop 1 IDLE the state; Loop 2 the job scheduler; Loop 3 the EWCET overrun monitor; Loop 4 the schedule reconfiguration protocol. Each one of these loops is a True or False result of a decision stage except Loop 4, which is a result of the brand new schedule for the system restart produced by the ATMP reconfiguration protocol. The following text explains each loop, followed by an explanation of the decision stages that lead to the activation of each loop.

Loop 1: IDLE represents the Idle state of the processor, indicating that no job has arrived yet to be executed, and keeps the system in a $S.l = LO$ mode. The system leaves any criticality mode ($S.l > 0$) and returns to the no-criticality mode ($S.l = LO$).

Loop 2: job scheduler is responsible for managing the normal scheduling of jobs. It operates by dispatching jobs from the filtered job queue and executing them. The loop continues dispatching jobs for execution until a job exceeds its WCET estimate, at which Loop 2 is left to handle the WCET overrun or a job has been scheduled and completed its execution.

Loop 3: overrun monitor is responsible for measuring the complete execution time of a task in case of a WCET estimate overrun. The WCET overrun of a permitted high criticality job or the continuation of a postponed low criticality job at their next instances is essential for accurately measuring the EWCET.

Loop 4: schedule reconfiguration is the reconfiguration process, which is entered in case the currently finished job has a WCET estimate overrun. This loop contains an optional run of ATMP to achieve a smooth degradation of the whole system to compensate for the updated EWCET values. Since ATMP is a relatively costly reconfiguration, one can decide to run it based on the criticality values of the overrunning tasks and their

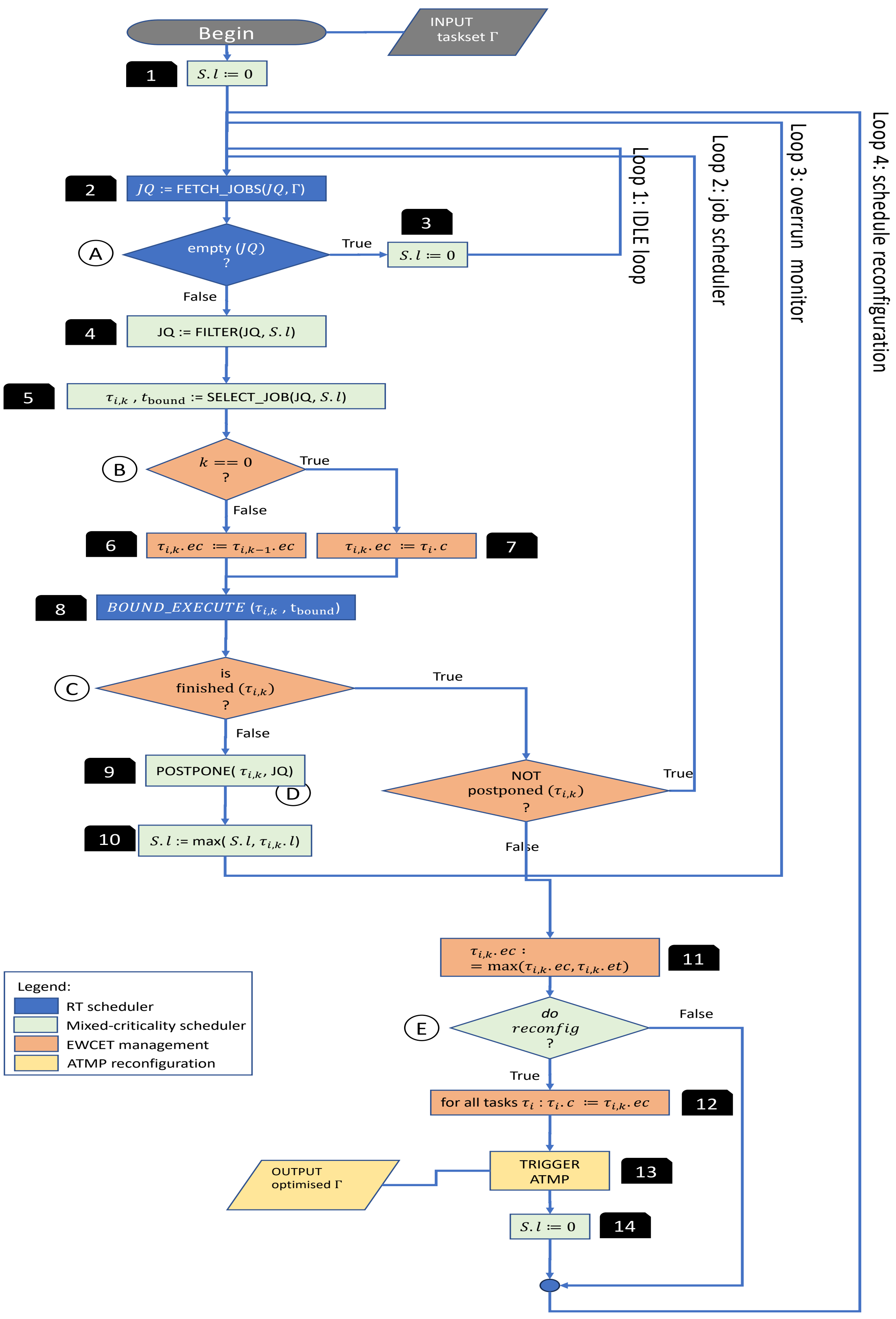

Begin
INPUT taskset Γ
1
$S.l := 0$
Loop 1: IDLE loop
Loop 2: job scheduler
Loop 3: overrun monitor
Loop 4: schedule reconfiguration
2
JQ := FETCH_JOBS(JQ, Γ)
A
empty (JQ) ?
True
3
$S.l := 0$
False
4
JQ := FILTER(JQ, $S.l$)
5
$\tau_{i,k}$, t_{bound} := SELECT_JOB(JQ, $S.l$)
B
$k == 0$?
True
False
6
$\tau_{i,k}.ec := \tau_{i,k-1}.ec$
7
$\tau_{i,k}.ec := \tau_i.c$
8
BOUND_EXECUTE ($\tau_{i,k}$, t_{bound})
C
is finished ($\tau_{i,k}$) ?
True
False
9
POSTPONE($\tau_{i,k}$, JQ)
D
NOT postponed ($\tau_{i,k}$) ?
True
False
10
$S.l$:= max($S.l$, $\tau_{i,k}.l$)
$\tau_{i,k}.ec := \max(\tau_{i,k}.ec, \tau_{i,k}.et)$
11
E
do reconfig ?
False
True
for all tasks τ_i : $\tau_i.c := \tau_{i,k}.ec$
12
TRIGGER ATMP
13
OUTPUT optimised Γ
$S.l := 0$
14
Legend:
🟦 RT scheduler
🟩 Mixed-criticality scheduler
🟧 EWCET management
🟨 ATMP reconfiguration

Figure 5.1: Flowchart of E-ATMP, using the EWCET

frequency of overruns. If ATMP is triggered, then one can be sure that future execution times up to the current EWCET values will not cause a WCET overrun, and consequently the system leaves any criticality mode ($S.l$ becomes zero again).

The system's decision-making process involves five distinct decisions. Decision A handles system status and job execution. Decision B focuses on the initial occurrence of tasks. Decision C assesses the job's WCET completion and overrun Decision D deals with job postponement and updates. Decision E considers system reconfiguration, influenced by the utility, criticality, and WCET overrun. These decisions collectively guide the E-ATMP framework for ensuring a smooth degradation. In the following, we detail each one of these stages.

Decision A: decides whether the system is currently idle, in which case the criticality mode gets reset ($S.l = 0$) and Loop 1 is taken. In case there is at least one job in the ready queue, one job $\tau_{i,k}$ gets fetched from the ready queue and executed.

Decision B: decides whether the job $\tau_{i,k}$ is the first instance of task τ_i ($k = 0$). If $\tau_{i,k}$ is the first instance, then the EWCET gets initialised with the current WCET estimate, otherwise, the EWCET value gets copied over from the previous job $\tau_{i,k-1}$ of the same task.

Decision C decides whether the job $\tau_{i,k}$ being just executed was able to finish within the given time budget $\tau_{i,k}.c$. If this was the case, it proceeds with Decision D whether $\tau_{i,k}$ was already a postponed job. Otherwise, the job gets postponed and Loop 3 is taken.

Decision D decides whether the currently finished job $\tau_{i,k}$ was postponed or not. If it was not postponed (i.e., it finished without a WCET-estimate overrun) then Loop 2 is taken. If it was postponed, the system proceeded with an EWCET update, leading to Decision E.

Decision E decides whether a system reconfiguration via ATMP should be triggered after job $\tau_{i,k}$ finished its WCET estimate overrun. This decision might not always be true, as a reconfiguration by itself is costly. Thus, the decision will be a heuristics that takes into account the criticality level of all the previously overrunning jobs, their factor by how much they overrun and how often they overrun. A concrete logic for this decision is presented in the Evaluation chapter based on the criticality of the tasks in question.

With the overall understanding of all the loops and decisions involved in E-ATMP, we now explain the individual actions performed in **Steps 1 . . . 14**

Step 1 $S.l = 0$
The input of E-ATMP is a mixed-criticality taskset Γ. The E-ATMP starts in Step 1 where the criticality mode $S.l$ gets reset to non-critical mode ($S.l = 0$). $S.l$ represents the criticality ceiling of all the jobs that have overrun their WCET estimate since the last reconfiguration (or beginning, if there was no reconfiguration so far).

Step 2 `JQ = FETCH_JOBS(`JQ`, `Γ`)`

In Step 2 the ready queue JQ gets updated by adding any new arriving jobs at the end of JQ via the function `FETCH_JOBS(`JQ`,`Γ`)`. The new arriving jobs are derived and created from the task set Γ. Afterward, in Decision A it is tested if JQ is currently empty (the system is idle), in which case E-ATMP returns to Step 2 via Loop 1.

Step 3 On the way back via Loop 1 the criticality mode gets reset in Step 3 ($S.l = 0$).

Step 4 If the ready queue JQ was not empty, then in Step 4 JQ gets updated via the function `FILTER(`JQ`, `Γ`)`, as described in Algorithm 3, to make space for postponed jobs. `FILTER(`$JQ, S.l$`)` removes ready tasks that are going to be replaced by a postponed task in JQ. For any postponed job $\tau_{i,k}$ the next job of the same task $\tau_{i,k+1}$ will be removed from JQ to allow the continuation of $\tau_{i,k}$. Algorithm 3`FILTER(`JQ`, `l`)` shows the implementation of Step 4 algorithm.

Step 5 Afterwards, Step 5 continues the mixed-criticality scheduler with the function `SELECT_JOB(`JQ`, `$S.l$`)`, given in Algorithm 4, which selects the job $\tau_{i,k}$ to be executed next, based on the following order: We only consider jobs $\tau \in JQ$ that have a criticality that is larger or equal to the current criticality mode: $\tau \leq l$. From those selected jobs it then chooses the next one based on the underlying real-time protocol.

Algorithm 4`SELECT_JOB` (JQ, l) shows the implementation of Step 4 algorithm.

Step 6 and 7 The function `SELECT_JOB()` also returns the time budget t_{bound} for this job, which is the WCET estimate of the job. Then, based on Decision B the EWCET for the selected job $\tau_{i,k}$ gets initialised either with the EWCET of the previous job $\tau_{i,k-1}$ (Step 6) in case that $k > 0$, or gets otherwise initialised with the WCET estimate $\tau_i.c$ (Step 7).

Step 8 In Step 8 the job $\tau_{i,k}$ gets executed via the function `BOUND_EXECUTE(`$\tau_{i,k}$`, `t_{bound}`)`, till it finishes, or the time budget t_{bound} runs out,

Algorithm 3: Step 4 `FILTER(`JQ`,` l`)`

```
1 This function detects postponed jobs $\tau_{i,k}$ and removes the next job of
    the same task $\tau_{i,k+1}$, to make space for the execution of $\tau_{i,k}$.
  Input  : $JQ$ ... queue structure of jobs to be scheduled
           $l$ ... System criticality mode;
  Output: $JQ$;
2 begin
3     $JQ_{tmp} \leftarrow JQ$ $t \leftarrow real\_time()$        // current absolute real-time for
       $\tau_{i,k} \in JQ_{temp}$ do
4         if isPostponedJob($\tau_{i,k}$) then
5             $JQ = JQ \setminus \tau_{i,k+1}$                  // make place for postponed
6         end
7         if $(\tau_{i,k}).d < t)$ then
8             $JQ = JQ \setminus \tau_{i,k}$                    // remove, if deadline passed
9         end
10    end
11    return $JQ$
12 end
```

Algorithm 4: Step 5 `SELECT_JOB` (JQ, l)

```
1 This function selects the next job to be scheduled. The considered jobs
    $\tau \in JQ$ have a criticality that is larger or equal to the current
    criticality mode: $\tau \leq l$. If the criticality mode $l$ is zero, then all jobs
    are considered. From those selected jobs it then chooses the next one
    based on the underlying real-time protocol, written as next_RT($JQ_{tmp}$).
  Input  : $JQ$: ready queue of jobs to be scheduled
           $l$: Current system-criticality mode
  Output: $\tau_{i,k}$, $\tau_{i,k}$.c
2 begin
3     $JQ_{tmp} = []$                                  // init empty queue for filtering
4     for $\tau_{i,k} \in JQ$ do
5         if $\tau_i.l \geq l$ then                       // only keep high criticality jobs
6             $(JQ_{tmp})$.add($\tau_{i,k}$)
7         end
8     end
9     $\tau_{i,k}$ = next_RT($JQ_{tmp}$)                       // real-time scheduler
10    $JQ = JQ \setminus \tau_{i,k}$                             // remove from ready queue
11    return $\tau_{i,k}$, $\tau_{i,k}.c$
12 end
```

whichever one comes first. After the execution of job $\tau_{i,k}$, in Decision C it is checked, whether $\tau_{i,k}$ finished, or else it was pre-empted because it consumed the time budget $\tau_{i,k}.c$. In case $\tau_{i,k}$ finished, it continues with Decision D, where it is tested if $\tau_{i,k}$ was not a postponed job (i.e., it did not overrun its WCET estimate), If it was not an overrun, then it continues via Loop 2 back to Step 2. Algorithm 5`BOUND_EXECUTE` (JQ, l) shows the implementation of Step 8 algorithm.

Algorithm 5: `Step 8 BOUND_EXECUTE` ($\tau_{i,k}$, t_{bound})

```
Input : $\tau_{i,k}$
        $t_{bound}$
begin
    Try:
        $TIMER\_COUNTDOWN(\ t_{bound}\ )$
        $executeJob(\tau_{i,k});$
        $TIMER\_STOP();$
        $setFinished(\tau_{i,k});$
    Exception $TIMER\_COUNTDOWN$:
        return
    end
end
```

Step 9 Coming back to Decision C, if $\tau_{i,k}$ was pre-empted because it consumed the time budget c, then in Step 9 the job $\tau_{i,k}$ gets marked as postponed and is put back into the ready queue JQ with its deadline extended by another period to allow continuation: $(\tau_{i,k}).d = (\tau_{i,k}).d + (\tau_{i,k}).p$. Algorithm 6`POSTPONE(`$\tau_{i,k}$`,` JQ`)` shows the implementation of Step 9 algorithm.

Algorithm 6: Step 9 `POSTPONE(`$\tau_{i,k}$`,` JQ`)`

```
Input : $\tau_{i,k}$
        $JQ$
begin
    $removeJobFromJobQueue(\tau_{i,k+1}, JQ);$
    $insertJobIntoJobQueue(\tau_{i,k}, JQ);$
end
return $\tau.i, k$
```

Step 10 Afterwards in Step 10 the criticality ceiling $S.l$ is updated with the criticality $\tau_{i,k}.l$ of job $\tau_{i,k}$, and the system continues via Loop 3 at Step 2.

Step 11 Coming back to Decision D, if $\tau_{i,k}$ is a job that was postponed before, then the system proceeds with Step 11, where EWCET of $\tau_{i,k}$ gets updated with the total execution time, including all the postponed executions.

Step 12 and 13 Afterwards, we reach Decision E, where it is decided whether the whole task set should be reconfigured via ATMP in Step 13 or not. As mentioned above in the description of Decision E, this decision can use different heuristics to find a balance between the benefit of smooth degradation and disruption due to reconfiguration. If it is chosen to call ATMP in Step 13, then before in Step 12 for all tasks their EWCET is assigned as the new WCET estimate. This way, ATMP will now be able to find a new smooth degradation where the tasks will not overrun with the EWCET values observed so far.

Step 14 Regardless of whether ATMP was triggered or not, the system continues afterward via Loop 4 in Step 2, which concludes all cases of iterations, and the system will try to schedule the next job, and so on.

5.2.2 E-ATMP ILP Problem Formulation

In this section, we shall illustrate the E-ATMP optimisation method. First, we define the decision variables, which are the individual task utilities. Second, we define the objective function, which is the maximised system utility calculated by the sum of optimised task utilities. Third, we define the problem constraints, which are resource and performance metrics constraints. Finally, we present an example of E-ATMP optimisation in action.

Optimisation Variables The decision variable is the utility of a task $\tau.u$. E-ATMP maximise the relative utility $\tau_i.u$ of each task τ_i and multiplies it by its numerical value (weight) for criticality $\tau_i.l$ and the resulting value represents the absolute utility of the optimised task $\tau_i.U$.

$$\tau.U = \tau.W \cdot \tau.u, \tau.W = \tau.l \tag{5.2}$$

Task criticality *tau.l* The designated criticality of the task, either LO or HI. **Task weight** *tau.W* $\tau.W$ maps a numerical value to the task criticality level $\tau_i.l$ as follows: when $\tau.l$ equals 2, it represents $\tau_i.l$ as HI, and when W equals 1, it corresponds to $\tau_i.l$ being LO.

Objective Function The goal of E-ATMP optimisation is to maximise the overall system utility SU. This overall system utility is calculated as the sum of the absolute utility levels for individual tasks $\tau.U$ for all tasks in the supplied task set Γ.

$$SU_{opt} = \sum_{\tau_i \in \Gamma} \tau_i.U \tag{5.3}$$

SU_{opt} optimised system utility that is the sum of optimised utilities by all tasks.

Resource Constraint The key concept of E-ATMP is that it calibrates the overall throughput within safe bounds according the a certain EWCET update (EWCET overrun). The resource constraint is the sum of each task load $\tau.$ in the task set. The maximum load of a task is the EWCET $\tau_i.ec$ divided by the primary period p_{prim,τ_i}.

$$SR = \sum_{\tau_i \in \Gamma} load_{\tau_i} \leq 1.0 \tag{5.4}$$

where the load of each task $load_{\tau_i}$ is calculated by:

$$maxload_{\tau_i} = \frac{\tau_i.ec}{\tau_i.uf.p_{prim}} \tag{5.5}$$

The *resource constraints* are used to limit the workload at each of the available cores $cr_i \in Cores$. The maximum workload a core cr can take is its computing capacity $Cap(cr)$:

$$\sum_{\tau_i \in \Gamma} load_{\tau_i} \leq \sum_{cr \in Cores} Cap(cr) \tag{5.6}$$

Period Constraints This constraint ensures that the optimised period $\tau.p$ is constrained to be less than the $\tau.uf.p_{prim}$, and greater than $\tau.uf.p_{tol}$:

$$\tau.uf.p_{tol} \leq \tau.p \leq \tau.uf.p_{prim} \tag{5.7}$$

5.3 Motivation: The Reconfiguration of Mixed-Criticality Scheduling

In this section, we show how a mixed-criticality task set would be scheduled in a system with/without the possibility of measuring the WCET and reconfiguring the system accordingly at runtime. This includes the case where the actual execution time is greater than the job's execution time budget, or WCET overrun.

Using the task set example presented in Tables and 5.1 and 5.2, we understand what the impact of WCET overrun in the context of mixed-criticality scheduling on uni-core system using the AMC protocol, and the potential benefit of applying the EWCET reconfiguration model in case of WCET overrun.

In this example, we consider two tasks, Task τ_A and Task τ_B, each with distinct characteristics. Task τ_A has a primary-period p_{prim} of 2.0 ms, a tolerance-period p_{tol} of 3.0 ms, a tolerance-utility u_{tol} of 0.3, an optimistic WCET estimate $\tau_A.c$ of 0.5 ms, and a criticality level $\tau_A.l$=1 (LO). On the other hand, Task τ_B has a primary-period p_{prim} of 5.0 ms, a tolerance-period p_{tol} of 6.0 ms, a tolerance-utility u_{tol} of 0.6, an optimistic WCET estimate $\tau_B.c$ of 0.5 ms, and a criticality level $\tau_B.l$=2 (HI).

As a concrete scheduling protocol, we use the AMC scheduling scheme [123]. First, we understand the AMC and its system criticality modes using an example that shows how AMC handles WCET overruns 5.3.1. Then, we apply the knowledge of an online update for the empiric execution EWCET, presented in the previous section, and reconfigure the schedule accordingly. Based on the example, we can motivate that our reconfiguration at runtime based on the EWCET provides a smoother service degradation in case of resource shortages due to WCET estimate overruns. An extended evaluation of the EWCET and E-ATMP framework is presented under the Experimental Evaluation chapter 7.

5.3.1 Mixed-Criticality Scheduling without Reconfiguration

The AMC scheduling scheme defines two system runtime modes: low-criticality mode and high-criticality mode. HI criticality tasks are assigned two WCET estimates: the optimistic WCET and the pessimistic WCET. In contrast, low-criticality tasks have only an optimistic WCET estimate. The low-criticality mode represents the normal behaviour of the system where no resource shortage such as WCET overrun. In this mode, all task executions are assumed to be bounded by the optimistic WCET estimates.

The high-criticality mode represents the system state after response to a resource shortage due to WCET overrun. In the HI criticality mode, HI criticality tasks are guaranteed and the execution times are bounded by the pessimistic WCET, which is greater than the optimistic WCET. However, LO criticality tasks are dropped until the system experiences an idle state (no task instance to execute). It is worth noting that the return of AMC to LO criticality during an idle instant is acceptable as defined by Santy et al. [77]. The transition from the LO criticality mode to the high-criticality mode occurs when any low or high-criticality task exceeds its optimistic WCET budget without completion.

Conversely, the transition from the high-criticality mode to the LO criticality mode occurs when no job arrives to execute or when the processor is idle. AMC guarantees that all deadlines are met in the LO criticality mode, while only HI criticality tasks are guaranteed in the HI criticality mode.

In the following example, using the task set presented in table 5.1, we understand how AMC schedules handle WCET overrun by dropping LO criticality jobs, using a mixed-criticality task set of two tasks, one HI criticality task, and one low-criticality task. Table 5.1 shows the example task set, which includes a

Task ID	Period Deadline [ms]	WCET Estimate [ms]	Real WCET [ms]	Criticality
A	2.0	1.0	2.0	1 (= LO)
B	5.0	1.0	2.0	2 (= HI)

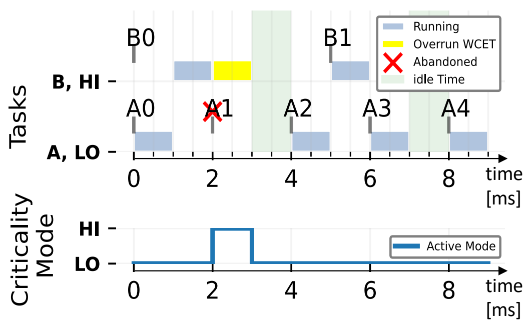
Table 5.1: Example: Mixed Criticality Task Set

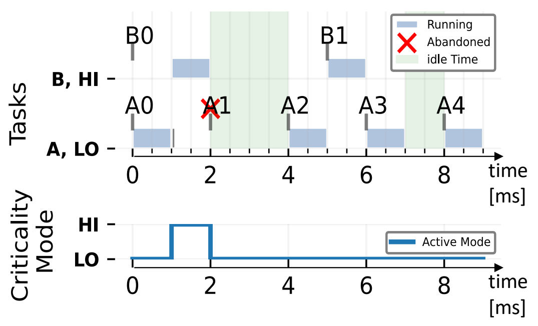
high-criticality task `B` and a low-criticality task `A`. In this example, we see that under the AMC scheduling protocol, whenever a high LO criticality job of a task, exceeds its execution time budget without completion, AMC activates the HI criticality mode, and removes all LO criticality jobs as long as HI criticality mode is activated, and LO criticality jobs are rescheduled again if an idle instant occurs.

While this approach ensures the schedulability and safety of the HI criticality jobs, it can result in inefficient utilisation of available resources, including the CPU and memory. Additionally, this removal of jobs can cause disruptive degradation in the services provided by the mixed-criticality system that AMC serves.

Table 5.1 provides information about our example two tasks, denoted as Task `A` and Task `B`. Each row represents a task and contains the following columns:

- Task ID: It identifies the task $\tau.id$ labelled as `A` and `B`.
- Period/Deadline: represents the time interval between consecutive instances of a task $\tau.p$. Task `A` period is 2.0 units of time, indicating that it repeats every 2.0 ms. task `B` period is 5.0 units of time.
- WCET Estimate: It represents the task $\tau.p$ estimated WCET, which in general might be an optimistic or safe WCET bound $\tau.c$. For both tasks, the WCET estimate is 1.0, which in our case optimistic WCET estimate, so we can show the scenarios of WCET estimate overruns.





a) High-criticality job B0 overrun by 1 ms   b) Low-criticality job A0 attempt overrun but postponed

Figure 5.2: Example of AMC scheduling, where a task exceeds its WCET estimate

- Real WCET: It represents the real WCET under worst-case conditions. The real WCET is typically not known, only the WCET estimates are known. For both tasks, the real WCET is assumed to be 2.0 in each overrun case.

- Criticality: It denotes the criticality level of a task $\tau.l$, indicating its importance or priority. For task `A`, the criticality level is LO (LO, numeric value 1), and for task `B`, it is HI(HI, numeric value 2).

Figure 5.2 shows the AMC schedule for the tasks presented above, along with a run-time monitor that handles the optimistic WCET overrun problem. Additionally, the transitions between run-time modes in AMC are recorded within each schedule and two cases of WCET overrun: the first case involves the overrun of a HI criticality task, while the second case involves the overrun of a LO criticality task. In Figure 5.2.a, we illustrate the first case, and in Figure 5.2.b, we illustrate the second case.

In Figure 5.2.a, we observe that the job *B0* exceeds its optimistic WCET starting from time 2, which is highlighted in yellow. To address this situation, the AMC activates the HI criticality mode at the start of the overrun, as seen in the bottom half of Figure 5.2.a. Consequently, any LO criticality job that was scheduled for execution, such as job *A1*, is dropped.

To handle interference during the overrun of *B0*, AMC activates the high mode. Once the HI mode is activated, AMC abandons new LO criticality job releases. Specifically, due to the arrival of *A1* at the start of the *B0* overrun, AMC abandons the execution of *A1*. Despite *A1* having a higher priority than

B0, it is prevented from utilising the underlying processor due to its lower criticality level. The system returns to its normal behaviour with the activation of the $S.l = 0$ at time 3.

In Figure 5.2.b, the low-criticality job *A0* exceeds its WCET and fails to complete at time 2, entering the overrun state. To handle the overrun, AMC takes the following actions: it activates the HI criticality mode at time 1, aborts *A1* at time 1, and deactivates the $S.l = HI$ at time 2. Since the HI criticality mode is activated, AMC ignores the arrival of *A1* because the HI criticality task *B0* is completed within its estimated WCET, resulting in the system being idle. Subsequently, AMC deactivates the HI mode, but *A0* is aborted, and *A1* is abandoned. The outcome of the *A0* overrun in this example is the dropping of *A1* to ensure the safety and schedulability of task *B* jobs.

From the two cases presented above, the overrun of HI criticality job (*B0*) and a LO criticality job (*A0*), it is evident that AMC takes an over-pessimistic approach in addressing WCET overrun. It achieves safety by dropping jobs with low-criticality and ensure the schedulability of high-criticality jobs. This approach cannot provide smooth degradation for the system's quality of service in case we consider a larger task set.

In the next section, we demonstrate how E-ATMP can provide a more smooth service degradation, by optimising the schedule in case of a WCET estimate overrun. E-ATMP generally aims to minimise the number of dropped tasks of lower criticality, by allowing to degrade of tasks based on empiric WCET.

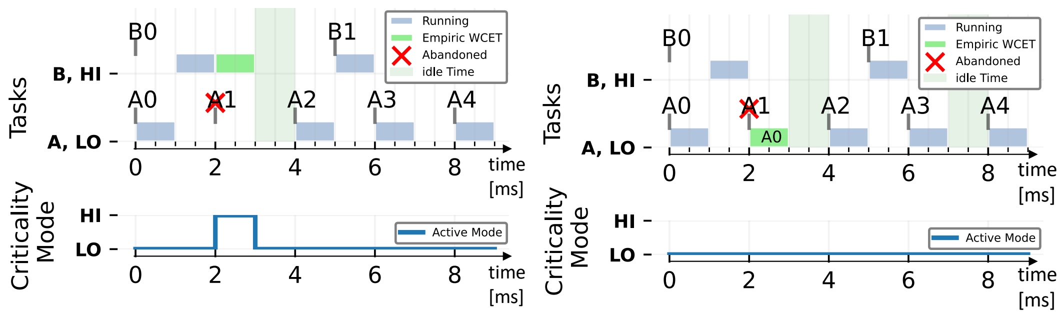
5.3.2 Mixed-Criticality Scheduling with E-ATMP Reconfiguration

In this section, we show how the E-ATMP protocol handles the WCET overrun using the same scenario handled by AMC in the previous subsection 5.3.1.

Task ID	Primary Period [ms]	Tolerance Period [ms]	WCET [ms]	Relative Utility	Criticality
A	2.0	3.0	0.3	0.5	1 (LO)
B	5.0	6.0	0.6	0.5	2 (HI)

Table 5.2: Example: Tolerance-based Mixed Criticality Task Set

As shown in Figure 5.1.a, $\tau_{B,0}.ec$ is twice the task's WCET bound $\tau_{B,0}.c$, which leads E-ATMP to activate the HI mode ($S.l = 2$) and stops job $\tau_{A,0}$ at time 2 ms, trigger the ATMP optimiser for a new schedule with the new $\tau_{B,k}.ec$ execution time. The optimised schedule eliminates the need to abandon



a) High-criticality job B0 overrun by 1 ms   b) Low-criticality job A0 attempt overrun but postponed

Figure 5.3: Example of mixed-criticality scheduling, where a task exceeds its WCET estimate

all jobs $\tau_{A,k}$ of LO criticality task τ_A while ensuring the safety of all jobs of HI criticality task τ_B with new updated WCET estimate $\tau_B.c$ (from the latest EWCET value). At time 6, E-ATMP allowed Task $\tau_{B,1}$ to overrun again without the need to activate the HI criticality mode. This demonstrates the flexibility of the E-ATMP protocol in handling overrun situations while maintaining efficient operation. Additionally, during the execution, Task $\tau_{A,3}$ safely interrupts $\tau_{B,1}$, executes, and once completed, $\tau_{B,1}$ resumes its execution. Figure 5.3.b shows that $\tau_{A,0}.ec$ completed without completion and attempts to overrun at time 1.0. E-ATMP aborts $\tau_{A,0}$ and allows $\tau_{A=B,0}$ to execute normally, but $\tau_{A,0}$ is postponed and replaces $\tau_{A,1}$ execution while keeping the system in LO criticality level.

5.4 Chapter Summary

This chapter presented the methodology for the development of the Empiric execution time EWCET and E-ATMP framework. The EWCET is initialised with the available optimistic WCET estimate updated during runtime each time a WCET overrun by a task happens, and the E-ATMP framework exploits the enabled information by EWCET. The EWCET and E-ATMP methodologies are explained in Section 5.2, and an example of WCET overrun is presented in Section 5.3.

Chapter 6

Implementation of ATMP-CA, LBP-CA, EWCET, and E-ATMP

This chapter describes the implementation of the protocols evaluated in this thesis, both existing and novel. The implementation for Mixed-criticality and Criticality Arithmetic classes is presented first. Following that, implementation of Bailout-based protocols, BP, LBP, and LBP-CA classes. Finally, it presents the Empiric EWCET and E-ATMP framework implementation. The organisation of this chapter is as follows: Section 6.1 presents the implementation for Mixed-criticality and Criticality Arithmetic classes. Section 6.2 presents the implementation for Bailout-based protocols, BP, LBP, and LBP-CA classes. Section 6.3 presents the implementation for the BP framework. Section 6.4 concludes the chapter.

The simulator is developed using Python programming language. Figure 6.1 shows the main elements of the developed simulator for this thesis. These elements are:

- Configure Tasks: here we define the number of tasks and the timing requirements for each task.
- Generate Jobs: here we generate instances for each task, where the number of these instances per task is either based on the hyper-period - Least Common Multiple (LCM) - of the tasks periods in the task set or based on the largest period.
- Real-Time Configuration: here we define the priority assignment and schedulability test. Implemented Priority assignment algorithms, are: Rate Monotonic (RM), Deadline Monotonic (DM), Earliest Deadline First (EDF), and Least Laxity First (LLF). Schedulability analyses and testing

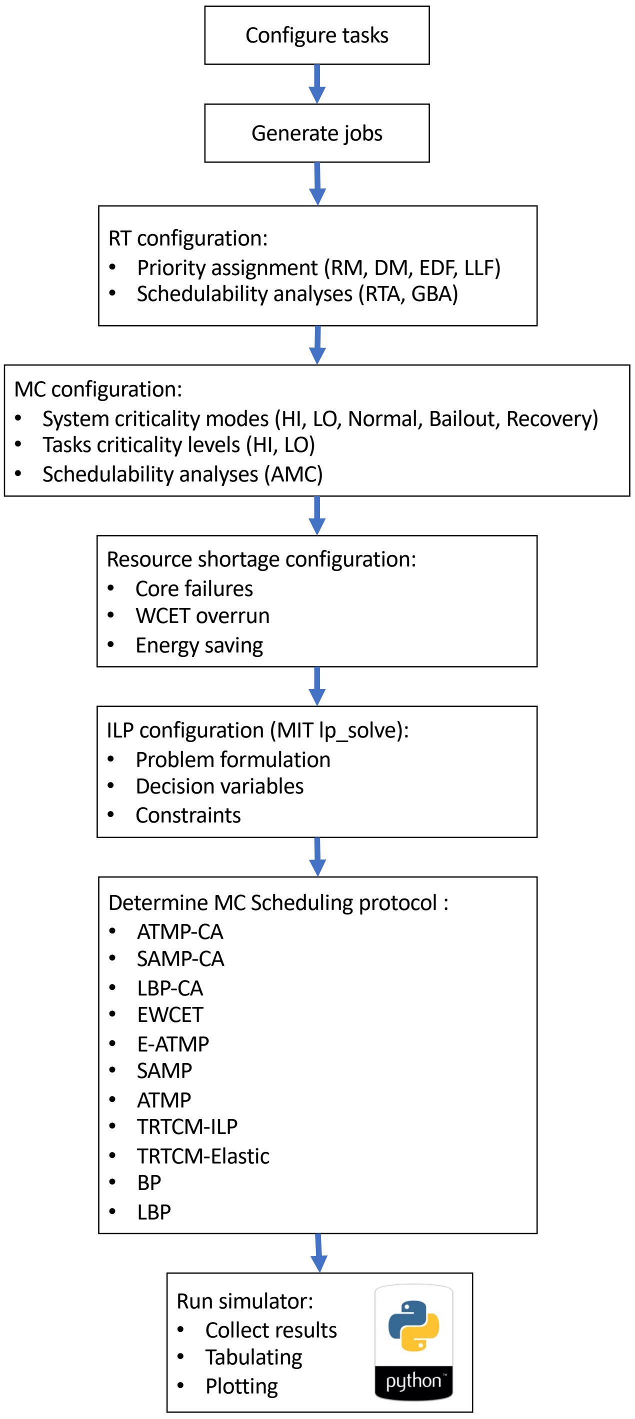


Figure 6.1: Developed Simulator for the Thesis

are based on the Response Time Analysis (RTA) and Guaranteed Bound Analysis.

- Mixed-Criticality (MC) Configuration: Define system criticality modes that are activated in response to resource shortages to trigger adaptation facilities. These modes can be either High (HI) and Low (LO) criticality modes for classic mixed-criticality scheduling, or Normal, Bailout, and Recovery in case of Bailout scheduler is used. Tasks criticality levels are also identified in this stage which can be either LO or HI criticality tasks, and the schedulability analysis used is the AMC .

- Resource Shortage Configuration: here er define the desired fault model if needed for the running experiment, which can be either core failures, WCET overruns, or energy-saving modes.

- ILP Configuration (MIT lp_solve): lp_solve is a free - under the GNU lesser general public license - linear programming solver software. lp_solve allows describing the linear problem in a text file, including the goal function, decision variables, and the problem constraints.

- Determine MC Scheduling Protocol: this defines the mixed-criticality protocol that provision the execution of services during normal and critical systems modes.

- Run Simulator: After the previous stages for configuration, we execute the simulation and collect data on task executions and system behaviour. Then, we tabulate the collected results and visualize performance metrics and system behaviour.

6.1 Implementation of Criticality Arithmetic-based Protocols

Criticality Arithmetic is designed and implemented to support survivability of Mixed-criticality schedulers in both, multi-core and uni-core platforms. The multi-core platforms consist of a central class that implements basic functionality shared by existing protocols in the literature, and developed ones in this thesis. The central class is `MultiCoreScheduler`. This class defines the attributes and methods for mixed-criticality protocols. Two derived classes and four multi-level inheritance classes are created from the `MultiCoreScheduler` class. The

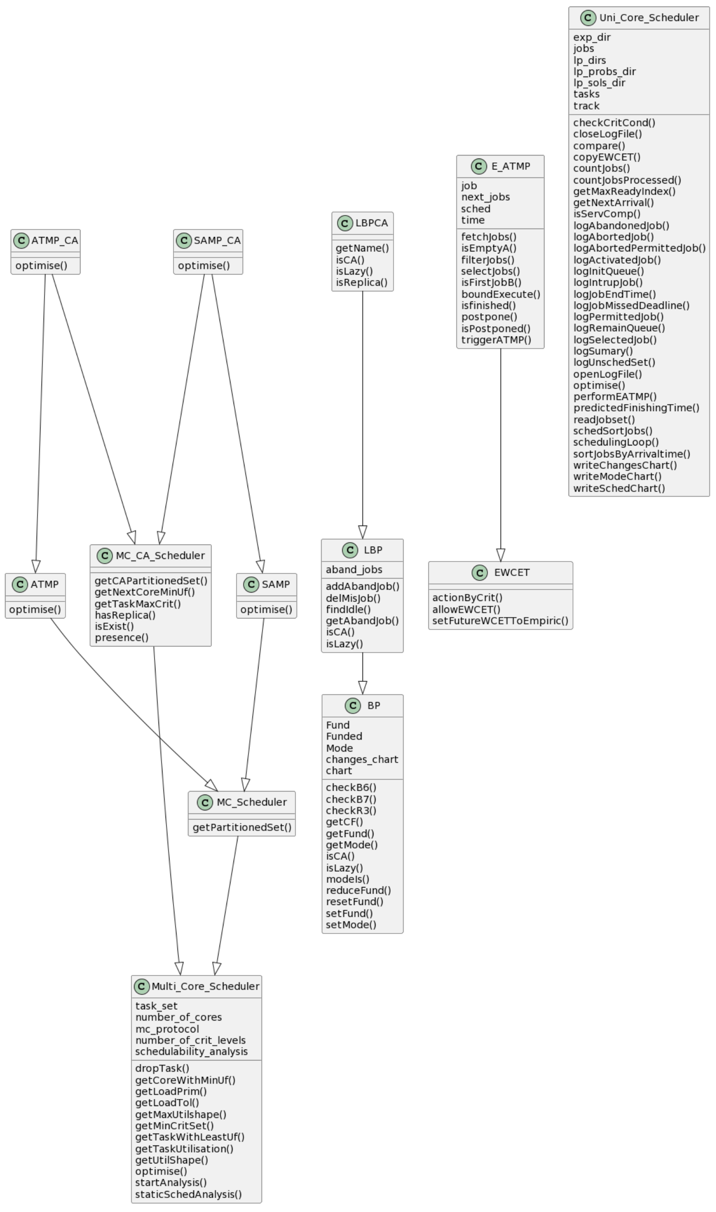


Figure 6.2: Developed Mixed-criticality Classes for the Thesis

two direct derived classes are `MC_Scheduler` and the `MC_CA_Scheduler`. Two of the four multi-level inheritance classes are CA-aware classes (SAMP-CA and ATMP-CA), and the other two classes are CA-agnostic (SAMP and ATMP).

The attributes in `MultiCoreScheduler` class are: the experiment identifier `experiment_id` and the necessary directories paths `experiment_dir`, the task set `task_set` to be allocated on available cores, the number of the available cores in the system `number_of_cores`, the mixed-criticality protocol `mc_protocol` chosen to allocated the provided task set, criticality levels `number_of_crit_levels`. In case the criticality level equals one, then `MultiCoreScheduler` behaves as a mixed-criticality-agnostic scheduler and operates as a normal Real-time multi-core scheduler.

The methods implemented in this class are: `dropTask`: to drop a task from the scheduling in case of resource shortage, `getCoreWithMinUf` retrieve core with the minimum utilization factor, `getLoadPrim` calculates get the task load using the primary period, `getLoadTol` calculates get the task load using the tolerance period, `getMinCritSet` gets the set of tasks with minimum criticality, `getTaskUtilisation` returns task load, `getTaskWithLeastUtilisation` returns a task with minimum WCET and maximum period in the task set, `getUtilShape` Method to get the utilization shape, `getMaxUtilshape` determine the task's largest tolerance range and highest tolerance utility, `optimise` This method is overridden by the inherited multi-level inheritance classes, `staticSchedAnalysis` This method runs certain schedulability tests to check the feasibility of the task set. `startAnalysis` triggers the `staticSchedAnalysis` method.

In the following we briefly explain the classes inherited from the `MultiCoreScheduler` class:

`MC_Scheduler` and `MC_CA_Scheduler` Allocation of tasks is based on awareness about criticality arithmetic. As seen in 6.2, class `MC_Scheduler` implements the `getPartitionedSet` method that performs the allocation process of mapping tasks to cores but allows tasks implement a service to be allocated on the same core. class `MC_CA_Scheduler` implements the method `getCAPartitionedSet` that performs the allocation process of tasks to cores and disallows CA-ware tasks of the same service to be allocated on the same core and ensures that each replica is allocated on a separate core.

In case of a shortage in computing resources, for example, core-failure, one of the four multi-level inheritance protocols texttt SAMP , ATMP , texttt SAMP-CA and `ATMP-CA` can be used to reconfigure the system. Each protocol provides a certain level form of graceful degradation.

SAMP and ATMP These classes integrate mixed-criticality protocols from the existing literature and are implemented to evaluate the applicability of criticality-arithmetic [43, 94].

SAMP is built upon the AMC mixed-criticality protocol and manifests the behavior of most mixed-criticality scheduling protocols. It extends the AMC protocol to support multi-core scheduling. The graceful degradation of SAMP is determined offline by limiting the impact of resource shortage through the dropping of all `LO-criticality` tasks.

ATMP class is based on AMC and the TRTCM model. It enables graceful degradation by using the TRTCM utility function to maximize task utility to maximize the system utility in case of resource shortage. It extends the AMC protocol to multi-core scheduling. ATMP reconfigure the system whenever a resource shortage occurs, but tests the schedulability of the new configuration using AMC response time analysis, before the deployment of the new configuration.

`SAMP-CA and ATMP-CA` These novel classes are developed during the research in the thesis. They manifest the modularity of criticality arithmetic and its ability to integrate into existing multi-core fixed-priority mixed-criticality protocols in the literature.

6.2 Implementation of Bailout-based Protocols

Mixed-criticality Bailout-based scheduling [46, 47, 98] differs from traditional Mixed-criticality models by that it defines three criticality modes - `normal`, `bailout`, `recovery` - to enable the execution of mixed-criticality tasks: Normal, Bailout, and Recovery. Compared to existing mixed-criticality models which use two criticality modes: `HI` and `LO-criticality` modes, the three modes model enables finer graceful degradation.

The central class is `BP`. This class defines the attributes and methods for Bailout-based mixed-criticality protocols. One derived class and one multi-level inheritance class are created from the `BP` class. The directly derived class is `LBP` and the novel multi-level inheritance CA-aware class implemented for this thesis, is the `LBP_CA` class.

The attributes in `BP` class are: the execution bank of funds `fund` and the boolean flag for already funded tasks `funded`, the list of bailout-modes `mode`, scheduling `changes_chart` which logs the traverse between bailout-modes during runtime.

The methods implemented in this class are: `checkB6` checks the necessity to activate the recovery mode, `checkB7` checks the ability to return to `normal` mode from `bailout` mode, `checkR3` checks the ability to return to `normal` mode from `recovery` mode, `getFund` reads the current available amount of funds, `isCA` checks Criticality Arithmetic awareness, it returns true if the object is an LBP-CA object or false otherwise. `isLazy` check if it survives jobs released during `bailout` or `recovery` modes. `modeIs` returns active mode. `reduceFund` deduct execution time budget from funds to HI-criticality job. `resetFund` sets bailout fund to zero. `setMode` applies mode transition from one mode to another.

In case of a shortage in computing resources, for example, WCET -overrun, one of the criticality-arithmetic-agnostic classes BP and LBP but they lack the awareness about criticality-arithmetic as LBP-CA. Each class provides a certain level form of graceful degradation.

In the following we briefly explain the classes inherited from the `BP` class:

LBP implements the Lazy Bailout protocol. `LO-criticality` jobs released during `bailout` and `recovery` are dropped by `BP`, where in `LBP-CA` are inserted in low-priority queue for possible execution in the future [98].

LBP_CA uses criticality-arithmetic with LBP for quick return to `normal` mode. CA relaxes the condition for entering the recovery mode, which in turn allows the system to activate the normal mode earlier than reference classes, `BP` and `LBP`.

6.3 Implementation od E-ATMP Framework

Mixed-criticality Empiric Worst-case Execution Time (EWCET), is a novel model for adapting to the WCET overrun by real-time tasks. The methods performed by the EWCET are: `actionByCrit` specifies the course of actions according to the criticality of the job about to overrun its WCET or most recent EWCET . `allow EWCET` prepares HI-criticality jobs to measure new overrun. Only tasks with higher criticality and higher priority may interrupt the HI-criticality job in question. `setFutureWCETToEmpiric`

The model is based on measuring the WCET overrun during runtime and updating the system accordingly. The EWCET model works in conjunction with a mixed-criticality protocol. In this thesis, EWCET is integrated into the

ATMP protocol. The following explains the methods performed by the E-ATMP framework class.

`fetchJobs` updates the jobs queue by the incoming ready tasks. `isEmptyA` decides whether the system is currently idle or not. `filterJobs` removes ready tasks that are going to be replaced by a postponed task in the jobs queue. `selectJobs` selects the next job to be executed `isFirstJobB` decides whether a job is the first instance of a task in question. `boundExecute` monitor and control jobs EWCET . `isfinished` decides whether a job completed its computation or not. `postpone` `isPostponed` set the job to execute at the next instance of the same task.

6.4 Chapter summary

This chapter presented the implementation of Mixed-criticality and Criticality Arithmetic protocols in this thesis, including existing and novel ones. Followed by the implementation of Bailout-based protocols, BP, LBP, and LBP-CA classes. Finally, it presented the implementation of the Empiric EWCET and E-ATMP framework. The implementation of Criticality Arithmetic is presented in Section 6.1, and the implementation of Bailout-based protocols is presented in Section 6.2. The implementation of Empiric EWCET and E-ATMP framework in Section 6.3.

Chapter 7

Experimental Evaluation

This chapter presents the evaluation of the existing and novel contributions presented in this thesis. First, an evaluation of ATMP-CA and LBP-CA protocols in Section 7.1. Second, an evaluation of SAMP and ATMP protocols in Section 7.2. Third, an evaluation of TRTCM-ILP and TRTCM-Elastic protocols in Section 7.3 Fourth, an evaluation of the E-ATMP framework for measuring and handling WCET overrun.

The organization of this chapter is as follows: Section 7.1 presents ATMP-CA and LBP-CA evaluation. Section 7.2 presents SAMP and ATMP protocols evaluation. Section 7.3 presents TRTCM-ILP and TRTCM-Elastic evaluation. Section 7.4 presents E-ATMP evaluation. Section 7.6 concludes the chapter.

The first evaluation evaluates criticality arithmetic protocols (SAMP-CA, and ATMP-CA), and criticality arithmetic agnostic protocols (SAMP and ATMP) on a multi-core platform. The criticality arithmetic aware protocols SAMP-CA and ATMP-CA achieve smooth degradation by dropping services based on their criticality, utility, adaptation, and the state of low criticality replicas. The evaluation also assesses criticality arithmetic protocol (LBP-CA), and criticality arithmetic agnostic protocols BP and LBP. The experimental setup 7.1.1, results and analysis 7.1.2, and discussion 7.1.3 are presented in Section 7.1.

The second evaluation compares achieved system utility between ATMP and SAMP protocols for service allocation on a multi-core system. The ATMP achieves smooth degradation by dropping services based on their criticality, utility, and adaptation. The SAMP achieves smooth degradation by dropping services based on criticality only. The experimental setup 7.2.1, results and analysis 7.2.2, and discussion 7.2.3 are presented in Section 7.2.

The third evaluation assesses smooth degradation on the uni-core system between two variants of the TRTCM model: TRTCM-ILP and TRTCM-Elastic .

TRTCM-ILP optimises services for smooth degradation by formulating an `ILP` problem and solving it using LP solver to maximise system utility and subject to each service utility function and resource load constraints. TRTCM-Elastic optimises services by applying a modification for spring compression and decompression functions, to compress and decompress services load to satisfy desired resource load constraint. The experimental setup 7.3.1, results and analysis 7.3.2, and discussion 7.3.3 are presented in Section 7.3.

The fourth evaluation assesses smooth degradation on the uni-core system between EWCET aware and `EWCET -agnostic` protocols. The EWCET aware framework is E-ATMP and the EWCET agnostic is the AMC scheme. The E-ATMP measures new WCET for services once they overrun their execution time bounds during the system mission, and provides smooth degradation by reducing the number of times the system enters high criticality mode, hence, avoiding dripping all low criticality services in case of resource shortages The AMC uses the existing WCET without new measurements during the system mission and provides a low level of smooth degradation by dropping all low-criticality services in case of resource shortages. The experimental setup 7.4.1, results and analysis 7.4.2, and discussion 7.4.3 are presented in Section 7.4.

Evaluaion Metrics: Evaluating smooth transitions between protocols studied in this thesis to answer RQ3 has been accomplished using: Offline and Online Feasibility Analysis, Number of successful services per protocol, Number of failed services per protocol, Number of compromised services per protocol, Number of postponed services per protocol, Ratio of achieved system/service utility between the protocols.

Offline and Online Feasibility Analysis In the context of real-time systems, offline and online feasibility analysis is applied to a set of services and a given number of cores/processors to ensure the correct execution of these services in terms of meeting their deadlines. Offline analysis is conducted before the system runs, while online analysis is conducted during the run. The offline analysis ensures that, before comparing and evaluating the performance of certain protocols, no service from the given set interferes with another service meeting its deadline during normal execution on the specified number of cores/processors. The online analysis tests the system's recovery from failures, whether core failure or WCET overrun.

Number of successful services per protocol A service is considered successful in two cases within the context of this thesis: either it is mapped to be executed on a certain processor, or it completes its task/s within its worst-case execution time (WCET) estimate and before its deadline. This metric is essential for evaluating any MC protocol or comparing it aginst another MC protocol. . In ATMP-CA, SAMP-CA, ATMP, and SAMP, a successfull task is a task that has been assigned to one of the existing cores. In LBP-CA, and E-ATMP a successfull task is a task that has completed its execution within its WCET estimate.

Number of failed services per protocol A service is considered failed in two cases: either its mapping has failed and it cannot be executed on a processor, or it missed its deadline due to being dropped or postponed to avoid its interference on higher criticality tasks. This metric is essential for evaluating any MC protocol or comparing it aginst another MC protocol. In ATMP-CA, SAMP-CA, ATMP, and SAMP, a failed task is defined as a task that has failed to be allocated to one of the existing cores. In LBP-CA, and E-ATMP a failed task is a task that has missed its deadline before completing its execution.

Number of compromised services per protocol A service is considered compromised when it executes below the optimal performance but remains at an acceptable level of operation due to resource shortages, which may result from permanent or transient faults. I used the TRTCM model and ILP solver to maximise the utility of each service, thereby maximising the overall system utility. In the case of permanent faults, such as core failure, the ILP solver is triggered to reconfigure the system for allocating a number of services on the remaining cores or processors. In the case of transient faults, such as WCET estimate overrun, the LP solver is triggered to generate an optimised schedule that acknowledges recent overruns information.

Number of postponed services per protocol A service is considered to be postponed to continue its execution in future when its of low criticality and has cosumed its WCET budget without completion. This metirc will only be found under E-ATMP chapter, Chapter 5 and its evaluation section in the Evaluation chapter, Section 7.4 in Chapter 7. A service is considered to be deferred to continue its execution in the future when it is of low criticality and has consumed its WCET budget without completion. If a postponed service completes its execution in the future, then it is considered successful; otherwise, it is considered failed. This postponed-service metric will only be found under the E-ATMP Framework in 5, and in its evaluation section in the 7 7.4

Ratio of achieved system/service utility between the protocols The sum of maximised utility per service defines the overall utility of the system. Failed services, whether due to core allocation or deadline misses, have zero utility. Only successful and compromised services contribute to the overall system utility. Therefore, after designing, running, and collecting the results of an experiment in this thesis, the achieved overall system utility from each protocol is used to compare which protocol provided smoother degradation than the other. For example, 7.6 shows ratio od absoluate utility between ATMP and SAMP protocol.

7.1 Experimental Evaluation of ATMP-CA and LBP-CA

7.1.1 Experimental Setup

In the following, we describe the setup of criticality arithmetic for mixed-criticality systems in multi-core and uni-core systems. This includes: system configuration, reference protocols, and the generation of tasks implement services..

System and Resource-shortage Configurations: I simulated a multi-core system with 6 cores for the evaluation of ATMP-CA and a uni-core system for the evaluation of LBP-CA. In the ATMP-CA experiment setup, we failed one core for each run, starting from 6 cores down to 2 cores. For the LBP-CA experiment setup, we induce WCET overrun at the critical instant of a HI criticality task. The reference protocols for comparison in the ATMP-CA evaluation are SAMP, ATMP, and SAMP-CA, whereas reference protocols for comparison in the LBP-CA evaluation are: BP and LBP protocols. As such, we simulated the resulting overall system utility for different cases of resource shortages caused by core failure and WCET overrun, on multi and uni-core mixed-criticality systems.

Reference Protocols: The reference protocols for the ATMP-CA evaluation, ATMP, and SAMP, were introduced in [43], but the SAMP-CA is developed for this evaluation. In essence, ATMP is similar to ATMP-CA in the sense that it also performs utility optimisation, but its core allocation and ILP constraints for utility maximisation do not take into account any of the services using criticality arithmetic. So, the comparison of ATMP-CA and ATMP shows the potential benefit of supporting any knowledge about criticality arithmetic in the scheduler. The other reference scheduler, SAMP, is generally less capable than ATMP and is only included for further reference. I developed SAMP-CA, which is basically the simple SAMP protocol, but using the new Algorithm 1 for core allocation, which also uses knowledge about criticality arithmetic. As such,SAMP-CA might perform better for systems with criticality arithmetic than SAMP, but it is not supposed to be able to compete with the utility optimisation performed by ATMP or ATMP-CA.

The reference protocols for the LBP-CA evaluation, BP and LBP, were introduced in [98]. LBP can schedule more services than BP, rather than dropping low criticality tasks released during the high criticality modes, bailout and recovery modes, are appended to another queue for future execution at idle instants where the system returns to the normal criticality mode.

Services/Tasks Generation A task set generated with random parameters for WCET $\tau.c$ and the utility-function $\tau.uf$. For the ATMP-CA evaluation, each task deadline is implicit $\tau.d$, where the task's period equals its deadline $\tau.d = \tau.p$. Note that at the start of the system, the task's period $\tau.p$ is equal to its primary period $\tau.uf.p_{prim}$, but gets reconfigured whenever the TRTCM optimisation function is triggered due to core-failure. For the LBP-CA evaluation, task deadlines are constrained, where the task deadline is less than or equal to their periods $\tau.d \leq \tau.p$. The criticality of a task $\tau_i.l$ or service $S_i.l$ is either high or low, which corresponds to a numeric value of either 2.0 or 1.0 respectively.

For the ATMP-CA evaluation, the task generation was constrained such that it includes two normal high criticality services, *S1* and *S2*, two high criticality services that use criticality arithmetic (*S3* AND *S4*), and a few other low services (*S5*, *S6*, *S7*,*S8*). The whole structure of this task set is shown in Table 7.1. As shown in the table, the tasks *T1* and *T2*, which implement the high criticality high services S1, S2, have the same criticality as the service itself. However, the high services S3 and S4, which use criticality arithmetic, are both implemented by two redundant tasks $T3_a$, $T3_b$ respectively $T4_a$, $T4_b$, which all have low criticality. For the LBP-CA evaluation, the generated task set was constrained such that it includes one three high criticality service, S1, one high criticality service that uses criticality arithmetic, (S2), implemented by tasks $T2_a$ and $T2_b$. The structure of this task set is shown in Table 7.2. services S1, S3, and S4 have the same criticality as the service itself. Services S2 is implemented by two low-criticality tasks $T2_a$, $T2_b$.

The following presents the results and analysis for ATMP-CA and LBP-CA

Service:	*name*	S1	S2	S3	S4	S5	S6	S7	S8
	crit.	high	high	high	high	low	low	low	low
Task:	*name*	T1	T2	$T3_a$, $T3_b$	$T4_a$, $T4_b$	T5	T6	T7	T8
	crit.	high	high	low	low	low	low	low	LO
	id.	A	B	C D	E F	G	H	I	J

Task ID	Period [ms]	Tolerance [ms]	WCET1 [ms]	WCET2 [ms]	Task Criticality	Service ID	Service Criticality
A	21	37.8	15.12	16.8	2	S1	2
B	10	18	7.2	8	2	S2	2
C	17	30.6	12.24	13.6	1	S3	2
D	17	30.6	12.24	13.6	1	S3	2
E	17	30.6	10.36	20.4	1	S4	2
F	17	30.6	12.24	13.6	1	S4	2
G	5	9	3.6	4	1	S5	1
H	5	9	3.6	4	1	S6	1
I	5	9	3.6	4	1	S7	1
J	11	19	7.92	8.8	1	S8	1

Table 7.1: ATMP-CA: Services and asks (the criticality arithmetic aware services are S3 and S4)

experiments.

7.1.2 Results and Analysis

In this subsection, we present the results obtained from the ATMP-CA and LBP-CA experiments. First, we analyse the ATMP-CA results in data, including absolute utility achieved and the number of dropped tasks, collected from running the task set starting from 6 cores and fail core per run down to 2 cores. Second, we analyse the LBP-CA results data, the number of dropped and executed tasks, and the total time spent on each criticality mode, collected from running the task set on a uni-core mixed-criticality system.

ATMP-CA Figure 7.1 shows the results for our experiments with 6 cores down to 2 available cores. The *MAX* line denotes the maximum possible absolute utility for each task, which is either 2.0 or 1.0. Simultaneously, Table 7.3 provides a detailed breakdown of the corresponding numerical values for absolute and relative utility, optimised periods, and the allocation of tasks to assigned cores.

Service:	*name*	S1	S2		S3	S4
	crit.	high	high		high	low
Task:	*name*	T1	T2$_a$	T2$_b$	T3	T4
	crit.	high	low	high	low	low
	id	A	B	C	D	E

Task ID	Period [ms]	Deadline [ms]	WCET1 [ms]	WCET2 [ms]	Task Criticality	Service ID	Service Criticality
A	25	12	8	8	1	**S1**	1
B	27	12	4	4	1	**S2**	2
C	49	24	4	10	2	**S3**	2
D	33	32	8	8	2	**S2**	2
E	93	92	12	12	1	**S4**	1

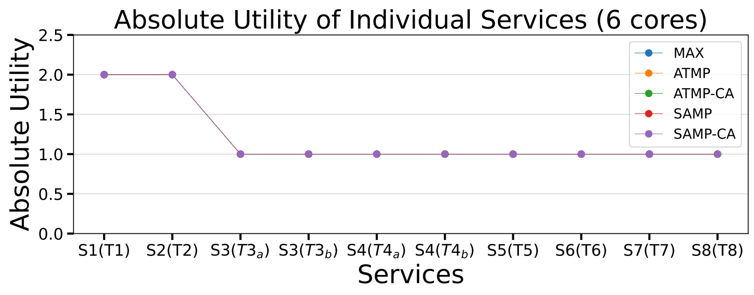
Table 7.2: LBP-CA: Set of Services and Tasks (S2 uses criticality arithmetic)

Each protocols presents the core identifier cr_i, task id $\tau.id$, optimised period $\tau.p$, optimised relative utility $\tau.util$, resulted absolute utility $\tau.util \cdot \tau.l$, and the service id $S.id$ a task $\tau.id$ implements.

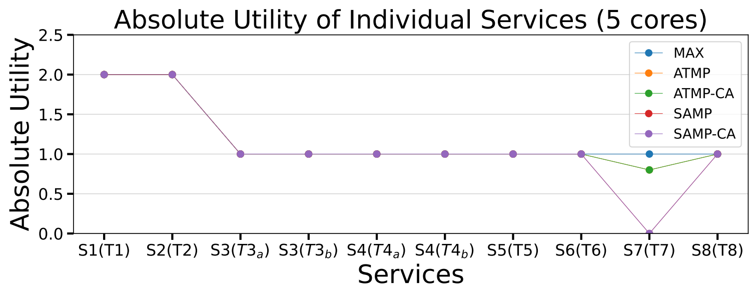
Figure 7.1.a shows the case with no resource shortage, i.e. all 6 cores out of 6 cores are available. This case represents the optimal allocation, which means for each task it was possible to assign them their primary period $\tau.uf.p_{prim}$, resulting in the maximum relative utility of 1.0, and no service is dropped at all.

Figure 7.1.b shows the case with 5 cores out of 6 cores available. SAMP and SAMP-CA dropped low criticality task, T7, that implements low criticality service, S7. ATMP and ATMP-CA successfully allocated the low criticality task, T7, but with minor degradation.

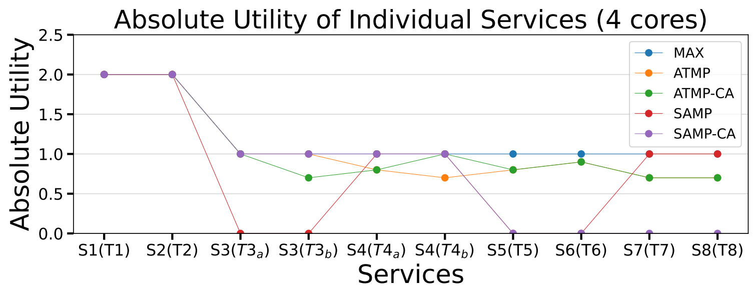
Figure 7.1.c shows the case with 4 cores out of 6 cores available. We see that SAMP allocated tasks that implement services S1, S2 and S4, except the two low criticality task replicas T4$_a$,T4$_b$ that implement service S3, and SAMP-CA assigned the high-criticality tasks at the cost of dropping all low criticality tasks. Also, SAMP and SAMP-CA show an equal number of allocated and dropped tasks but differ in the behaviour of tasks dropping when considering criticality arithmetic. ATMP and ATMP-CA protocols have assigned all services to available processing cores, with minor degradation of their absolute utility without dropping any tasks at all. This resulted by the criticality-arithmetic-aware algorithm that uses the ILP objective function mentioned in Section 4.1.3



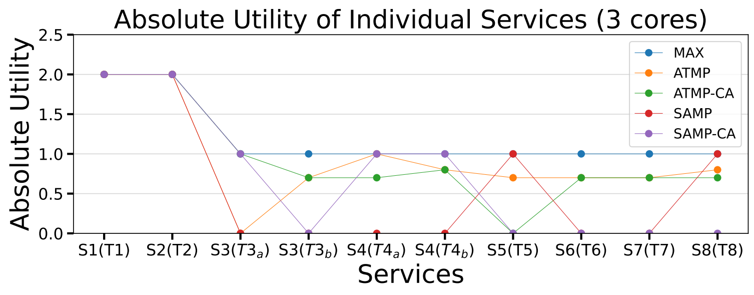
a) Six available cores, with zero core failures



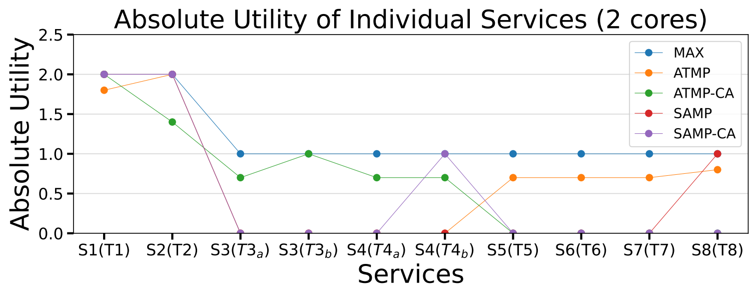
b) Five available cores, with one core failures



b) Four available cores, with two core failures



b) Three available cores, with three core failures



c) Two available cores, with four core failures

Figure 7.1: Achieved Absolute Utility by tasks ($T3_a$,$T3_b$ implement service S3, and $T4_a$,$T4_b$ implement service S4)

for service S3. Under ATMP-CA, the first task $T3_a$ has been set to full utility, resulting in the degraded utility in $T3_b$, which allows to give resources to other tasks. Because both tasks execute the service S3, there is no need to assign maximum utility to both of them during system overload, as observed in ATMP. Also with service S4, ATMP-CA allows a degraded utility for task $T4_a$ with the full utility to $T4_b$, whereas under ATMP noth tasks implement S4 are degraded.

Figure 7.1.d shows the case with 3 cores out of 6 cores available. It's clear that SAMP-CA have dropped all low criticality services including service tasks, S3, to retain all high criticality services. ATMP-CA shows smoother degradation than ATMP while providing an acceptable level of remaining services.

Figure 7.1.e shows the case with 2 cores out of 6 cores available. Here, SAMP retained the tasks of high criticality services S1 and S2 and just one low criticality task T8, but dropped all the tasks of all other high criticality services, including S3 and S4. SAMP-CA performed already a bit better by retaining the tasks of high criticality services S1 and S2 and also retaining one task $T4_b$ of high criticality service S4, while dropping all other services. This shows that the criticality-arithmetic-aware core allocation in SAMP-CA shows some benefit, but generally, both SAMP and SAMP-CA have limited performance, as they don't support the flexibility with the tolerance range. On the other hand, ATMP and ATMP-CA have shown a significant difference in dropping high-criticality services. ATMP retained two high-criticality services and four low-criticality tasks but dropped both S3 and S4 replicas, where ATMP-CA successfully allocated all high-criticality services including the replicated tasks dropped all low-criticality ones. In addition, ATMP-CA has allocated S3 replica task D with maximum utility as a result of the degradation of task C, and both S4 replicas have been degraded which shows that the modified optimisation process couldn't find a solution that allocates task F at the maximum utility.

LBP-CA The purpose of our experiment was to show that the LBP-CA returns to normal mode with the least number of abandoned Low-critical tasks compared to reference protocols. Figure 7.3 shows the schedule for the task set presented in Table 7.2 by LBP-CA, which uses criticality arithmetic, and reference Bailout-based schedulers, BP and LBP [47, 98] The figure presents three sub-figures, each presenting the generated schedule by each protocol. At each sub-figure, the top half of the sub-figure shows the real-time execution for all tasks in the system uni-core system. The other half shows the criticality modes each protocol traverses by the induced WCET overrun on the first job of the high criticality service S3, implemented by task C. At the schedule part of each sub-figure, the x-axis represents the real time or absolute time, and the y-axis

Number of Cores	SAMP						SAMP-CA						ATMP						ATMP-CA					
	cr_{id}	$\tau.id$	$\tau.p$	$\tau.util$	$\tau.abs$	$S.id$	Core	$\tau.id$	$\tau.p$	$\tau.util$	$\tau.abs$	$S.id$	Core	$\tau.id$	$\tau.p$	$\tau.util$	$\tau.abs$	$S.id$	Core	$\tau.id$	$\tau.p$	$\tau.util$	$\tau.abs$	$S.id$
6 CORES	0	A	21	1	2	S1	0	A	21	1	2	S1	0	A	21	1	2	S1	0	A	21	1	2	S1
	1	B	10	1	2	S2	1	B	10	1	2	S2	1	B	10	1	2	S2	1	B	10	1	2	S2
	2	C	17	1	1	S3	2	C	17	1	1	S3	2	C	17	1	1	S3	2	C	17	1	1	S3
	3	D	17	1	1	S3	3	D	17	1	1	S3	3	D	17	1	1	S3	3	D	17	1	1	S3
	4	E	17	1	1	S4	4	E	17	1	1	S4	4	E	17	1	1	S4	4	E	17	1	1	S4
	5	F	17	1	1	S4	5	F	17	1	1	S4	5	F	17	1	1	S4	5	F	17	1	1	S4
	2	G	5	1	1	S5	2	G	5	1	1	S5	2	G	5	1	1	S5	2	G	5	1	1	S5
	3	H	5	1	1	S6	3	H	5	1	1	S6	3	H	5	1	1	S6	3	H	5	1	1	S6
	4	I	5	1	1	S7	4	I	5	1	1	S7	4	I	5	1	1	S7	4	I	5	1	1	S7
	5	J	11	1	1	S8	5	J	11	1	1	S8	5	J	11	1	1	S8	5	J	11	1	1	S8
5 CORES	0	A	21	1	2	S1	0	A	21	1	2	S1	0	A	21	1	2	S1	0	A	21	1	2	S1
	1	B	10	1	2	S2	1	B	10	1	2	S2	1	B	10	1	2	S2	1	B	10	1	2	S2
	2	C	17	1	1	S3	2	C	17	1	1	S3	2	C	17	1	1	S3	2	C	17	1	1	S3
	3	D	17	1	1	S3	3	D	17	1	1	S3	3	D	17	1	1	S3	3	D	17	1	1	S3
	4	E	17	1	1	S4	4	E	17	1	1	S4	4	E	17	1	1	S4	4	E	17	1	1	S4
	2	F	17	1	1	S4	2	F	17	1	1	S4	2	F	17	1	1	S4	2	F	17	1	1	S4
	3	G	5	1	1	S5	3	G	5	1	1	S5	3	G	5	1	1	S5	3	G	5	1	1	S5
	4	H	5	1	1	S6	4	H	5	1	1	S6	4	H	5	1	1	S6	4	H	5	1	1	S6
	x	I	**0**	**0**	**0**	S7	x	I	**0**	**0**	**0**	S7	0	I	**6.6**	**0.8**	**0.8**	S7	0	I	**6.6**	**0.8**	**0.8**	S7
	1	J	11	1	1	S8	1	J	11	1	1	S8	1	J	11	1	1	S8	1	J	11	1	1	S8
4 CORES	0	A	21	1	2	S1	0	A	21	1	2	S1	0	A	21	1	2	S1	0	A	21	1	2	S1
	1	B	10	1	2	S2	1	B	10	1	2	S2	1	B	10	1	2	S2	1	B	10	1	2	S2
	x	C	**0**	**0**	**0**	S3	2	C	17	1	1	S3	2	C	17	1	1	S3	2	C	17	1	1	S3
	x	D	**0**	**0**	**0**	S3	3	D	17	1	1	S3	3	D	17	1	1	S3	3	D	31	**0.7**	**0.7**	S3
	2	E	17	1	1	S4	2	E	17	1	1	S4	2	E	27	**0.8**	**0.8**	S4	2	E	27	**0.8**	**0.8**	S4
	3	F	17	1	1	S4	3	F	17	1	1	S4	3	F	31	**0.7**	**0.7**	S4	3	F	17	1	1	S4
	x	G	**0**	**0**	**0**	S5	x	G	**0**	**0**	**0**	S5	0	G	6.6	**0.8**	**0.8**	S5	0	G	6.6	**0.8**	**0.8**	S5
	x	H	**0**	**0**	**0**	S6	x	H	**0**	**0**	**0**	S6	1	H	6.2	**0.9**	**0.9**	S6	1	H	6.2	**0.9**	**0.9**	S6
	2	I	5	1	1	S7	x	I	**0**	**0**	**0**	S7	2	I	9	**0.7**	**0.7**	S7	2	I	9	**0.7**	**0.7**	S7
	3	J	11	1	1	S8	x	J	**0**	**0**	**0**	S8	3	J	20	**0.7**	**0.7**	S8	3	J	20	**0.7**	**0.7**	S8
3 CORES	0	A	21	1	2	S1	0	A	21	1	2	S1	0	A	22	1	2	S1	0	A	22	1	2	S1
	1	B	10	1	2	S2	1	B	10	1	2	S2	1	B	10	1	2	S2	1	B	10	1	2	S2
	x	C	**0**	**0**	**0**	S3	2	C	17	1	1	S3	x	C	0	**0**	**0**	S3	2	C	17	1	1	S3
	x	D	**0**	**0**	**0**	S3	x	D	**0**	**0**	**0**	S3	0	D	31	**0.7**	**0.7**	S3	0	D	31	**0.7**	**0.7**	S3
	x	E	**0**	**0**	**0**	S4	2	E	17	1	1	S4	2	E	17	1	1	S4	2	E	31	**0.7**	**0.7**	S4
	x	F	**0**	**0**	**0**	S4	1	F	17	1	1	S4	1	F	27	**0.8**	**0.8**	S4	1	F	27	**0.8**	**0.8**	S4
	2	G	5	1	1	S5	x	G	**0**	**0**	**0**	S5	2	G	9	**0.7**	**0.7**	S5	x	G	0	**0**	**0**	S5
	x	H	**0**	**0**	**0**	S6	x	H	**0**	**0**	**0**	S6	0	H	9	**0.7**	**0.7**	S6	0	H	9	**0.7**	**0.7**	S6
	x	I	**0**	**0**	**0**	S7	x	I	**0**	**0**	**0**	S7	1	I	9	**0.7**	**0.7**	S7	1	I	9	**0.7**	**0.7**	S7
	2	J	11	1	1	S8	x	J	**0**	**0**	**0**	S8	2	J	17	**0.8**	**0.8**	S8	2	J	20	**0.7**	**0.7**	S8
2 CORES	0	A	21	1	2	S1	0	A	21	1	2	S1	0	A	26	0.9	1.8	S1	0	A	22	1	2	S1
	1	B	10	1	2	S2	1	B	10	1	2	S2	1	B	10	1	2	S2	1	B	18	**0.7**	**1.4**	S2
	x	C	**0**	**0**	**0**	S3	x	C	**0**	**0**	**0**	S3	x	C	**0**	**0**	**0**	S3	0	C	31	**0.7**	**0.7**	S3
	x	D	**0**	**0**	**0**	S3	x	D	**0**	**0**	**0**	S3	x	D	**0**	**0**	**0**	S3	1	D	17	1	1	S3
	x	E	**0**	**0**	**0**	S4	x	E	**0**	**0**	**0**	S4	x	E	**0**	**0**	**0**	S4	0	E	31	**0.7**	**0.7**	S4
	x	F	**0**	**0**	**0**	S4	1	F	17	1	1	S4	x	F	**0**	**0**	**0**	S4	1	F	29	**0.7**	**0.7**	S4
	x	G	**0**	**0**	**0**	S5	x	G	**0**	**0**	**0**	S5	0	G	9	**0.7**	**0.7**	S5	x	G	**0**	**0**	**0**	S5
	x	H	**0**	**0**	**0**	S6	x	H	**0**	**0**	**0**	S6	1	H	9	**0.7**	**0.7**	S6	x	H	**0**	**0**	**0**	S6
	x	I	**0**	**0**	**0**	S7	x	I	**0**	**0**	**0**	S7	0	I	9	**0.7**	**0.7**	S7	x	I	**0**	**0**	**0**	S7
	1	J	11	1	1	S8	x	J	**0**	**0**	**0**	S8	1	J	17	**0.8**	**0.8**	S8	x	J	**0**	**0**	**0**	S8

Table 7.3: Allocation of Tasks to Cores (red x indicates failed allocation on the respective core, bolded numbers indicate compromised service either degraded or dropped)

shows the progress of periodic tasks presented in Table 7.2. At the criticality part in each sub-figure, the x-axis represents the real time or absolute time, and the y-axis shows the system criticality modes Normal, Bailout, and Recovery modes, where active mode in the system is represented by the blue horizontal line, and the vertical lines associates the time and duration of a mode change according to the schedule sub-figure at the top part of the sub-figure.

Figure 7.2 shows the complete Fixed-priority based schedule with no WCET overrun for the job set presented in Table 7.4. In the Figure 7.3, job C0 overrun its optimistic WCET estimates, WCET1, by 6 ms from time 16 to 22 ms. The following code fragment shows how we induced this overrun in the experiment.

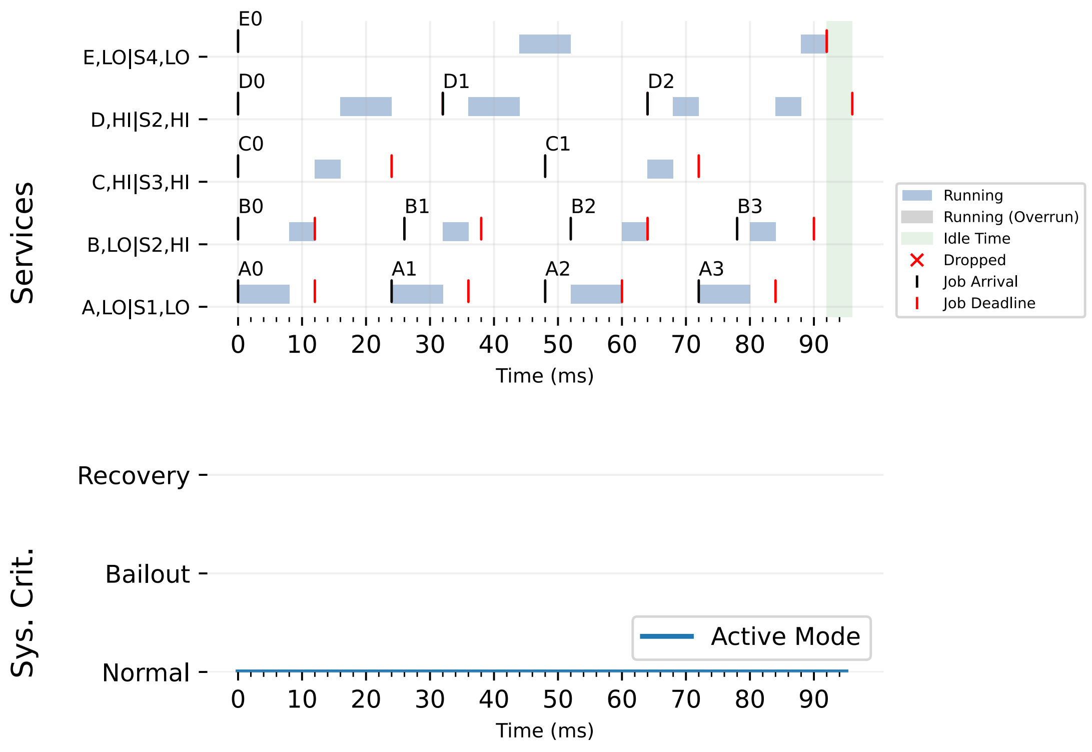
```
def set_overrun(job): job.et = 10
jobs_queue = [set_et(x) for x in sched.jobs_queue if x.id == ‘C0’]
```

All protocols: - BP, LBP, LBP-CA - activate the bailout mode at the start of the overrun. BP and LBP show identical mode changes, but different schedules. BP and LBP activate the bailout modes at time 16 and remain C0 completion at time 22. BP and LBP activate the recovery modes at time 22 until the last HI criticality task D0 completes execution at time 30. BP and LBP spent 14 ms in bailout and recovery high criticality modes

LBP-CA shows unique mode changes, with a criticality arithmetic perspective for a smoother degradation schedule. LBP-CA enters bailout mode at time 16 until C0 completion at time 22. Since LBP-CA is using criticality arithmetic, it realizes that service S2 already executed by task B job, B0, and activation for the recovery mode can be neglected to return to normal mode at time 22. LBP-CA spent 6 ms in bailout mode, about 50% of time spent by BP and LBP protocols in high criticality modes, 14 ms.

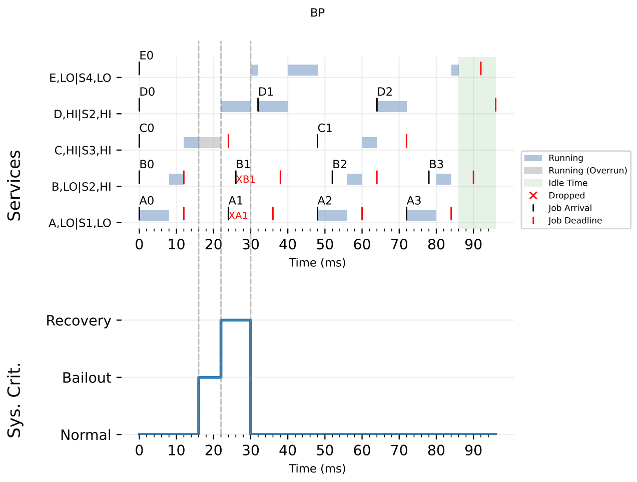
In the schedule part of the protocol, we see different responses for C0 overrun. Though BP and LBP show identical mode changes, they differ in their resilience to the induced C0 overrun. LBP shows smoother degradation than BP. BP abandons the release of low criticality jobs A1 and B1 during the recovery mode. Note that job E0 is released before BP and LBP activate the criticality modes of bailout and recovery, and it's abandoned according to the AMC analysis. LBP tolerated the situation by append releasing low criticality jobs to a low priority queue for a possible future execution, which occurred after the completion of D0 at time 30. LBP schedules low criticality job B1 for execution, but A1 is dropped after B1 finishes due to the arrival of high criticality job D1, and job E0 is delayed by 12 ms. BP and LBP schedule complete at time 90 ms. LBP-CA degradation features quickly to normal mode by using criticality arithmetic.

LBP-CA schedules low criticality job E0 at its release time and returns to normal mode quicker than BP and LBP. Service S2 is already completed by the execution of replica B0, D0 that collaborates B0 in implementing S2 can be neglected. The system returns to normal mode and E0 scheduled at its scheduling time for normal execution LBP-CA schedule completes at time 84 ms. Table 7.5 presents a summary of the degradation caused by CO overrun. Competed and dropped jobs are presented for each protocol, and the mode change duration for the start of time 0 - critical instant - up to time 30 where the identical system returns to normal mode by the reference protocols BP and LBP. BP and LBP. remained in normal mode for 48% of the time, 20% in bailout mode, and 26.67% of the time in recovery mode. In contrast, LBP-CA shows the earliest return to normal compared to BP and LBP, resulting in staying up to 80% normal behavior, 20% in bailout mode, and 0% of the time in recovery mode.

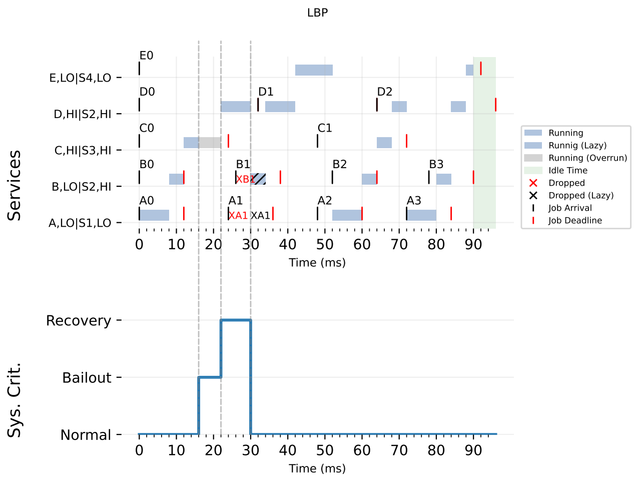


a) BP, BP, and LBP-CA (0 dropped, 14 allocated)

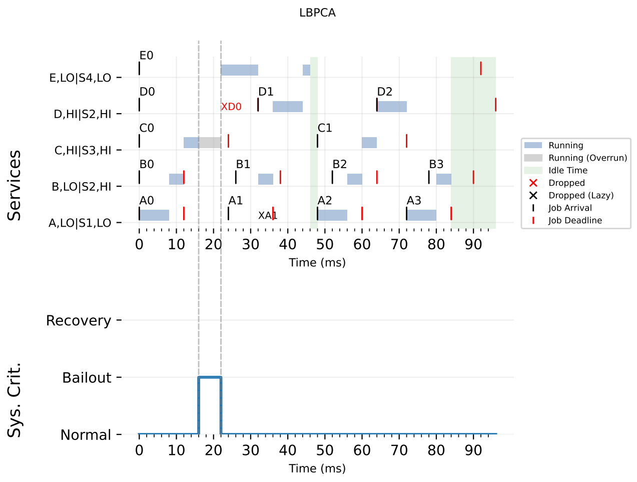
Figure 7.2: Comparison of scheduling mixed-criticality tasks between BP, LBP, and LBP-CA



a) BP protocol (2 dropped jobs, 12 completed jobs)



b) LBP protocol (1 dropped jobs, 13 completed jobs)



b) LBP-CA protocol (2 dropped jobs, 12 completed jobs)

Figure 7.3: Comparison of scheduling mixed-criticality tasks between BP, LBP, and LBP-CA

Task ID	Arrival	Priority	Abs. Dealine	Actual WCET	WCET1	WCET2	Task Criti-claity	Service Criti-claity	Service Criti-claity
A0	0	5	12	8	8	8	1	1	S1
B0	**0**	**4**	**12**	**4**	**4**	**4**	**1**	**2**	**S2**
C0	0	3	24	10	4	10	2	2	S3
D0	**0**	**2**	**32**	**8**	**8**	**8**	**2**	**2**	**S2**
E0	0	1	92	12	12	12	1	1	S4
A1	24	5	36	8	8	8	1	1	S1
B1	**26**	**4**	**38**	**4**	**4**	**4**	**1**	**2**	**S2**
D1	**32**	**2**	**64**	**8**	**8**	**8**	**2**	**2**	**S2**
A2	48	5	60	8	8	8	1	1	S1
C1	48	3	72	4	4	10	2	2	S3
B2	**52**	**4**	**64**	**4**	**4**	**4**	**1**	**2**	**S2**
D2	**64**	**2**	**96**	**8**	**8**	**8**	**2**	**2**	**S2**
A3	72	5	84	8	8	8	1	1	S1
B3	**78**	**4**	**90**	**4**	**4**	**4**	**1**	**2**	**S2**

Table 7.4: Jobs Releases for LBP-CA experiment (jobs use criticality arithmetic agnostic are Bolded)

						BP		**LBP**		**LBP-CA**	
Job	**Arrival**	**Priority**	**Absolute Dealine**	**Criticality**	**Service ID**	**Completed**	**Dropped**	**Completed**	**Dropped**	**Completed**	**Dropped**
A0	0	5	12	1	S1	Y		Y		Y	
B0	**0**	**4**	**12**	**1**	**S2**	**Y**		**Y**		**Y**	
C0	0	3	24	2	S3	Y		Y		Y	
D0	**0**	**2**	**32**	**2**	**S2**	**Y**		**Y**			**Y**
E0	0	1	92	1	S4	Y		Y			
A1	24	5	36	1	S1		Y		Y		Y
B1	26	4	38	1	S2		Y	Y		Y	
						Normal	53.33	Normal	53.33	Normal	80.00
						Bailout	20.00	Bailout	20.00	Bailout	20.00
						Recovery	26.67	Recovery	26.67	Recovery	0.00

Table 7.5: Responses impact on jobs induced by C0 WCET overrun BP, LBP, and LBP-CA Protocols (S2 jobs, B0 and D0, use criticality arithmetic), and Modes Duration (LBP-CA shows earliest return)

7.1.3 Discussion

This experiment sets out with the aim of assessing the importance of incorporating Criticality Arithmetic for building dependable mixed-criticality systems. The most obvious finding to emerge from the analysis is that integrating criticality arithmetic into mixed-criticality systems allows better survivability and smooth degradation when tolerating permanent faults in system cores such as core failures. In such situations, criticality arithmetic enables efficient usage of the remaining cores by allocating tasks according to the criticality of the service they implement.

The present study raises the possibility that integrating criticality arithmetic into existing Mixed-criticality allocation and scheduling algorithms, will empower such systems with fine-grained graceful degradation during overload and computing resource shortages. In addition, the study discovers that there is no need to allocate all critical service replicas at their maximum quality of service at the cost of dropping another critical service.

One limitation of this experiments is that its limited to systems with two criticality levels. Future work would be extending these findings by ones obtained from systems with more than three criticality levels.

The next section presents the experimental evaluation for comparing SAMP and ATMP mixed-criticality protocols for assessing smooth degradation under reduced computing capacity resource shortage.

7.2 Experimental Evaluation of SAMP and ATMP Protocols

7.2.1 Experimental Setup

The following presents the setup for evaluating smooth degradation between mixed-criticality protocols, Adaptive Tolerance-based Mixed-criticality Protocol (ATMP), and Standard Adaptive Mixed-criticality Protocol (SAMP), on a multi-core platform with reduced computing capacity. This includes: system configuration, protocols, and the generation of tasks implement services.

System and Resource-shortage Configurations: In order to assess smooth degradation between ATMP and SAMP, we applied a frequency scaling algorithm on both protocols over 8 cores with a frequency slowdown factor of 2% and cores turn-down factor of 1. The experiment was performed by running the same task set on both protocols, every time we decreased the frequency by a factor of 2% on the available number of cores. Then, we record the achieved absolute utility and number of dropped tasks by each protocol. The frequency scaling algorithm starts with 8 cores and runs the task set under 50 degrees of processing speed. Subsequently, we reduce the number of cores by turning one core and start all over again on the available number of cores.

Mixed-criticality Protocols:, SAMP is a multi-core scheduling protocol that partitions the multi-core computing resources into single partitioned cores, each core tests and runs its allocated services under the AMC analysis and scheduling scheme [43, 8]. The ATMP protocol is an application of the tolerance-based real-time computing model. ATMP is a multi-core scheduling protocol that partitions the multi-core computing resources that maximises each core utility using linear programming to optimise task rates and tolerance ranges in case of core failure, constrained to the remaining number of cores after the failure. The main value that distinguishes ATMP from the traditional SAMP model, is that abandoning system tasks from execution to reduce the workload when tolerating resource shortage, is based on the usefulness and run-time adaptation capability of these tasks.

Tasks Generations: We generated a mixed-criticality task set consisting of 20 tasks, with two criticality levels, 12 low-criticality, and 8 high-criticality tasks. The maximum absolute utility of each task depends on its criticality level as seen in the previous section, each criticality task has an absolute utility of 2.0, whereas a low-criticality task's absolute utility is 1.0, and the maximum system utility is the sum of achieved utility by all tasks in the system, which is 28 in the current task set.

The first set of analyses examined smooth degradation on the SAMP approach. In this part of the experiment, disruptive degradation for the overall system utility is seen in Figure 7.4. The achieved absolute utility has decreased significantly from 28 to 22 when slowing core speed from 100% to 80% while showing a stable performance between 26 to 22, with more reduction for speed from 80% to 40%, before we see a significant decline for the system utility with lower frequencies from 40% to 2% and the lowest absolute utility achieved is between 14 and 16 in the 8 cores case and 2 in the two cores case.

The second set of analyses examined smooth degradation under the ATMP approach. We observe that the absolute utility degradation is smoother and finely graded with a greater reduction in processing speed compared to SAMP. As seen in Figure 7.5, the achieved absolute utility is decreasing gracefully with the overall reduction of frequency. The smooth degradation is seen in two ranges of speeds, between 100% to 60% and between 60% to 2%. It's apparent in

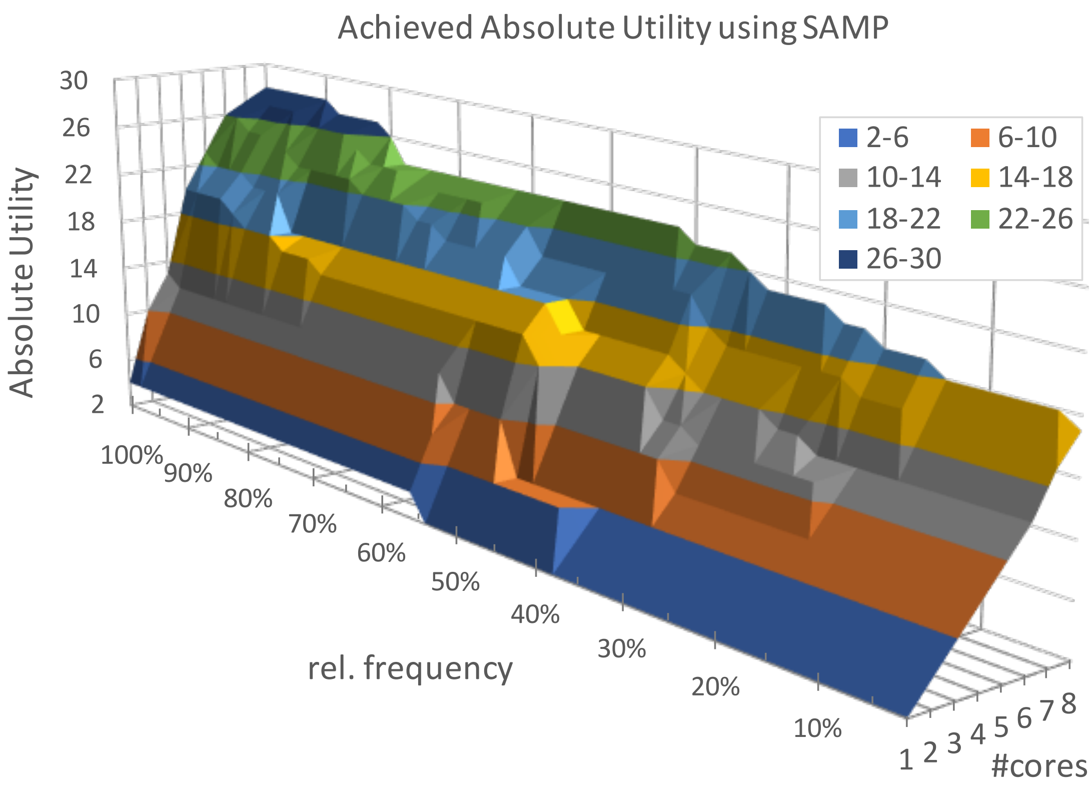


Figure 7.4: Achieved Absolute Utility Under SAMP Protocol (8 cores, 50 processing speeds)

the first range, that the lowest utility achieved in the 8 cores case is between 28 to 22, were in the two cores case is between 2 to 6. In the second range, the lowest utility achieved in the 8 cores was about 22 and about 3 in the two cores case. I compared the two sets of analyses for smooth degradation between

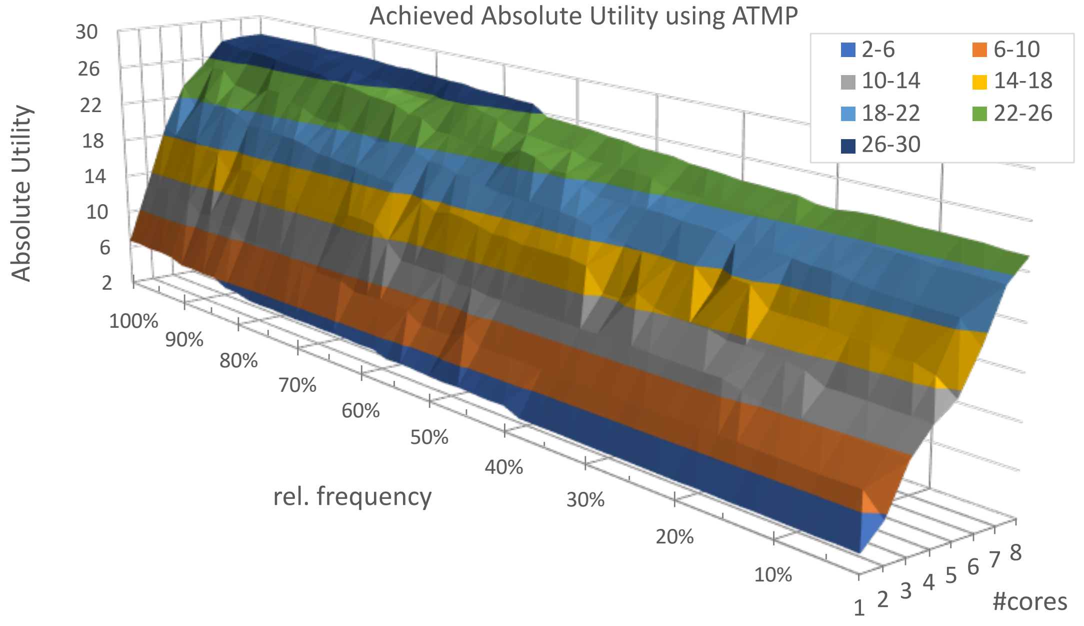


Figure 7.5: Achieved absolute utility under ATMP protocol (8 cores, 50 processing speeds)

ATMP & SAMP. The comparison is conducted by recording the total absolute utility for each run by ATMP and dividing it by the achieved utility under SAMP as presented in Figure **??**. The table presents the numerical data for the ratio comparison figure. The main point is that ATMP shows a higher ratio of achieved absolute utility between 40% and 60% of frequency speed. Therefore, in the next section, we present experimental results and analysis for task dropping and maximized utility by ATMP and SAMP, using distinct use cases from the 8, 4, and 2 cores with 50% and 100% frequency experiment data.

7.2.2 Results and Analysis

The following presents results obtained from the three sets of analysis for utility degradation under frequency scaling, SAMP, ATMP, and SAMP versus ATMP. First, we show the absolute utility achieved and the number of dropped tasks when running the task set over three cases 8, 4, and 2 cores under full and

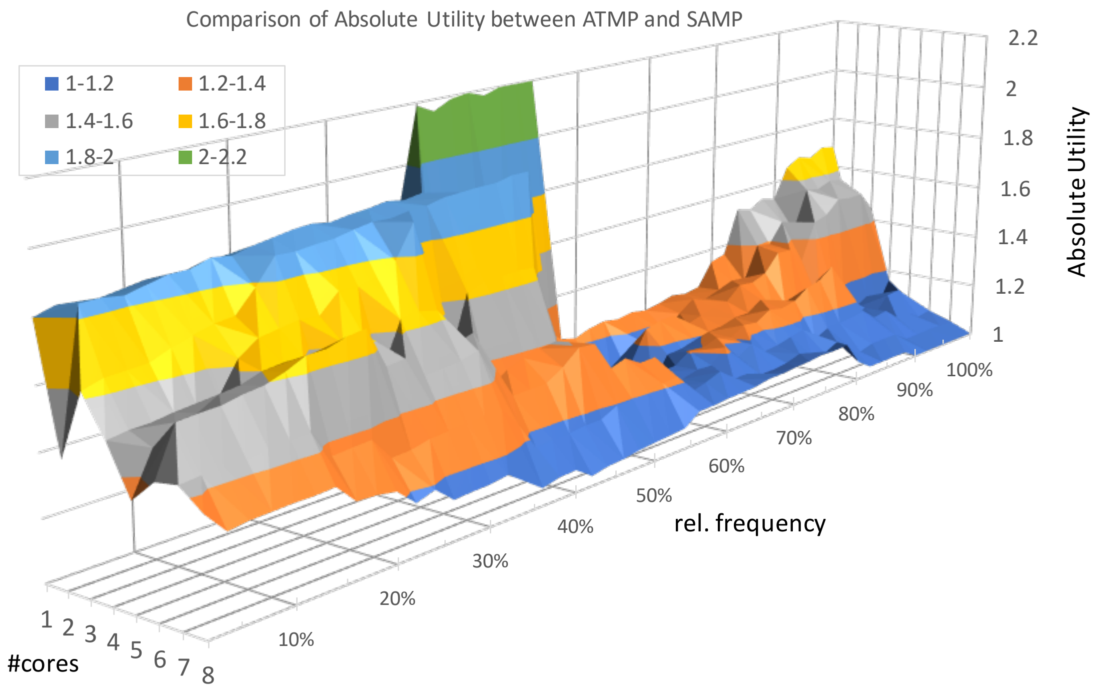


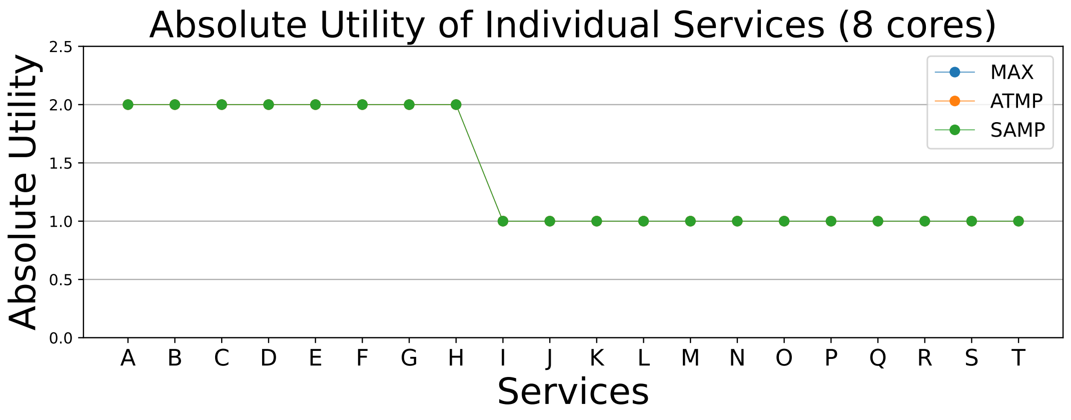
Figure 7.6: Ratio of absolute utility achieved between ATMP & SAMP (8 cores, 50 processing speeds)

Processing Speed	Number of Cores 1	2	3	4	5	6	7	8
1.00	1.68	1.27	1.45	1.16	1.10	1.06	1.04	1.00
0.96	1.66	1.58	1.52	1.13	1.15	1.09	1.03	1.00
0.92	1.65	1.57	1.48	1.16	1.13	1.13	1.03	1.00
0.88	1.55	1.47	1.32	1.15	1.16	1.16	1.07	1.03
0.84	1.53	1.40	1.38	1.21	1.31	1.15	1.05	1.03
0.80	1.35	1.29	1.30	1.21	1.26	1.12	1.13	1.07
0.76	1.26	1.25	1.26	1.23	1.26	1.17	1.11	1.15
0.72	1.25	1.23	1.24	1.21	1.23	1.15	1.12	1.14
0.68	1.25	1.18	1.27	1.19	1.22	1.16	1.11	1.14
0.64	1.25	1.10	1.25	1.17	1.20	1.17	1.14	1.13
0.60	1.22	1.10	1.03	1.12	1.16	1.16	1.16	1.12
0.56	1.12	1.09	1.23	1.09	1.13	1.27	1.22	1.11
0.52	2.25	1.30	1.19	1.06	1.10	1.31	1.21	1.07
0.48	2.18	1.45	1.94	1.19	1.23	1.39	1.20	1.06
0.44	2.19	1.29	1.93	1.65	1.40	1.35	1.17	1.05
0.40	2.19	1.29	1.93	1.65	1.36	1.31	1.16	1.08
0.36	1.86	1.92	1.91	1.42	1.52	1.40	1.25	1.06
0.32	1.86	1.92	1.90	1.90	1.63	1.48	1.32	1.15
0.28	1.84	1.90	1.89	1.75	1.63	1.47	1.31	1.14
0.24	1.84	1.91	1.87	1.74	1.62	1.54	1.29	1.19
0.20	1.82	1.89	1.87	1.69	1.46	1.67	1.37	1.23
0.16	1.83	1.88	1.72	1.67	1.61	1.65	1.36	1.24
0.12	1.83	1.86	1.69	1.52	1.58	1.60	1.45	1.31
0.08	1.81	1.83	1.65	1.52	1.53	1.57	1.44	1.30
0.04	1.82	1.84	1.64	1.51	1.47	1.57	1.43	1.29

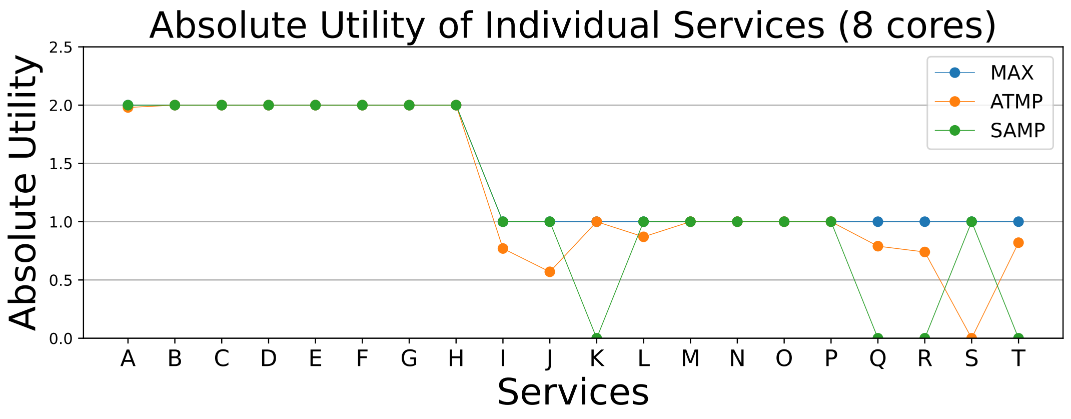
Table 7.6: Ratio of Achieved Absolute Utility Between SAMP and ATMP over 8 cores at Various Processing Speeds

half frequency speed. Second, we present a comparison between the three cases. Table 7.6 shows an overview of the ratio of achieved absolute utility between SAMP and ATMP over 8 cores at various processing speeds.

Figure 7.7 provides the results for task dropping obtained from the 8 cores case. The maximum system utility 28 is reached when running the task set with full frequency on both protocols and no drop of any task as shown in Figure 7.7.a. However, when we reduce frequency speed to half as seen in Figure 7.7.b, SAMP protocol dropped 4 low criticality tasks, where only one task dropped by ATMP in the cost of the quality of 6 low criticality tasks.



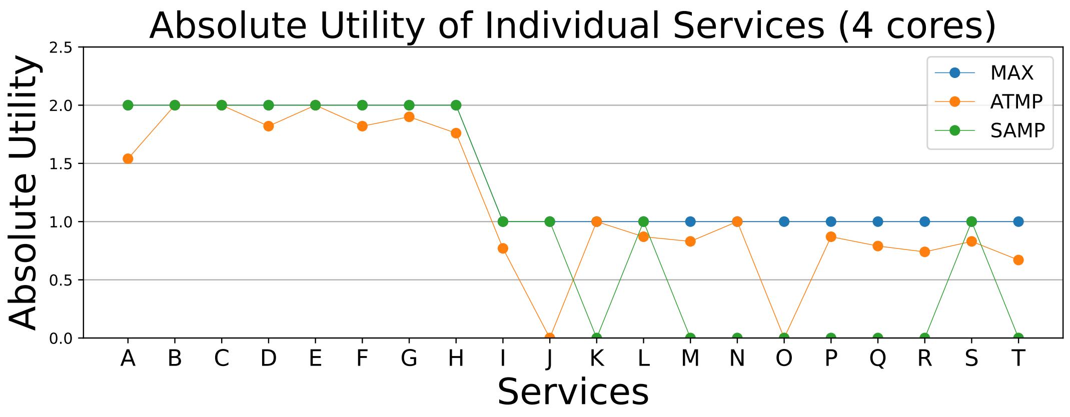
a) frequency = 100%



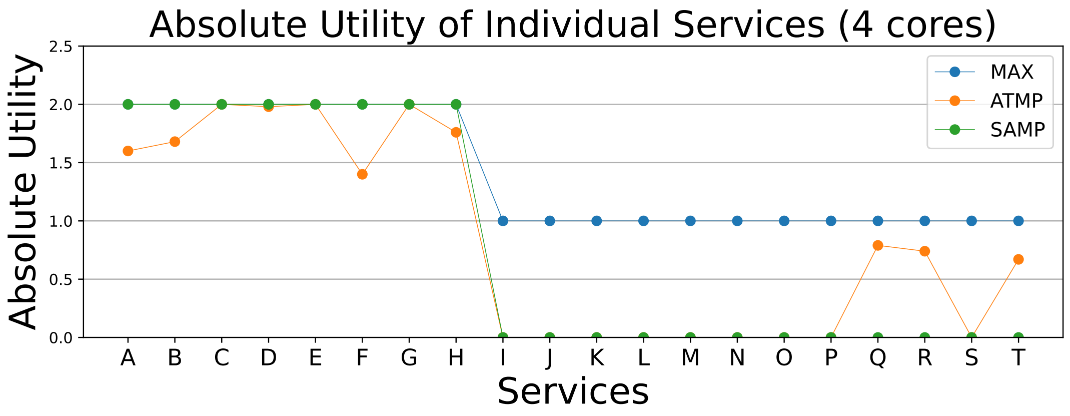
b) frequency = 50%

Figure 7.7: Achieved absolute utility by tasks under ATMP & SAMP on 8 cores

In the second case of the comparison, the 4 cores case, Figure 7.8 displays the result for task dropping by each protocol. It's clear from the Figure 7.8.a that with 100% frequency, SAMP abandoned the execution of 8 low criticality tasks and no drop of any high criticality tasks. Though that ATMP has abandoned only two low criticality tasks, 5 high criticality tasks, and 8 low criticality tasks have been executed with compromised performance. However, when we reduce frequency speed to half as seen in Figure 7.8.b, SAMP protocol dropped 4 low criticality tasks, while a single task dropped by ATMP in the expenses of the quality of 6 low criticality tasks. Thus, with half frequency speed, more compromise for system performance and quality of execution. SAMP protocol has dropped the execution of all 12 low criticality tasks but maintained the maximum performance of the high criticality tasks. On the other hand, two low-criticality tasks have succeeded in execution, and 5 high-criticality tasks have experienced an acceptable delay in the ATMP case.



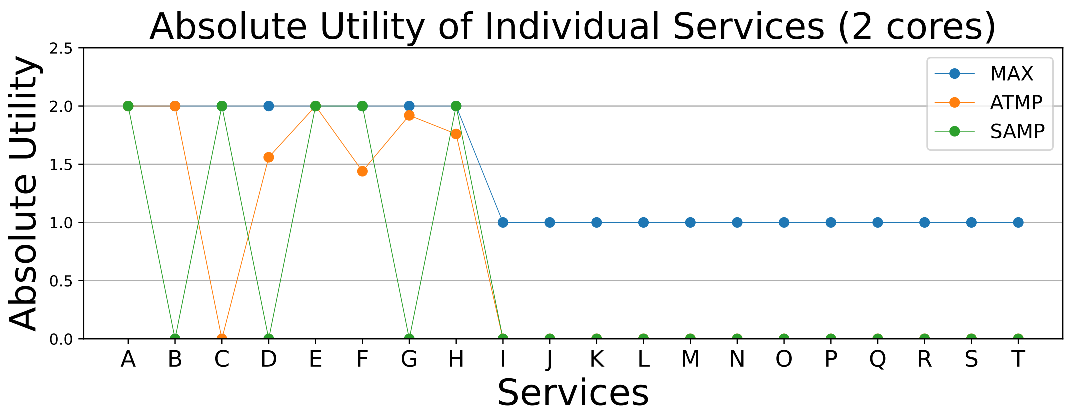
a) frequency = 100%



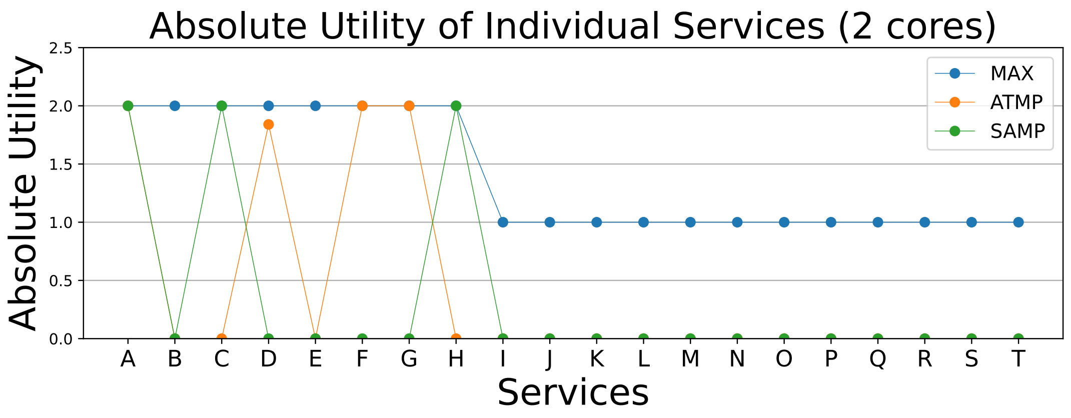
b) frequency = 50%

Figure 7.8: Achieved absolute utility by tasks under ATMP & SAMP on 4 cores

In the third case, the 2 cores case. Figure 7.9 illustrates the results under this severe reduction in the number of cores. In Figure 7.9.a, we see that SAMP and ATMP dropped all low criticality tasks. ATMP dropped one high criticality and SAMP dropped 3 high criticality tasks. Both protocols achieved similar performance when we reduced the frequency to half as shown in Figure 7.9.b. It can be seen that the performance is almost identical in dropping all low-criticality tasks and almost half of the high-criticality tasks in both protocols.



a) frequency = 100%



b) frequency = 50%

Figure 7.9: Achieved absolute utility by tasks under ATMP & SAMP on 2 cores

At the end of this section, we illustrate an overview of the main results on two frequency speeds, 100%, and 50%, between SAMP and ATMP. Then, we compare the achieved utility and count the number of dropped tasks between SAMP and ATMP.

Table 7.7 demonstrates the performance between the three cases in both protocols. We ignore the optimal case; 8 cores with maximum speed in the top half of the table for SAMP and ATMP columns. Further vital points can be concluded from this table. First, under the SAMP approach column, the frequency speed reduction for all available cores achieves higher utility than the

		SAMP		ATMP	
Frequency	#Cores	Abs. utility	#Tasks dropped	Abs utility	#Tasks dropped
100%	8	28	0	28	0
	4	20	8	23.21	2
	2	10	15	12.68	13
50%	8	24	4	25.54	1
	4	16	12	16.62	9
	2	6	17	7.84	16

Table 7.7: Slowdown-Frequency/Turndown-Cores Effect on ATMP and SAMP

Frequency	#Cores	Abs. Utility ATMP/SAMP	#Tasks dropped ATMP - SAMP
100%	8	1.00	0
	4	1.16	-6
	2	1.27	-2
50%	8	1.06	-3
	4	1.04	-3
	2	1.31	-1

Table 7.8: Comparison of achieved absolute utility & dropped tasks between ATMP & SAMP

turn-down half number of the available cores and half the number of dropped tasks. Second, under the ATMP column, minimising the processing speed of overall available cores achieves higher utility than the turn-down half number of the available cores and half the number of dropped tasks. We end the comparison by dividing the achieved utility for ATMP against SAMP and determining the difference in the number of dropped tasks per protocol.

Both approaches deliver higher performance when reducing frequency speed to half. Table 7.8 shows that the obtained ATMP utility Is higher than that in SAMP when turning down 4 cores, and operating the remaining cores at full speed, which retains 6 tasks from being dropped compared to SAMP.

7.2.3 Discussion

The comparison shows that ATMP fulfilled a smoother degradation than SAMP. In addition, ATMP guarantees more tasks than SAMP, while at the same time maximise system utility higher than SAMP protocol. It was found from the analysis that slowing down frequency is more efficient than turn-down cores, despite which model is used, the traditional or TRTCM-based model. These findings broadly support the work of other studies in the tolerance-based model, which allows us to extend the development of resilient systems for mixed-criticality systems based on the TRTCM-based model.

In this next section, we evaluate the TRTCM utility function, as defined by the TRTCM model that features smooth degradation as seen in this chapter. The evaluation involves a comparison between TRCTM-ILP and a heuristic function, which represents an integration between TRTCM and the Elastic Real-time model (TRTCM-Elastic). TRTCM-Elastic is agnostic to the two concepts of service utility and criticality. I Integrated the two concepts into the original Real-time Elastic model for evaluating smooth degradation under different types of resource shortages, and the reduced desired load, between TRTCM-ILP and TRTCM-Elastic .

7.3 Experimental Evaluation of TRTCM-ILP and TRTCM-Elastic

This section describes the experimental setup, results, and analysis for evaluating smooth degradation between TRTCM-ILP and TRTCM-Elastic .

7.3.1 Experimental Setup

System and Resource-shortage Configurations: The first part of the evaluation compares the achieved absolute utility between TRTCM-ILP and TRTCM-Elastic . I applied a desired-load reduction algorithm to both models to observe the degradation in overall system utility each time we reduced the desired load on the processor. The second part compares the execution time overhead for finding each solution between the methods. Each time we reduce the desired load in each run, we record the execution time by each method.

TRTCM translates the resource shortage problem into a utility function that is constrained to the reduced desired load. TRTCM-ILP uses integer linear programming for optimising the task set according to a given load, and TRTCM-Elastic uses a compression/decompression heuristic that compresses and decompresses tasks to meet the given desired load. Under the TRTCM-Elastic model, tasks with 0 elasticity cannot change their configuration and always receive the initial. Similarly, under TRTCM-ILP , tasks with a primary period equal to their tolerance periods, cannot change their configuration. The experiment was performed by running the same task set on both optimisation models. I decreased the load by a factor of 0.01 starting from 1.0, which is the maximum load bound for using a single processing core. The maximum load of the tasks in the experiment exceeds the processing core load bound of 1.4 and the minimum load is 0.71, therefore, the desired load degradation starts from 1.0 and stops at 0.71. For clarification, no core failure or WCET overrun is assumed here, only reducing the desired load.

Task-set The task set consists of two tasks presented in 7.9, one high and one low criticality task, each defined with primary and tolerance periods, implicit deadlines, relative utility, single WCET estimate, criticality, and elasticity parameters. The maximum utility of the two tasks is 3.0 (sum of tasks criticality weight). The task set maximum load is 1.4 and the minimum load is 0.7. A task with a relative utility of 1.0 or elasticity of 0.0, indicates that the task reconfiguration is prohibited and has to remain constant, therefore, these two cases are eliminated in both tasks in the task set. Task A is a high-criticality task with a weight of 2.0, whereas task B is of low criticality with a weight of 1.0.

7.3.2 Results and Analysis

Here we describe the results and analysis obtained from the experiment. First, we analyse the results for the achieved absolute system utility between TRTCM-ILP and TRTCM-Elastic . Then, we analyse the approach taken by each method in finding a configuration for each task for the desired configurations, which implies the total absolute utility achieved by the method. Finally, we analyse the execution time overhead by each method.

Figure 7.12 presents the achieved absolute utility against a reduction in the desired load. We can observe that the solution obtained by TRTCM-ILP , using the linear programming solver, consistently optimizes the overall system utility better than the heuristic function used by TRTCM-Elastic . As seen in Figure 7.12, TRTCM-ILP provides steadier degradation than TRTCM-Elastic , although TRTCM-Elastic heuristic shows relatively smooth degradation. As seen in Figure 7.12, the maximum utility achieved by TRTCM-ILP is greater than 2.6, whereas the maximum utility achieved by TRTCM-Elastic is less than 2.5. These results are expected, since TRTCM-Elastic solutions are sub-optimal, where TRTCM-ILP always finds the optimal solution if exists. In addition, the ratio of the difference between TRTCM-ILP and TRTCM-Elastic shows that the more the scheduling problem gets easier, the ratio of difference decreases, and the more the problem gets harder by reducing the desired load, the ratio of difference increases. This reveals that when the desired load problem is considered to be less complex according to a given task set, it is better to find a sub-optimal solution quickly with reduced overhead using TRTCM-Elastic , than executing the linear programming solver by TRTCM-ILP for finding an optimal solution, where sub-optimal solution can deliver accepted level of service with minimum cost in terms of execution time and memory overhead.

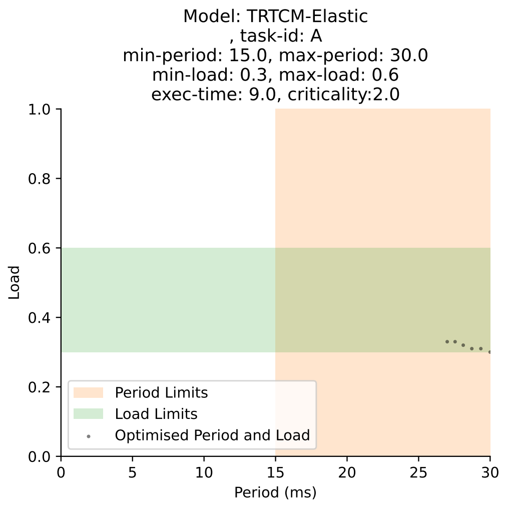
This optimal performance by TRTCM-ILP comes in cost in terms of execution time as seen in sub-figure Figure 7.13.a, TRTCM-Elastic execution time is consistently below 10 ms, remaining steady as far as the problem of finding a solution or configuration gets more complex. In contrast, TRTCM-ILP execution time ranges between 30 and 45.

In the sub-figure, Figure 7.13.b, the ratio of execution time between TRTCM-Elastic and TRTCM-ILP is presented. We can see that the execution time overhead of the TRTCM-ILP utility function is almost 10 times higher than the TRTCM-Elastic heuristic function, and the execution time ratio between TRTCM-ILP and TRTCM-Elastic is the same despite the complexity of the scheduling problem.

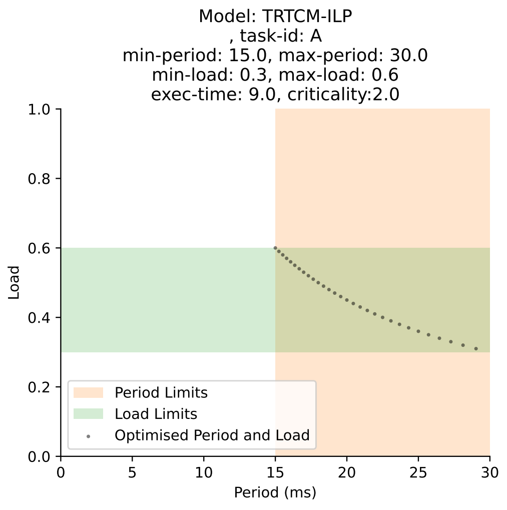
Figures 7.10 and 7.11 illustrate the impact of reducing the desired load on the two tasks in the task set, task A and task B. In sub-figures 7.10.a and 7.10.b, the results from each method for task A are presented, while 7.11.a and 7.11.b show the results for task B. The intersection area between the two colours/constraints represents the solution space for possible configurations of the task. The x-axis represents the tolerance range of task periods, while the y-axis represents the desired load. Each sub-figure highlights the task period and desired load constraints with orange and green respectively. Task A has a tolerance range for its period between 15 and 30, and task B between 20 and 40, where the maximum and minimum load for task A are 0.6 and 0.3, and for task B, they are 0.8 and 0.4 respectively.

In sub-figure 7.10.a, we observe that TRTCM-Elastic exploits task A to be in the range between 26 and 30, each time we reduce the desired load. On the other hand, TRTCM-ILP exploits most of the task A's solution space with smooth degradation, as seen in figure 7.10.b. Also, TRTCM-ILP has determined schedulable periods for task A when, where TRTCM-Elastic selected periods with load is less than or equal to 0.3. For task B, as shown in sub-figures 7.11.a for the TRTCM-Elastic case and figure 7.11.b for the TRTCM-ILP case, we can see that TRTCM-Elastic configured task B to be between 22 and 30, whereas TRTCM-ILP solution in the degrade task B to minimum level of service at period 40.0.

Table 7.10 presents the ratio of achieved utility by each task and the result total system utility. Table 7.11 presents the ratio of achieved utility by each task

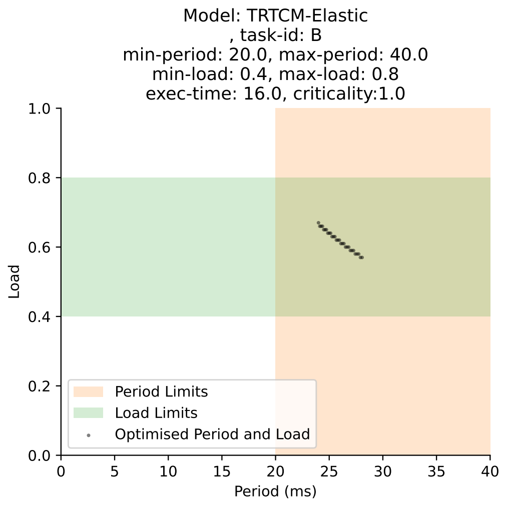


a) Task A TRTCM-Elastic

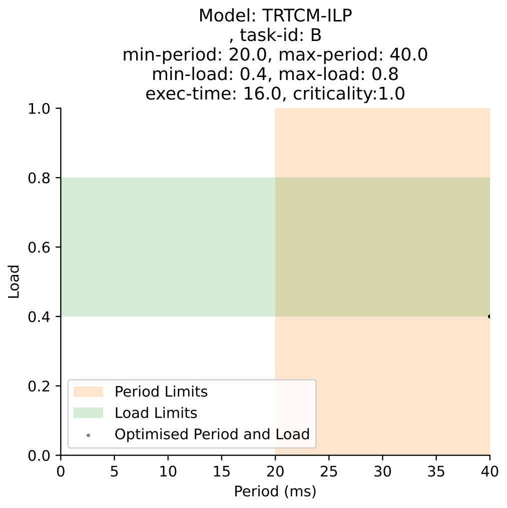


b) Task A TRTCM-ILP

Figure 7.10: Task A Degradation between TRTCM-ILP and TRTCM-Elastic under Reduced Desired Load



a) Task B TRTCM-Elastic



b) Task B TRTCM-ILP

Figure 7.11: Task B Degradation between TRTCM-ILP and TRTCM-Elastic under Reduced Desired Load

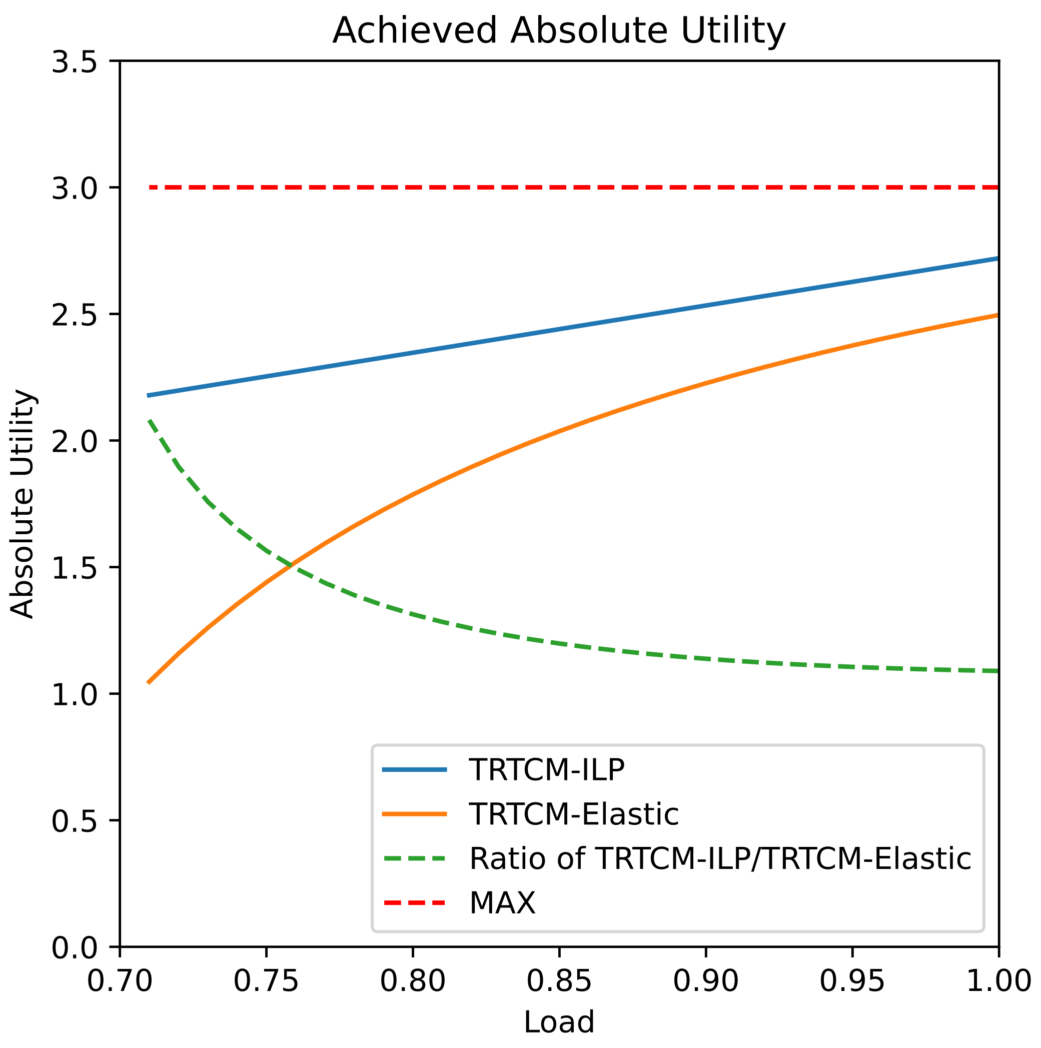
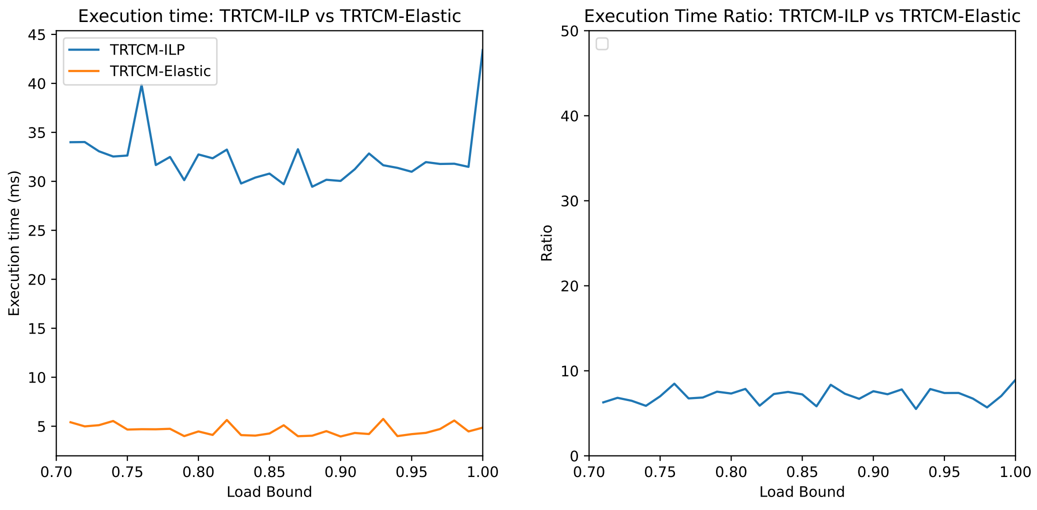


Figure 7.12: Degradation of tasks, A and B, between TRTCM-ILP and TRTCM-Elastic under reduced desired-load

Task ID	Primary Period [ms]	Tolerance Period [ms]	WCET [ms]	Relative Utility	Criticality
A	15	30	0.72	9	2
B	20	40	0.72	16	1

Table 7.9: Example: TRTCM-Elastic and TRTCM-ILP Task set



a) Execution Time b) Execution Time Ratio

Figure 7.13: Execution Time Overhead by TRTCM-ILP and TRTCM-Elastic

and the result total system utility.

7.3.3 Discussion

The more the scheduling problem gets easier, the ratio of difference decreases, and the more the problem gets harder by reducing the desired load, the ratio of difference increases. This reveals that when the problem is considered to be less complex, according to a given task set and certain desired load, it is better to find a sub-optimal solution quickly with reduced overhead using TRTCM-Elastic , than executing the linear programming solver by TRTCM-ILP for finding an optimal solution with high execution time overhead, where sub-optimal solution can deliver accepted utility with minimum cost in terms of execution time overhead.

The limitation of the current experiments is that it does not consider other forms of resource shortages or the comparison, for example, the core, failures, computing capacity, and the WCET overrun. Future work can be extending the

	TRTCM-ILP			TRTCM-Elastic			$Ratio(\tau_i) = \frac{ILP(\tau_i)}{Elastic(\tau_i)}$		$\frac{Total_{ILP}}{Total_{Elastic}}$
Load	A	B	Total	A	B	Total	Ratio A	Ratio B	Ratio Total
1.00	2.00	0.72	2.72	1.55	0.94	2.50	1.29	0.76	1.09
0.99	1.98	0.72	2.70	1.53	0.94	2.47	1.29	0.76	1.09
0.98	1.96	0.72	2.68	1.51	0.94	2.45	1.30	0.77	1.09
0.97	1.94	0.72	2.66	1.49	0.94	2.43	1.31	0.77	1.10
0.96	1.93	0.72	2.65	1.46	0.94	2.40	1.31	0.77	1.10
0.95	1.91	0.72	2.63	1.44	0.94	2.38	1.32	0.77	1.11
0.94	1.89	0.72	2.61	1.41	0.93	2.35	1.33	0.77	1.11
0.93	1.87	0.72	2.59	1.39	0.93	2.32	1.35	0.77	1.12
0.92	1.85	0.72	2.57	1.36	0.93	2.29	1.36	0.77	1.12
0.91	1.83	0.72	2.55	1.33	0.93	2.26	1.38	0.78	1.13
0.90	1.81	0.72	2.53	1.30	0.93	2.23	1.39	0.78	1.14
0.89	1.79	0.72	2.51	1.27	0.92	2.19	1.42	0.78	1.15
0.88	1.78	0.72	2.50	1.23	0.92	2.16	1.44	0.78	1.16
0.87	1.76	0.72	2.48	1.20	0.92	2.12	1.47	0.78	1.17
0.86	1.74	0.72	2.46	1.16	0.92	2.08	1.50	0.78	1.18
0.85	1.72	0.72	2.44	1.12	0.92	2.04	1.54	0.79	1.20
0.84	1.70	0.72	2.42	1.08	0.91	1.99	1.58	0.79	1.22
0.83	1.68	0.72	2.40	1.03	0.91	1.95	1.63	0.79	1.23
0.82	1.66	0.72	2.38	0.99	0.91	1.90	1.69	0.79	1.26
0.81	1.65	0.72	2.37	0.93	0.91	1.84	1.76	0.79	1.28
0.80	1.63	0.72	2.35	0.88	0.91	1.79	1.85	0.79	1.31
0.79	1.61	0.72	2.33	0.82	0.90	1.73	1.96	0.80	1.35
0.78	1.59	0.72	2.31	0.76	0.90	1.66	2.09	0.80	1.39
0.77	1.57	0.72	2.29	0.69	0.90	1.59	2.27	0.80	1.44
0.76	1.55	0.72	2.27	0.62	0.90	1.52	2.50	0.80	1.50
0.75	1.53	0.72	2.25	0.54	0.90	1.44	2.82	0.80	1.56
0.74	1.51	0.72	2.23	0.46	0.89	1.35	3.29	0.81	1.65
0.73	1.50	0.72	2.22	0.37	0.89	1.26	4.06	0.81	1.76
0.72	1.48	0.72	2.20	0.27	0.89	1.16	5.49	0.81	1.90
0.71	1.46	0.72	2.18	0.16	0.89	1.05	9.12	0.81	2.08

Table 7.10: Achieved Absolute Utility at Various Load Levels Between TRTCM-ILP and TRTCM-Elastic

	TRTCM-ILP		TRTCM-Elastic		$Ratio(\tau_i) = \frac{ILP(\tau_i)}{Elastic(\tau_i)}$	
load	A	B	A	B	Ratio A	Ratio B
1.00	15.00	40.00	27.00	24.00	0.56	1.67
0.99	15.25	40.00	27.55	24.12	0.55	1.66
0.98	15.52	40.00	28.12	24.24	0.55	1.65
0.97	15.79	40.00	28.72	24.37	0.55	1.64
0.96	16.07	40.00	29.35	24.49	0.55	1.63
0.95	16.36	40.00	30.00	24.62	0.55	1.63
0.94	16.67	40.00	30.68	24.74	0.54	1.62
0.93	16.98	40.00	31.40	24.87	0.54	1.61
0.92	17.31	40.00	32.14	25.00	0.54	1.60
0.91	17.65	40.00	32.93	25.13	0.54	1.59
0.90	18.00	40.00	33.75	25.26	0.53	1.58
0.89	18.37	40.00	34.62	25.40	0.53	1.58
0.88	18.75	40.00	35.53	25.53	0.53	1.57
0.87	19.15	40.00	36.49	25.67	0.52	1.56
0.86	19.57	40.00	37.50	25.81	0.52	1.55
0.85	20.00	40.00	38.57	25.95	0.52	1.54
0.84	20.45	40.00	39.71	26.09	0.52	1.53
0.83	20.93	40.00	40.91	26.23	0.51	1.53
0.82	21.43	40.00	42.19	26.37	0.51	1.52
0.81	21.95	40.00	43.55	26.52	0.50	1.51
0.80	22.50	40.00	45.00	26.67	0.50	1.50
0.79	23.08	40.00	46.55	26.82	0.50	1.49
0.78	23.68	40.00	48.21	26.97	0.49	1.48
0.77	24.32	40.00	50.00	27.12	0.49	1.48
0.76	25.00	40.00	51.92	27.27	0.48	1.47
0.75	25.71	40.00	54.00	27.43	0.48	1.46
0.74	26.47	40.00	56.25	27.59	0.47	1.45
0.73	27.27	40.00	58.70	27.75	0.46	1.44
0.72	28.12	40.00	61.36	27.91	0.46	1.43
0.71	29.03	40.00	64.29	28.07	0.45	1.43

Table 7.11: Optimised Period by Each Task at Various Load Levels Between TRTCM-ILP and TRTCM-Elastic

Task ID	Primary Period [ms]	Tolerance Period [ms]	WCET [ms]	Relative Utility	Criticality
A	2	3	0.3	0.5	1 (=LO)
B	4	6	0.6	0.5	1 (=LO)
C	6	9	0.2	0.5	1 (=LO)
D	8	12	0.8	0.5	1 (=LO)
E	10	15	0.3	0.7	1 (=LO)
F	12	18	0.5	0.7	2 (=HI)
G	14	21	0.5	0.7	1 (=LO)
H	16	24	0.7	0.7	2 (=HI)
I	18	27	0.3	0.7	1 (=LO)
J	20	30	0.5	0.7	2 (=HI)
K	22	33	0.9	0.7	1 (=LO)
L	30	36	0.2	0.5	2 (=HI)

Table 7.12: Example: Airbus 380 Flight Control System Mixed-criticality Task-set (to improve readability, in the text we use synonyms for the two criticality levels l: 1=LO, 2= HI)

evaluation to consider different types of resource shortages.

7.4 Experimental Evaluation of EWCET model and E-ATMP framework

7.4.1 Experimental Setup

We created a simulation tool that models task scheduling on a single processor platform. The simulator allows to definition of scenarios where specific task instances exceed the estimated WCET and projects the aborted and abandoned tasks during the transition between low and high criticality modes. Using Mahenni's research on modelling safety-critical systems using the SysML modelling language [124], show how E-ATMP shows better handling for jobs execution time overrun than AMC .

To demonstrate how AMC and E-ATMP handle low and high criticality tasks WCET overrun situations, we configured the simulator to execute the Airbus A380 task set. We intentionally induced overruns for three jobs, one job of a low criticality task and two instances of a high criticality task. Specifically, we focus on the first job ($\tau_{A,0}$) of task τ_A and the first and second jobs ($\tau_{F,0}$ and $\tau_{F,1}$) of task τ_F to examine the handling of overruns by these jobs.

7.4.2 Results and Analysis

This section presents the experiment results.

Figure 7.14.a presents the effect of optimistic-WCET by the induced overrun on low jobs under the AMC protocol. A total of 15 jobs having criticality low criticality $\tau_{i,k}.l = LO$ are abandoned from execution as a result of the effect of overrun on jobs $\tau_{A,0}, \tau_{F,0}, \tau_{F,1}$. Please note that the numerical representation of low criticality jobs is $\tau_{i,k}.l$=1 and high criticality jobs is $\tau_{i,k}.l$=2.

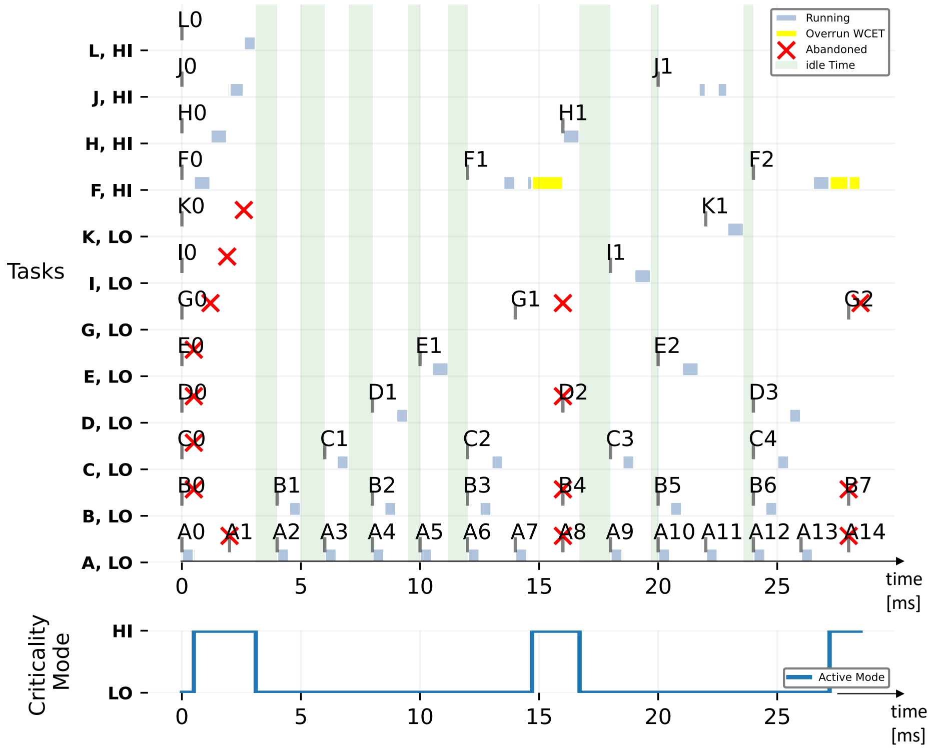
During the first occurrence of an overrun by job $\tau_{A,0}$, the AMC protocol aborted job $\tau_{A,0}$ and dropped it, along with eight low criticality $(\tau_{i,k} = LO, 1)$ jobs. It also activated the high criticality mode $(S.l = HI)$. AMC activates the $S.l = HI$ mode to prevent future releases of jobs from tasks with low criticality $\tau_i.l = LO, 1$ during this mode. The dropped jobs due to the overrun caused by $\tau_{A,0}$ include $\tau_{B,0}$, $\tau_{C,0}$, $\tau_{A,0}$, and $\tau_{E,0}$, and jobs dropped because they were released during the $S.l = HI, 2$ mode are $\tau_{G,0}$, $\tau_{I,0}$, and $\tau_{K,0}$. Similarly, during the second occurrence of an overrun by job $\tau_{F,0}$, the AMC protocol dropped four low criticality jobs and activated the $S.l = HI$ mode. The dropped jobs due to this overrun include $\tau_{A,8}$, $\tau_{B,4}$, $\tau_{D,2}$, and $\tau_{G,1}$. Finally, during the third occurrence of an overrun by job $\tau_{F,1}$, the AMC protocol dropped 8 low criticality jobs and activated the $S.l = HI$ mode. The dropped jobs due to this overrun include $\tau_{A,A14}$, $\tau_{B,7}$, and $\tau_{G,2}$.

Figure 7.14.b presents how E-ATMP minimises the effect of empiric-WCET overrun on low-criticality jobs. During the first occurrence of an overrun by job $\tau_{A,0}$, the E-ATMP protocol postponed $\tau_{A,0}$ to replace $J_{A,1}$, and without activate the $S.l = HI$ mode. Only one low criticality job dropped due to this replacement that is job $\tau_{A,1}$.

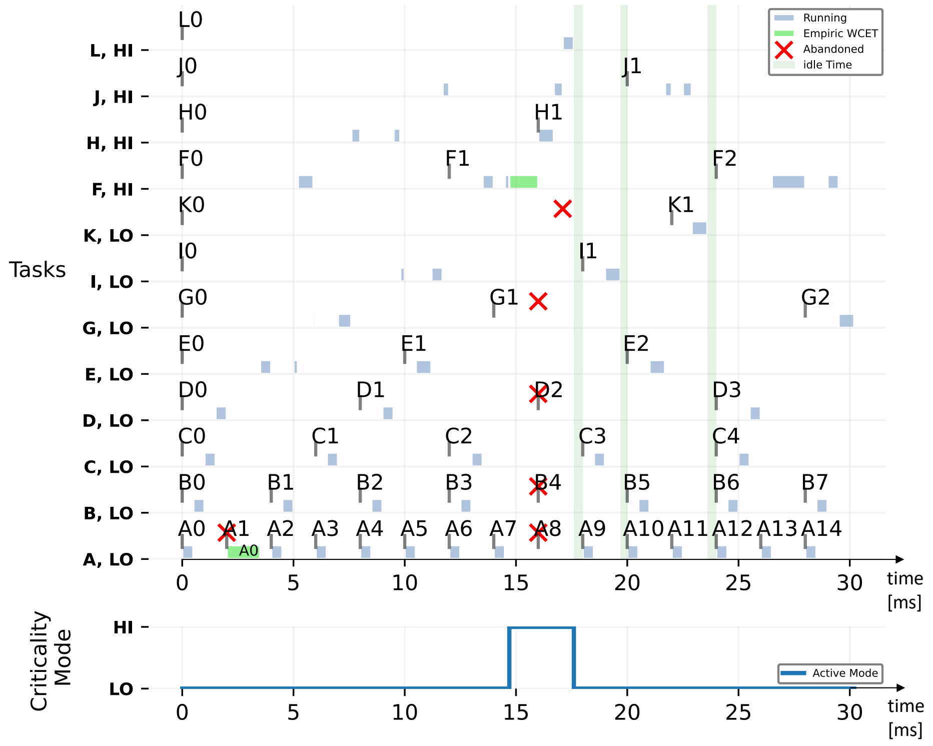
Similarly, during the second occurrence of an overrun by job $\tau_{F,0}$, the E-ATMP protocol activated the $S.l = HI$ mode and dropped five low criticality jobs. The dropped jobs due to this $\tau_{F,0}$ overrun include $\tau_{A,8}$, $\tau_{B,4}$, $\tau_{D,2}$, $\tau_{G,1}$, and $\tau_{K,0}$.

7.4.3 Discussion

This experiment introduced the concept of an empiric worst-case execution time (EWCET), which mandates a continuous update of the WCET estimate whenever an overrun of the current WCET of a task happens. The EWCET is useful in real-time systems, as it allows triggering a reconfiguration of the system with smooth degradation of services in cases of too many deadline overruns happening.



(a) AMC schedule with overrun: Overrun jobs: A0, F1, F2; Number of abandoned low jobs: 15



(b) E-ATMP schedule with overrun: Overrun jobs: A0, F1, (F2 is running the Empiric EWCET); Number of abandoned low jobs: 5

Figure 7.14: AMC and E-ATMP Schedule for Airbus Flight Control Services

However, while the idea of the EWCET is simple in itself, it is not straightforward to use, as one has to provide a scheduling framework that can take advantage of the EWCET . In particular, for the EWCET to be deployed, the scheduling framework has to allow for the extended execution of a task even after the overrun of its current WCET budget.

To demonstrate the applicability of the EWCET , we introduced in this chapter E-ATMP , which is a reconfiguration method for mixed-criticality systems to provide smooth degradation of services. E-ATMP uses the existing ATMP framework and adds additional mechanisms to facilitate the deployment of the EWCET . E-ATMP provides a powerful line of defense against the threat of WCET underestimations in a world where is increasingly difficult to provide safe and accurate WCET estimates due to the ever-increasing processor HW complexity. I use a mixed-criticality scheduling example to demonstrate that E-ATMP can provide a smooth degradation of service in case of WCET overruns.

7.5 Chapter Discussion

The criticality arithmetic aware protocols LBP-CA and ATMP-CA achieve smoother degradation than reference protocols SAMP, BP, and LBP. ATMP-CA degrades services based on their criticality, utility, adaptation, and the state of low criticality replicas. LBP-CA enables a quicker return to normal mode of operation after failures. The ATMP achieves smoother degradation by dropping services based on their criticality, utility, and adaptation. The SAMP achieves smooth degradation by dropping services based on criticality only. TRTCM-ILP optimises services for smooth degradation by formulating an `ILP` problem and solving it using LP solver to maximise system utility and subject to each service utility function and resource load constraints. TRTCM-Elastic optimises services by applying a modification for spring compression and decompression functions, to compress and decompress services load to satisfy desired resource load constraint. The EWCET aware framework is E-ATMP and the EWCET agnostic is the AMC scheme. The E-ATMP measures new WCET for services once they overrun their execution time bounds during the system mission, and provides smooth degradation by reducing the number of times the system enters high criticality mode, hence, avoiding dripping all low criticality services in case of resource shortages

7.6 Chapter Summary

This chapter presented an evaluation of the contributions during the research and development of this thesis. First, an evaluation of ATMP-CA and LBP-CA protocols for the evaluation of criticality arithmetic and dependability in Section 7.1. Second, an evaluation of SAMP and ATMP protocols for evaluating the smooth degradation between the two protocols in Section 7.2. Third, an evaluation of TRTCM-ILP and TRTCM-Elastic protocols for evaluating the smooth degradation between the two protocols in Section 7.3. Fourth, an evaluation of the E-ATMP framework for measuring and handling WCET overrun in Section 7.4.

Chapter 8

Conclusions

This chapter provides a summary and conclusion of my dissertation. Additionally, it presents an outlook on future work.

The organization of this chapter is as follows: Section 8.1 presents research findings. Section 8.2 presents contributions to knowledge. Section 8.3 presents the future work. Section 8.4 closes the thesis.

8.1 Research Questions and Findings

This thesis aimed to define and evaluate smooth degradation in mixed-criticality protocols on both uni and multi-core processor platforms in case of transient and permanent resource shortages, facilitate adaptivity in response to WCET overrun, and build highly dependable components from less dependable components.

RQ: *How to provide smooth degradation (also called graceful degradation) for integrated systems with services of mixed-criticality in case of resource shortages?*

RQ1: *What is a suitable definition of smooth degradation?*

RQ2: *What protocols are possible to grant resources to services of different criticality in case of resource shortage?*

RQ3: *How can we evaluate and compare the smooth degradation of one protocol against another?*

RQ4: *How can we provide higher levels of dependability from less dependable components?*

RQ5: *How can the system adapt based on occurrences of services' overruns of WCET estimates?*

RQ6: *What is a novel mixed-criticality scheduling protocol that provides an improvement on smooth degradation of service in case of resource shortage?*

Answer to RQ1: The first objective was to investigate different adaptation approaches from existing literature to understand these adaptive techniques for smooth degradation and summarise a concrete definition. During the research of this dissertation, I researched mixed-criticality literature and developed experiments for Mixed Criticality systems under four types of resource shortages, which are: core failures, WCET overrun, reduced computing capacity, and reduced desired/available load. Drawing a universal definition for smooth degradation is challenging. Many sources present solutions that feature smooth degradation, but a consensus for a concrete definition in the literature is absent. **Answer to RQ2:** The second objective acknowledges state-of-the-art approaches and adaptation techniques for mixed-criticality systems and identifies existing Fixed-priority-based approaches for granting resources to mixed-criticality services. The identified approaches are AMC, SAMP, ATMP, BP, and LBP. They have been studied, implemented, and used in experimental evaluation for comparing smooth degradation between mixed-criticality protocols as seen in the experimental evaluation chapter, Chapter 7, and as reference schedulers against novel approaches developed in this dissertation, ATMP-CA, LBP-CA, E-ATMP, EWCET, TRTCM-Elastic.

Answer to RQ3: The third objective was to design and implement a simulator to simulate resource shortages on mixed-criticality systems and evaluate each protocol adaptation in terms of achieved system utility, and number of successful and compromised services. This provided the method for comparing smooth degradation between different approaches. Reduced computing capacity, core failures, and WCET overrun have been simulated and investigated. Then, evaluate the smooth degradation performed between the given protocols by comparing the achieved overall system utility by each protocol, the number of successfully scheduled services, and the number of dropped or compromised services. The evaluation method for smooth degradation between ATMP and SAMP, under reduced computing capacity, is in Chapter 7 Section 7.2. Smooth degradation overhead between TRTCM-ILP and TRTCM-Elastic, under reduced desired/available l is in Chapter 7 Section 7.3. The evaluation for novel Criticality Arithmetic aware protocols developed in chapter 4: ATMP-CA, SAMP-CA, and LBP-CA, against Criticality Arithmetic agnostic protocols: SAMP, ATMP, BP, and LBP, and evaluated in Chapter 7 Section 7.1. The evaluation for novel EWCET and the E-ATMP framework developed in 5, EWCET mixed-criticality scheme, AMC, is in Chapter 7 Section 7.4.

During the research of this thesis, I implemented over 40 classes classified into modules, with a primary focus on modularity throughout development to facilitate reusability and seamless integration of various functionalities. The simulator allows experimenting with different configurations for system resources and running services by defining different types of resource shortages as mentioned above and reports the achieved overall system utility by each protocol, the number of successfully scheduled services, and the number of dropped or compromised services. Then, comparing the ratio of achieved system utility and the difference in dropped or compromised services in the reported results reveals which protocol provides smoother degradation than the other protocol. The experiments presented in this thesis provide the foundation for comparing the smooth degradation of services and systems, between mixed-criticality protocols. Chapter 6 presents the significant classes developed for this object.

Answer to RQ4:

The fourth objective aimed to establish reliable mixed-criticality systems using components that are inherently less dependable. For this objective, protocols were derived from the Criticality Arithmetic model, which itself drew inspiration from the industrial SIL arithmetic model. These protocols, namely ATMP-CA for mid-term scheduling and LBP-CA for short-term scheduling, were developed to enhance the dependability of mixed-criticality services constructed from less reliable services. ATMP-CA implements a core allocation and utility-optimization algorithm that uses information about criticality arithmetic. This way, ATMP-CA can consider the property that, in case of a core failure, the scheduler does not have to ensure that all replicated tasks of the service have to get rescheduled. Also, ATMP-CA can consider that when running multiple tasks as instances of a service with criticality arithmetic, it might be sufficient to ensure that only one task runs at the full frequency, i.e., providing maximum throughput. Therefore, ATMP-CA maximises the system utility whenever case of resource shortages occur, while providing an acceptable level of degradation for services implemented via criticality arithmetic.

LBP-CA provides quicker returns from high criticality modes, Bailout, and Recovery modes, using the information about criticality arithmetic. LBP-CA cancels the activation of the transition from Bailout to Normal mode when the lowest-priority, high-criticality job service has already been executed by another replica. As such, LBP-CA information about criticality arithmetic prevents the transition from Bailout mode to Recovery mode, which provides smoother degradation than Bailout-based schedulers BP and LBP.

The quick return to normal operation, criticality arithmetic aware allocation, and formulation for granting system resources in mixed-criticality systems enable building dependable high-criticality services, from less dependable low-criticality services. The evaluation for novel ATMP-CA and LBP-CA developed in 4 is in Chapter 7 Section 7.1.

Answer to RQ5: The fifth objective was to address the issue of WCET overruns. This involved exploring various static and dynamic analysis methods for measuring WCET in existing literature. In this thesis, a novel approach called EWCET was developed, which initialised using one of the static or dynamic estimates, and updated whenever a service exceeds the known WCET at the present and assuming it for all future arrivals of this service. Additionally, a framework was created that incorporates adaptive responses for each overrun, the E-ATMP. This framework generates a new system configuration with an acceptable level of service rather than enabling a high-criticality mode of operation in the future for the same WCET overrun. The evaluation for novel Empiric Worst-case execution time EWCET and the E-ATMP framework developed in 5, EWCET mixed-criticality scheme, AMC, is in Chapter 7 Section 7.4.

Answer to RQ6: The sixth objective is the resultant of achieving previous objectives which is the realisation of novel mixed-criticality protocols and a framework. This achievement allows for smoother degradation in the event of resource shortages while maintaining an equivalent level of safety but higher overall system utility as provided by the reference scheduler in experimental conditions. The protocols based on Criticality Arithmetic, namely ATMP-CA and LBP-CA, along with the E-ATMP time framework, contribute to enhancing the smooth degradation of services. ATMP-CA and LBP-CA improve the response to core failure, while the E-ATMP framework addresses issues related to resource shortages arising from WCET overruns.

ATMP-CA can optimise the overall system utility in case of resource shortages while providing an acceptable level of degradation for services implemented via criticality arithmetic, better than reference protocols, SAMP and ATMP. LBP-CA system transitions and mode activations are always better than or equal to the one provided by reference protocols, LBP, and BP.

EWCET and E-ATMP framework to find a stable schedule after the occurrence of WCET overruns with smooth degradation of service quality. The proposed solution is based on dynamic adaptation by smooth service degradation in case of WCET overruns due to the optimistic nature of the available WCET estimates.

8.2 Contribution to Knowledge

The research has produced three main contributions. Building highly dependable components, from less dependable components. Evaluating and improving the state-of-the-art smooth degradation for mixed-criticality scheduling protocols on uni and multi-core. Developing a framework that provides adaptive responses to WCET overrun, while improving the smooth degradation. The following summarises these novel contributions.

- I introduced a concrete model for criticality arithmetic theory, that enables realising high criticality components from low criticality components. The research found that the awareness of criticality arithmetic by mixed protocols enhances the system's dependability, and smoother degradation for free. The limitation of criticality is that it is limited to two and three criticality levels.

- I introduced the ATMP-CA protocol, which is a mid-term multi-core criticality arithmetic protocol. ATMP-CA uses information about criticality arithmetic to enable building highly dependable systems from less dependable components. ATMP-CA shows smoother degradation than compared protocols, SAMP, and ATMP. The limitation of the study is that it's based on two criticality levels, and has no solution for when several computing cores is only one. In this case, ATMP-CA has no difference against ATMP.

- I introduced the LBP-CA protocol, which is a short-term uni-core criticality arithmetic protocol. LBP-CA uses criticality arithmetic to enable building dependable systems for a quicker return to normal or low criticality mode, which improves the smooth degradation compared to the reference protocols, BP, and LBP. The limitation of this study is that it allows the drop of high criticality task replica in case another replica of the service is completed priorly.

- I introduced the EWCET model for measuring WCET overrun online. EWCET for the first job of each service gets initialised with a given WCET estimate of that service, and for each succeeding job, the EWCET is the maximum of the previous job EWCET and the current job's execution time. Therefore, any update of the EWCET is based on the maximum with the previous job's EWCET.

- To demonstrate the applicability of the EWCET, I introduced the E-ATMP framework, which is a reconfiguration method for mixed-criticality systems to provide smooth degradation for services in case of a WCET overrun occurs. E-ATMP uses the existing ATMP framework and adds additional mechanisms to facilitate the deployment of the EWCET updates during runtime. E-ATMP provides a powerful line of defense against the threat of WCET underestimations in a world where is increasingly difficult to provide safe and accurate WCET estimates due to the ever-increasing processor HW complexity.

- I evaluated results obtained from the three sets of analysis for smooth degradation of the achieved system utility between SAMP and ATMP protocols, under computing capacity shortage. I show the absolute utility achieved and the number of dropped tasks when running the task set over three cases 8, 4, and 2 cores under full and half frequency speed. Then, I compared the three cases between the two protocols, SAMP and ATMP.

- I evaluated the ATMP utility function, as defined by the TRTCM model, and integrated it with the E-MC . The evaluation is a comparison of smooth degradation between TRTCM-ILP and TRTCM-Elastic. TRTCM-Elastic is an integration of the TRTCM utility function and the E-MC . E-MC Elastic provides a heuristic function for finding an acceptable configuration for the system services but lacks the utility metrics assigned to each service as featured in TRTCM.

8.2.1 Research Significance

The concrete Criticality Arithmetic (CA) protocols in this dissertation allow the integration of mixed-criticality systems services using redundant per acr-shortseooc for building dependable mixed-criticality systems. Software Element out-of Context (SEooC) approach is used for developing Software (SW), Hardware (HW), or System elements where the developer is sure that this element will be used as a Safety element not just the context of one Safety program, but the Safety element finds use in several Safety goals or Safety requirements, and probably cuts across items and vehicles (i.e., Electronic Control Unit (ECU)). In such cases, it is much more efficient in terms of development costs and effort to integrate the same element SEooC in several programs instead of developing separate Safety elements for every program.

8.3 Future work

Alternative Methodologies: probabilistic approaches.

- **Dynamic Priority-based Scheduling:** All contributions in this thesis assume a Fixed-priority assignment-based system. Extending Criticality Arithmetic protocols to include dynamic-based priority assignment opens new questions regards smooth degradation for integrated mixed criticality systems.

- **Verifying Simulated Findings on Physical Platform** The findings in this thesis have to be verified on modern real computing platforms such as Kalray Massively Parallel Processor Array (MPPA) [125]. MPPA2-256 is the second processor of the MPPA processor family. MPPA2-256 is specially designed support to networking, storage, and high-performance embedded applications with high integrity safety, and mission-critical systems. In MPPA2-256, 256 cores organised in 16 computing clusters each containing 16 cores. The system's core operating system in each computing cluster provisions the scheduling and execution among the 16 cores.

- **Extending Criticality Arithmetic**: The ATMP-CA protocol can be extended to support systems with more than two criticality runtime operation modes, this will allow the applicability ofATMP-CA to different core allocation and ILP formulation. Also, LBP-CA can be extended to support migration to different cores with resource shortages, which may return the system to normal criticality mode quicker than the uni-core model presented in this dissertation. In this system model, BP fund is distributed among cores, which in turn better allocation for WCET budgets system services by exploiting the unused execution time from all cores in the system.

- **Extending EWCET**: Currently, EWCET is not included in the prior verification for the schedulability of system services before deployment, therefore, it relies on a framework to provision the adaptation process during WCET overrun. This aspect of EWCET can be seen as a feature from one perspective, and a limitation from another. From the feature perspective, it does not rely on specific priori verification analysis, and hence, both priority assignment algorithms, dynamic and static, can use the EWCET model either with the E-ATMP framework or any framework that enables an optimisation function or heuristics. From the limitation perspective, the predictability of system behaviour and assurance for interference-free execution for high criticality services, EWCET may not be useful in systems with tight tolerance ranges or limited solution spaces for possible configurations to the system to adapt in case of WCET overrun. Therefore developing a schedulability analysis that considers EWCET for priori verification is important and can be realised by bounding the WCET by available slack in the system.

- **Extending E-ATMP**: Different optimisation and reconfiguration algorithms can be integrated into E-ATMP. Currently, it uses the LP solver to find an optimal schedule with every WCET update in the system. The LP solver overhead is relatively high. Heuristics alike TRTCM-Elastic can find a sub-optimal schedule with small overhead for less complex problems or with resource shortages that don't require as much computation as in the LP solver.

- **Undergraduate Courses:** In addition, this thesis lays the groundwork by providing materials and tools for the design of courses centred on mixed-criticality systems for undergraduate education. The background chapter in this thesis can be expanded to encompass the course titled "Mixed-Criticality Systems Concepts and Practices". The contributions and evaluation chapters can be repurposed for a course named "Theory and Analysis for Mixed-Criticality Systems". Collaborative efforts with industry partners could further enhance the practical applicability of the courses, bridging the gap between academic knowledge and real-world implementation.

8.4 Closing

Building highly dependable components, from less dependable components, evaluating and improving smooth degradation for mixed-criticality scheduling, and developing a framework that provides adaptive responses to WCET overrun, have been presented in this thesis.

This implementation sets the stage for researchers to consider criticality arithmetic and EWCET approaches, with the implementation of future work and further technological improvements, this research can be truly transcended into a real-world solution for building highly dependable mixed-criticality systems from less dependable components, that degrade smoothly in case of resource shortage due to core-failures or WCET overrun.

Acronyms

ABS Anti-lock Braking System. 1, 46, 47

AMC Adaptive Mixed-Criticality. xii, 22, 27, 34, 37, 38, 40, 41, 47, 58, 76–79, 83, 86, 90, 99, 104, 121–124, 128–130

AMP Asymmetric Multiprocessing. 21

ARP Aerospace Recommended Practice 4654. 2

ASIL Automotive Safety Integrity Levels. 2, 45–47

ATMP Adaptive Tolerance-based Mixed-criticality Protocol. viii, xii–xiv, 9, 10, 12, 14, 36, 38, 47, 50, 52, 58, 64, 68, 70, 74, 79, 85, 86, 88, 89, 91–93, 97, 103, 104, 106–112, 124, 125, 128, 130–132

ATMP-CA Critical Arithmetic Adaptive Tolerance-based Mixed-criticality Protocol. iii, 12, 14, 36, 38, 40, 41, 45, 47, 49–51, 54, 56, 62, 85, 86, 89, 91–93, 95, 97, 124, 125, 128–131, 133

BCET Best-Case Execution Time. xi, 3, 18

BF Bailout Fund. 57

BP Bailout Protocol. 9, 14, 38, 39, 55, 58, 59, 61, 81, 87, 89, 92, 93, 97, 99–101, 128–131

CA Criticality Arithmetic. iii, 9, 12–14, 36, 38, 45, 49, 58, 62, 132

CBS Constant Bandwidth Server. 40

CPU Central Processing Unit. 4, 17, 18, 22, 56, 77

DAL Design Assurance Levels DO-178C. 2

DM Deadline Monotonic. 19, 28, 58

DVFS Dynamic Voltage and Frequency Scaling. 35, 42

E-ATMP Empiric Adaptive Tolerance-based Mixed-criticality Protocol. iii, ix, xi, xii, 10, 13, 14, 32, 40, 41, 43, 63, 64, 67–71, 74–76, 79–81, 88–91, 121–125, 128–130, 132

E-ATMP Empiric ATMP. 79

E-MC Elastic Mixed-Criticality Model. 10, 132

ECU Electronic Control Unit. 132

EDF Earliest Deadline First. 3, 19, 20, 28, 29

EDF-VD Earliest Deadline First with Virtual Deadline. 34, 40, 41

EWCET Empiric Worst Case Execution Time. iii, ix, xi, 9, 10, 12–14, 32, 33, 43, 63, 64, 66–71, 74–76, 80, 81, 87, 88, 90, 121–124, 128–132

FP Fixed-priority. 59, 64

FT Fault-tolerance. 36

FTMC Fault Tolerant Mixed Criticality Scheduling. 40

HI High Criticality. 2, 22–24, 28, 32, 33, 37, 38, 76–80

HW Hardware. 7, 45, 132

IEC International Electrotechnical Commission. 1, 46

ILP Integer Linear Programming. 14, 40, 47, 49, 52–54, 91, 133

ISO International Standardisation Organisation. 2, 45

LBP Lazy Bailout Protocol. 9, 14, 39, 55, 58, 59, 61, 81, 87, 92, 93, 97, 99, 101, 128–131

LBP-CA Criticality Arithmetic Lazy Bailout Protocol. iii, xi, xiii, 9, 14, 36, 38–41, 45, 55, 58, 59, 61, 62, 81, 87–89, 91–94, 97, 99–102, 124, 125, 128–131, 133

LO Low Criticality. 2, 22–24, 28, 33, 37, 38, 76–80

LP Linear Programming. 36

LUB Least Upper Bound. 20, 29

MC Mixed Criticality. 22, 23, 33, 38, 91

MPPA Kalray Massively Parallel Processor Array. 133

RM Rate Monotonic. 3, 19, 28, 29

RTA Response Time Analysis. 19, 37

RTSh Real-time Scheduling Theory. 2

SAMP Standard Adaptive Mixed-Criticality Protocol. viii, xi–xiv, 9, 10, 12, 14, 85, 86, 89, 91–93, 95, 97, 103–112, 124, 125, 128, 130–132

SAMP-CA Critical Arithmetic Standard Adaptive Mixed-Criticality Protocol. 14, 85, 86, 91, 128

SEooC Software Element out-of Context. 132

SIL Safety Integrity Level. 2, 45, 62, 129

SMP Symmetric Multiprocessing. 21

SS Standby-sparing. 35

SW Software. 7, 45, 132

TRTCM Tolerance-based Real-Time Computing Model. 7, 10, 12, 36, 38, 41, 42, 47, 89, 93, 113, 132

TRTCM-Elastic Elastic Tolerance-based Real-Time Computing Model. ix, xii, xiv, 10, 12, 14, 89, 90, 113–120, 124, 125, 128, 132

TRTCM-ILP Tolerance-based Real-Time Computing Model Interger Linear Programming. ix, xii, xiv, 10, 12, 14, 89, 90, 113–120, 124, 125, 128

TTA Time-Triggered Architecture. 35

TU Time/Utility. 29

UML Unified Modeling Language. 14

WCET Worst-Case Execution Time. iii, xi, xiii, 3–10, 12, 14, 17, 18, 20–25, 27, 28, 30–35, 37–39, 48, 56–58, 60, 63–68, 70, 71, 73–80, 83, 85, 87, 89–93, 97, 99, 102, 114, 118, 121, 122, 124, 125, 128, 130–132

WCRT Worst-case Response Time. 19, 41

ZSRM Zero-Slack Rate Monotonic. 40

Bibliography

[1] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckman, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenstrom, "The worst-case execution time problem - overview of methods and survey of tools," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, Apr. 2008.

[2] R. Kirner, S. Iacovelli, and M. Zolda, "Optimised adaptation of mixed-criticality systems with periodic tasks on uniform multiprocessors in case of faults," in *Proc. 11th IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS'15)*, (Auckland, New Zealand), April 2015.

[3] A. Frigerio, B. Vermeulen, and K. Goossens, "Component-level asil decomposition for automotive architectures," *Proceedings of the 2019 International Conference on Dependable Systems and Networks Workshops*, pp. 62–69, 2019.

[4] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Proc. 28th IEEE International Real-Time Systems Symposium (RTSS'07)*, pp. 239–243, Dec. 2007.

[5] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[6] S. Baruah and S. Vestal, "Schedulability analysis of sporadic tasks with multiple criticality specifications," in *2008 Euromicro Conference on Real-Time Systems*, pp. 147–155, IEEE, 2008.

[7] S. Baruah and A. Burns, "Expressing survivability considerations in mixed-criticality scheduling theory," *Journal of Systems Architecture*, vol. 109, 2020.

[8] S. K. Baruah, A. Burns, and R. I. Davis, “Response-time analysis for mixed criticality systems,” in *2011 IEEE 32nd Real-Time Systems Symposium*, pp. 34–43, IEEE, 2011.

[9] S. K. Baruah, L. Cucu-Grosjean, R. I. Davis, and C. Maiza, “Mixed criticality on multicore/manycore platforms (dagstuhl seminar 15121),” in *Dagstuhl Reports*, vol. 5, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[10] P. Puschner and C. Koza, “Calculating the maximum execution time of real-time programs,” *Real-time systems*, vol. 1, no. 2, pp. 159–176, 1989.

[11] P. Puschner and A. Burns, “” a review of worst-case execution-time analysis”; real-time systems, 18 (2000), 2/3; s. 115-128.,” *Real-Time Systems*, vol. 18, no. 2/3, pp. 115–128, 2000.

[12] R. Kirner and P. Puschner, “Classification of wcet analysis techniques,” in *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC’05)*, pp. 190–199, IEEE, 2005.

[13] S. Altmeyer, B. Lisper, C. Maiza, J. Reineke, and C. Rochange, “Wcet and mixed-criticality: What does confidence in wcet estimations depend upon?,” in *15th International Workshop on Worst-Case Execution Time Analysis (WCET 2015)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[14] H. Kopetz, *Real-time systems: design principles for distributed embedded applications.* Springer Science & Business Media, 2011.

[15] G. C. Buttazzo, G. Lipari, and L. Abeni, “Elastic task model for adaptive rate control,” in *Proc. 19th IEEE Real-Time Systems Symposium (RTSS’98)*, pp. 286–295, Dec. 1998.

[16] E. D. Jensen, C. Locke, and H. Tokuda, “A time-driven scheduling model for real-time systems,” in *Proc. IEEE Real-Time Systems Symposium*, pp. 112–122, IEEE, Dec. 1985.

[17] E. D. Jensen, “Asynchronous decentralized realtime computer systems,” in *Real Time Computing*, pp. 347–371, Springer, 1994.

[18] R. Kirner, S. Iacovelli, and M. Zolda, “Optimised adaptation of mixed-criticality systems with periodic tasks on uniform multiprocessors in case of faults,” in *2015 IEEE International Symposium on*

Object/Component/Service-Oriented Real-Time Distributed Computing Workshops, pp. 17–25, IEEE, 2015.

[19] A. Avižienis, "Fault-tolerance and fault-intolerance: Complementary approaches to reliable computing," in *ACM SIGPLAN Notices*, vol. 10, pp. 458–464, ACM, 1975.

[20] A. Burns, R. I. Davis, S. Baruah, and I. Bate, "Robust mixed-criticality systems," *IEEE Transactions on Computers*, vol. 67, pp. 1478–1491, Oct 2018.

[21] C. P. Shelton, *Scalable graceful degradation for distributed embedded systems.* PhD thesis, Carnegie Mellon University, 2003.

[22] M. Catherine, I. Saverio, and K. Raimund, "Analysis for systems modelled in Matlab/Simulinkusing sil arithmetic to design safe and secure systems," in *Proc. 23rd IEEE International Symposium on Real-time Distributed Computing*, pp. 213–218, IEEE (2020), May 2020.

[23] R. Kirner, "A uniform model for tolerance-based real-time computing," in *Proc. 17th IEEE Int'l Symposium on Object/Component/Service-oriented Real-Time Distributed Computing*, (Reno, Nevada, USA), pp. 9–16, June 2014.

[24] A. Burns and R. I. Davis, "Mixed criticality systems-a review:(february 2022)," *Department of Computer Science, University of York, Tech. Rep*, 2022.

[25] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," in *RTSS*, vol. 89, pp. 166–171, 1989.

[26] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM computing surveys (CSUR)*, vol. 43, no. 4, p. 35, 2011.

[27] P. K. Harter Jr, "Response times in level-structured systems," *ACM Transactions on Computer Systems (TOCS)*, vol. 5, no. 3, pp. 232–248, 1987.

[28] P. K. Harter Jr, "Response times in level-structured systems," *Technical Report CU-CS-269-94*, 1987.

[29] L. Sha, T. Abdelzaher, K.-E. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok, "Real time scheduling theory: A historical perspective," *Real-time systems*, vol. 28, pp. 101–155, 2004.

[30] M. Pandya and M. Malek, "Minimum achievable utilization for fault-tolerant processing of periodic tasks," *IEEE Transactions on Computers*, vol. 47, no. 10, pp. 1102–1112, 1998.

[31] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings, "Hard real-time scheduling: The deadline-monotonic approach," *IFAC Proceedings Volumes*, vol. 24, no. 2, pp. 127–132, 1991.

[32] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal*, vol. 29, no. 5, pp. 390–395, 1986.

[33] M. S. Fineberg and O. Serlin, "Multiprogramming for hybrid computation," in *Proceedings of the November 14-16, 1967, fall joint computer conference*, pp. 1–13, 1967.

[34] M. Spuri and G. Buttazzo, "Scheduling aperiodic tasks in dynamic priority systems," *Real-Time Systems*, vol. 10, no. 2, pp. 179–210, 1996.

[35] R. Davis and A. Burns, "A survey of hard real-time scheduling algorithms and schedulability analysis techniques for multiprocessor systems,‖ university of york, department of computer science," tech. rep., Tech. Rep. YCS-2009-443, 2009.

[36] D. De Niz, K. Lakshmanan, and R. Rajkumar, "On the scheduling of mixed-criticality real-time task sets," in *Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE*, pp. 291–300, Citeseer, 2009.

[37] L. Sha, J. P. Lehoczky, and R. Rajkumar, "Solutions for some practical problems in prioritized preemptive scheduling.," in *Unknown Host Publication Title*, pp. 181–191, IEEE, 1986.

[38] N. C. Audsley, *Optimal priority assignment and feasibility of static priority tasks with arbitrary start times.* Citeseer, 1991.

[39] N. C. Audsley, "On priority asignment in fixed priority scheduling," *Information Processing Letters*, vol. 79, no. 1, pp. 39–44, 2001.

[40] V. Izosimov, P. Pop, P. Eles, and Z. Peng, "Scheduling of fault-tolerant embedded systems with soft and hard timing constraints," in *Proceedings of the conference on Design, automation and test in Europe*, pp. 915–920, 2008.

[41] S. Baruah, V. Bonifaci, G. d'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie, "Scheduling real-time mixed-criticality jobs," *IEEE Transactions on Computers*, vol. 61, no. 8, pp. 1140–1152, 2011.

[42] Z. Al-bayati, J. Caplan, B. H. Meyer, and H. Zeng, "A four-mode model for efficient fault-tolerant mixed-criticality systems," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 97–102, IEEE, 2016.

[43] S. Iacovelli, R. Kirner, and C. Menon, "Atmp: An adaptive tolerance-based mixed-criticality protocol for multi-core systems," in *2018 IEEE 13th International Symposium on Industrial Embedded Systems (SIES)*, pp. 1–9, IEEE, 2018.

[44] N. Chen, S. Zhao, I. Gray, A. Burns, S. Ji, and W. Chang, "Msrp-ft: Reliable resource sharing on multiprocessor mixed-criticality systems," in *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 201–213, IEEE, 2022.

[45] Q. Zhao, Z. Al-Bayati, Z. Gu, and H. Zeng, "Optimized implementation of multirate mixed-criticality synchronous reactive models," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 22, no. 2, pp. 1–25, 2016.

[46] I. Bate, A. Burns, and R. I. Davis, "A bailout protocol for mixed criticality systems," in *Real-Time Systems (ECRTS), 2015 27th Euromicro Conference on*, pp. 259–268, IEEE, 2015.

[47] I. Bate, A. Burns, and R. I. Davis, "An enhanced bailout protocol for mixed criticality embedded software," *IEEE Transactions on Software Engineering*, vol. 43, no. 4, pp. 298–320, 2017.

[48] L. Pautet, T. Robert, and S. Tardieu, "Litmus-rt plugins for global static scheduling of mixed criticality systems," *Journal of Systems Architecture*, vol. 118, p. 102221, 2021.

[49] D. Liu, J. Spasic, N. Guan, G. Chen, S. Liu, T. Stefanov, and W. Yi, "Edf-vd scheduling of mixed-criticality systems with degraded quality guarantees," in *2016 IEEE Real-Time Systems Symposium (RTSS)*, pp. 35–46, IEEE, 2016.

[50] X. Gu and A. Easwaran, "Dynamic budget management with service guarantees for mixed-criticality systems," in *2016 IEEE Real-Time Systems Symposium (RTSS)*, pp. 47–56, IEEE, 2016.

[51] H. Silva, M. Vieira, and A. Neto, "Are safety-critical systems really survivable to attacks?," in *2023 IEEE International Systems Conference (SysCon)*, pp. 1–8, IEEE, 2023.

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

[53] A. K.-L. Mok, *Fundamental design problems of distributed systems for the hard-real-time environment.* PhD thesis, Massachusetts Institute of Technology, 1983.

[54] R. Kirner, *Extending optimising compilation to support worst-case execution time analysis: von Raimund Kirner.* PhD thesis, Citeseer, 2003.

[55] J. Souyris, E. Le Pavec, G. Himbert, G. Borios, V. Jégu, and R. Heckmann, "Computing the worst case execution time of an avionics program by abstract interpretation," in *5th International Workshop on Worst-Case Execution Time Analysis (WCET'05)*, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2007.

[56] L. Tan, "The worst-case execution time tool challenge 2006," *International journal on software tools for technology transfer*, vol. 11, pp. 133–152, 2009.

[57] R. Kirner, P. Puschner, and A. Prantl, "Transforming flow information during code optimization for timing analysis," *Real-Time Systems*, vol. 45, pp. 72–105, 2010.

[58] P. K. Saraswat, P. Pop, and J. Madsen, "Task migration for fault-tolerance in mixed-criticality embedded systems," *ACM SIGBED Review*, vol. 6, no. 3, pp. 1–5, 2009.

[59] A. Thekilakkattil, R. Dobrin, S. Punnekkat, and H. Aysan, "Optimizing the fault tolerance capabilities of distributed real-time systems," in *2009 IEEE Conference on Emerging Technologies & Factory Automation*, pp. 1–4, IEEE, 2009.

[60] G. Liu, Y. Lu, and S. Wang, "An efficient fault recovery algorithm in multiprocessor mixed-criticality systems," in *2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, pp. 2006–2013, IEEE, 2013.

[61] R. M. Pathan, "Fault-tolerant and real-time scheduling for mixed-criticality systems," *Real-Time Systems*, vol. 50, no. 4, pp. 509–547, 2014.

[62] M. Lindgren, H. Hansson, and H. Thane, "Using measurements to derive the worst-case execution time," in *Proceedings Seventh International Conference on Real-Time Computing Systems and Applications*, pp. 15–22, IEEE, 2000.

[63] G. von der Brüggen, K.-H. Chen, W.-H. Huang, and J.-J. Chen, "Systems with dynamic real-time guarantees in uncertain and faulty execution environments," in *2016 IEEE Real-Time Systems Symposium (RTSS)*, pp. 303–314, IEEE, 2016.

[64] J. Caplan, Z. Al-Bayati, H. Zeng, and B. H. Meyer, "Mapping and scheduling mixed-criticality systems with on-demand redundancy," *IEEE Transactions on Computers*, vol. 67, no. 4, pp. 582–588, 2018.

[65] G. von der Brüggen, L. Schönberger, and J.-J. Chen, "Do nothing, but carefully: Fault tolerance with timing guarantees for multiprocessor systems devoid of online adaptation," in *2018 IEEE 23rd Pacific Rim International Symposium on Dependable Computing (PRDC)*, pp. 1–10, IEEE, 2018.

[66] S. Safari, M. Ansari, G. Ershadi, and S. Hessabi, "On the scheduling of energy-aware fault-tolerant mixed-criticality multicore systems with service guarantee exploration," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 10, pp. 2338–2354, 2019.

[67] F. Reghenzani, Z. Guo, L. Santinelli, and W. Fornaciari, "A mixed-criticality approach to fault tolerance: integrating schedulability and failure requirements," in *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 27–39, IEEE, 2022.

[68] F. Reghenzani and W. Fornaciari, "Mixed-criticality with integer multiple wcets and dropping relations: new scheduling challenges," in *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, pp. 320–325, 2023.

[69] H. Li, I. Puaut, and E. Rohou, "Traceability of flow information: Reconciling compiler optimizations and wcet estimation," in *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*, pp. 97–106, 2014.

[70] A. Prantl, *High-level compiler support for timing analysis.* PhD thesis, Vienna University of Technology, 2010.

[71] I. Wenzel, R. Kirner, B. Rieder, and P. Puschner, "Measurement-based timing analysis," in *Leveraging Applications of Formal Methods, Verification and Validation: Third International Symposium, ISoLA 2008, Porto Sani, Greece, October 13-15, 2008. Proceedings 3*, pp. 430–444, Springer, 2008.

[72] B. Lisper and M. Santos, "Model identification for wcet analysis," in *2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 55–64, IEEE, 2009.

[73] J. Nowotsch, M. Paulitsch, A. Henrichsen, W. Pongratz, and A. Schacht, "Monitoring and wcet analysis in cots multi-core-soc-based mixed-criticality systems," in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–5, IEEE, 2014.

[74] L. Cagnizi, F. Reghenzani, and W. Fornaciari, "Run-time dynamic wcet estimation," in *Proceedings of the 8th ACM/IEEE Conference on Internet of Things Design and Implementation*, pp. 458–460, 2023.

[75] Z. Jiang, S. Zhao, P. Dong, D. Yang, R. Wei, N. Guan, and N. Audsley, "Re-thinking mixed-criticality architecture for automotive industry," in *2020 IEEE 38th International Conference on Computer Design (ICCD)*, pp. 510–517, IEEE, 2020.

[76] T. Kadeed, B. Nikolic, and R. Ernst, "Safe online reconfiguration of mixed-criticality real-time systems," in *2020 IEEE 25th Pacific Rim International Symposium on Dependable Computing (PRDC)*, pp. 140–149, IEEE, 2020.

[77] F. Santy, L. George, P. Thierry, and J. Goossens, "Relaxing mixed-criticality scheduling strictness for task sets scheduled with fp," in *2012 24th Euromicro Conference on Real-Time Systems*, pp. 155–165, IEEE, 2012.

[78] S. Draskovic, R. Ahmed, P. Huang, and L. Thiele, "Schedulability of probabilistic mixed-criticality systems," *Real-Time Systems*, vol. 57, pp. 397–442, 2021.

[79] X. Dong, G. Chen, M. Lv, W. Pang, and W. Yi, "Flexible mixed-criticality scheduling with dynamic slack management," *Journal of Circuits, Systems and Computers*, vol. 30, no. 10, p. 2150306, 2021.

[80] I. Koren and C. M. Krishna, *Fault-tolerant systems.* Morgan Kaufmann, 2020.

[81] H. Aydin, "Exact fault-sensitive feasibility analysis of real-time tasks," *IEEE Transactions on Computers*, vol. 56, no. 10, pp. 1372–1386, 2007.

[82] P. Huang, H. Yang, and L. Thiele, "On the scheduling of fault-tolerant mixed-criticality systems," in *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE*, pp. 1–6, IEEE, 2014.

[83] J. Zhou, M. Yin, Z. Li, K. Cao, J. Yan, T. Wei, M. Chen, and X. Fu, "Fault-tolerant task scheduling for mixed-criticality real-time systems," *Journal of Circuits, Systems and Computers*, vol. 26, no. 01, p. 1750016, 2017.

[84] A. Thekkilakattil, R. Dobrin, and S. Punnekkat, "Fault tolerant scheduling of mixed criticality real-time tasks under error bursts," *Procedia Computer Science*, vol. 46, pp. 1148–1155, 2015.

[85] M. Zhao, D. Liu, X. Jiang, W. Liu, G. Xue, C. Xie, Y. Yang, and Z. Guo, "Cass: Criticality-aware standby-sparing for real-time systems," *Journal of Systems Architecture*, vol. 100, p. 101661, 2019.

[86] S. Safari, S. Hessabi, and G. Ershadi, "Less-mics: A low energy standby-sparing scheme for mixed-criticality systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 4601–4610, 2020.

[87] A. Naghavi, S. Safari, and S. Hessabi, "Tolerating permanent faults with low-energy overhead in multicore mixed-criticality systems," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 2, pp. 985–996, 2021.

[88] F. Kempf and J. Becker, "Leveraging adaptive redundancy in multi-core processors for realizing adaptive fault tolerance in mixed-criticality systems," in *2023 12th Mediterranean Conference on Embedded Computing (MECO)*, pp. 1–5, IEEE, 2023.

[89] L. Behera, "A fault-tolerant time-triggered scheduling algorithm of mixed-criticality systems," *Computing*, pp. 1–23, 2022.

[90] B. Ranjbar, A. Hosseinghorban, M. Salehi, A. Ejlali, and A. Kumar, "Toward the design of fault-tolerance-aware and peak-power-aware multicore mixed-criticality systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 5, pp. 1509–1522, 2021.

[91] A. Thekkilakattil, R. Dobrin, and S. Punnekkat, "Mixed criticality scheduling in fault-tolerant distributed real-time systems," in *Embedded Systems (ICES), 2014 International Conference on*, pp. 92–97, IEEE, 2014.

[92] S. Safari, H. Khdr, P. Gohari-Nazari, M. Ansari, S. Hessabi, and J. Henkel, "Therma-mics: Thermal-aware scheduling for fault-tolerant mixed-criticality systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 7, pp. 1678–1694, 2021.

[93] J. Choi, H. Yang, and S. Ha, "Optimization of fault-tolerant mixed-criticality multi-core systems with enhanced wcrt analysis," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 24, no. 1, pp. 1–26, 2018.

[94] S. Fadlelseed, R. Kirner, and C. Menon, "Atmp-ca: Optimising mixed-criticality systems considering criticality arithmetic," *Electronics*, vol. 10, no. 11, p. 1352, 2021.

[95] S. Fadlelseed, R. Kirner, and C. Menon, "Lbp-ca: A short-term scheduler with criticality arithmetic," in *Proceedings of Abstracts, School of Physics, Engineering and Computer Science Research Conference 2022*, University of Hertfordshire, 2022.

[96] S. Iacovelli, R. Kirner, and C. Menon, “ATMP: An adaptive tolerance-based mixed-criticality protocol for multi-core systems,” in *Proc. 13th International Symposium on Industrial Embedded Systems (SIES’18)*, (Graz, Austria), June 2018.

[97] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings, “Applying new scheduling theory to static priority pre-emptive scheduling,” *Software engineering journal*, vol. 8, no. 5, pp. 284–292, 1993.

[98] S. Iacovelli and R. Kirner, “A lazy bailout approach for dual-criticality systems on uniprocessor platforms,” *Designs*, vol. 3, Feb. 2019. Special Issue: Challenges and Directions Forward for Dealing with the Complexity of Future Smart Cyber–Physical Systems.

[99] H. Su and D. Zhu, “An elastic mixed-criticality task model and its scheduling algorithm,” in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 147–152, IEEE, 2013.

[100] H. Su, D. Zhu, and D. Mossé, “Scheduling algorithms for elastic mixed-criticality tasks in multicore systems,” in *2013 IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 352–357, IEEE, 2013.

[101] H. Su, N. Guan, and D. Zhu, “Service guarantee exploration for mixed-criticality systems,” in *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 1–10, IEEE, 2014.

[102] C. Gill, J. Orr, and S. Harris, “Supporting graceful degradation through elasticity in mixed-criticality federated scheduling,” in *Proc. 6th Workshop on Mixed Criticality Systems (WMC), RTSS*, pp. 19–24, 2018.

[103] A. Taherin, M. Salehi, and A. Ejlali, “Stretch: Exploiting service level degradation for energy management in mixed-criticality systems,” in *2015 CSI Symposium on Real-Time and Embedded Systems and Technologies (RTEST)*, pp. 1–8, IEEE, 2015.

[104] A. Taherin, M. Salehi, and A. Ejlali, “Reliability-aware energy management in mixed-criticality systems,” *IEEE Transactions on Sustainable Computing (SUSC)*, vol. 3, no. 3, pp. 195–208, 2018.

[105] J. Orr, C. Gill, K. Agrawal, S. Baruah, C. Cianfarani, P. Ang, and C. Wong, “Elasticity of workloads and periods of parallel real-time tasks,” in *Proceedings of the 26th International Conference on Real-Time Networks and Systems (RTNS)*, pp. 61–71, 2018.

[106] J. Orr, C. Gill, K. Agrawal, J. Li, and S. Baruah, “Elastic scheduling for parallel real-time systems,” *Leibniz Transactions on Embedded Systems (LITES)*, vol. 6, no. 1, pp. 05–1, 2019.

[107] J. Boudjadar, S. Ramanathan, A. Easwaran, and U. Nyman, “Combining task-level and system-level scheduling modes for mixed criticality systems,” in *2019 IEEE/ACM 23rd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pp. 1–10, IEEE, 2019.

[108] J. Orr and S. Baruah, “Multiprocessor scheduling of elastic tasks,” in *Proceedings of the 27th International Conference on Real-Time Networks and Systems (RTNS)*, pp. 133–142, 2019.

[109] J. Orr, J. C. Uribe, C. Gill, S. Baruah, K. Agrawal, S. Dyke, A. Prakash, I. Bate, C. Wong, and S. Adhikari, “Elastic scheduling of parallel real-time tasks with discrete utilizations,” in *Proceedings of the 28th international conference on real-time networks and systems (RTNS)*, pp. 117–127, 2020.

[110] S. Baruah, A. Easwaran, and Z. Guo, “Mc-fluid: simplified and optimally quantified,” in *2015 IEEE Real-Time Systems Symposium*, pp. 327–337, IEEE, 2015.

[111] R. M. Pathan, *Three aspects of real-time multiprocessor scheduling: Timeliness, fault tolerance, mixed criticality.* Chalmers Tekniska Hogskola (Sweden), 2012.

[112] P. Huang, P. Kumar, G. Giannopoulou, and L. Thiele, “Energy efficient dvfs scheduling for mixed-criticality systems,” in *Proceedings of the 14th International Conference on Embedded Software*, p. 11, ACM, 2014.

[113] S.-H. Kang, H.-w. Park, S. Kim, H. Oh, and S. Ha, “Optimal checkpoint selection with dual-modular redundancy hardening,” *IEEE Transactions on Computers*, vol. 64, no. 7, pp. 2036–2048, 2014.

[114] Z. Al-bayati, B. H. Meyer, and H. Zeng, “Fault-tolerant scheduling of multicore mixed-criticality systems under permanent failures,” in *2016 IEEE*

international symposium on defect and fault tolerance in VLSI and nanotechnology systems (DFT), pp. 57–62, IEEE, 2016.

[115] L. Zeng, P. Huang, and L. Thiele, "Towards the design of fault-tolerant mixed-criticality systems on multicores," in *Proceedings of the international conference on compilers, architectures and synthesis for embedded systems*, pp. 1–10, 2016.

[116] S. Safari, M. Ansari, G. Ershadi, and S. Hessabi, "On the scheduling of energy-aware fault-tolerant mixed-criticality multicore systems with service guarantee exploration," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 30, no. 10, pp. 2338–2354, 2019.

[117] B. Ranjbar, B. Safaei, A. Ejlali, and A. Kumar, "Fantom: Fault tolerant task-drop aware scheduling for mixed-criticality systems," *IEEE access*, vol. 8, pp. 187232–187248, 2020.

[118] G. Chen, N. Guan, K. Huang, and W. Yi, "Fault-tolerant real-time tasks scheduling with dynamic fault handling," *Journal of Systems Architecture*, vol. 102, p. 101688, 2020.

[119] H. Sobhani, S. Safari, J. Saber-Latibari, and S. Hessabi, "Realism: Reliability-aware energy management in multi-level mixed-criticality systems with service level degradation," *Journal of Systems Architecture*, vol. 117, p. 102090, 2021.

[120] International standards Organisation, "Iso26262: Road vehicles – functional safety." ISO/DIS standard 26262, Nov 2011.

[121] International Electrotechnical Commission, "Functional safety of electrical / electronic / programmable electronic safety-related systems." IEC standard 61508, 1998.

[122] J. Y.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance evaluation*, vol. 2, no. 4, pp. 237–250, 1982.

[123] S. Baruah, A. Burns, and R. Davis, "Response-time analysis for mixed criticality systems," in *Proc. 32nd Real-Time Systems Symposium (RTSS)*, pp. 34–43, IEEE, 2011.

[124] F. Mhenni, J.-Y. Choley, N. Nguyen, and C. Frazza, “Flight control system modeling with sysml to support validation, qualification and certification,” *IFAC-PapersOnLine*, vol. 49, no. 3, pp. 453–458, 2016.

[125] B. D. de Dinechin, P. G. de Massas, G. Lager, C. Léger, B. Orgogozo, J. Reybert, and T. Strudel, “A distributed run-time environment for the kalray mppa®-256 integrated manycore processor,” *Procedia Computer Science*, vol. 18, pp. 1654–1663, 2013.