Optimization and Analysis for Dependable Application Software on Unreliable Hardware Platforms

Dissertation

zur Erlangung des Grades eines

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

der Technischen Universität Dortmund
an der Fakultät für Informatik

von

Kuan-Hsun Chen

Dortmund

2019

Tag der mündlichen Prüfung:	22. Mai 2019
Dekan / Dekanin:	Prof. Dr. Gernot A. Fink
Gutachter / Gutachterinnen:	Prof. Dr. Jian-Jia Chen
	Prof. Dr. Rolf Ernst

Acknowledgments

First and foremost I would like to sincerely appreciate my advisor (Doktorvater) Prof. Dr. Jian-Jia Chen for his supervision, support, and motivation over the past six years. He never hesitated to share his vast knowledge and precious time to me throughout this long journey. Without his guidance and trust, I would not have been possible to get this far in this Ph.D. study. I would also like to thank Prof. Dr. Rolf Ernst for his commitment to be the second reviewer of my dissertation. Special thanks to Prof. Dr. Peter Marwedel for his scientific feedback and mentoring.

The research leading to this thesis has received funding partly from the Deutsche Forschungsgemeinschaft (DFG) as part of the priority program "Dependable Embedded Systems" (SPP 1500) - Generating and Executing Dependable Application Software on UnReliable Embedded Systems (GetSURE), and the Collaborative Research Center SFB876 - Resource optimizing real time analysis of artifactious image sequences for the detection of nano objects - Project B2. Especially, I would like to thank all my project collaborators in GetSURE, Prof. Dr. Muhammad Shafique, Prof. Dr. Semmen Rehman, and Florian Kribel for their helpful ideas and extensive data.

Moreover, I would like to thank several people from the LS12 respectively for their helpful - professional or personal - funny conversations and technical supports: Alexander, Anas, Boguslaw, Björn, Ching-Chi, Claudia, Helena, Hendrik, HuanFui, Horst, Ingo, Jan, Junjie, Kevin, Lars, Lea, Marco, Markus, Micheal, Mikail, Niklas, Olaf, Sebastian, Ulrich, and Wei. Special thanks go out to Georg von der Brüggen. Thank you for all the time and efforts you spent for me while I often took days off right before several deadlines unintendedly.

Last but certainly not least, I would like to thank my lovely family - especially my parents for having supported in the first place, my sister for her encouragement, and my beloved wife Yen-Jen, for her unconditional love, and for everything else. Without her caring and support this would not have been possible.

List of publications

Parts of this thesis are published in the following international workshop, conference, and peer-reviewed journal papers:

Kuan-Hsun Chen, Jian-Jia Chen, Florian Kriebel, Semeen Rehman, Muhammad Shafique, and Jörg Henkel. *Reliability-Aware Task Mapping on Many-Cores with Performance Heterogeneity.* ESWEEK Workshop on 1st International Workshop on Resiliency in Embedded Electronic Systems, pages 10-11, Amsterdam, The Netherlands, October 2015. [KCK+15]

Kuan-Hsun Chen, Björn Bönninghoff, Jian-Jia Chen, and Peter Marwedel. *Compensate or ignore? meeting control robustness requirements through adaptive soft-error handling.* Proceedings of the 17th ACM SIGPLAN/SIGBED Conference on Languages, Compilers, Tools, and Theory for Embedded Systems (LCTES), pages 82-91, Santa Barbara, CA, USA, June 2016. doi: 10.1145/2907950.2907952 [KBC+16]

Kuan-Hsun Chen, Jian-Jia Chen, Florian Kriebel, Semeen Rehman, Muhammad Shafique, and Jörg Henkel. *Task Mapping for Redundant Multithreading in Multi-Cores with Reliability and Performance Heterogeneity.* In IEEE Transactions on Computers, vol. 65, no. 11, pp. 3441-3455, 1 Nov. 2016. doi: 10.1109/TC.2016.2532862 doi: 10.1109/TC.2016.2532862 [KCK+16]

Kuan-Hsun Chen, Georg Von Der Brüggen, and Jian-Jia Chen. *Overrun Handling for Mixed-Criticality Support in RTEMS.* Proceedings of Workshop on Mixed Criticality Systems (WMC). Porto, Portugal, November 2016. [KBC16]

Kuan-Hsun Chen and Jian-Jia Chen. *Probabilistic schedulability tests for uniprocessor fixed-priority scheduling under soft errors.* Proceedings of the 12th IEEE International Symposium on Industrial Embedded Systems (SIES), Toulouse, 2017, pp. 1-8. doi: 10.1109/SIES.2017.7993392 [KC17]

Kuan-Hsun Chen, Georg Von Der Brüggen, Jian-Jia Chen. *Reliability Optimization on Multi-Core Systems with Multi-Tasking and Redundant Multi-Threading.* In IEEE Transactions on Computers, vol. 67, no. 4, pp. 484-497,1 April 2018. doi:

10.1109/TC.2017.2769044 [KBC18b]

Mikail Yayla, **Kuan-Hsun Chen**, and Jian-Jia Chen. *Fault Tolerance on Control Applications: Empirical Investigations of Impacts from Incorrect Calculations*. Proceedings of the 4th International Workshop on Emerging Ideas and Trends in Engineering of Cyber-Physical Systems (EITEC), Porto, Portugal, April 2018. doi: 10.1109/EITEC.2018.00008 [YKC18]

Georg von der Brüggen, Nico Piatkowski, **Kuan-Hsun Chen**, Jian-Jia Chen, and Katharina Morik, *Efficiently Approximating the Probability of Deadline Misses in Real-Time Systems*. Proceedings of the 30th Euromicro Conference on Real-Time Systems (ECRTS), Barcelona, Spain, July 2018. doi: 10.4230/LIPIcs.ECRTS.2018.6 [BPK+18]

Kuan-Hsun Chen, Georg Von Der Brüggen, and Jian-Jia Chen. *Analysis of Deadline Miss Rates for Uniprocessor Fixed-Priority Scheduling*. Proceedings of the 24th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Hakodate, Japan, August 2018. doi: 10.1109/RTCSA.2018.00028 [KBC18a]

Kuan-Hsun Chen, Niklas Ueter, Georg Von Der Brüggen, and Jian-Jia Chen. *Efficient Computation of Deadline-Miss Probability and Potential Pitfalls*. Proceedings of the ACM/IEEE Conference of Design Automation and Test in Europe (DATE), Florence, Italy, March 2019. doi:10.23919/DATE.2019.8714908 [KUB+19]

The following publications, which are parts of my Ph.D. research, are not included in this dissertation, since the topics of them are out of the scope from the dissertation:

Muhammad Shafique, Philip Axer, Christoph Borchert, Jian-Jia Chen, **Kuan-Hsun Chen**, Björn Döbel, Rolf Ernst, Hermann Härtig, Andreas Heinig, Rüdiger Kapitza, Florian Kriebel, Daniel Lohmann, Peter Marwedel, Semeen Rehman, Florian Schmoll, Olaf Spinczyk. *Multi-layer software reliability for unreliable hardware*. Special Issue: Dependable Embedded Systems it - Information Technology, 57.3 (2015): 170-180. Oct. 2017. [SAB+15]

Semeen Rehman, **Kuan-Hsun Chen**, Florian Kriebel, Anas Toma, Muhammad Shafique, Jian-Jia Chen, and Jörg Henkel. *Cross-Layer Software Dependability on Unreliable Hardware*. In IEEE Transactions on Computers, vol. 65, no. 1, pp. 80-94, Jan. 1 2016. doi: 10.1109/TC.2015.2417554 [RKK+16]

Georg Von Der Brüggen, **Kuan-Hsun Chen**, Wen-Hung Huang, and Jian-Jia Chen. *Systems with Dynamic Real-Time Guarantees in Uncertain and Faulty Execution Environments*. Proceedings of the IEEE Real-Time Systems Symposium

(RTSS), pages 303-314, Porto, Portugal, November 2016. doi: 10.1109/RTSS.2016.037. [BKH+16]

Junjie Shi, **Kuan-Hsun Chen**, Shuai Zhao, Wen-Hung Huang, Jian-Jia Chen, and Andy Wellings. *Implementation and Evaluation of Multiprocessor Resource Synchronization Protocol (MrsP) on LITMUS^{RT}*. Proceedings of Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT). Dubrovnik, Croatia. June 2017. [SKB+17]

Zheng Dong, Cong Liu, Soroosh Bateni, **Kuan-Hsun Chen**, Junjie Shi, Jian-Jia Chen, Georg Von Der Brüggen, James Anderson. *Shared-Resource-Centric Limited Preemptive Scheduling: A Comprehensive Study of Suspension-based Partitioning Approaches*. Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS). Porto, Portugal. April 2018. doi: 10.1109/RTAS.2018.00026 [DLB+18]

Nils Hölscher, **Kuan-Hsun Chen**, Georg Von Der Brüggen, Jian-Jia Chen. *Examining and Supporting Multi-Tasking in EV3OSEK*. Proceedings of Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT). Barcelona, Spain. July 2018. [HKB+18]

Lea Schönberger, Wen-Hung Huang, Georg Von Der Brüggen, **Kuan-Hsun Chen**, and Jian-Jia Chen. *Schedulability Analysis and Priority Assignment for Segmented Self-Suspending Tasks*. Proceedings of the 24th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Hakodate, Japan, August 2018. doi: 10.1109/RTCSA.2018.00027 [SHK+18]

Sebastian Buschjäger, **Kuan-Hsun Chen**, Jian-Jia Chen, and Katharina Morik. *Realization of Random Forest for Real-Time Evaluation through Tree Framing*. Proceedings of IEEE International Conference on Data Mining (ICDM). Singapore, November 2018. doi: 10.1109/ICDM.2018.00017 [BKC+18]

Mikail Yayla, Anas Toma, Jan Eric Lenssen, Victoria Shpacovitch, **Kuan-Hsun Chen**, Frank Weichert, and Jian-Jia Chen. *Resource-Efficient Nanoparticle Classification Using Frequency Domain Analysis*. Bildverarbeitung für die Medizin 2019. [YTL+19]

Abstract

As chip technology keeps on shrinking towards higher densities and lower operating voltages, memory and logic components are now vulnerable to electromagnetic inference and radiation, leading to transient faults in the underlying hardware [Bau05], which may jeopardize the correctness of software execution and cause so-called soft errors. To mitigate threats of soft errors, embedded-software developers have started to deploy Software-Implemented Hardware Fault Tolerance (SIHFT) techniques, e.g., [CCK+06; MD11; LCP+09]. However, the main cost is the significant amount of *time* due to the additional computation of using SIHFT techniques. To support safety critical systems, e.g., computing systems in automotive and avionic devices, real-time system technology has been primarily used and been widely studied. While considering hardware transient faults and SIHFT techniques with real-time system technology, novel scheduling approaches and schedulability analyses are desired to provide a less pessimistic off-line guarantee for timeliness or at least to provide a certain degree of performance for new application models. Moreover, reliability optimizations also need to be designed thoughtfully while considering different resource constraints.

In this dissertation, we present three treatments for soft errors. Firstly, we study how to allow erroneous computations without deadline misses by modeling inherent safety margins and noise tolerance in control applications as *(m, k) constraints*. We further discuss how a given (m, k) requirement can be satisfied by individual error detection and *flexible compensations* while satisfying the given *hard real-time constraints*. Secondly, we analyze the *probability of deadline misses* and the *deadline miss rate* in *soft real-time systems*, which allow to have occasional deadline misses without erroneous computations. Thirdly, we consider how to deploy redundant multi-threading techniques to improve the system reliability under two different system models for multi-core systems: 1) Under *core-to-core frequency variations*, we address the *reliability-aware task-mapping problem*. 2) We decide on *redundancy levels* for each task while satisfying the given real-time constraints and the limited redundant cores even under *multi-tasking*. Finally, an enhancement for real time operating systems is also provided to maintain the strict periodicity for task overruns due to potential transient faults, especially on one popular platform named Real-Time Executive for Multiprocessor Systems (RTEMS).

Contents

CHAPTER 1

Introduction

Contents

Due to the invention of semiconductor-based integrated circuits [Kil76], computer systems these days have been widely adopted in all areas of technology. Compared to earlier tube-based solutions, embedded systems - with massively lower weight and energy consumption - have become ubiquitous even in safety critical domains, such as computing systems in automotive, robotics, avionic devices, and nuclear power plants. In these domains, computing systems have to perform real-time control operations on time to react to dynamic changes imposed by their environments. Such real-time activities are typically modeled as tasks with *hard* deadlines. Any task finishing after its deadline is considered to deliver not only a late, but also wrong result, as it could jeopardize the whole system behavior. Therefore, the correctness of the system behaviors depends not only upon the functional correctness of the delivered results, but also upon the *timeliness* of the time instant at which these results are delivered.

In addition to timeliness, *dependability* has emerged as one of the most prominent design constraints as well. Since chip technology keeps on shrinking towards higher densities and lower operating voltages [Gar00], hardware faults caused by electromagnetic radiation, process variations, temperature, aging, and voltage fluctuations have

become unavoidable and increased dramatically [Con02; SKK+02; Con03; Bor05; Bau05b; NX06; DW11]. Nowadays, the magnitude of these hardware faults can be observed in all areas of computing. For example, the Japan Aerospace Exploration Agency (JAXA) reported that the Japanese satellite Hitomi accidentally crashed in March 2016 and led to a severe financial damage of more than two hundred million Euros. According to the investigations provided by JAXA [Jap16], the Attitude Control System (ACS) determined to activate the Reaction Wheel (RW) though the satellite was not rotating, to counteract against the wrong attitude provided by the Inertial Reference Unit (IRU), which was caused by the particle strikes. In fact, this should not have been a fatal problem. However, it initiated a series of cascading failures. Eventually the satellite rotated abnormally and separated into several pieces [Ale16]. More severe outcomes were reported for the car manufacturer Toyota, i.e., several Toyota vehicles were reported to accelerate unintendedly in the years between 2000 to 2010 [CBS10], at least 89 casualties and monetary losses of over one billion U.S. dollars for the car manufacturer, On the one hand, radiation-induced bit flips were assumed to be one possible miscreant [Jun13]. On the other hand, the death of tasks caused by overload may also have led to the loss of throttle control on Electronic Throttle Control System (ETCS) and have caused the vehicle to accelerate out of control [Koo14].

Functional safety standards such as the industrial-oriented IEC-61508 [Int10] and the automotive domain specific ISO-26262 [Int00] are provided by certification authorities in the industry, which require a low (or very low) probability of failure per hour (e.g., due to deadline misses). Extensive researches have been conducted at the hardware level, such as N-modular redundant components with an additional voter (commonly used in avionic systems [Skl76]), radiation hardening, or memory protected with an Error-Correcting Code (ECC). However, for cost-sensitive mass products such as cars, applying solely hardware techniques against hardware faults is sometimes not affordable due to its severe impact on performance, circuit area, and energy consumption. As reported by the developers of the ARGOS satellite, using radiation hardened hardware was simply too slow for the "*data processing job intended for it*" [LWW+02]. Even though such hardware faults occur rarely, this waste of resources cannot be avoided if fault-tolerant techniques are implemented at the hardware level.

Before detailing the state of the art in real-time systems with respect to fault-tolerance and identifying potentials therein that lead to the contributions of this dissertation in Section 1.2, the following Section 1.1 provides the context that motivates this dissertation - Software-Implemented Hardware Fault Tolerance *(SIHFT) Techniques on Real-Time System*. Section 1.3 provides an outline for this dissertation, and Section 1.4 shows an overview of my own contributions to research results obtained in cooperation with other researchers.

1.1 SIHFT Techniques on Real-Time Systems

Instead of solely addressing hardware faults at the hardware level, embedded-software developers have started to deploy application-specific error detection [Hil00; HJS02; OSM02; NV03; RCV+05a; LCP+09; HAN12] and error recovery [CCK+06; PGZ08; BSS13]. These SIHFT techniques can *selectively* harden the critical tasks and sensitive spots in the software stack. To adopt such techniques, the main cost is significantly the additional amount of *time*. For example, CRAFT [RCV+05b] duplicates instructions that lead to relatively high susceptibility towards software failures (jump, branches, calls, etc), but these techniques incur beyond 40% performance loss. Another well-known example is adopting Simultaneous Redundant Multi-threading (SRT) [RM00] to provide fault detection and recovery, by which multiple copies of the same program are executed as separate threads on the same processor leading to significant performance overhead.

In order to support those safety-critical systems as mentioned at the very beginning of this chapter, real-time systems technology has been primarily used and widely studied. Even without considering any aforementioned impact from hardware faults or any performance overhead due to SIHFT techniques, making a predictable real-time system itself is already a challenging matter. Typically, real-time systems are designed by considering several pessimistic assumptions on system behavior and on the environment, e.g., the Worst-Case Execution Time (WCET) of all computational activities. Under such a design approach, system designers can perform an off-line schedulability analysis and develop a scheduling algorithm to guarantee that the system is able to satisfy a given timing requirement under all operating conditions predicted in advance. If the real-time application can be modeled by a few fixed parameters, classical schedulability analysis can be effectively applied to provide an off-line guarantee. However, when taking hardware faults and SIHFT techniques into consideration, classic stories in real-time systems now might *turn over a new leaf*.

Everything Adapted from Hard to Soft

Depending upon the types of considered faults, e.g., permanent (remaining for indefinite periods), intermittent (appearing frequently and irregularly), and transient (occurring arbitrarily) hardware faults [Muk08; SS82], classical arguments and analyses in real-time computing technology can be refined accordingly. To handle such permanent hardware faults, the hardware should be hardened with redundancy techniques such as replication [Pra96]. In this dissertation, mainly the effect of *transient* hardware faults is investigated (also known as Single Event Upset (SEU) [WA08; Alt13] or *soft errors*). Namely, each task instance may only be affected by a transient hardware fault. All hardware faults are assumed to have either no effect (the fault was masked by a subsequent operation) or be detected (and recovered) by SIHFT techniques. There are no unnoticed faults or timeouts in the considered systems.

Without relying on specific hardware features, there are several well-known SIHFT techniques to handle transient hardware faults, e.g., retry/re-execution [MD11; PM98; BDP96], checkpoint-based recovery [BDP96; ELS+13], etc. This dissertation assumes that additional computations, which are incurred by some necessary protections preventing the considered systems from system crashes and control flow errors, can be integrated into the execution time of tasks, and mainly focuses on Incorrect Computation Faults (ICF), i.e., that the delivered results may be wrong in response to correct inputs. Based on these assumptions, the effect of applying different SIHFT techniques against ICF is simplified to the different additional workload for each task instance and the different value of the reliability metrics.

Since transient hardware faults are supposed to appear *occasionally*, the additional workload incurred by SIHFT techniques should not be directly considered as part of the normal execution behavior. For example, consider a real-time control task which is protected by a re-execution mechanism. At the end of each execution, a fault detection mechanism *always* verifies if the delivered results are wrong due to transient hardware faults. Depending upon the indication of the fault detection, the re-execution mechanism may be triggered to derive a new output without further detections. Suppose that the WCET excluding the time spent on fault detection is $\mathbb{X}$ time units and that the fault detection takes additional 20% of $\mathbb{X}$. This means that normally the WCET of this task is $1.2\mathbb{X}$ time units, but the WCET of this task becomes $2.2\mathbb{X}$ if a fault is detected[1]. Under such a scenario, directly applying classical analyses of hard real-time systems to deliver an off-line guarantee by always integrating workload of SIHFT can be very pessimistic.

Hence, novel scheduling approaches and schedulability analyses are desired to provide a less pessimistic off-line guarantee for timeliness or at least to ensure a certain degree of performance for new application models considering SIHFT techniques against transient hardware faults. Instead of assuming that recovery mechanisms always take place, the additional workload incurred by SIHFT techniques should be modeled more flexibly, e.g., by adopting multiple versions or probabilistic distributions. Depending upon the considered application models, different real-time guarantees or further reduction on the overall utilization can be achieved.

Resource-Efficient Reliability on Multi-Core Systems

When modern mainstream processors gradually become multi-core systems[2], the demand of achieving resource-efficient reliability on multi-core systems is also a foreseeable objective in the near future. Considering the available resources in such systems, adopting redundant cores to mitigate the impact of soft-errors by using software/hardware redundancy techniques like Redundant Multi-Threading

[1] We assume that, there is no need for the additional detection after detecting one fault.

[2] Commercial examples are: Tilera chip with 100 cores [Til], Intel's Xeon Phi [Intb] and SCC [Inta], Nvidia GPUs with 1024 processing elements [Tes]. Due to increasing core integration, emerging on-chip systems are envisaged to contain 1000s of cores (according to the ITRS prediction: approximately 1500 cores by 2020 and > 5000 cores in 2026) [ITR].

(RMT) [Rot99; RM00; SGF+06] is an expected solution to enable spatial and temporal redundancy.

When we are aware of the additional overhead incurred by SIHFT techniques, we notice that optimizing the overall system reliability also requires to thoughtfully consider the given timing-constraints. For example, the frequencies of individual cores in multi-core hardware may not be trivially homogeneous due to *process variations* [BDM02], *aging* effects [SGH+14], and performance heterogeneous (micro-) architecture designs [ARM13]. Under these circumstances, the reliability-aware task-mapping problem even limited to one-by-one mapping can not be trivially solved by a greedy approach [RKS+14b] while satisfying the given timing constraints.

Furthermore, existing results in the literature typically assume that Triple Modular Redundancy (TMR)-based RMT has to be applied using three cores in parallel, i.e., Chip-level Redundant Multi-threading (CRT) [KCK+16; RKS+14b], or that three replicas are executed on one core sequentially, i.e., SRT [RM00]. However, solely adopting any of them is too inefficient in practice. For example, applying SRT on high utilization tasks to run identical copies on the same core is likely to lead to a system overload. Moreover, the number of redundant cores in the system is limited. Even if the number of cores is adequate to activate CRT for all tasks, the utilization of the dedicated cores may be unnecessarily low (see some examples in Section 1.2.3).

Therefore, several optimization approaches are provided in this dissertation to address the distinct aforementioned design constraints. The key objective is to allocate the tasks by using RMT to improve the overall reliability defined by any applicable metric, e.g., the sum of reliability penalty or the maximum reliability penalty. The timing guarantee is either defined as the tolerated deadline miss rate, which is typically adopted as the quality of service (QoS) metric, or defined as hard real-time constraint(s). The potential applications may have competing scenarios of concurrent execution, e.g., image recognition, data encryption, and secure video conferences, in which various thread instances have to process different sets of data.

1.2 Contributions of this Dissertation

This dissertation focuses on the analysis and optimization for hardware-fault resilience of system software, in particular embedded system software. The presented context mainly addresses the effect of transient hardware faults under the SEU assumption, so-called *soft errors*, that manifest the causes on the software level as *bit flips* [WA08; Alt13] in the main memory or in CPU registers. In this section, the studied problems and the proposed approaches are described in detail. Moreover, the contributions of this dissertation are explicitly listed.

Chapter 2 introduces more background information about real-time systems, basic terms and metrics for better understanding this dissertation.

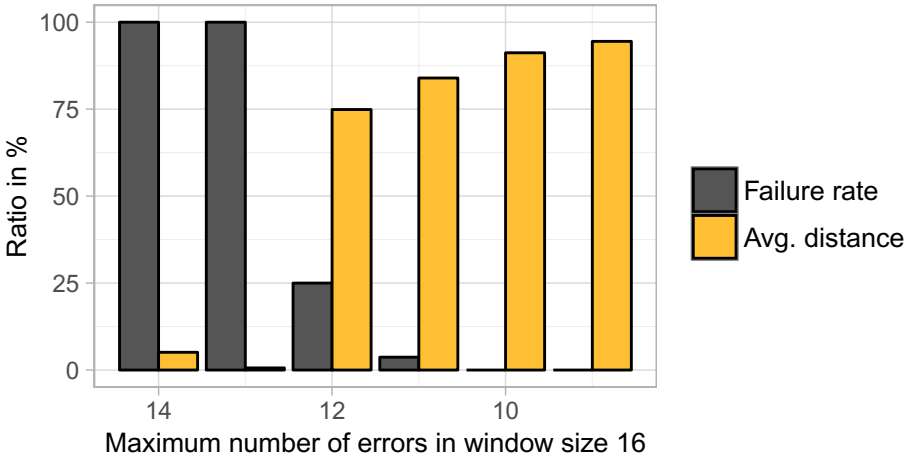


Figure 1.2: Average performance and failure rate in a LEGO NXT path-tracing experiment as Figure 1.1 in relation to the maximum observed number of erroneous task instances for any sliding window size $k = 16$.

1.2.1 Control Robustness and Soft-Error Compensation

In some control applications, a limited number of soft errors might be tolerable and may only downgrade the control performance, e.g., the mission can still be completed but might be imperfect. Due to the potential inherent safety margins and noise tolerance of control tasks, the subsequent iterations might gradually correct the faulty behavior without any need for explicit fault tolerance routines, which is called *inherent fault-tolerance* [VCT+99; FGI09].

An initial experiment as shown in Figure 1.1 demonstrates this effect on a simple LEGO NXT path-tracing application. While constantly going forward, light sensors are read periodically so that the robot can follow and stay on a circular track. During this activity, a fault is randomly injected in each job to derive a wrong control decision. When many jobs are affected by faults, it could result in the robot steering towards the outside of the track. Such soft errors leads either to an increase of steering actions, or in the worst case to the robot leaving the track, which is marked as a failed run.

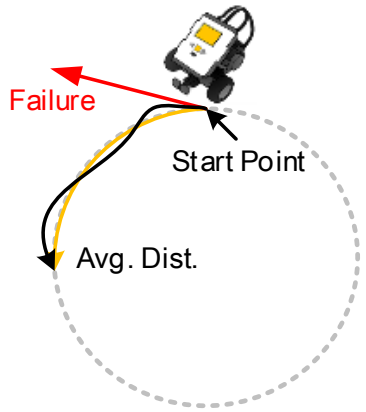


Figure 1.1: An initial experiment on a path-tracing application.

Figure 1.2 illustrates the average covered distance (compared to the maximum recorded) as well as the rate at which the experiment failed in binned sets of errors per window with a given window size $k = 16$.

The window started from any instance with a sliding-window policy. We observe that a control task can tolerate limited number of errors, which can be modeled as a (m, k) *control robustness constraint*[3], requiring a number m of correct runs out of any k consecutive instances to be correct. For example, the results in Figure 1.2 suggest that $(6, 16)$ is possible, namely that the failure rate is zero if there is at most 10 errors out of 16 any consecutive instances.

One trivial way to comply with given (m, k) robustness constraints is to adopt static (m, k) patterns that preselect m instances executed with SIHFT techniques for reliable executions in every repeated window. However, such over-provisioning at the expense of high workloads is likely unnecessarily for low fault rates, since erroneous executions incurred by transient faults are not always more than the number of preselected reliable executions. Although the classical hard real-time approach pessimistically provides an off-line guarantee based on given (m, k) constraints, it also provides an opportunity to use SIHFT techniques adaptively on-the-fly only when it is necessary. The aforementioned observations lead to the first contribution of this dissertation:

Contribution 1: Static Reliable Execution and Dynamic Compensation

- (m, k) control robustness is introduced to quantify the minimal constraint for inherent fault tolerance. By preselecting the reliable instances with a static (m, k) pattern, the given (m, k) constraint can be satisfied, which is called *Static Pattern-Based Reliable Execution*.
- A sufficient schedulability test based on a multi-frame task model is provided accordingly to ensure the given hard real-time guarantee.
- A runtime adaptive approach, called *Dynamic Compensation*, is presented to determine the executing task versions by monitoring the erroneous instances with sporadic replenishment counters, such that the number of expensive reliable executions can be greatly reduced under low soft-error rates.

Chapter 4 describes how and when the proposed approaches can safely compensate, or even ignore errors, while satisfying the given hard real-time and (m, k) control robustness constraints.

1.2.2 Deadline Misses and Probabilistic Analysis

For safety-critical systems, *hard* real-time guarantees are of importance to ensure that results are not just functionally correct but also always delivered according to given timing constraints. In hard real-time systems, it is assumed that any deadline miss

[3]In the literature regarding real-time systems, the term (m, k) is often used for modeling firm real-time constraints. Here we borrow this insight to represent the robustness of control tasks.

can lead to a catastrophe and must be avoided. By contrast, *soft* real-time system can tolerate certain deadline misses. However, intuitively, deadline misses should still be avoided as far as possible, and it is important to quantify the deadline misses to justify whether the considered systems are acceptable.

In such systems, quantifying the deadline misses is more meaningful than validating if every deadline can be strictly met. Based on one natural assumption that fault rates are low (or very low), tasks may seldom have longer execution times incurred by fault recovery operations but these may lead to deadline misses. Due to the usage of SIHFT techniques, such various execution times for each task are modeled by a probabilistic distribution function, by which the system designers are allowed to provide a statistical quantification such as the *deadline miss rate* [MEP04] and the *probability of deadline misses* [BPK+18; BMC16; DGK+02; KC17].

Both the *probability of deadline misses* as well as the *deadline miss rate* are important performance indicators to evaluate the extent of requirements compliance of soft real-time systems. To derive the probability of deadline misses, *probabilistic response-time analyses* [AE13; MC13; DGK+02; BMC16; TBE+15] are typically applied. However, most of them are based on convolution operations to calculate the probability density functions, indicating exponential time complexity. Alternatively, we provide Probabilistic Schedulability Tests (PST) by using analytical upper bounds [KC17; BPK+18] and the *moment generating function* to calculate the probability of deadline misses without any convolution.

By applying existing probabilistic approaches, i.e., [AE13; MC13; DGK+02; BMC16; TBE+15], the probability of the first deadline miss can be obtained while assuming that either jobs missing their deadlines are discarded or that the system is rebooted. In this situation the probability of one deadline miss is directly related to the deadline miss rate since all jobs can be considered individually. However, these very restrictive assumptions often do not hold in practice as aborting jobs or rebooting the system after a deadline miss is not always an option. If at the moment of a deadline miss, the job is neither discarded nor the system rebooted, the additional workload due to a deadline miss may cause a *domino effect* introducing further deadline misses for the subsequent instances. Hence, the actual deadline miss rate may be greater than the probability of the first deadline miss. Instead of deploying simulations, this contribution provides an analytical approach to derive the deadline miss rate.

The second contribution of this dissertation provides several analytical bounds on the probability of deadline misses and the deadline miss rate:

Contribution 2: Probability of Deadline Misses and Deadline Miss Rates

- Two PSTs are proposed to evaluate the upper bounds on the probability of deadline misses and ℓ–consecutive deadline misses for any positive integer ℓ. Compared to the state-of-the-art, the proposed approaches are very efficient with respect to the analysis runtime and the precision of the derived bounds.

- We prove that finding the optimal real valued parameter s for the Chernoff bound approach is a convex optimization problem.
- Leveraging on approaches in the literature, an analytical approach is proposed to derive a safe upper bound on the expected deadline miss rate.

Chapter 5 describes the proposed PSTs in detail, explains how they can derive the probability of ℓ–consecutive deadline misses, presents how they can be adopted in the analysis of deadline miss rates, and evaluates them with extensive simulations.

1.2.3 RMT on Homogeneous Multi-core Systems

RMT is a fault-tolerance technique adapted from hardware to software, providing temporal (or structural) redundancy by repeated execution of the same code. In combination with Double Modular Redundancy (DMR) and TMR, fault detection and recovery can be achieved by replica-comparing and majority-voting, respectively. As introduced in Section 1.1, SRT is one well-known approach activating RMT against transient hardware faults, but it increases the computation time significantly [RM00].

Instead of running replicas on the same core, exploiting idle cores for task redundancy provides an alternative against soft errors on multi-core systems. State-of-the-art techniques like [Rot99; RM00; SGF+06] exploit idle cores to enable spatial and temporal redundancy. In particular, Intel's CRT [MKR02] executes redundant copies of a given task on different cores in parallel and performs error detection and recovery using comparison and voting on the threads' outputs. However, in modern multi-core systems, individual cores may exhibit different frequencies due to *process variations* [BDM02], *aging* effects [SGH+14], and performance heterogeneous (micro-) architecture designs [ARM13].

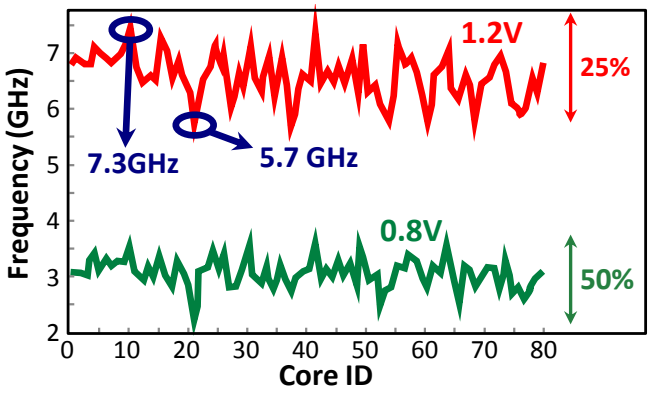
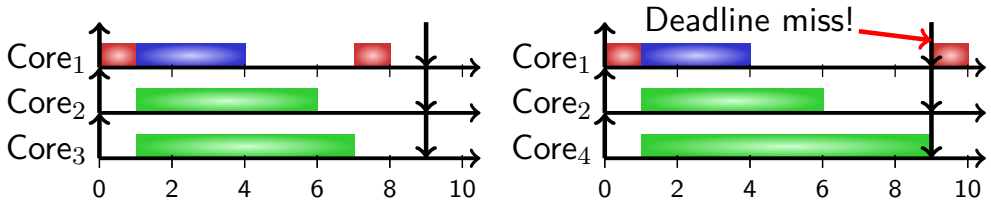


Figure 1.3: The core-to-core frequency variations (28 % at $1.2V$ and 59 % at $0.8V$) for an Intel 80-core test chip, adpated from [DVA+11].

Figure 1.3 shows the core-to-core frequency variations for an Intel's 80-core test chip [DVA+11]. Aging further aggravates this issue by inducing frequency degradation

at run-time. As shown in Figure 1.4, the performance of the task with RMT depends upon the lowest-frequency core in the assigned core group. As transient hardware faults might occur more within a longer execution time, executing on a higher frequency core is assumed to have a better (lower) reliability penalty than executing on the lower one. When applying RMT on such a multi-core system with performance heterogeneity, efficiently optimizing the reliability of the whole system is not a trivial problem.



(a) A task with TMR-based RMT can meet the deadline.

(b) The thread on the lowest core, i.e., Core_4, leads to a deadline miss.

Figure 1.4: Example of TMR-based RMT on three cores with different frequencies, where the core frequencies are $f_1 > f_2 > f_3 > f_4$. The red blocks represent the workload due to the necessary steps for synchronizing the majority-vote mechanism, and the green blocks are the redundant threads. The vote has to wait for the redundant thread on the lowest-frequency core among the assigned cores, i.e., Core_3 or Core_4, to derive a result.

Without performance heterogeneity, multi-tasking together with RMT is also a useful combination in practice, by which the multi-cores may be utilized as much as possible. Instead of solely using CRT— the utilization of dedicated cores may be unnecessarily low, or SRT— activating on high utilization cores may lead to a system overload, we propose a hybrid structure combining CRT and SRT, where two threads are executed on the same core and one is executed on another core in parallel, termed as Mixed Redundant Threading (MRT). Figure 1.5 illustrates different redundancy levels among the combinations of SRT, CRT, MRT, DMR, and TMR. Consider one task and two available cores. Since SRT-TMR imposes a higher computation demand on one processor, e.g., three times the execution time, it may lead to a deadline miss as shown in Figure 1.6a. Applying CRT-TMR is also not possible, as only two cores are available. However, if we execute the task with one replica on Core_1 sequentially and one on Core_2 in parallel like in Figure 1.6b, TMR-based RMT as MRT is possible without sacrificing timeliness. While considering RMT and task-mapping on multi-core systems, the performance heterogeneity and eventually using MRT motivate the third contribution of this dissertation:

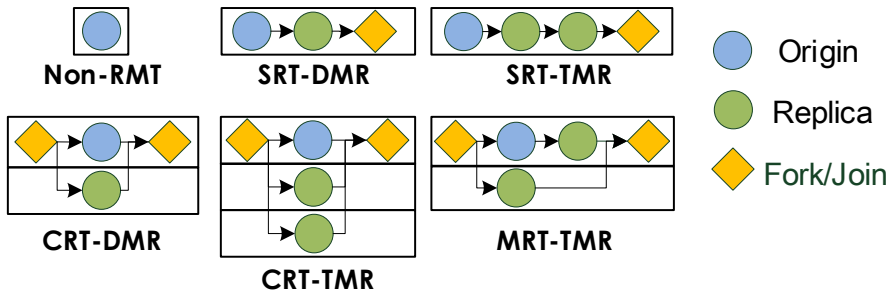
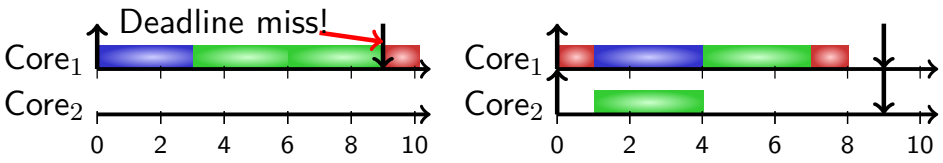


Figure 1.5: DAG abstractions of the different redundancy levels, where the blue nodes are original executions and the green nodes are replicas. The yellow nodes represent the workload due to the necessary steps for forking the original executions and replicas, joining, and comparing the delivered results from DMR/TMR at the end of RMT.



(a) SRT is not feasible. **(b)** MRT can activate TMR.

Figure 1.6: TMR-based RMT on two cores, where blue blocks are the original execution, green blocks are replicas, and red blocks represent the workload necessary for forking and joining the original execution and replicas. CRT is not possible since only two cores are available. While SRT is not feasible due to a deadline miss as shown in subfigure (a), MRT can feasibly schedule the task with TMR as shown in subfigure (b).

Contribution 3: Reliability-Aware Task Mapping Approaches
The third contribution of this dissertation is twofold. Firstly, we provide reliability optimizations through CRT on multi-core systems with performance variations:

- Given the redundancy level of each task, we show how an elegant approach called Hungarian Algorithm (HA) can be applied to provide an effective task-mapping in the course of the global optimization with respect to reliability while considering the effect of performance heterogeneity on RMT.
- An Iterative Level Adaptation (ILA) heuristic in combination with the above task-mapping approaches is proposed, such that the redundancy level of tasks can be determined under polynomial time complexity.

Secondly, we present resource-aware reliability optimizations through different RMT levels while complying multi-tasking on multi-core systems:

- MRT is proposed as a mixture of SRT and CRT, providing more flexibility on RMT selections, i.e., the tasks with one replica are executed on one core and a second replica is executed on a different core.
- Several dynamic programming approaches with different optimization granularities through the Federated Scheduling (FS) as a backbone are proposed to find an optimal selection of redundancy levels while satisfying given hard real-time constraints for multi-tasking.

Chapter 6 first describes how HA can be applied to determine an effective task-mapping and details the ILA heuristic for RMT. Without considering performance variations, Chapter 6 shows how SRT, CRT and MRT can be activated in different granularities under FS to optimize the system reliability while satisfying given hard real-time constraints.

1.2.4 Strict Periodicity and Overrun Handling

Let us consider periodic real-time task systems. In the literature, it is usually assumed that multiple task instances of a task are executed in a First-Come-First-Serve (FCFS) manner. Thus, it is sufficient to release the second task instance at the moment the first task instance finishes, assuming that the first task instance finishes after the time point at which the second task instance would have been released according to a *strictly periodic pattern*.

A task is said to *overrun* when a task instance is not finished at the end of its period. This may not only happen if a task misses its deadline but also when a task has an arbitrary deadline. When SIHFT techniques are considered, e.g., re-execution [MD11] or check-pointing [ELS+13], tasks may have a longer execution time in some rare cases due to fault recovery operations, leading to deadline misses as well.

In our work *Systems with Dynamic Real-Time Guarantees* [BKH+16], we analyzed the impact of transient hardware faults within the execution of a task instance. During these analyses, we discovered that the implementation in Real-Time Executive for Multiprocessor Systems (RTEMS) (version 4.11) does not behave as expected when these faults occur in a task missing its deadline.

To illustrate the behavior of the original design, we provide an example with two implicit-deadline sporadic[4] tasks in Figure 1.7: τ_1 with $C_1 = 6$ and $T_1 = 10$, and τ_2 with $C_2 = 1$ and $T_2 = 2$, where τ_1 is given a higher priority than τ_2[5]. We let τ_1 release only two jobs (at 0 and 10) and let τ_2 have a phase of 6, i.e., the first job of τ_2 is released at 6 but its follow-up jobs arrive with a period of 2.

[4] Chapter 2 explains the definition of different task models in detail.

[5] For the simplicity of presentation, here the priorities of tasks are given arbitrary to show the effects under different designs.

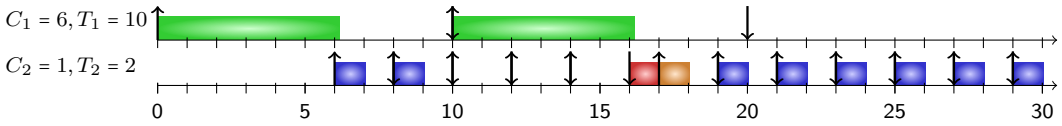


Figure 1.7: The original design in RTEMS [KBC16]. The red job is postponed from 10. The arrival pattern from 16 is changed due to the lateness of the red job, due to which all following jobs are released earlier.

We can see that τ_2 cannot be executed in the interval $[10, 16]$ as τ_1 has higher priority, resulting in 3 expired periods. Using the original overrun handling, the job of τ_2 released at 10 (colored red) is finished at 17, which leads to a new release of τ_2 at 17, as τ_2 is marked to have deadline misses. This results in a shift of the periodicity of τ_2 by 1 for this job and the following jobs. The job that should be released at 16 is released at 17 now (orange) and the following jobs are all shifted as well. It is worth noting that the jobs that should be released at 12 and 14 are never released to the system. Overall, we can see that the original design in RTEMS violates the strict periodicity and does not match the expectation of overrun handling in most applications and researches, as two jobs do not enter the system at all and the period of τ_2 is shifted due to the deadline miss of the job started at 10.

Contribution 4: Overrun Handling Support
Hence, the forth and last contribution of this dissertation tackles the design problems arising from the original implementation of RTEMS when overruns take place:

- Two major problems in the original implementation of RTEMS are discovered, namely, that the release patterns of tasks are shifted and that jobs are not released at all if the overrun of a task is longer than one task period.
- A provided extension enhances fixed-priority and dynamic-priority schedulers to provide a proper behavior (following the expectation in the literature) for overrun handling. It is now part of the mainstream version.

Chapter 7 describes the design flaw of overrun handling in detail and provides a comprehensive implementation included in the latest version of RTEMS.

1.3 Outline

This dissertation is structured as follows:

- Chapter 2 summarizes the background and related work required for understanding this dissertation.
- Chapter 3 provides the system model and the experimental platform used throughout the dissertation.

- Chapter 4 presents the first contribution of this dissertation: The design and implementation of the *static pattern-based* and *runtime compensation* approaches for soft-error handling. The (m, k) model is introduced to quantify robustness requirements, and to study the problem of scheduling tasks with multiple versions for soft-error handling while satisfying (m, k) robustness requirements.
- Chapter 5 describes the second contribution: The PST approaches for deriving the *probability of (ℓ-consecutive) deadline misses* and the analyses of *deadline miss rates*. Compared to conventional analyses, the PST approaches are efficient with respect to the derived probability of deadline misses and the needed calculation time. This is the first work deriving the deadline miss rate in *soft* real-time task system without considering rebooting after deadline misses.
- Chapter 6 shows the third contribution: We provide several *reliability-aware task mapping* approaches via RMT on a multi-core system. The proposed reliability optimizations consider two different systems: homogeneous multi-core systems with heterogeneous performance and homogeneous multi-core systems with multi-tasking.
- Chapter 7 addresses the fourth contribution: We provide a complete enhancement on fixed-priority and dynamic-priority schedulers for the overrun handling in RTEMS, which is accepted in version 4.11 and inherited by version 5.

Chapter 8 summarizes key results and current limitations in this dissertation, and discusses opportunities for future research.

1.4 Author's Contribution to this Dissertation

According to §10(2) of the "Promotionsordung der Fakultät für Informatik der Technischen Universität Dortmund vom 29. August 2011", my contribution to the presented results for each chapter is listed in the following:

- Chapter 4 describes the *soft-error compensation* approaches while satisfying given hard real-time constraints and control robustness requirements. The approach was published at LCTES 2016 [KBC+16], where I was the principal author, contributed the design, and coordinated self-balancing robot's implementation and evaluation with Björn Bönninghoff. The evaluation on linear dependences was developed in cooperation with Mikail Yayla and published at EITEC 2018 [YKC18], which I co-authored. The proposed approaches in this chapter are implemented in RTEMS by Mikail Yayla during the project — Summer of Code in Space (SOCIS) 2017 — funded by European Space Agency (ESA) [Mik17], which I mentored.
- Chapter 5 at first describes the *probabilistic schedulability tests* for deriving the probability of deadline misses. The approach was published at SIES 2017 [KC17], where I was the principal author. Especially, Niklas Ueter contributed to the theorem that finding the optimal real-valued parameter is a convex optimization

problem. It was published at DATE 2019, where I was the principal author. The analytical upper bounds based on *Hoeffding's inequality* and *Bernstein's inequality* were published at ECRTS 2018 [BPK+18], which I co-authored. I contributed the theorems and implementation involved in the evaluation deployed by Georg von der Brüggen. The analyses of *Deadline Miss Rates* were published at RTCSA 2018 [KBC18a], where I was the principle author and contributed concepts, implementation and evaluation of the analyses.

- Chapter 6 describes the *reliability-aware task mapping* approaches for redundant multi-threading on multi-core systems. The approaches were published at REES 2015 [KCK+15], TC 2016 [KCK+16] and TC 2018 [KBC18b]. In each of these publications, I was the principal author and contributed concepts, design, implementation, and evaluation of the approaches.
- Chapter 7 describes the design flaw of overrun handling and provides a comprehensive enhancement on RTEMS (accepted at version 4.11 and inherited by version 5). Details were already published at WMC 2016 [KBC16], where I was the principal author and provided concepts, design, and implementation of the enhancement.

CHAPTER 2

Background and Related Work

Contents

This chapter summarizes the background and related work required for understanding this dissertation. Each contribution in this dissertation is individually supplemented by its corresponding related work. Initially, several terminologies in real-time systems are introduced, e.g., task models and scheduling algorithms, since most of them are commonly used as standard models throughout the dissertation.

2.1 Terminology of Real-Time Systems

This section is partially based on the work of Buttazzo et al., i.e., *Soft Real-Time Systems: Predictability vs. Efficiency* [BLA+05], which is complemented in several places if appropriate.

2.1.1 Typical Task Models

Depending on the timing requirements, real-time tasks are classified according to the consequence of their potential deadline misses as follows:

- *Hard real-time:* All jobs must be completed within their deadlines. Any deadline miss leads to a total system failure.
- *Firm real-time:* Only a limited number of jobs is allowed to miss their deadlines. For example, the (m, k)-firm real-time guarantee allows to specify tasks in which m out of any consecutive k jobs must meet their deadlines [Ram99].
- *Soft real-time:* The usefulness of the produced result gracefully degrades with its increasing response time. If a deadline is missed, the system keeps working at a degraded level of performance. Although deadline misses are tolerable in the system, they should still be avoided as far as possible.
- *Non real-time:* It does not matter if deadline are missed or not. The usefulness of the produced result does not depend on the completion time of its computation.

In this dissertation, only hard and soft real-time systems are considered.

2.1.2 Deadline and Activation Models

Each task in a real-time system is associated with a relative deadline. Depending on the timing requirements, relative deadlines of tasks can be classified into three categories: *implicit*, *constrained*, and *arbitrary* deadlines.

- *Implicit-deadline:* Each deadline of a task is equal to its minimum inter-arrival time or period.
- *Constrained-deadline:* Each deadline of a task is no more than its minimum inter-arrival time or period.
- *Arbitrary-deadline:* No general relation between deadlines and periods exists.

Each job of a task in the system has its absolute deadline. It is the absolute time before which a task should be completed.

As stated above, a limited number of deadline misses are tolerable in firm or soft real-time systems. Depending on the action against deadline misses, the system may behave in the following two ways:

- Discards the job missing its deadline or even reboots the system. The remaining portions of the computation will not continue but no carry-in workload is propagated to the subsequent jobs [AE13; MC13; DGK+02; BMC16; TBE+15].

- Lets the overrun job continue until it finishes. However, the carry-in workload may trigger further deadline misses as a *domino effect* in the subsequent jobs [BKH+16; KBC18a].

Computational activities can either be activated by a timer at predefined time instants, named *time-triggered activation*, or by the occurrence of a specific event, named *event-triggered activation*. In real-time systems, the activations of jobs for each task are mostly triggered by time. Depending on how the activations are separated, there are three activation models:

- *Periodic model:* All activations are strictly separated by a fixed interval of time, called *period*. Suppose that the first job of task τ_i is activated at time ε_i. The j-th job must be activated at time $\varepsilon_i + (j-1)T_i$, where T_i is the task period.
- *Sporadic model:* A task has a minimum inter-arrival time T_i between the activations of any two adjacent jobs. This task is said to be *sporadic*.
- *Aperiodic model:* If a task has no regulation with respect to its job activations, this task is said to be *aperiodic*.

In general, periodic task model is usually a default system model in the implementation of Real Time Operating System (RTOS). For example, the deadline detection in Real-Time Executive for Multiprocessor Systems (RTEMS) is embedded in the routine of the task periodicity, by which the task deadline is expected to be less than – *constrained* – or equal to its period – *implicit*. Throughout this dissertation, only periodic and sporadic models are considered.

2.1.3 Scheduling Algorithms and Schedulability

Which job is executed on a processor can be decided by many different scheduling approaches, e.g., a priority-based scheduling approach, a budget-based scheduling approach, or a time-division scheduling approach, for example, Time Division Multiple Access (TDMA) [SCT10]. In a TDMA system, each task is assigned a particular time window, called *TDMA slot*, such that each task uses its own slot one after the other.

Depending on how the priorities are assigned, priority-based scheduling algorithms in real-time systems can be classified as follows [LKA04]:

- *Fixed task priority:* All jobs of each task are associated with a unique priority level. A task which has a shorter period gets a higher priority. This is widely used in the industrial practice and is supported in most RTOSs, also known as *fixed-priority scheduling*. Here two widely-used algorithms are Rate-Monotonic (RM) scheduling and Deadline-Monotonic (DM) scheduling, by which each task is given a fixed priority based on their periods and their deadlines, respectively.
- *Fixed job priority:* Each job of a task has a single priority, whereas different jobs of the same task may have different priorities. One well-known scheduling

algorithm is Earliest-Deadline-First (EDF) scheduling, by which the job in the system with the earliest absolute deadline is always executed first.
- *Dynamic priority:* Each job may have different priorities on-the-fly. For example, the job with the least laxity is always executed first by Least-Laxity-First (LLF) scheduling.

Furthermore, depending on if CPU occupation can be transferred from one running task to another, scheduling algorithms can be classified into two categories:

- *Preemptive:* A job or task instance can be preempted by another job or task instance during the execution at any time.
- *Non-Preemptive:* Once a job or task instance has started to run on a processor, it cannot be preempted by another job or task instance until it finishes its computation. The scheduler is only invoked at the end of a job (assuming there is no interrupt).

A real-time system is said to be *schedulable*, if there exists a scheduling algorithm that schedules the system without violating the given timing constraints. A scheduling algorithm is said to be *feasible* to be applied under a given real-time system if it schedules the system without violating the given timing constraints.

To test if a scheduling algorithm is *feasible* to be applied under a given real-time system, the Worst-Case Execution Time (WCET) of each task in a real-time system must be known in order to derive the timing guarantee. In this dissertation, assume that the WCET of each task is given in advance by either a *measurement-based* approach[1] or by a *static analysis*. How to derive tight and safe upper bounds on the WCET analyses is widely-discussed in the literature, which is out of the scope for this dissertation.

Knowing the WCET of each task, utilization bounds like the seminal result of the Liu and Layland (L&L) bound [LL73] can be used to validate the schedulability quickly. In addition, the well-known Time-Demand Analysis (TDA) developed in [LSD89] is applicable and even tighter than the L&L bound. However, considering the impact of hardware transient faults, both tests may reject many task sets which are actually schedulable if we pessimistically add the execution time of applying error-correcting mechanism to the WCET of each task[2]. This concept is explicitly discussed from different perspectives in Chapter 4 and 5.

2.2 Control Robustness and Soft-Error Compensation

2.2.1 Fault-Tolerance in Control Applications

In control theory, control systems are expected to deal with imperfect signals and the uncertainty induced by environments. Hence, controllers are typically designed to

[1] Please note that there is no guaranteed upper bound for measurement-based approaches, since the measurement-based approach in general will underestimate the actual WCET.

[2] We assume that the number of repeated errors is limited or given as a threshold.

(m,k)	R-pattern	E-pattern	(Reverse) E-pattern
(3,10)	1 1 1 0 0 0 0 0 0 0	1 0 0 1 0 0 1 0 0 0	0 0 0 1 0 0 1 0 0 1
(5,10)	1 1 1 1 1 0 0 0 0 0	1 0 1 0 1 0 1 0 1 0	0 1 0 1 0 1 0 1 0 1
(7,10)	1 1 1 1 1 1 1 0 0 0	1 1 1 0 1 1 0 1 1 0	0 1 1 0 1 1 0 1 1 1

Table 2.1: Iterations of different patterns over different (m,k), where binary $\{0,1\}$ represents two different purposes of instances, e.g., unreliable and reliable.

make the control output gradually closer to the desired control signal with erroneous signals in the feedback loop. Particularly, several techniques are proposed to handle delayed [Ram99; KGC+12] or dropped signal samples [HSJ08; BS15; GDD19]. These uncertain signals occur occasionally like soft-errors in general. If faults are not fatal, applying techniques such as interpolation, moving average, and fuzzy design can mitigate the effect of such soft-errors. In case a fault results in a completely wrong result, such samples can be dropped, while a new input can be computed using the previous inputs [Ram99; HSJ08; BS15]. Such a margin of tolerable errors, e.g., delayed, dropped, wrong, during task executions allows us to exploit different Software-Implemented Hardware Fault Tolerance (SIHFT) schemes or even to *ignore* soft-errors *occasionally*.

2.2.2 (m,k) Models and Static Patterns

In general, (m,k) models define that m out of k consecutive instances have to comply with a given requirement, where $m \leq k$. Such a model provides a flexible but robust requirement, which requires not only a ratio m over k but also the frequency of correctnesses, i.e., only m within every k. In the related literature, a (m,k) firm real-time guarantee is said that at least m out of k consecutive job instances of the corresponding task can meet their deadlines. To comply a given (m,k) constraint, several (m,k) patterns [QH00; NQ06; Ram99; KS95] are widely used, i.e., Deep Red Pattern (R-pattern) [KS95], Evenly Distributed Pattern (E-pattern) [Ram99], and Reverse E-pattern [QH00] by which the instances are preselected for different purposes.

In general, each pattern for task τ_i is a binary string $\mathbb{B}_i = \{\beta_{i,0}, \beta_{i,1}, \ldots \beta_{i,(k_i-1)}\}$. Each $\beta_{i,j}$ can be either "1" or "0", and $\sum_{j=0}^{k_i-1} \beta_{i,j} = m_i$. Since the aforementioned static patterns are used as a backbone to comply with the given (m,k) robustness constraints in Chapter 4, we define them formally in what follows:

Suppose that a job instance of task τ_i in a given pattern $\mathbb{B}_i$ is denoted as $\beta_{i,j}$, where $j = 0, 1, \ldots, k_i - 1$. The j-th job instance of task τ_i is determined as follows:

- **If $\mathbb{B}_i$ is R-pattern:**

$$\beta_{i,j} = \begin{cases} 1, & 0 \leq j \mod k_i < m_i \\ 0, & \text{otherwise} \end{cases} \qquad j = 0, 1, \ldots, k_i - 1 \tag{2.1}$$

- **If $\mathbb{B}_i$ is E-pattern:**

$$\beta_{i,j} = \begin{cases} 1, & \text{if } j = \left\lfloor \left\lceil \frac{j \times m_i}{k_i} \right\rceil \times \frac{k_i}{m_i} \right\rfloor \\ 0, & \text{otherwise} \end{cases} \quad j = 0, 1, \ldots, k_i - 1 \tag{2.2}$$

- **If $\mathbb{B}_i$ is Reverse E-pattern:**

$$\beta_{i,j} = \begin{cases} 0, & \text{if } j = \left\lfloor \left\lceil \frac{j \times (k_i - m_i)}{k_i} \right\rceil \times \frac{k_i}{(k_i - m_i)} \right\rfloor \\ 1, & \text{otherwise} \end{cases} \quad j = 0, 1, \ldots, k_i - 1 \tag{2.3}$$

Table 2.1 shows different static patterns over different (m, k) constraints. By repeating a given static pattern $\mathbb{B}_i$, it can be guaranteed for taskτ_i that there must be m_i job instances marked as "1" among any k_i consecutive jobs. Therefore, such job instances can be preselected for applying necessary techniques to satisfy the given (m, k) constraint.

2.3 Probabilistic Analysis and Deadline Misses

2.3.1 Probabilistic Timing Analysis

Several probabilistic timing analyses in the literature can calculate the probability that a deadline miss occurs in the system. Depending on the considered task models, they can be classified as follows:

- **For periodic real-time task systems:** Diaz et al. [DGK+02] provided a framework for calculating the deadline miss probability. Tanasa et al. [TBE+15] adopted the Weierstrass Approximation and applied a customized decomposition procedure to derive the deadline miss probability among all possible combinations. However, both of them only work for small examples with respect to the number of jobs, i.e., 7 and 25 jobs in the hyper-period, respectively.
- **For sporadic real-time task systems:** Axer et al. [AE13] iterated over the activations of released jobs to evaluate the response-time distribution for non-preemptive fixed-priority scheduling. Maxim and Cucu-Grosjean [MC13] proposed a probabilistic response time analysis and probabilistic minimum inter-arrival times based on job-level convolution. Ben-Amor et al. [BMC16] provided a probabilistic response time analysis based on [MC13], considering precedence constrained tasks. All above approaches convolute the probability at which a new job arrives in the interval of interest, but this convolution procedure expectedly lead to an extremely high analysis runtime. Recently, von der Brüggen et al. [BPK+18] proposed a novel approach to compute probabilistic response time based on using multinomial distributions and a *task-level* convolution procedure, which drastically improves the computational complexity but preserves full precision in terms of approximation quality.

For an extensive literature review, we refer the reader to the recent survey by Davis and Cucu-Grosjean [DC19].

2.3.2 Deadline Miss Rate

As soft real-time systems tolerate and allow a certain amount of deadline misses, the deadline miss rate is an important performance indicator to evaluate the proposed analyses, scheduling algorithms, etc. Most studies in the literature [KG94; SLS+99; BBL+03; LB05; HJS+06] use the deadline miss rate as a performance metric to empirically evaluate their proposed approaches.

Manolache et al. [MEP04] presented a stochastic approach for obtaining the expected deadline miss rate analytically. Based on this, Manolache et al. [MEP08] addressed the problem of task priority assignment and task mapping. They consider a non-preemptive scheduling to reduce the computational complexity of their convolution based approach, which is very restrictive.

2.4 Redundant Multi-Threading on Multi-Core Systems

2.4.1 N-Modular Redundancy

The most common *structural redundancy* solution against errors, known especially from highly critical systems in avionics, is N-Modular Redundancy (NMR) of important components. Particularly, Double Modular Redundancy (DMR) and Triple Modular Redundancy (TMR) [LV62; VZB+10; RM00; MKR02] are widely used. In a DMR setup, a functionally redundant *voter* component can continuously compare the internal state of the external behavior of two components and detect a deviation. In a TMR setup, the third component can be used to further correct one faulty instance without repeating previous work. In Redundant Multi-Threading (RMT) techniques, N-Modular Redundancy becomes available for both temporal and structural redundancy.

2.4.2 RMT techniques

RMT techniques [RM00; VPC02; MKR02] provide fault detection and recovery mechanisms by replicating a task into multiple identical executing threads and comparing the produced results. There are two ways to apply RMT, Simultaneous Redundant Multi-threading (SRT) and Chip-level Redundant Multi-threading (CRT).

SRT approaches like in [RM00; VPC02] provide transient fault coverage by running identical copies of the task on the same processor. Alternatively, CRT approaches in [RKS+14b; MKR02; KCK+16] use redundant cores on multi-core systems for RMT, in which redundant replicas of a given task are executed on different cores in parallel.

2.4.3 Schedulability Problem for Parallel Workloads

Some related researches for the schedulability problem of parallel workloads in fork-join and DAG task models are known, i.e., [LCA+14; AQN+13; KKP+15; RE17]. Axer et al. [AQN+13] focused on a worst-case response time analysis with a fork-join

task model. Li et al. [LCA+14] proposed a Federated Scheduling (FS) strategy to deal with parallel workloads in partitioned scheduling. Recently, Rambo et al. [RE17] proposed a replica-aware co-scheduling by considering replicas as gangs under a single error assumption for mixed-critical systems with much better performance than [AQN+13]. However, most existing research assumes that the parallel workload distribution is known, i.e., the number of threads per task is given beforehand, and only analyze the feasibility problem in terms of timeliness. Kwon et al. [KKP+15] tackled the scheduling problem with the global scheduling algorithm PD^2 [SA05] and determine the parallel executing options by testing all possible combinations. The above work focuses on the scheduling of given parallel workloads, whereas our work aims to optimize the system reliability by parallelizing the workloads, scheduling them under given timing constraints on a given multi-core platform.

2.5 Related Implementation for Maintaining Periodicity

In Chapter 7, we enhance the overrun handling mechanisms in RTEMS for keeping the task periodicity strict. In this section, we extensively review how the other real-time operating systems, e.g., FreeRTOS [Rea16] , LITMUS^{RT} [Bra06; BBC+07], and ERIKA-OS [Evi16], handle overruns and maintain the periodicity of tasks.

2.5.1 FreeRTOS

FreeRTOS is a well-known real-time operating system, which especially offers lighter and easier real-time processing. In `timers.c`, `prvProcessTimerOrBlockTask()` and `prvProcessExpiredTimer()` are responsible for the periodicity of the task. The feature of `prvProcessTimerOrBlockTask()` is similar to the function `Period()` in RTEMS that determines if a task should be blocked or if a timer has expired. In the function `prvProcessExpiredTimer()`, the expired timer is updated immediately with the next expiry time. To maintain the periodicity, all tasks' timers are listed in an expiry time order and the task which refers to the head of list expires first. Although the deadline of each task is assigned correctly in their timer, there exists no mechanism providing overrun handling in the FreeRTOS scheduler that enforces the correct number of postponed jobs required to be released for keeping strict periodicity.

2.5.2 LITMUS^{RT}

LITMUS^{RT} is a popular real-time extension of the Linux kernel. The implementation of overrun handling for the fixed-priority scheduler can be found in `job_completion()` in `sched_pfp.c`. With the common function `prepare_for_next_period()` and `setup_release()` in `jobs.c`, it has implemented a special counter called `job_no` to record how many jobs should be ideally released without overrun behavior. As reported in [BBC+07], the system will have already advanced to the next job by the time at which the job completion is signaled from user-space. It had another system

call named `wait_for_job_no()` to handle such a overrun situation, which is replaced completely by `sys_wait_for_job_release()` in the latest version of LITMUS^{RT}. With `sys_wait_for_job_release()` in `litmus.c`, a task is only going to sleep when its number of released job is greater than `job_no` by triggering `complete_job()`. This is similar to the case in `rate_monotonic_period()` where RTEMS decides if the period should be blocked. By this implementation, the postponed jobs should be released consecutively until there is no postponed job, which is similar to our enhancement in Chapter 7.

Interestingly, when adopting the EDF scheduling in LITMUS^{RT}, we noticed that once a job is overrunning, it gets the highest priority in the system. This results from the fact that the distance between the job missing its deadline and its absolute deadline becomes negative. This behavior is not well-defined in the literature.

2.5.3 ERIKA-OS

ERIKA-OS is an open-source Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen (OSEK)/Vehicle Distributed eXecutive (VDX) Kernel proposed in [GBL+00]. Its latest version is released in [Evi16]. According to the OSEK standard [OSE05], the periodicity of tasks is predefined in the OIL language file by setting up corresponding alarms to each task. Each core only maintains one sorted alarm queue and each alarm has an assigned time. Along with the system clock ticking, the difference between the current time and the assigned time of the first alarm is gradually decreased to zero and the first alarm is set to *ready*. As long as it is ready, one job from its corresponding task will be released and the alarm itself will be inserted back into the queue for the next period.

Due to the above design, jobs of all corresponding tasks from the ready alarms will be *sequentially* released until there are no more ready alarms in the alarm sorted queue. However, when there is one executing job in the system, all other alarms, which are corresponding to lower priority tasks, could be ready at the same time, but pending or even missing deadlines. Since there is no mechanism to monitor the amount of time for such pending alarms, there is no further information about how many jobs are actually postponed. Therefore, for such tasks missing their deadlines, only the first postponed job will be released in the system and the periodicity is no longer strict.

CHAPTER 3

System Model and Experimental Platform

Contents

This chapter summarizes the considered system models and the experimental platforms adopted in this dissertation.

3.1 Application Model for Uniprocessor Systems

In this dissertation, a considered application can be modeled as a set of periodic or sporadic, hard or soft real-time tasks with their implicit or constrained deadlines, such that there are in total N tasks, denoted as $\Gamma = \{\tau_1, \tau_2, ..., \tau_N\}$. Each task $\tau_i \in \Gamma$ is associated with a relative deadline D_i.

A preemptive fixed-priority scheduling policy is assumed where the priority of a task cannot be changed during runtime. The tasks are indexed from 1 to n where τ_1 has the highest priority and τ_n has the lowest priority. Let $hp(\tau_k)$ be the set

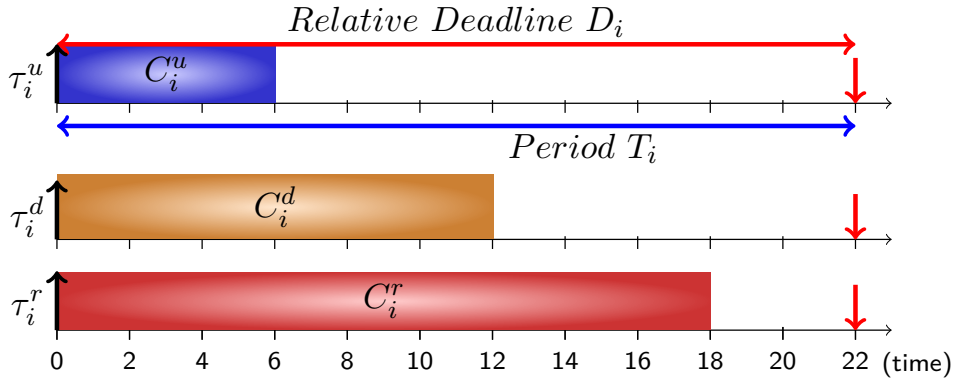


Figure 3.1: Multiple execution versions, where $\tau_i(\{C_i^u, C_i^d, C_i^r\},\ D_i,\ T_i)$.

of tasks with higher priority than τ_k and let $hep(\tau_k)$ be $hp(\tau_k) \cup \{\tau_k\}$. We assume preemption overheads to be negligible compared to the execution time of one task instance. If the overheads are not negligible, they can be integrated into the Worst-Case Execution Time (WCET) of tasks using standard approaches provided in the literature.

Each task τ_i releases an infinite number of task instances, called jobs, under a minimum inter-arrival time constraint (or period) T_i, which specifies the minimum time between two consecutive job releases of τ_i. Therefore a job of task τ_i released at time t_a must be completed not later than the absolute deadline $t_a + D_i$ and the next job of task τ_i must be released exactly at (or not earlier than) $t_a + T_i$ for periodic (or sporadic) tasks. We consider implicit-deadline, where $D_i = T_i\ \forall \tau_i \in \Gamma$, and constrained-deadline task systems, where $D_i \leq T_i\ \forall \tau_i \in \Gamma$.

Due to soft-error handling, e.g., error detection and recovery mechanisms, we consider that soft-errors induced by hardware transient faults only affect every task instance, namely job, at most once under the Single Event Upset (SEU) assumption. Without using error detection and recovery, the system can still be executed with wrong output values but without unnoticed faults, i.e., a Silent Data Corruption (SDC), or a even system crash. It means that if the fault rate is $\mathbb{F}$, then the probability that the first execution of a job is incorrect is $\mathbb{F}$. The following one or two executions for detection or correction, if they exist, are assumed to be hardened perfectly; error detection and correction always perform correctly. Without such a control of errors in the experiments, the conclusions may not be drawn correctly, since erroneous executions can be similar under different error rates.

Each task τ_i is assumed to have several (but a finite number of) execution versions τ_i^j. Each version τ_i^j is associated with a distinct value of execution time C_i^j. The utilization of τ_i^j can be calculated as $U_i^j = C_i^j/T_i$. Throughout the dissertation (besides

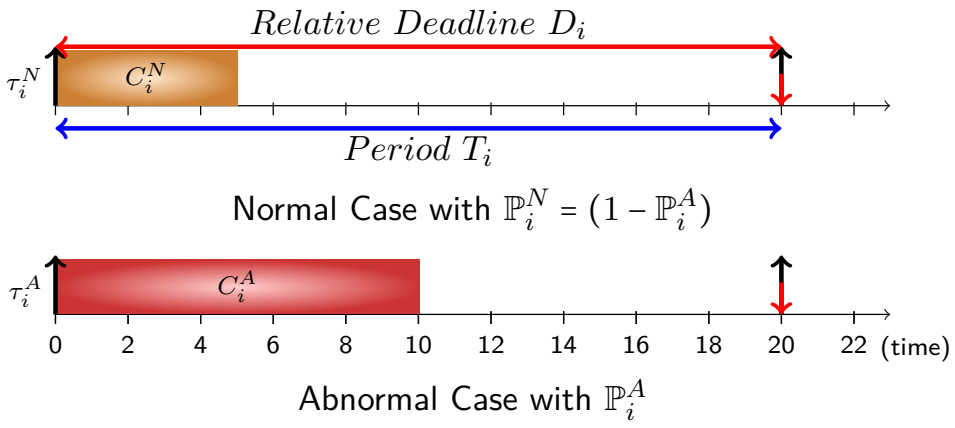


Figure 3.2: Dual modes with probabilistic execution time, where $\tau_i(\{C_i^N, C_i^A\},\ D_i,\ T_i)$.

Chapter 6), we assume that there is no data dependency among tasks, i.e., we only consider independent tasks. In this dissertation, two types of execution time models are considered, namely:

- *Multiple Execution Versions:* Each task has multiple versions as shown in Figure 3.1, regarding which the scheduler can decide which version τ_i^j to execute when a job of task τ_i arrives. In Chapter 4, we consider periodic tasks to be available in three versions, i.e., $\{\tau_i^u, \tau_i^d, \tau_i^r\}$, with different execution times $\{C_i^u, C_i^d, C_i^r\}$, concretely. Depending on selected Software-Implemented Hardware Fault Tolerance (SIHFT) methods, different versions provide different levels of soft-error handling. Since not all errors lead to critical failures of a task, but might only have an impact on the output [ESH+11], selective protection can increase the efficiency but also reduce the quality by allowing incorrect output [SAL+08].
  Providing only a low level of protection with SIHFT techniques, the unreliable version τ_i^u only protects the system from errors that lead to a system crash, while nevertheless allowing incorrect outputs. The error detection version τ_i^d, in contrast, requires additional effort to determine the correctness of the output values, for example, redundant execution, special encoding of data [SSF09], or control-flow checking [RCV+05a]. To enable error recovery mechanisms, the reliable version τ_i^r has full error detection and correction by applying error detection techniques on a larger scale than w.r.t. τ_i^d, e.g., increased redundancy and voters [CRA06].
- *Multiple Modes with Probabilistic Execution Time:* Each task τ_i has a set of h distinct execution modes and each mode j with $j \in \{1, \ldots, h\}$ is associated with a different WCET C_i^j. Assume those execution modes are ordered increasingly according to their WCETs, i.e., $C_i^j \leq C_i^{j+1}\ \forall j \in \{1, \ldots, h-1\}$. One of the

applicable cases, i.e., dual modes, is as shown in Figure 3.2 that each task has two distinct WCETs while using SIHFT techniques against transient faults.
When no fault occurs during the execution of task τ_i and therefore no error recovery is necessary, the execution is considered as a *normal* execution with a smaller WCET value denoted as C_i^N. If a fault is detected in a job of task τ_i, the related job has a longer WCET (*abnormal*), denoted as C_i^A, for potential error recovery [BKH+16]. It holds that $C_i^A \geq C_i^N \ \forall \tau_i$. The fault detection is assumed to perform perfectly and to be executed at predefined checkpoints or at the end of a job execution. This additional computation time required for the fault detection is integrated into C_i^N.
The occurrence of soft errors, i.e., that τ_i is executed abnormally, is modeled by a given probability $\mathbb{P}_i^A$. Thus, the probability of executing a job normally is $\mathbb{P}_i^N = 1 - \mathbb{P}_i^A$ for each job of τ_i. We assume that $\mathbb{P}_i^A$ is independent from previous errors and executions according to the following definition:

Definition 1 (Independent Random Variables)**.** Two random variables are (probabilistically) independent if they describe two different events such that whether one event take places or not does not have any impact on the probability that the other one occurs.

Namely, the random variables describing the execution times of jobs of the same or different tasks are assumed to be independent in this model.

3.2 Application Model for Multi-Core Processor Systems

Specifically in Chapter 6, we consider that an application is given as a task graph $G=(\Gamma, \mathbb{E})$, where Γ is a set of N nodes representing tasks, such that $\Gamma = \{\tau_1, \tau_2, ..., \tau_N\}$, and $\mathbb{E} = \{e_{xy} | (\tau_x, \tau_y)\}$ is the set of edges denoting task dependencies.

Each task τ_i has K_i task versions $\tau_i = \{\tau_{i,1}, \tau_{i,2}, ..., \tau_{i,K_i}\}$ generated by the reliability-aware compilation [SRA+13; RSK+11]. Moreover, each task τ_i has an individual relative deadline D_i for its all its versions. Although we consider homogeneous multi-core processor systems (detailed in Section 3.3), two different models need to be considered:

- Cores with homogeneous performance; each core has multi-threads.
- Cores with heterogeneous performance; each core has a single thread.

When the performance of cores is *heterogeneous*, the execution time of task version $\tau_{i,k}$ depends on the core it is assigned to. We assume the given mapping function $C_{i,k,m}(e)$ denotes the continuous cumulative distribution function of execution time, according to which the execution time of version $\tau_{i,k}$ is less than or equal to execution time e when it is executed on core c_m. With this mapping function, the deadline miss rate for a version $\tau_{i,k}$ on core c_m can be estimated as:

$$P_{\mathrm{dm}}(\tau_{i,k}, c_m) = 1 - C_{i,k,m}(D_i) \tag{3.1}$$

In addition, the expected execution time of a task version $\tau_{i,k}$ depending on the frequency of core c_m is denoted as $E(\tau_{i,k,c_m})$, which can be calculated by the continuous cumulative distribution function $C_{i,k,m}(e)$.

To guarantee the timeliness, we assume that the set of *tolerable* rates of deadline misses $\rho_\Gamma = \{\rho_1, \rho_2, \ldots, \rho_i\}$ is given as a hard real-time constraint in the system. Therefore, each task must guarantee its probability of deadline miss rate to be lower than the tolerable miss rate ρ_i. According to Eq.(3.1), we consider version $\tau_{i,k}$ to be feasible on core c_m. If it's deadline miss rate $P_{\mathrm{dm}}(\tau_{i,k}, c_m)$ is not greater than the given miss rate constraint ρ_i, i.e., $P_{\mathrm{dm}}(\tau_{i,k}, c_m) \leq \rho_i$, where $1 \leq k \leq K_i$. If there exists a task mapping such that all tasks comply with their tolerable miss rates for each of their feasible versions, we consider this task mapping as a feasible solution. Please note that in this considered model each core has only one single thread, so that the deadline miss rate is solely based on the probability distribution of the task execution time.

3.2.1 Task Redundancy Levels

We consider task-level redundancy levels formed by Triple Modular Redundancy (TMR) and also Double Modular Redundancy (DMR) [RM00; MKR02]. To execute a task with CRT-TMR, it requires three cores to provide a majority-voting mechanism (so CRT-DMR with two cores). In addition, we propose the mixed usage of Chip-level Redundant Multi-threading (CRT) and Simultaneous Redundant Multi-threading (SRT), called Mixed Redundant Threading (MRT), under which TMR can also be activated on two parallel cores with two replicas, i.e., the original execution and one replica are executed on one core while a second replica is executed on a second core. By using all possible combinations, the following six redundancy levels can be defined:

- NON-RMT (ϕ): the task without any redundancy.
- SRT-DMR : the original and one replica executed sequentially.
- SRT-TMR : the original and two replicas executed sequentially.
- CRT-DMR : the original and one replica executed in parallel.
- CRT-TMR : the original and two replicas executed in parallel.
- MRT-TMR : the original and one replica executed sequentially, one replica executed in parallel, i.e., original and two replicas in total.

These redundancy levels can be characterized as a set of directed acyclic graphs (DAGs) [LCA+14]. Figure 1.5 illustrates the possible DAGs, where each node (sub-task) represents a sequence of instructions and each edge represents execution dependencies between nodes. Each node is characterized by the WCET of the corresponding sub-task. A node is ready to be executed if all of its predecessors have been executed. Each box represents a processor, i.e., nodes in the same block are assigned to the same processor. We assume that each task τ_i has K_i given levels generated by the reliability-aware compilation [RSK+11; HWZ06], each matching one

of the above six redundancy levels, i.e., $\tau_i = \{\tau_{i,1}, \tau_{i,2}, \cdots, \tau_{i,K_i}\}$. For each task $\tau_i \in \Gamma$, one of the levels is chosen to be executed, denoted as $\theta = \{\theta_1, \theta_2, \ldots, \theta_N\}$. If task τ_i is executed without any Redundant Multi-Threading (RMT), θ_i is ϕ; otherwise, θ_i is the above any other levels. For each level $\tau_{i,j}$, two parameters are assumed to be given:

- The total execution time $C_{i,j}$: The sum of the WCETs of all the sub-tasks of $\tau_{i,j}$.
- The critical-path length $L_{i,j}$: The WCET of the critical-path in the given DAG, i.e., the sum of the WCETs of the path with the longest total WCET.

We assume that the correctness of the execution result is affected by soft-errors (faults). The WCET is profiled off-line in a fault-free system and is protected by the adoption of watchdog approaches. Each level $\tau_{i,j}$ is associated with two cost values; namely, the utilization $u_{i,j}$ and the number of required cores $H_{i,j}$. The utilization $u_{i,j}$ is given by $\frac{C_{i,j}}{T_i}$ and $H_{i,j}$ is determined by the scheduling policy detailed in Section 6.3.1. We assume that each task τ_i has at least a level $\tau_{i,\phi}$ without any redundancy and that its total execution time is not larger than its period.

Particularly for Section 6.2 dealing with performance heterogeneity, Γ_{TMR} and Γ_ϕ denote the set of tasks which are executed in CRT-TMR mode or without RMT, respectively. The cardinality of a set $\mathbf{X}$ is denoted as $|\mathbf{X}|$. $\delta_{\vec{\theta}}$ denotes the number of cores to satisfy all tasks activated in the decided redundancy levels $\vec{\theta}$, where $\delta_{\vec{\theta}} = |\Gamma_\phi| + 3 \times |\Gamma_{\text{TMR}}|$. We assume the voter and the interconnect between cores are both hardened by special treatments without further effects from errors as in [RSS+17; HLD+15]. Therefore, the reliability penalty of TMR-based RMT can be a negligible value ε, where $\varepsilon \geq 0$. With respect to the execution time, a task executed in TMR-based RMT mode has to wait for all redundant threads completing their jobs due to the usage of majority-voting mechanism. As shown in Fig. 1.4, we can observe the fact that the slowest thread on core C_3 will dominate the execution time of the task. If the redundant thread on core C_3 spends too much execution time, it may lead to a deadline miss. Therefore, we can safely estimate the execution time of the task in RMT mode by the execution time of its redundant thread on the lowest-frequency core of the assigned core group.

3.3 Hardware Model

For the chapters targeting uniprocessors (i.e., Chapters 4, 5, and 7), no specific hardware model is assumed. However, for Chapter 6, targeting multi-core systems, we consider a multi-core processor $\mathbb{C} = \{c_1, c_2, ..., c_M\}$ with M ISA-compatible Reduced Instruction Set Computing (RISC) cores that are connected in a communication fabric, e.g., a Network on Chip (NOC). Each core c_i has its own instruction and data cache to execute tasks. Let $\mathbb{G}_i$ be a subset of cores from a many-core processor $\mathbb{C} = \{c_1, c_2, \ldots, c_M\}$, which is detailed in the next section. If $\mathbb{G}_i$ has three cores, it

is eligible for the task activated with CRT-TMR, which is called the *core group* for brevity. Throughout this dissertation, we assume that all additional communication overhead can be integrated into the execution time of tasks, e.g., NOC overhead, cache coherency traffic, or shared resource accesses.

3.4 Experimental Platforms

In this dissertation, several experimental evaluations are conducted to evaluate the proposed scheduling algorithms and analyses. The applied platforms are introduced hereinafter.

3.4.1 Synthesized Task Sets

For uniprocessor systems, we apply the UUniFast [BB05] method to synthesize task sets for numerical simulations or an event-driven simulator. Given a utilization value U^N_{sum} and a number of tasks N for a desired set, the UUniFast method can randomly generate implicit-deadline task sets. According to the suggestion from Emberson et al. [ESD10], the task periods are generated according to a log-uniform distribution with targeted orders of magnitude. For each task τ_i, the normal utilization is assigned with a utilization value U^N_i, while the execution time C^N_i is set to $U^N_i \cdot T_i$[1].

For multi-core systems, we apply the UUniFast-Discard [DB09] method, which is an extended version of UUniFast for the multiprocessor (or even multi-core) domain. This algorithm applies UUniFast unchanged for total utilization $U^N_{\text{sum}} > 100\%$, and discards any task set which contains an individual task utilization U^N_i greater than 100%. Unfortunately, this algorithm becomes increasingly inefficient for large numbers of task number N with values of U^N_{sum} close to $N/2$ [ESD10].

3.4.2 Lego Mindstorms NXT Robot

To demonstrate the scheduling approach proposed in Chapter 4, a Lego Mindstorms NXT robot [Leg] was used as a computing unit for control applications. The microcontroller is powered by an Atmel 32-bit ARM main processor, which runs at 48 Mhz, while having 64 KB RAM and 256 KB flash memory. The co-processor is an 8-bit AVR ATmega48 processor running at 8 Mhz with 512 Bytes RAM and 4 KB flash. The robot has up to four sensors at its disposal as well as three output ports on its top to control up to three motors.

A Real Time Operating System (RTOS), named nxtOSEK [Chi13] is developed, allowing us to program custom applications with C/C++ and providing access to the Lego robot's sensors and motors. NxtOSEK as a port of the Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen (OSEK) standard relies on two types of files: The OIL file describes the properties of tasks and

[1] The execution time for a unreliable version C^u_i is assigned to $U^u_i \cdot T_i$.

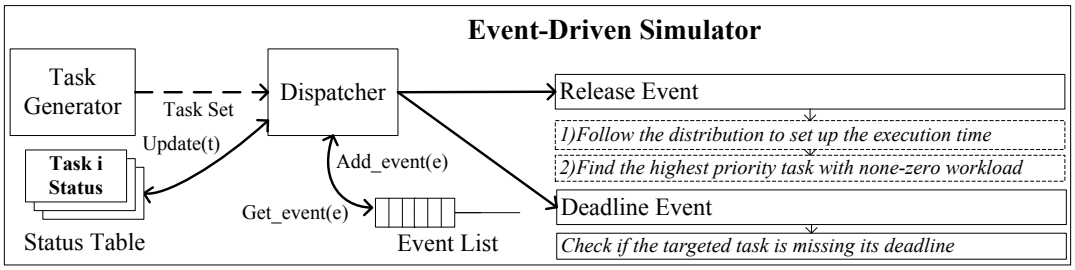


Figure 3.3: Overview of the event-driven uniprocessor simulator released on [Kua18].

their alarms, stacksize, etc. For each task, the task body which contains the motor-, display-, sensor- and other functions should be designed in the C (or C++) files.

3.4.3 Event-Driven Uniprocessor Simulator

We also implemented an event-driven simulator[2] written in Python 2.7 to simulate the Rate-Monotonic (RM) policy as well as fault occurrences.

An overview about the simulator is shown in Figure 3.3. For each task τ_i, there are only two types of events in the simulator: `release` and `deadline`. A `release` event of τ_i adds its new workload to the entry of τ_i in the status table and places a `deadline` event of τ_i into the event list. The `deadline` event of τ_i will check if the remaining workload of τ_i is zero in the status table, i.e., τ_i meets its deadline, or not, i.e., one deadline is missed. The main components of the simulator are listed in the following:

- **Task Generator:** By applying the aforementioned UUniFast method [BB05] and the suggestion from Davis et al. made in [DZB08], the task generator outputs a set of tasks under a given utilization value U^N_{sum}, where $U^N_{\text{sum}} = \sum_{\tau_i \in \Gamma} U^N_i$.
- **Dispatcher:** It checks if the number of jobs released by the targeted task (by default the lowest priority task) is equal to the targeted number. If not, it continues to dispatch the next event in the event list.
- **Event List:** This linked list[3] is used to keep tracking the following events in the simulated task system. When a new event is inserted by another `release` event, the events in the list are sorted according to their future occurring time.
- **Status Table:** It records the number of deadline misses, the number of released jobs, and the remaining workload for each unfinished job of a task.

According to the considered model in Chapter 5, jobs are never aborted in the simulator. If a job misses its deadline, the remaining portion of execution time is still executed at the same priority level before the next job of the task can start executing. Whenever the dispatcher gets a new event, the workloads of the tasks in the status

[2]The complete scripts are available at [Kua18].
[3]This can be easily upgraded to any sortable data structure with a lower time complexity.

table are updated with respect to the elapsed time from the previous to the current event and the processor is assigned to the highest priority task under a fixed-priority scheduling policy. If no ready task exists in the system, the processors runs idle until the next job is released.

Please note that the two dashed blocks in Figure 3.3 belonging to the release events are compatible with different considered models: 1) For this dissertation, we followed the considered task model to implement a fault injection module, so that each task can only be released in two different execution modes, either related to C_i^N with high probability P_i^N, or related to C_i^A with low probability P_i^A. This part can be easily revised to fit any execution time distribution; 2) As we consider a preemptive fixed-priority scheduling policy in this dissertation, the task with the highest priority, which has non-zero workload, will be executed. This part can be easily extended for non-preemptive task systems and dynamic-priority scheduling policies.

3.4.4 Reliability-Aware Many-Core Simulator

In Chapter 6, we use an in-house reliability-aware many-core simulator provided by our partner in the SPP1500-Project – Generating and Executing Dependable Application Software on UnReliable Embedded Systems, Prof. Dr.-Ing. Muhammad Shafique, to build up our framework. We refer the reader to more details in [RSK+11; Reh15; RCK+16].

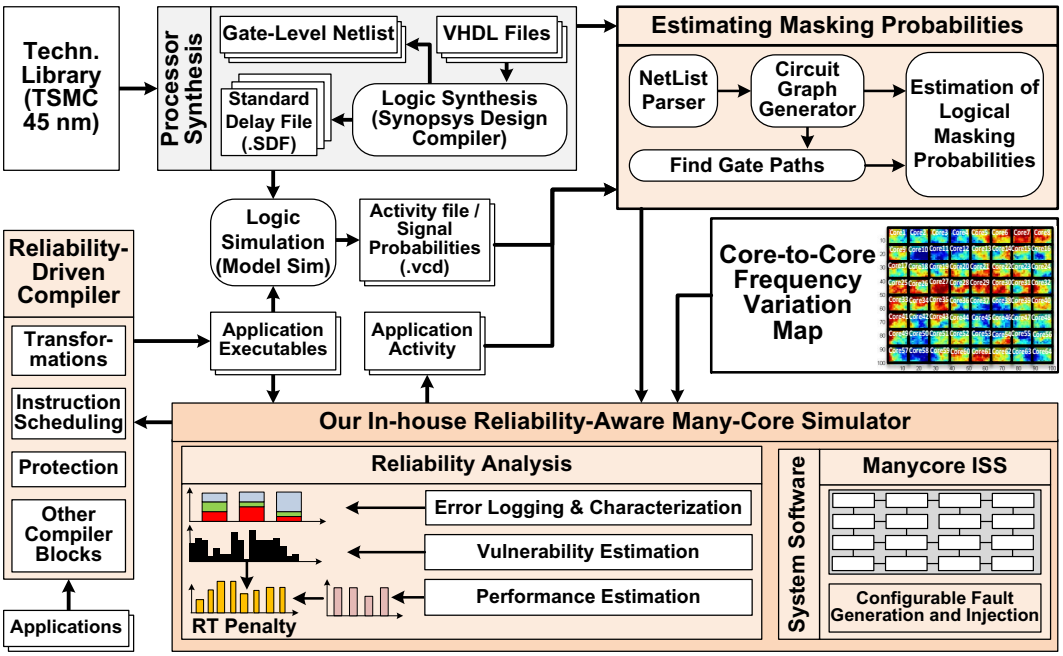


Figure 3.4: Experimental setup with reliability-driven compiler, system software, and processor simulator.

Overall, the framework as shown in 3.4 employs a reliability-aware many-core simulator with integrated configurable fault generation and injection modules. Each core implements the SPARC-v8 ISA (used in LEON2 and LEON3 cores), which is generated using the ArchC architecture description language and related tools [ARB+05]. We extended the simulator with in-house developed configurable fault generation and injection modules and error analysis and logging functionalities. These are required to perform an in-depth reliability and vulnerability analysis.

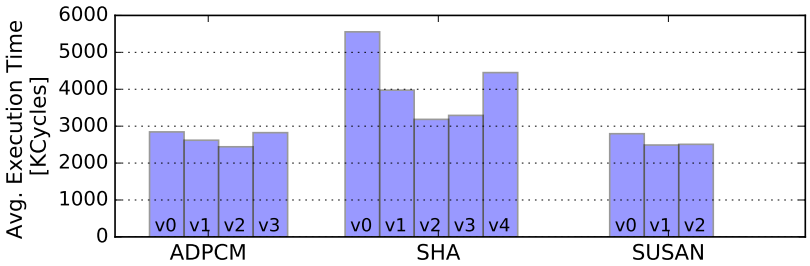
For accurate reliability estimation, we synthesized the LEON3 cores using the Synopsys Design Compiler for a TSMC 45nm technology library to obtain area, frequency, and logical masking probabilities. We performed gate-level error masking and propagation analysis on the netlists to obtain logical masking probabilities of different processor components. These probabilities are then used to obtain the instruction vulnerabilities that are later used to estimate task reliability penalties; (see detailed procedure in [RSK+11; RCK+16]). For the fault scenario generation, different parameters (e.g., the number of bit flips per fault, the fault rate using the neutron flux calculator [Wil06], and the coordinates of a given location, fault distribution, etc.) are used.

Faults in different processor components are randomly injected (as also done in [SWK+05; MWE+03]) during the execution of a given function version. Their effects on the application output are monitored using an error logger. Possible Errors are categorized w.r.t their severity from the user's perspective (e.g., application failure, incorrect output, correct output). The results of the fault injection experiments are used (1) to estimate the software-level vulnerability and masking properties of the applications; (2) to analyze the reasons of application failures, e.g., accessing prohibited memory regions and non-decodable instructions. Again, we assume the voter and the interconnect between components are both hardened by special treatments as in [RSS+17; HLD+15].

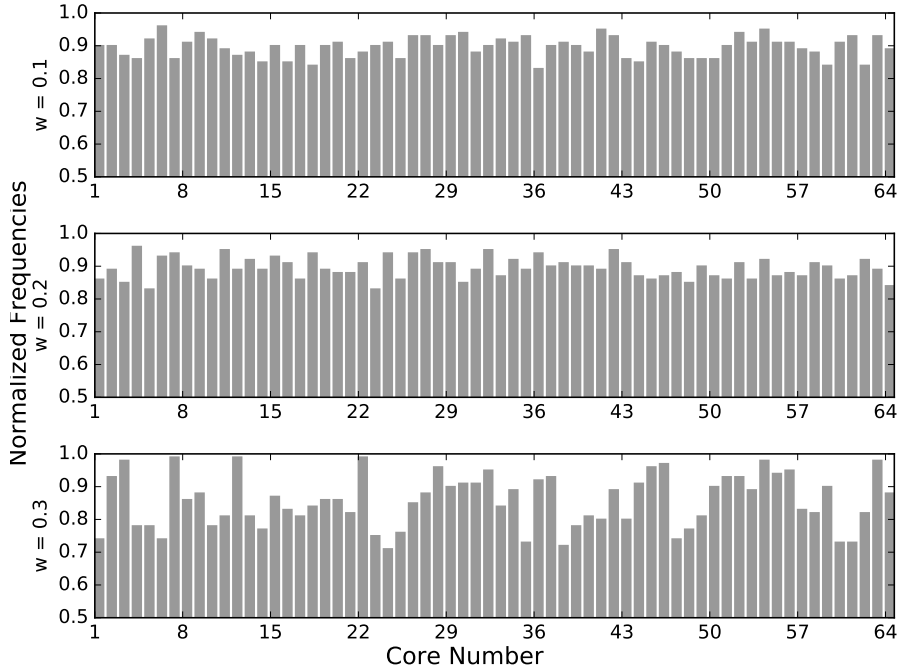
For the task set we analyzed, seven tasks were chosen from the embedded benchmark MiBench [GRE+01]: (1) SAD, (2) ADPCM, (3) CRC, (4) SusanS, (5) SHA, (6) SATD, and (7) DCT, where the data dependencies are as follows:

- DCT → SAD → SATD
- ADPCM → CRC
- SUSAN → SHA

Each task is compiled with a reliability-driven compiler, which is based on the GCC framework. Different reliable function versions are generated by applying different reliability-driven transformations [RSK+11; RKS+14a] and a reliability-driven instruction scheduling algorithm [RSH12]. These reliable function versions provide trade-off points for reliability vs. performance. Fig. 3.5(a) shows an example in our setup, according to different function versions under the same frequency core have different performance. A subset of Pareto-optimal versions is selected and used by the run-time system. For each compiled version, we estimate the performance and the values of reliability penalties under two different fault rates, i.e., 1 fault/1MCycles



(a) Performance of tasks on the same frequency core.



(b) Core-to-core frequency variation under normal distribution for 8×8 cores.

Figure 3.5: Performance heterogeneity of the experimental setup.

and 10 faults/1MCycles, which are computed from the neutron flux, fault probability, processor layout, and the processor frequency, which conforms to the test conditions opted by the related work [HWZ06; LDV+04]. The reliability penalty for each function (task) is estimated using the approach in [RSK+11; RCK+16].

3.4.5 Real System Simulation

To analyze the behavior of *Systems with Dynamic Real-Time Guarantees* [BKH+16], we select Real-Time Executive for Multiprocessor Systems (RTEMS) [RTE13] version 4.11[4] to deploy experimental evaluations, where the used kernel is enhanced

[4]The latest release in 2018 is version 5, but it is still an ongoing branch. The proposed enhancement is inherited from 4.11 without further changes.

by the contribution in Chapter 7. In [BKH+16], we adopted QEMU emulators to deploy testing instances on RTEMS with the chosen board support package, *RealView Platform Baseboard Explore for Cortex-A9*.

```
rtems_task Task_i(
    rtems_task_argument unused
)
{
    rtems_id            selfid = rtems_task_self();
    rtems_status_code   status;
    rtems_id            RM_period;
    \\...

        period_name = rtems_build_name( 'P', 'E', 'R', 'i' );
        status = rtems_rate_monotonic_create(period_name, &RM_period);
        if( RTEMS_SUCCESSFUL != status ) {
                printf("RM failed with status: %d\n", status);
                exit(1);
        }
        while(TRUE){
                status = rtems_rate_monotonic_period(RM_period, givenTi);

                update_monitor();
                LOOP(normal_wcet);

                task_fault = faultInjection();
                if(task_fault == TRUE){
                        update_monitor();

                        remaining_time = abnormal_wcet - normal_wcet;
                        if(remaining_time != 0)
                                LOOP(remaining_time);

                }

                check_deadline(deadline, task_type, selfid);
        }
}
```

Listing 3.1: Illustrated example of task body in RTEMS version 4.11.

For each task, the period and the priority are given and configured off-line. Listing 3.1 presents the pseudo code of the task body implemented in RTEMS. As we can see that `LOOP()` is implemented by simulating the computation time with the given WCET. To simulate the occurrence of faults, a specific procedure is triggered in the moment in which a task instance finishes its normal execution with a random draw, based on the given probability of faults per millisecond. Here C_i^N determines if the execution is prolonged to run up to a longer WCET C_i^A or not. To monitor the system state, the enhancement proposed in Chapter 7 provides a helper function to calculate the carry-in workload for each task potentially incurred by deadline misses for each task.

CHAPTER 4

Meeting Control Robustness via Soft-Error Compensation

Contents

4.1 Overview

To avoid catastrophic events like unrecoverable system failures caused by soft errors, Software-Implemented Hardware Fault Tolerance (SIHFT) techniques have been proposed without the needs of special hardware supports. However, maintaining the correctness of all executions by trivially using them to each job can be very costly due to their additional computation time. As a result, how and when to apply such software-based approaches should consider the properties of applications and the scheduling strategy to keep the timeliness guarantees.

From the initial experiment we did in [KBC+16] (shown in Chapter 1.2.1), we observed that, a limited number of errors may not be that fatal due to the inherent safety margins and noise tolerance in control applications. In control theory literature, several techniques have been proposed to aid control applications to be stable if some signal samples are delayed [Ram99; KGC+12] or even dropped [HSJ08; BS15]. Although they can describe the tolerance for such a delayed/dropped occurrence in the (m, k) requirement or even more flexible model for varying intervals, none of them have discussed how to actively decide when to maintain the correctness of executions by using scheduling techniques. In most of control systems, having a high quality control is the main objective. However, satisfying the minimal requirement may only provide a minimum acceptable control performance. In order for the system to meet both control robustness requirements and timeliness requirements, an adaptive deployment providing high quality of control most of time without utilizing too much timing resource is thus desired.

In this chapter, we introduce how the above fault-tolerance can be modeled as a (m, k) robustness requirement and how a given (m, k) robustness requirement can be satisfied by adopting patterns of task jobs with individual error detection and compensation capabilities. Based on an off-line scheduling analysis, we propose a dynamic compensation approach to guarantee the timeliness even under the worst case that all the jobs are affected by soft errors, while reducing the average utilization of the system, which can result in energy reduction for embedded systems. *To the best of our knowledge, this is the first work presenting a selective runtime adaption for soft-error handling without skipping any job.*

The presentation is organized as follows: In Section 4.2, a pattern-based method to enforce the reliable executions is introduced. In Section 4.3, we propose a runtime adaptive approach called *dynamic compensation*. In Section 4.5, the experimental evaluation demonstrates the effectiveness of the proposed approaches. Section 4.6 summarizes the chapter. Parts of this chapter were originally published on LCTES 2016 [KBC+16] and EITEC 2018 [YKC18].

4.1.1 Motivational Example

Suppose that we are given two tasks τ_1 and τ_2 with properties as defined in Table 4.1. To satisfy the given requirement $(m_2, k_2) = (1, 1)$, only τ_2^r is taken for executions,

Task	(m_i, k_i)	C_i^u	C_i^d	C_i^r	T_i
τ_1	$(2,4)$	1	$1+\varepsilon$	2	4
τ_2	$(1,1)$	x	x	5	8

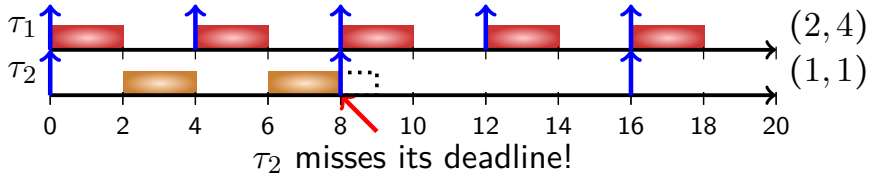
Table 4.1: Example task set properties.

which requires computation time $C_2^r = 5$ for each job. Assuming transient faults occur at $t = 0$ and $t = 8$ on task τ_1, the example in Figure 4.1 demonstrates execution scenarios for different compensation strategies. For simplicity of presentation, the provided diagram starts from time point $t = 0$.

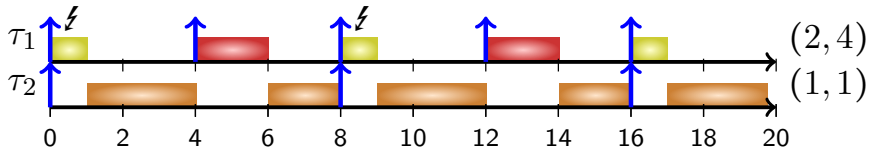
If all τ_1 jobs are trivially activated with τ_1^r to prevent any effects from soft errors like Figure 4.1a, τ_2 is clearly not schedulable due to the processor overload, by which the system utilization is over 100%, i.e., $\frac{2}{4} + \frac{5}{8} > 1$. To comply the requirement (m_1, k_1) for τ_1, one way is to statically distribute the execution of reliable jobs τ_1^r and unreliable jobs τ_1^u in an alternating pattern. This static approach is introduced as *Static Pattern-Based Reliable Execution* in Section 4.2. As shown in Figure 4.1b, directly executing τ_1^r on the second and forth jobs will guarantee satisfaction of the requirement $(m_1, k_1) = (2, 4)$ and avoid the processor overload even all the jobs are erroneous. However, it is obvious that this approach is over-provisioning, since the fault does not occur on the second job under this distribution of errors, in which the potential correctness of the second job. In addition, the overall utilization now is 100%, which may not be good in terms of energy-saving.

In this chapter, the considered system allows limited number of erroneous executions but requires to satisfy all the given hard real-time constraints. To achieve this treatment against soft errors, we provide a run-time adaptive approach called *Dynamic Compensation* that enhances *Static Pattern-Based Reliable Execution* by recognizing the need to execute reliable jobs dynamically instead of having a static schedule. As shown in Figure 4.1c, we can see that reliable execution is only activated once on the fourth job, since satisfaction of the requirement $(2, 4)$ would only be broken if an error occurs in this job. If the failure rate of the system is low or k_i is larger than m_i greatly, the amount of expensive reliable executions can be reduced significantly in this way. However, if there is an additional fault which occurs at $t = 4$, the above dynamic approach is not schedulable as Figure 4.1d. This can be solved by using a more schedulable pattern to avoid consecutive reliable executions, i.e., once there is a fault at 0, the following instance at 4 must execute the recovery version.

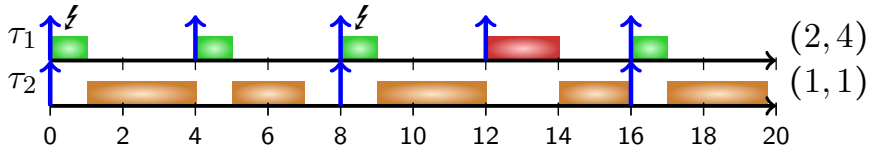
Up to here, we can see that applying SIHFT techniques efficiently is not a trivial task. While (m_i, k_i) control robustness requirements need to be satisfied, the timeliness of the real-time system also needs to be guaranteed. *If the reliable execution can only be activated right before the moment that the requirement would not be satisfied, the resulting reduction of execution time can be utilized to save energy, which may be good to most mobile and embedded devices with the limitation of battery capacity.*



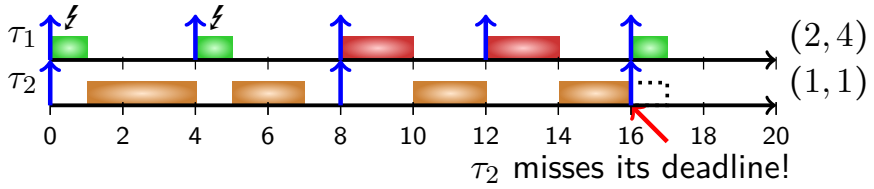
(a) τ_1 is fully protected with reliable versions.



(b) τ_1^r is statically distributed.



(c) τ_1 is dynamically compensated with a reliable execution on the forth instance.



(d) τ_1 is not feasible if an additional fault occurring at $t = 4$.

Figure 4.1: To deal with two occurring faults (ϟ), there are different ways: The red block presents the reliable executions τ_i^r, the green block presents the executions with error detection τ_i^d, and the yellow block presents the unreliable version τ_i^u.

4.1.2 Fault Model

In this chapter, we deal with potentially wrong values in the data transfer of the motor and the light sensor values caused by soft errors in the system. The probability of the occurrence of a fault is assumed, and every task job has at most one fault under Single Event Upset (SEU) [WA08; Alt13]. Without using error detection and correction, the considered system can still be executed without a system crash. However, the wrong values in the data transfer may degrade the control performance

and finally lead to a mission failure, e.g., under soft errors the robot deviates from or leaves the track, but its operating system does not crash.

4.1.3 Problem Definition

From the above example, the problem addressed in this chapter can be stated as follows: Suppose that we are given a set of timing-independent and preemptive control tasks $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$. Each task τ_i is associated to an individual (m_i, k_i) control robustness requirement, which means that m_i out of any k_i consecutive jobs must be correct, e.g., see [GNB18]. We assume this (m_i, k_i) requirement can be given by other means analytically [KGC+12; GDD19] or empirically [BS15; YKC18].

Without restriction to a specific method, each task is assumed to have one unreliable version τ_i^u with its Worst-Case Execution Time (WCET) C_i^u, one unreliable version with fault detection (namely detection version) τ_i^d with its WCET C_i^d, and one reliable version τ_i^r with its WCET C_i^r, respectively. Due to the rising overhead for error detection as well as for error correction, we assume that $C_i^u < C_i^d < C_i^r$, $\forall \tau_i \in \Gamma$ holds. The objective of this chapter is to derive an analytical solution utilizing the processor efficiently by reducing the amount and thus the overhead of reliable jobs τ_i^r such that the system can satisfy its hard real-time constraints and (m_i, k_i) requirements without skipping any job.

To schedule all the above control tasks on a uniprocessor system, the Rate-Monotonic (RM) scheduling is considered throughout this chapter. All the control tasks are indexed from 1 to n where τ_1 has the highest priority and τ_n has the lowest priority. As stated in Chapter 2.1.3, using Liu and Layland (L&L) bound [LL73] and the Time-Demand Analysis (TDA) may unnecessarily reject many task sets which are schedulable actually. In this chapter, a sufficient schedulability test based on a multi-frame task model [MC97] is provided to validate the proposed scheduling approaches.

The objective of this chapter is how to efficiently utilize the processor by reducing the number and thus the overhead of reliable executions τ_i^r such that the system can satisfy both its hard real-time and (m_i, k_i) requirements while maintaining low overall utilization without skipping any job. The off-line solution to this problem is presented in Section 4.2, whereas the on-line solution is presented in Section 4.3. In Section 4.4, an empirical approach to obtain (m, k) requirements is introduced.

Please note that the proposed approaches are not limited to the RM scheduling. They can be easily extended for constrained-deadline tasks, in which $D_i \leq T_i$, $\forall \tau_i \in \Gamma$, and the priority assignment policy should be changed to Deadline-Monotonic (DM), by which the shorter the relative deadline, the higher the assigned priority.

4.2 Static Pattern-Based Reliable Execution

In this section, we first show how to enforce the (m_i, k_i) requirement by applying (m, k) static patterns to allocate the reliable executions for a specific task τ_i. While the adopted pattern may affect the schedulability, stability, and flexibility, deciding the most suitable pattern is out-of-scope of this work. The scheduling analysis and the illustrated example are presented at the end of this section.

4.2.1 Static Pattern and Soft-Error Handling

To fully utilize the fault tolerance, it should be clear that the most efficient way is to execute the reliable version of task τ_i only at the essential jobs by which the amount of reliable jobs is equivalent to m_i for every k_i consecutive jobs for a (m_i, k_i) requirement. To ease the static analysis as well as to reduce the implementation cost, we adopt the well-known concept of (m, k)-patterns [QH00; NQ06; Ram99; KS95] that defines a partitioning of jobs within any k_i consecutive jobs. To adapt the concept for the targeted purpose, the partitioning is redefined as follows:

Definition 2 ((m, k)-pattern)**.** (m, k)-pattern of task τ_i is a binary string $\mathbb{B}_i = \{\beta_{i,0}, \beta_{i,1}, \ldots \beta_{i,(k_i-1)}\}$ which satisfies the following properties: 1) $\beta_{i,j}$ is a reliable job if $\beta_{i,j} = 1$ and a unreliable job if $\beta_{i,j} = 0$ and 2) $\sum_{j=0}^{k_i-1} \beta_{i,j} = m_i$.

By repeating the (m, k)-pattern $\mathbb{B}_i$, a job pattern of τ_i can be obtained. If we can guarantee that all reliable jobs in (m, k)-pattern are all correct, a (m_i, k_i) requirement for τ_i must be satisfied. A trivial way is to directly execute the reliable version τ_i^r for every m_i jobs, which is called *Reliable Execution* (RE) for the rest of chapter. However, directly applying the reliable version on each reliable job is not the only option. Giving a try with an unreliable version before directly executing the reliable version in a same period may also be feasible to deliver the correct jobs, which is called *Detection and Recovery* (DR). To notate briefly, both static approaches (static-RE and static-DR) for the rest of this chapter are denoted as Static Pattern-Based Reliable Execution (SRE) and SDR, respectively.

For implementation, each control task τ_i can use an index to identify the current job on a (m, k)-pattern $\mathbb{B}_i$ with a given (m_i, k_i) requirement. When the current job in $\mathbb{B}_i$ is reliable, the reliable version τ_i^r and the detection version τ_i^d should be executed accordingly depending upon the adopted strategy, i.e., RE or DR. In contrast (index points to an unreliable job), the control task keeps executing the unreliable version τ_i^u without fault detection safely. After all, (m_i, k_i) requirement will be satisfied through RE or DR with a static (m, k)-pattern that the number of reliable jobs within window size k_i must be equal to m_i.

4.2.2 Offline Scheduling Analysis

Due to the availability of multiple versions for each τ_i, the periodic control tasks may have different distinguishable execution times depending on the executing versions.

To validate the system schedulability, the multi-frame task model proposed by Mok and Chen [MC96] can be applied for describing the studied task set. Each task can be transformed to a multi-frame real-time task τ_i with k_i frames, period T_i, and an array of different execution times, i.e., $\{C_i^0, C_i^1, \ldots, C_i^{k_i-1}\}$, in which the array of execution times for each task can be determined by given (m,k) patterns. Without loss of generality, each task is assumed to has at least two frames, i.e., $k_i \geq 2$. If a task has a $(1,1)$ requirement, we can artificially create a multi-frame task with two same execution time frames.

Definition 3 (Maximum of the sum of the execution times)**.** For any σ consecutive frames of task τ_i, let $\Omega_i(\sigma)$ be the maximum of the sum of the execution times. For brevity, we define $\Omega_i(0) = 0$.

Therefore, $\Omega_i(1)$ is $\max_{j=0}^{k_i-1} C_i^j$ and $\Omega_i(2)$ is $\max_{j=0}^{k_i-1}(C_i^j + C_i^{((j+1) \mod k_i)})$. It is not difficult to see that $\Omega_i(\sigma)$ is equal to $\Omega_i(\sigma \mod k_i) + \lfloor \frac{\sigma}{k_i} \rfloor \sum_{j=0}^{k_i-1} C_i^j$ when $\sigma > k_i$.

With the critical instant of multi-frame task by Definition 5 in [MC96], the schedulability test of task τ_q can be given as follows[1]:

Lemma 1. Suppose that all the multi-frame tasks in $hp(\tau_q)$ under fixed priority scheduling on a uniproccessor are schedulable. Multiframe task τ_q is schedulable, if

$$\exists t \text{ with } 0 < t \leq T_q \text{ such that } \Omega_q(1) + \sum_{\tau_i \in hp(\tau_q)} \Omega_i\left(\left\lceil \frac{t}{T_i} \right\rceil\right) \leq t \tag{4.1}$$

Proof. This directly comes from Theorem 5 and Lemma 6 by Mok and Chen in [MC96]. By using the definition of critical instant [MC96], we can ensure that task τ_q must be schedulable under fixed-priority assignment, if there exists a time point t, where the worst case response time is less than deadline T_q. □

To calculate Eq. (4.1), $\Omega_i(\sigma)$ for each task τ_i must be found out, where $\sigma = 1, 2, \ldots, k_i - 1$. One trivial way is to construct a look-up table for the first k_i entries, and derive $\Omega_i(\sigma)$ in $O(k_i^2)$ for $\sigma = 1, 2, \ldots, k_i - 1$. To test the schedulability in off-line, all the task frames in the test must be considered under the assumption that reliable executions always follow a given pattern to take place statically. Depending on which strategy is applied for reliable executions, i.e., RE or DR, the peak frames with the maximum execution time and $\Omega_i(\sigma)$ for each task τ_i should be different in the worst case. Suppose that each task is given a (m,k) pattern $\mathbb{B}_i$, the precise rules to calculate $\Omega_i(\sigma)$ can be defined as follows:

- **Detection and Recovery (DR)**: For each unreliable job marked as "0", the execution time should be considered as C_i^u for the unreliable version τ_i^u. As the worst case is re-executing τ_i^r right after τ_i^d in the same period, each reliable job marked as "1" in $\mathbb{B}_i$ should be considered as $C_i^d + C_i^r$.

[1] Typically in the literature regarding scheduling and priority assignment, τ_k is often used instead of τ_q when describing a specific task. However, k in this chapter is already used in (m,k) requirement and thus a different parameter q is used here.

Task	(m_i, k_i)	C_i^u	C_i^d	C_i^r	T_i
τ_1	$(2,4)$	1	2	3	4
τ_2	$(1,1)$	x	x	5	10

Table 4.2: Task set properties for demonstrating schedulability tests.

- **Reliable Execution (RE)**: For each unreliable job marked as "0", the execution time is set as C_i^u. As the worst case is executing τ_i^r directly with RE, the execution time of each reliable job marked as "1" in $\mathbb{B}_i$ is C_i^r.

The following example illustrates how Eq. (4.1) can be applied for different strategies:

Example 1. Suppose that there are two tasks τ_1 and τ_2 in Γ, and the corresponding properties for τ_1 and τ_2 are given as shown in Table 4.2. The given pattern for task τ_1 is E-pattern [Ram99], by which $(2,4)$ requirement can be represented as $\mathbb{B}_1 = \{0,1,0,1\}$. For simplicity, τ_2 requires $(1,1)$ and only has τ_2^r to execute.

For DR strategy, according to the above rule, the pattern $\mathbb{B}_1$ should be transferred as $\left\{C_1^u, C_1^d + C_1^r, C_1^u, C_1^d + C_1^r\right\}$. By checking with Eq. (4.1), none of t from 0 to 10 can pass the schedulability test. For example, when $t = T_2 = 10$,

$$\Omega_2(1) + \Omega_1\left(\left\lceil \frac{10}{4} \right\rceil\right) > 10 \tag{4.2}$$

where $\Omega_2(1) = C_2^r$ and $\Omega_1(3) = C_1^u + 2C_1^d + 2C_1^r$. Hence, task τ_2 is deemed to be unschedulable with DR strategy. For RE strategy, pattern $\mathbb{B}_1$ can be transferred to $\{C_1^u, C_1^r, C_1^u, C_1^r\}$. Based on Eq. (4.1), we can test whether task τ_2 is schedulable. As shown in Eq (4.3), when $t = 8$:

$$\Omega_2(1) + \Omega_1\left(\left\lceil \frac{8}{4} \right\rceil\right) \leq 8 \tag{4.3}$$

where $\Omega_2(1) = C_2^r$ and $\Omega_1(2) = C_1^u + C_1^r$. Therefore, the given task set can be schedulable with RE strategy. □

4.3 Dynamic Compensation

As revealed in the motivational example, it is pessimistic to allocate the reliable jobs strictly due to the fact that soft errors randomly occur from time to time and limited faults are tolerable. To obtain the timeliness guarantee, Lemma 1 must be applied with the assumption that reliable executions follow a given pattern to take place statically. In fact, it is possible to adaptively schedule tasks in run-time but still guarantee the behaviors in the worst-case scenarios. Here we propose a runtime adaptive approach, called Dynamic Compensation (DC), to decide the executing task version in run-time by enhancing Static Pattern-Based Reliable Execution and monitoring the

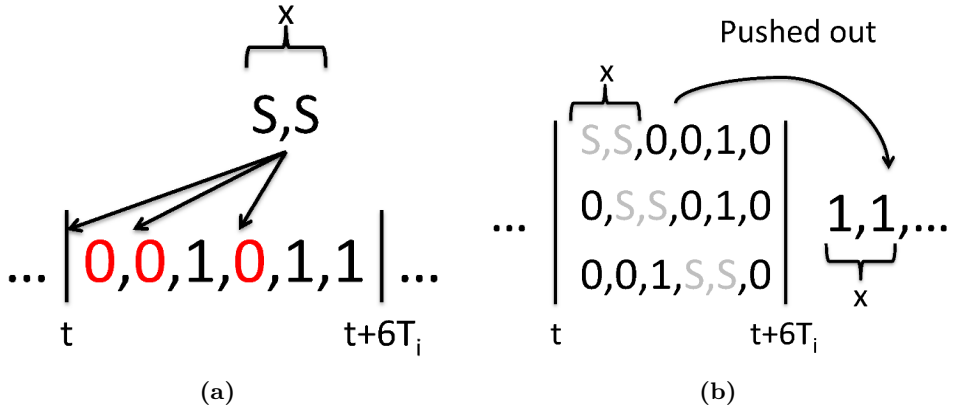


Figure 4.2: Example of successful executions in the proof of Theorem 2, in which (a) is the original jobs of τ_i and (b) is the jobs of τ_i after the insertion. x number of S insertions at most only push out x reliable jobs.

erroneous jobs with sporadic replenishment counters. The main idea is to execute the unreliable jobs as many as possible and exploit their successful executions to postpone the moment that the system must enforce the (m_i, k_i) requirement. It is worth noting that the resulting distribution of execution jobs can still follow the binary string of (m, k) static patterns even in the worst case. Please note that, we only consider the detection version τ_i^d for the execution of unreliable jobs in the DC approach in order to know whether the derived result is correct or not on-the-fly.

4.3.1 Preprocessing

Suppose that a static pattern $\mathbb{B}_i$ is a binary string and given as the initial input. In Section 4.2.1, only the minimum amount of reliable executions is used to comply the (m_i, k_i) requirement without considering the positive impact of successful unreliable jobs. However, a successful execution of an unreliable job can still be counted as a correct run like a reliable execution. Such cases can be handled carefully in the DC approach to ensure that the future jobs can still satisfy the (m_i, k_i) requirement.

The key idea is to *postpone the adoption of the original binary string* $\mathbb{B}_i$ *due to unreliable jobs with successful executions*. For the simplicity of presentation, an S denotes a successful execution for each unreliable job, by which each successful execution can insert one S into the original binary string $\mathbb{B}_i$. Theorem 1 shows that the above procedure can still satisfy the (m_i, k_i) requirement:

Theorem 1. Given a control task τ_i with a (m_i, k_i) requirement, a task is executed repeatedly based on a (m, k)-pattern $\mathbb{B}_i$. If there are x successful executions of τ_i^d denoted as S inserting into the sequence of jobs, task τ_i can still comply the (m_i, k_i) requirement with the given pattern $\mathbb{B}_i$ for any consecutive k_i jobs, in which $x \geq 0$.

Proof. We can prove this by contradiction. Suppose that the insertion of x successful executions S violate the (m_i, k_i) requirement from time t to $t + k_i \cdot T_i$. By definition of the (m_i, k_i) requirement, the total amount of successful executions and reliable jobs must be *less than* m_i within time interval $[t, t + k_i \cdot T_i]$. The interval must start with an original job 0/1 or a successful execution S including k_i consecutive executions.

For k_i consecutive executions, suppose that there are x successful executions. x successful executions S are inserted into the original sequence of jobs, and x jobs thus are pushed out from the time interval. For example, the original jobs of τ_i can be shown as Figure 4.2a, in which x is 2 and $(m_i, k_i) = (3, 6)$. By the assumption of not satisfying the (m_i, k_i) requirement, the amount of reliable jobs "1"s must be less than $m_i - x$ within time interval $[t, t + k_i \cdot T_i]$. However, there are only at most x of reliable executions denoted as "1" being pushed out from the time interval by inserting x successful executions S as shown in Figure 4.2(b). It means that, the total amount of successful executions and reliable jobs is *at least* m_i within the time interval $[t, t + k_i \cdot T_i]$. Thus, the contradiction is reached. □

From Theorem 1, we notice that, the number of consecutive unreliable executions before a reliable job, i.e., jobs marked as "0" before a job marked as "1", represents the number of tolerant faults before the moment that reliable executions must be executed for enforcing (m, k) requirement. On the one hand, an erroneous execution can only affect $k_i - 1$ number of the following executions. On the other hand, unreliable jobs may likely provide successful executions like reliable executions. Hence, they should be executed as many as possible but under a counter-based control. To realize this idea, we use a set of sporadic replenishment counters to monitor the current status of fault tolerance and aid the runtime adaptation.

To exploit the most amount of unreliable jobs in the (m_i, k_i) requirement, the given (m, k) pattern $\mathbb{B}_i$ is always rearranged so that the binary string always starts from 0 and ends with 1, i.e., the first job is unreliable and the last job is reliable. After rearranging, the number of partitions as p_i is counted, such that one partition is composed of a group of consecutive unreliable jobs and a group of consecutive reliable jobs. For example, given a pattern $\mathbb{B}_i = \{0, 1, 1, 0, 0, 1\}$, p_i is set to 2, since there are two partitions, i.e., $\{0, 1, 1\}$ and $\{0, 0, 1\}$. To describe the partitions for implementation, two sets of counters are required, i.e., counter $o_{i,j} \in \mathbb{O}_i$ and counter $a_{i,j} \in \mathbb{A}_i$, where $j \in \{1, \ldots, p_i\}$ and p_i is the number of partitions in task τ_i. In each partition, counter $o_{i,j}$ is prepared to describe the number of unreliable jobs, whereas counter $a_{i,j}$ records the number of reliable jobs. For the above pattern $\mathbb{B}_i$, the set of counters $\mathbb{A}_i$ will be set as $\{2, 1\}$, and $\mathbb{O}_i$ will be set as $\{1, 2\}$.

4.3.2 Compensation and Replenishment

For each task τ_i, a mode indicator Π_i is used to indicate the behaviors of dynamic compensation under different statuses, i.e, $\Pi_i \in \{\texttt{tolerant, safe}\}$. If task τ_i cannot tolerate any error in the following jobs, the mode indicator is set to `safe` and the compensation is activated for complying the robustness requirements accordingly. If

Algorithm 1 Dynamic compensation of task τ_i with (m_i, k_i)

```
1:  procedure dyn_Compensation(mode Π_i, index j)
2:  if Π_i is tolerant mode then
3:      result = execute(τ_i^d);
4:      if Fault is detected in result then
5:          o_{i,j} = o_{i,j} − 1;
6:          Enqueue_Error(o_{i,j});
7:          if o_{i,j} is equal to 0 then
8:              Set Π_i to safe mode;
9:              Set ℓ_i to a_{i,j};
10: else
11:     either Detection_Recovery() or Reliable_Execution();
12:     ℓ_i = ℓ_i − 1;
13:     if ℓ_i is equal to 0 then
14:         Set Π_i to tolerant mode;
15:         j = (j + 1) mod k_i;
16: Update_Age(𝕆_i);
17: end procedure
```

task τ_i can still tolerate an error, the mode indicator is set to `tolerant` and the task executes the detection version τ_i^d in the next job. The pseudo-code is presented in Algorithm 1, and detailed as follows:

- Whenever an erroneous result is observed, the current counter $o_{i,j}$ is decreased by one unit (Lines 4-5). After k jobs, one unit needs to be increased back to the same counter $o_{i,j}$ (Lines 6 and 16).
- When the current tolerance counter $o_{i,j}$ is equal to 0, task τ_i is required to be executed in the *safe mode*. ℓ is set to $a_{i,j}$ (Lines 7-9).
- In `safe` mode, ℓ will be decreased iteratively. When ℓ is reduced to 0, the task turns back to `tolerant` mode and update the index of partition j (Lines 13-15).

Particularly, there are two different strategies (Line 11):

- **Detection and Recovery (DR):** The task will first execute τ_i^d. If there is a fault detected in the result, the system has to re-execute the job with the reliable version immediately in the same period.
- **Reliable Execution (RE):** In *safe mode*, the task will execute the following jobs with the amount of $a_{i,j}$ of reliable versions τ_i^r obstinately.

Due to the flexibility of counters $\mathbb{O}_i$ and $\mathbb{A}_i$, Algorithm 1 is applicable for any given pattern. To notate briefly, the DC approach (Algorithm 1) in combined with two strategies is denoted as DRE and DDR for the rest of this chapter.

In fact, we can notice that, the resulting jobs sequence will perform the same as Static Pattern-Based Reliable Execution in the worst case as the following:

Lemma 2. Given a (m, k)-pattern $\mathbb{B}_i$, in the worst case that all the unreliable jobs are erroneous executions, the DC approach (Algorithm 1) will follow the static pattern $\mathbb{B}_i$ to execute detection and reliable versions accordingly.

Proof. This is based on the proof of Theorem 1, by taking the fact that there is no successful unreliable execution inserting to the static pattern. If there is no insertion in the binary string of static pattern $\mathbb{B}_i$, $\mathbb{B}_i$ is exactly the same as it is. Therefore, the DC approach (Algorithm 1) behaves the same execution sequence of jobs as Static Pattern-Based Reliable Execution. □

4.3.3 Feasibility Test

Based on Lemma 2, thus, the schedulability test in Section 4.2.2 can be directly applied to test the feasibility for the worst case, where all unreliable jobs are applying an error detection technique. For each task τ_i, the following theorem shows that (m_i, k_i) requirement can be satisfied by applying the DC approach (Algorithm 1):

Theorem 2. By applying the DC approach (Algorithm 1) with a given pattern $\mathbb{B}_i$, the control task τ_i always comply its (m_i, k_i) requirement in any consecutive k_i jobs even in the worst case.

Proof. This property can be proved directly. Suppose that given a interval of k_i consecutive executions of task τ_i, there must be two cases, either some of unreliable jobs are correct or all unreliable jobs are never correct.

For the first case, if the output of unreliable jobs are correct, by applying the DC approach (Algorithm 1), the system will keep execute the detection versions without changing the dynamic counters. From Theorem 1, the correct execution of unreliable jobs only postpone the adoption of static patterns $\mathbb{B}_i$, so that the amount of correct jobs is at least m_i and (m_i, k_i) requirement is still enforced in any consecutive k jobs jobs. For the second case that all the unreliable jobs are never correct, Lemma 2 shows that the DC approach performs as same as SRE, which enforces (m_i, k_i) requirement by a given pattern $\mathbb{B}_i$. Therefore, (m_i, k_i) requirement for task τ_i will be satisfied by applying the DC approach even in the worst case that all executions of unreliable jobs are erroneous. □

4.4 Empirically Obtaining (m, k) Requirements

In this section, we present one empirical approach to obtain (m, k) robustness requirements. First of all, we need to find the potential candidates (m, k) requirements that can *prevent the targeted application from a mission failure*. The potential candidates are defined as follows:

Definition 4 (Finding (m, k) candidates)**.** The potential robustness requirement candidates (m, k) can be found by finding the minimum number m' of the correct jobs, given sliding window size j, among all the possible (i, j) *observed*, i.e., $(m, k) = (m', k)$.

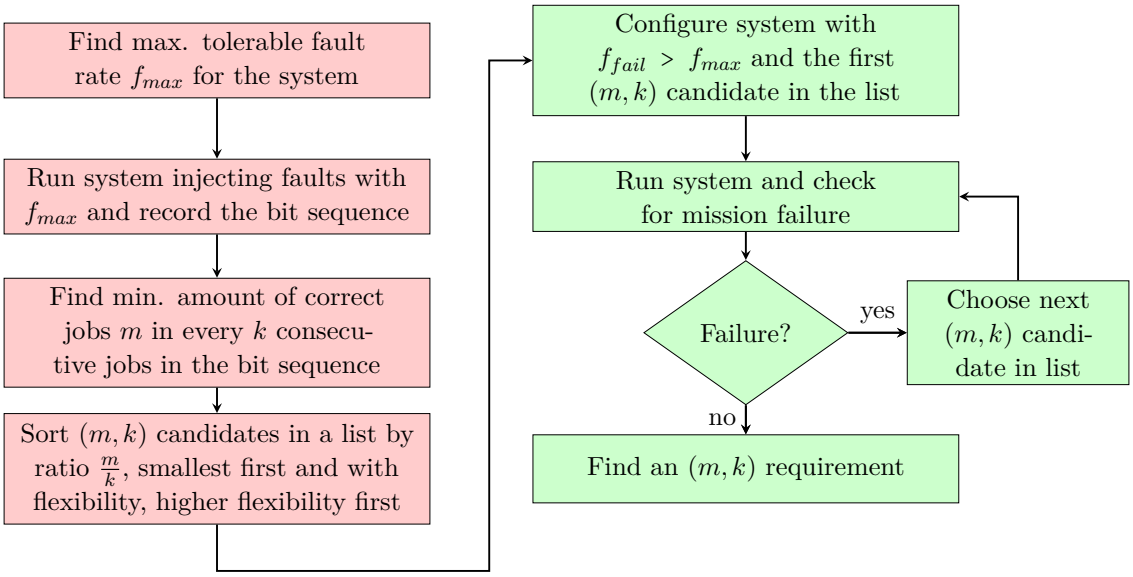


Figure 4.3: The empirical process of finding and verifying (m, k) robustness requirements. Red blocks are for finding, green instructions for verifying (m, k) requirements.

Among the candidates, the best one is selected as follows:

Definition 5 (Determining the best (m, k) candidate)**.** Out of all candidates, the best (m, k) candidate has the lowest m compared to its k, while guaranteeing the prevention of mission failure. If the ratio $\frac{m}{k}$ of (m, k) candidates is equal, the candidate with the larger difference $k - m$ is chosen.

Please note that if the ratio of k to m of two robustness requirements is equal, then (m, k) with the higher difference between k and m is easier to be satisfied. For example, $(1, 2)$ needs a much strict pattern such as $\{0, 1\}$, whereas $(2, 4)$ can also allow an execution pattern like $\{0, 0, 1, 1\}$, which has a higher flexibility to apply the proposed compensation approaches. Therefore, we pick $(2, 4)$ rather than $(1, 2)$ in this case. In Figure 4.3, we present how to empirically find and verify (m, k) candidates with the instructions step-by-step. In the red blocks in Figure 4.3, we specify the steps for finding (m, k) candidates, and in the green blocks we specify the steps for verifying (m, k) candidates.

4.4.1 Finding (m, k) candidates

At first, we find the maximum fault rate, denoted as f_{max}, for which the system always runs without a mission failure. The fault rate should be high enough so that any increase of it would cause a mission failure. Afterwards, we run several experiments and inject faults into jobs of τ_i based on f_{max} without any fault detection and correction routines, and record the information about whether a fault occurred

in the jobs of τ_i in a bit sequence and do this for all jobs of τ_i that are executed by the system in each experimental run. With sufficient amount of bits, we quantify the correctness to prevent mission failure by finding the minimal amount of correct jobs m in every k consecutive jobs with a size k sliding window through the bit sequence to derive certain (m, k) requirements and find the tightest one.

4.4.2 Verifying (m, k) candidates

Up to here, we should have several distinguishable (m, k) candidates. To find out the best one over these candidates, we configure the system with a fault rate $f_{fail} > f_{max}$, which definitely causes the system to fail when no protection in executions. Afterwards, we sequentially start from the first, which is the potentially best, (m, k) requirement in the list to verify if the selected candidates are able to prevent mission failures. We adopt (m, k) R-pattern [QH00; NQ06] to comply the (m, k) requirement by executing reliable versions accordingly. If the system run over the application without a mission failure, we then say the tested (m, k) empirically works. If not, we then try the next best requirement in the list, till we can reach a stable (m, k) requirement.

4.5 Experimental Evaluation

In this section, we use experiments to demonstrate the effectiveness of the proposed approaches. Given an user preferred (m, k) pattern $\mathbb{B}$, the proposed approaches and some baseline approaches as shown in Figure 4.4, are listed as follows:

- Fully Robust (FR): The system only runs the reliable versions. This is the most robust way against potential errors.
- SRE-$\mathbb{B}$: The system directly executes a reliable version if the current job of $\mathbb{B}$ is marked as reliable (see Section 4.2).
- SDR-$\mathbb{B}$: The system gives an additional chance to execute a detection version τ_i^d when the current job of $\mathbb{B}$ is marked as reliable. If any fault is detected, a reliable version is executed immediately in the same period (see Section 4.2).
- DRE-$\mathbb{B}$: By applying the DC approach (Algorithm 1), the system starts to execute reliable versions if the current fault tolerance counter is depleted (see Section 4.3).
- DDR-$\mathbb{B}$: By applying the DC approach (Algorithm 1), the system executes a detection version τ_i^d again when the tolerance counter is depleted. If the result is not correct, a reliable version is executed immediately in the same period (see Section 4.3).

For simplicity of presentation, we let all tasks have the same pattern $\mathbb{B}$. In general, the proposed approaches in this chapter can work well with the other techniques in the literature which require the bounded occurrence of delayed/dropped samples [Ram99; HSJ08; BS15; KGC+12]. These existing solutions can be considered as one of the above baseline approaches, i.e., FR or SRE. Specifically, applying static

patterns to guarantee the presence of mandatory jobs in [Ram99] can be considered as SRE. Running in an open loop for each invalid sample followed by a certain number of reliable jobs in [KGC+12] is also similar as SRE. In [HSJ08; BS15], while the sample does not appear in time, the previous control value is held for the next loop, in which all the jobs are fully reliable as FR to prevent from further soft errors.

The evaluation is performed in two separate experiments: a case study with a practical robotic application and a numerical simulation for synthesized task sets. For the case study, we extend a self-balancing robotic application, i.e., NXTway-gs [YYa10], with a fault injection mechanism and apply the proposed compensation approaches. Here all the (m,k) robustness requirements are obtained by several empirical experiments in advance. We evaluate the proposed approaches based on the overall utilizations for varying fault-rates and (m,k) requirements for this robotic application. To calculate the overall utilization after finishing a test track, we consider the number of periods, denoted as $|p|$, and the numbers of executed unreliable, detection, and correction versions of a task, denoted as $|u|$, $|d|$, and $|r|$, respectively. The overall utilization is calculated as follows:

$$\mathbb{U}_{\text{SRE}} = \frac{|u|C_i^u + |r|C_i^r}{|p|T_i} \tag{4.4}$$

$$\mathbb{U}_{\text{SDR}} = \frac{|u|C_i^u + (|d| - |r|)C_i^d + |r|(C_i^d + C_i^r)}{|p|T_i} \tag{4.5}$$

$$\mathbb{U}_{\text{DRE}} = \frac{|d|C_i^d + |r|C_i^r}{|p|T_i} \tag{4.6}$$

$$\mathbb{U}_{\text{DDR}} = \frac{(|d| - |r|)C_i^d + |r|(C_i^d + C_i^r)}{|p|T_i} \tag{4.7}$$

In the numerical simulation, we use Lemma 2 to report the success ratio in terms of the schedulability for different proposed approaches with different given (m,k)-patterns. We only apply two well-known static (m,k)-patterns, which are the Deep Red Pattern (R-pattern) [KS95] and the Evenly Distributed Pattern (E-pattern) [Ram99] as shown in Table 2.1. Please note that, in the DC approach, the E-pattern is actually be transferred as the Reverse E-pattern [QH00] in the preprocessing phase. Please refer back to Chapter 2 for more details of (m,k) patterns.

4.5.1 Case Study: Two Wheeled Mobile Robot

We consider a self-balancing application, i.e., a two wheeled mobile robot on LEGO Mindstorms NXT equipped with a modified boot loader to run the nxtOSEK [YYa10]. There are three periodic real-time control tasks in this robotic application: (1) Balance Control, (2) Path Control, (3) Distance Control, which are related to a Gyroscopic Sensor, two Light Sensors, and an Ultrasonic Sensor respectively. Each sensor samples the environment at a given rate and is connected as a slave to an I^2C peripheral

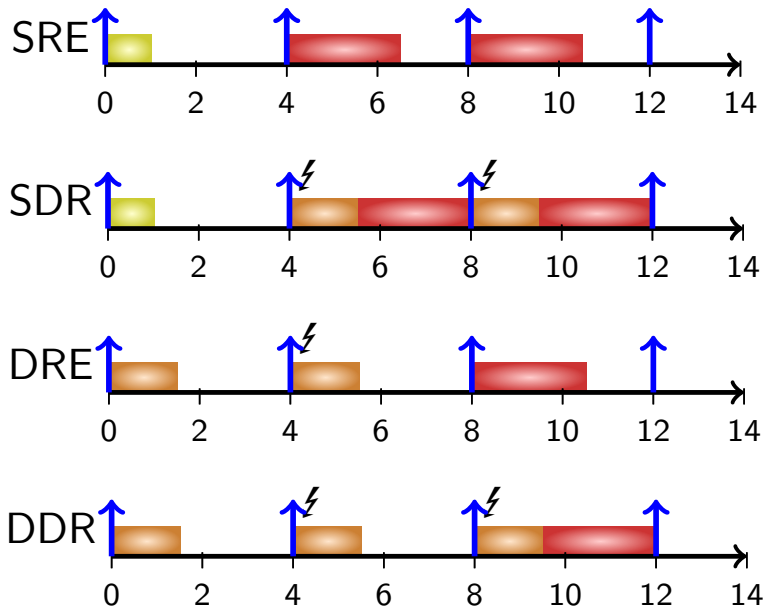


Figure 4.4: Example illustrates different compensation approaches. Given the R-pattern $(2, 3)$, i.e., $\mathbb{B} = (0, 1, 1)$. Suppose faults (ϟ) occur at the second and third jobs. Yellow blocks are unreliable executions. Brown blocks are the unreliable executions of detection versions. Red blocks are reliable executions.

bus. Sampled values are obtained by a master controller that initiates reads from the sensors.

It has been shown that this operation can be suspected to radiation-induced faults and software-based hardening is applicable [NVS+02]. While different techniques are available to harden the complete application, it lies beyond the scope of this chapter to apply and evaluate system-wide fault-tolerance, e.g., control-flow and memory errors. Hence, we solely consider the vulnerable access to the sensors in this case study. While the applicability of sophisticated software fault-tolerance mechanisms has been shown for I^2C implementations [NVS+02], the sensor data is also crucial to the control application and thus serves the targeted purpose.

Fault Injection and Task Versions

To demonstrate the system under the threat of transient faults, we use a simplified error model and define that for each independent sampling, the value may deviate from the true value with a probability p_{fault} per job. By providing proxies to the original calls that effectively access the bus to read the sensor values, we provide an unreliable version that heuristically injects errors to the returned value. An error detecting proxy is then provided with an according overhead [NVS+02], and a reliable

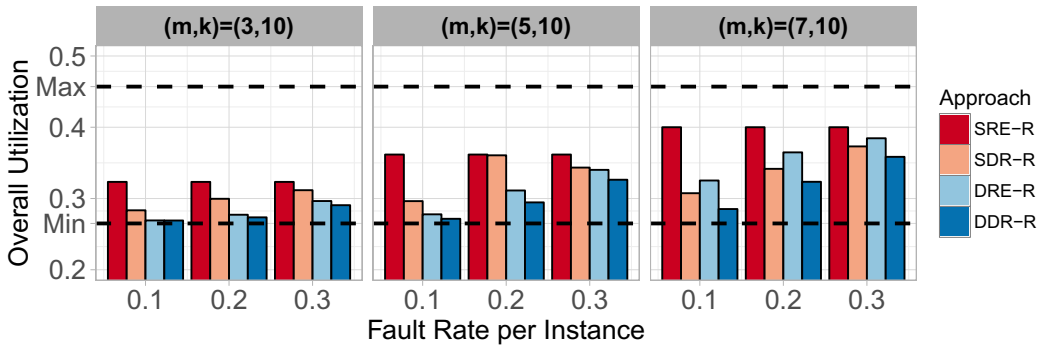


Figure 4.5: Overall Utilization after applying different compensation approaches on Task Path, where lower is better. Two horizontal and dashed bars represent the maximum (0.457) and the minimum utilization (0.265).

Task Name	m	k	Period	Unreliable (μs)	Detection (μs)	Reliable (μs)
Balance	1	1	4000	X	X	435
Path	3	10	1000	99.267	102.598	291.139
Distance	3	5	3000	99.933	103.93	173.217

Table 4.3: Properties of task versions in nxtOSEK-GS [YYa10], which are associated with data sampling of Gyroscopic Sensor, Light Sensor, and Ultrasonic Sensor respectively. The time unit here is microsecond (μs).

proxy that uses majority voting. Within these three versions of the control tasks, all calls to read the sensors are hooked with the proxies, and, for the error detection version, the comparison result is propagated to signal the success of the respective task. The average execution time for each task version is profiled and shown in Table 4.3, along with the respective task periods in microsecond and feasible (m_i, k_i) requirements, i.e., $(1,1), (3,10), (3,5)$ respectively. The robustness requirements are again derived from empirical experiments where the self-balancing robot needs to follow a given monitor while keeping balance, and the fault rate was kept at 30%. Within the empirical experiment, the R-pattern was used for both dynamic and static approaches.

Experimental Results

In this experiment, we evaluate the effectiveness of different compensation approaches based on the overall utilization. In addition, we vary the (m_i, k_i) requirement of the Path Control task to show the corresponding impact on utilization. In order to calculate the overall utilization, we monitor the number of executed jobs of each task version and multiply these by the profiled execution times. In addition, we acquire a maximum utilization resulting from applying FR, which is 0.457, and serves as the baseline as it represents the overall utilization in absence of the proposed method,

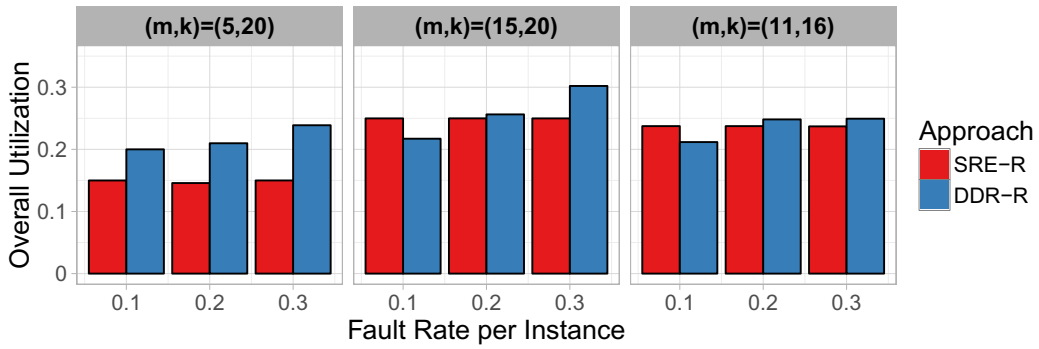


Figure 4.6: The overall utilization of SRE-R can be lower than DDR-R.

and with full protection against errors. The minimum overall utilization is 0.265 and is obtained by solely using the unreliable version for all task jobs, resulting in no protection against soft errors.

Figure 4.5 presents the results for the self-balancing application described above for different (m_i, k_i) requirements and varying fault rates. When the fault rate increases, the overall utilization of dynamic compensations also rises, since the requirement of reliable executions is increased within the application execution. On the other hand, we can notice that SRE-R will always be constant for a fixed (m_i, k_i) requirement, as the overall utilization is deterministic by the amount of job partitions. Using SDR-R results in lower utilizations, as it benefits from the dynamic reaction according to the fault distribution. When the fault rate is as low as 10% and the (m_i, k_i) requirement equals to $(3, 10)$, the probability of activating reliable executions is rare, and, hence, both dynamic compensation approaches, i.e., DRE-R and DDR-R, can closely achieve the minimum overall utilization. On the other hand, when the fault rate is as large as 0.3 and the (m_i, k_i) requirement is tight, i.e., $(7, 10)$, the difference between SRE-R and both dynamic approaches is limited. Interestingly, given a tight (m, k) requirement, SDR-R results in lower utilization than DRE-R. However SDR-R will likely compensate for an error that can safely be neglected while for small m.

Interplay of Different (m, k) and Fault Rates

In Figure 4.5, we can see that DDR-R always dominates the other approaches. However, we also noticed that it is not always the case when the gap between the execution times of C_i^r and C_i^d is closer under some circumstances as shown in Figure 4.6. This motivates us to investigate the impact of the difference between the reliable version τ_i^r and the detection version τ_i^d by adopting a scaling factor S to control the gap between the execution times of detection version and reliable version, where $S = \frac{C_i^r}{C_i^d}$. In addition to S, the considered parameters here are the fault rate and the ratio $\frac{m}{k}$.

Figure 4.7 presents the linear dependences between the scaling factor S and the ratio $\mathbb{U}_{\text{SRE}}/\mathbb{U}_{\text{DDR}}$. Generally, if the S is higher, DDR-R can obtain more benefit. If

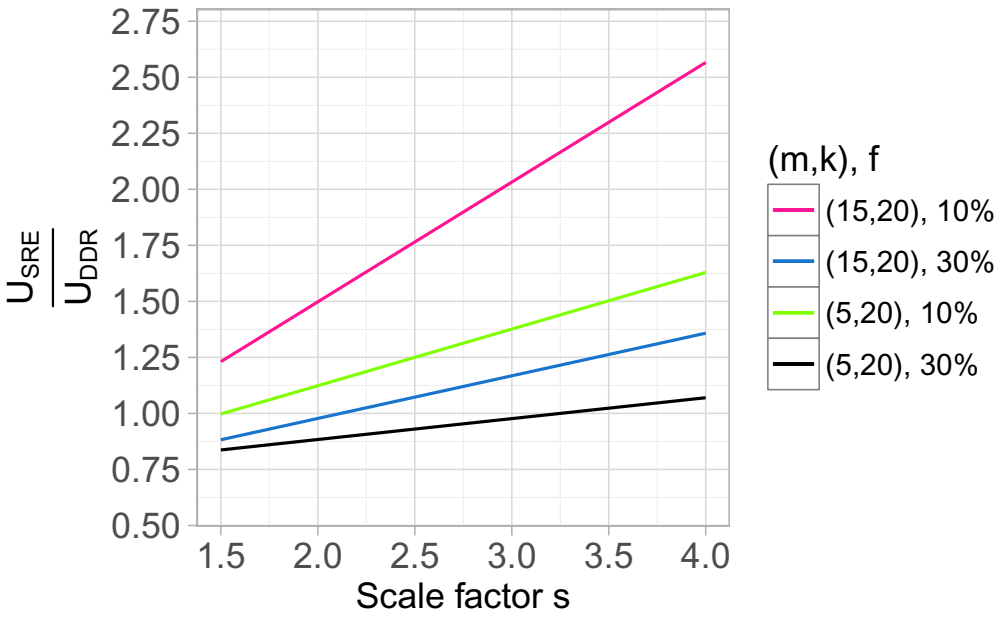


Figure 4.7: The linear dependences between the scaling factor S and the ratio $\frac{\mathbb{U}_{\text{SRE}}}{\mathbb{U}_{\text{DDR}}}$ describe how much DDR-R can outperform SRE-R in the overall utilization.

only the fault rate $\mathbb{F}$ is increased, the gradient decreases (see blue and black plots), since more faults enforce DDR-R to execute the detection and the recovery versions within one period. If only m is increased, the gradient increases because $\mathbb{U}_{\text{SRE}}$ generally increases more than $\mathbb{U}_{\text{DDR}}$. When $S = 1.5$, SRE-R only has 1.2 times overall utilization compared to the DDR for the red plot, and one for the green plot. For blue and black plots, SRE-R outperforms DDR where $\mathbb{F} = 30\%$ in the region that $S < 2$. As a result, it is not always beneficial to solely adopt DDR-R, though it seems to be the dominant technique in many cases. When the fault rate is higher, the margin between DDR-R and SRE-R gets smaller.

4.5.2 Synthesized Task Sets

We have shown that DDR-R generally outperforms the other compensation approaches in reducing the overall utilization. However, recalling that the DDR-R executes a detection version followed by a reliable version in case that an error is detected, the DDR-R requires much execution time in the worst-case, i.e., when having a sufficient amount of consecutive errors. Therefore, DDR-R approach is relatively harder to be scheduled. On the other hand, SRE-R approach does not execute the detection version by default, and it can also perform well as shown in Figure 4.6. These observations motivate us to evaluate different compensation approaches regarding schedulability with the synthetic task sets.

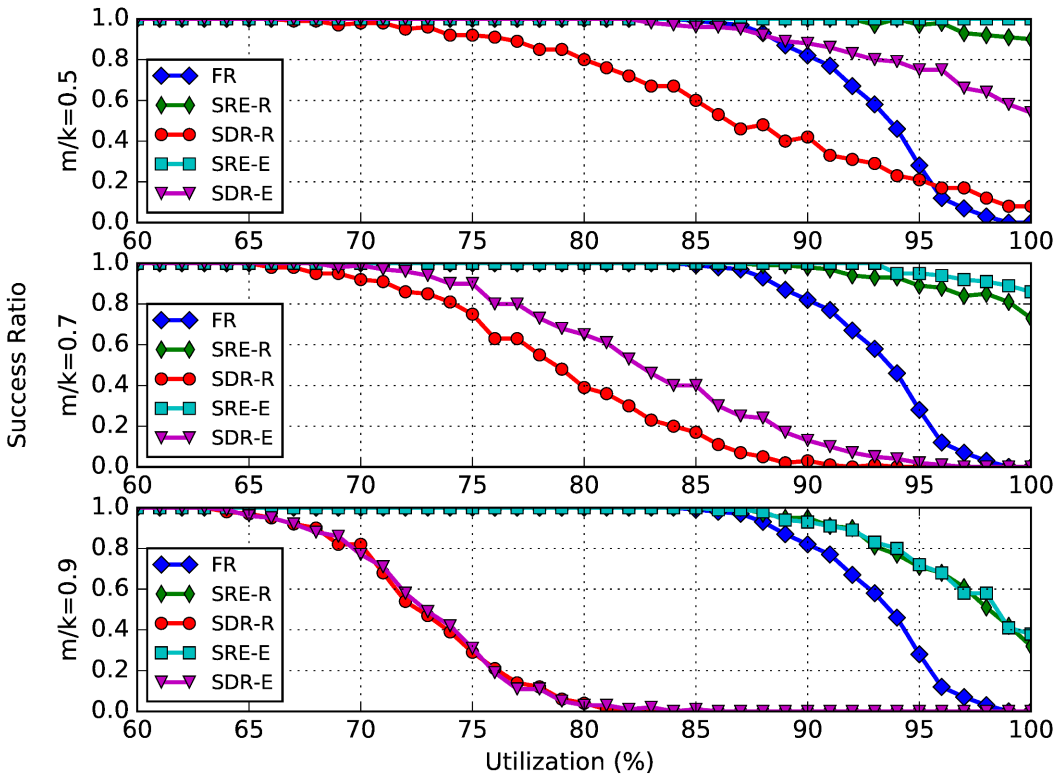


Figure 4.8: Success ratio comparison for different static approaches with two (m, k) patterns, i.e., R-pattern and E-pattern.

In this experiment, we evaluate the success ratio of different compensation approaches with synthesized task sets by using *Lemma* 1. For each task set, if all the tasks in it can pass Eq. (4.1), it is deemed to be schedulable and therefore successful. As stated in Lemma 2, the DC approaches, i.e., DRE and DDR, in the worst case perform the same as the static compensation approaches, i.e., SRE and SDR, respectively. Therefore, we only test SRE and SDR with two different patterns to compare their differences. To empirically evaluate the schedulability of different compensation approaches, we apply the UUniFast [BB05] method to generate a set of utilization values and follow the suggestion from Davis and Burns in [DZB08] to generate the task periods for each task set according to an exponential distribution. The utilization U_i of multi-frame task τ_i is based on its peak frame, and the generated periods lie within the range from 1 to 1000. Since there are only three frame types (versions) in this study, i.e., τ_i^u, τ_i^d, and τ_i^r, we take τ_i^r as the peak frame and set its WCET as $C_i^r = T_i U_i$. For the other task versions, we set $C_i^u = C_i^r/3$ and $C_i^d = C_i^u \cdot 121\%$ to emulate the software-only fault detection (i.e., SWIFT+PROFiT [RCV+05b]) and error recovery by using Simultaneous Redundant Multi-threading (SRT) technique. The cardinality of the task sets is 10, and k_i is uniformly distributed in the range $[3, 10]$. For each k_i, m_i is set accordingly by different ratio of over all $\frac{m}{k}$.

Please refer to Chapter 3.4 for more detailed informations about the experimental setup.

Figure 4.8 illustrates the derived results from the numerical simulation. It should be clear that the success ratios of the schedulability tests for the approaches (except FR) are highly dependent on the ratio $\frac{m}{k}$. If $\frac{m}{k}$ increases, the flexibility of using different protection approaches decreases. No matter which pattern the approaches use, we can observe that the maximum of the execution times Ω_i among the frames of task τ_i are really close when $\frac{m}{k}$ ratio is high. SRE with both patterns, i.e, SRE-R and SRE-E, perform better (with respect to the success ratio of schedulability) than the other SDR approaches in all the simulated cases.

Interestingly, the strategies using the E-pattern, i.e., SRE-E and SDR-E, are always better than the same strategies using the R-pattern, i.e., SRE-R and SDR-R, in terms of the success ratio. The reason comes from the distribution of reliable jobs. As the E-pattern evenly distributes the reliable jobs, in general, there are less consecutive reliable jobs in a strategy using the E-pattern than those in the same strategy using the R-pattern. Therefore, for a low priority task, the interference from the higher priority tasks under the E-pattern is usually less than the case with the R-pattern, so it is relatively easier to be scheduled. When $\frac{m}{k}$ is high, e.g., to 0.7 or even 0.9, both SDR approaches, i.e., SDR-R and SDR-E, are clearly inferior to the others, because SDR approach needs to additionally provide fault detection and re-execution.

Overall, we can see that SRE-E will be the most suitable approach to adopt if the considered system is heavily loaded. The type of given patterns matters, i.e., E-pattern is better, but is not significant as the compensation strategies, i.e., RE or DR, regarding to the schedulability. DR strategy can save more utilization in run-time if the fault rate is relatively low, but it is more difficult to be scheduled in the worst case. For the dynamic compensation approaches, the R-pattern is more suitable than the E-pattern, since the consecutive unreliable jobs at the beginning of the R-pattern provides the most flexibility and fault-tolerance.

4.6 Summary

While embedded systems used for control applications are liable to both hard real-time constraints and fulfillment of operational objectives, the inherent safety margins and noise tolerance in control applications can be exploited when applying software-based error-handling approaches against soft errors induced by the uncertain environment. Hence, in this chapter, we propose to adopt (m, k) model to express the control robustness requirement and present two scheduling approaches, i.e., Static Pattern-Based Reliable Execution and Dynamic Compensation, to determine when/how to compensate or even ignore soft errors safely.

The experimental evaluations compared the efficiency and the applicability of proposed approaches with the baseline approaches. The results show that the overall

utilization of using the DC approach in general outperforms the SRE approach, but can also be inferior to the situation that the given robustness requirement is tight. Moreover, the proposed approach with DR strategy can further reduce the overall utilization but makes the given task sets more difficult to be scheduled as shown in the results. These results suggest that the proposed approaches can be used to serve different applications with inherent fault-tolerance depending upon their perspectives, thus avoiding over-provision under robustness and hard real-time constraints.

The proposed approaches in this chapter have been implemented in Real-Time Executive for Multiprocessor Systems (RTEMS) by Mikail Yayla within the project — Summer of Code in Space (SOCIS) 2017 — funded by European Space Agency (ESA), which I mentored. The detailed report can be found in [Mik17]. His contribution in RTEMS allows people to easily port the proposed approaches to several other platforms rather than only on LEGO NXT [YYa10]. This might potentially provide the flexibility for the fault tolerance of space vehicles to mange redundant executions more efficiently in the future.

CHAPTER 5

Probabilistic Analyses for Deadline-Misses

Contents

5.1 Overview

In Chapter 4, we have presented how a system can allow limited erroneous executions (so decrease the result quality) while satisfying given hard real-time timing constraints.

In this chapter, we proposes another treatment for soft errors incurred by the transient faults: the considered system allows rare deadline misses but without any erroneous execution, e.g., *soft* real-time systems. To handle soft errors, many SIHFT techniques have been proposed in the need of additional computation, e.g., re-execution [MD11], redundancy [RM00], check-pointing [ELS+13], etc. While the appearance of transient faults is usually assumed to be *very low*, the additional amount of time incurred by such techniques may still lead to deadline misses. Since the transient faults are usually approximated and interpreted as random and independent events [Bau05a], the WCET of distinct execution modes can also be modeled as independent random variables. Therefore, it is reasonable to model the timing behavior in conjunction with sporadic soft-error handling events based on probabilistic arguments.

The deadline miss probability is one important metric considered in the literature to quantify the timeliness of real-time systems. To derive the probability, several researches propose to analyze the probabilistic response time by using the convolution-based approaches [MC13; BMC16; DGK+02] and obtain the probability the response time succeeding the targeted deadline as the deadline miss probability. To derive the probability, they calculate the joint probability density function of the worst-case pending execution times at a given time instant by convolving the probabilistic demand whenever a job arrives in the interval of interest. Naturally they are computationally expensive and not scalable with respect to the number of jobs in the interval of interest. Lately in [BPK+18], a task-level convolution-based approach greatly reduces the analysis runtime by using multinomial distributions, but it still requires significant runtime when the number of tasks is large.

Alternatively, we propose to use analytical upper bounds to over-approximate the probability of deadline misses. In [KC17], an analytical approach is proposed to over-approximate the probability of (ℓ-consecutive) deadline misses based on the *Chernoff bound* and the Moment Generating Function (MGF). In [BPK+18], we additionally provide two more analytical approaches by applying the Hoeffding's inequality [Hoe63] and the Bernstein's inequality [FR13], which are several orders of magnitude faster, e.g., 10^3x, than the task-level convolution-based approach in [BPK+18], but with several orders of magnitude more errors. If a sufficiently low deadline miss probability can be derived, these analytical approaches whilst providing reasonable quality.

Likewise, the deadline miss rate is also an important performance indicator to evaluate the extent of requirements compliance for real-time systems. Existing probabilistic approaches, i.e., [BPK+18; MC13; BMC16; KC17], all assume that, after a deadline miss the system either discards the job missing its deadline, or reboots itself. Under this assumption, the probability of one deadline miss directly relates to the deadline miss rate, since no backlog incurred from the jobs missing deadlines needs to be considered. However, such restrictive assumptions do not always hold in practice, as aborting jobs or rebooting the system after a deadline miss is not always an option [BKH+16; KBC16]. If this is the case, the additional workload due to a deadline miss may trigger further deadline misses. Hence, the actual deadline miss

rate may be greater than the probability of the first deadline miss as shown in the following subsection.

Theoretically, the expected miss rate for a task can be determined by counting the number of jobs that miss the deadline and the number of total releases in an *infinitely* long sequence of jobs, considering all tasks under the given constraint, the related scheduling algorithm, and the given fault rate. Nevertheless, as Butler and Finelli stated in [BF93], *Life-testing of ultrareliable software is infeasible*, i.e., the amount of time needed to perform the simulations is too large or even impossible. Therefore, a statistical quantification that can efficiently derive the deadline miss rate is desired.

In this chapter, we provide an analytical approach to safely over-approximate the expected deadline miss rate for a specific sporadic real-time task under fixed-priority preemptive scheduling in uniprocessor systems. The proposed approach is compatible with the existing techniques in the literature that calculate the probability of deadline misses either based on the convolution-based approaches or analytically. To the best of our knowledge, this is the first approach providing a safe upper bound on the expected deadline miss rate.

The presentation in this chapter is organized as follows: In Section 5.1.1, two motivational examples are presented. In Section 5.2, we introduce how to apply analytical upper bounds to form the probabilistic schedulability test, which can derive the probability of (ℓ-consecutive) deadline misses. In Section 5.4, an analytical approach is proposed to adopt the probabilistic approaches in the literature to derive a safe upper bound on the expected deadline miss rate. In Section 5.5, the experimental evaluations firstly demonstrate the effectiveness of the Chernoff bound approaches and secondly present the efficiency and the pessimism of the proposed analytical approach while deriving the expected deadline miss rate. Finally Section 5.6 summarizes the chapter. Parts of this chapter were originally published on SIES 2017 [KC17], ECRTS 2018 [BPK+18], RTCSA 2018 [KBC18a], and DATE 2019 [KUB+19].

5.1.1 Motivational Example

In the following, two examples are given to motivate the presented results of this chapters: The first example[1] shows why the traditional job-level convolution-based methods are not scalable due to state space explosion; the second example shows that in fact the deadline miss rate may be significantly different, if aborting jobs or rebooting the system after a deadline miss is not an option.

Job-level Convolution leads to state space explosion!

Suppose that we have two periodic tasks τ_1 and τ_2 that periodically release jobs, starting from time 0. Each task τ_i has two modes of execution times C_i^N and C_i^A with probability $\mathbb{P}_i^N$ and $\mathbb{P}_i^A$, respectively. The period of task τ_1 is 1 and the period

[1] This example is adapted from [BPK+18] to illustrate the time complexity of convolution-based approaches.

of task τ_2 is 100. Task τ_1 is assumed to always has a higher priority than task τ_2 and always meet its deadline. We assume these two tasks are running on a uniprocessor system with a fixed-priority preemptive scheduling. The system is further assumed to reboot if a job of task τ_2 is not finished before the next job of task τ_2 is released.

What we are interested here is how likely a job of task τ_2, arriving at time t_a, can finish its execution before the next period $t_a + 100$. An intuitive procedure is to *convolute* the probability density functions of the execution times over these two tasks and evaluate the probability of the accumulative execution time, namely *workload*, of the jobs released from time t_a to $t_a + \ell - 1$ (inclusive), starting from $\ell = 1, 2, 3, \ldots, 100$. When ℓ is 1, we have 2^2 combinations of the workload formed by the two jobs of τ_1 released at time t_a. When ℓ is 2, we can have up to 2^3 combinations of the workload, and so on so forth. Intuitively, we can notice that it will lead to 2^{101} combinations of the workload when ℓ is 100, which is *exponential* with respect to the number of jobs that may interfere with a job of task τ_2. This intuitive approach, which enumerate all possible combinations, is based on job-level convolution generally used in [BMC16; DGK+02; MC13].

Since there are only two modes of task τ_1, there are only $\ell + 1$ different workload combinations of the ℓ jobs released from time t_a to time $t_a + \ell - 1$. Therefore, there are only $2(\ell + 1)$ different workload combinations of the jobs released from time t_a to $t_a + \ell - 1$. It is possible to evaluate all of them from $\ell = 1, 2, \ldots, 100$, but it should be clear that it is not practical especially when the number of combinations can be easily larger than this example. Essentially, the only thing we are interested is the probability of the deadline miss at time $t_a + 100$. We do not actually care about the individual execution modes of the 100 jobs of task τ_1 released from t_a to $t_a + 99$. Instead, only their overall workload matters with respect to the probability of the deadline miss at time $t_a + 100$, which can be calculated by using a binomial distribution over 100 independent random variables with the same distribution. Eventually, there are only 101 different workload combinations taken into consideration for the jobs of τ_1. Together with the job of task τ_2, there are in fact only 2×101 different workload combinations. This approach is based on task-level convolution proposed by von der Brüggen et al. in [BPK+18], which is the state-of-the-art.

The aforementioned approaches are different ways to realize the same concept to convolute the probability density functions of the jobs' execution times. Although the task-level convolution-based approach proposed by von der Brüggen et al. [BPK+18] significantly dominates the traditional job-level convolution-based approach with respect to the related runtime and the scalability, the proposed analytical upper bounds in this chapter are still several orders of magnitude faster than it. Nevertheless, the error of analytical upper bounds is also not negligible compared to the convolution-based approaches. More evaluation details are provided in Section 5.5.

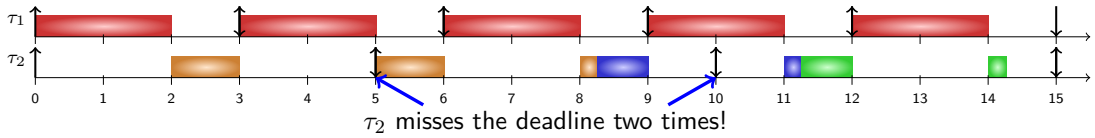


Figure 5.1: An example schedule showing that a deadline miss of the first job of τ_2 (at time 5) directly leads to an additional deadline miss of the second job of τ_2 (at time 10). For τ_2, same color blocks represent the same j-th job. The upward arrows represent the arrival time for each job, whereas the next downward arrows represent the related deadline.

Probability and Miss Rate could be significantly different!

Consider two implicit-deadline periodic tasks τ_1 and τ_2 under fixed-priority preemptive scheduling. Task τ_1 has a WCET of 2 and a period of 3. Task τ_2 has a period of 5 and two distinct versions identified by the different resulting WCETs, which is either 1 or 2.25. Assume that for each job of τ_2 one of the two versions is executed with 50% probability. Since the first job of τ_2 will meet its deadline if it is executed for 1 time unit and miss the deadline if it is executed for 2.25 time units, the probability of deadline misses for the first job is 50%. However, as shown in Figure 5.1, once the first job of τ_2 executes 2.25 time units, which leads to its deadline miss, the second job definitely misses its deadline as well. In this example, the probability that the first job of τ_2 misses its deadline is 50% and therefore at least 2 of the 3 jobs of τ_2 miss their deadlines in this case. Obviously, the occurrence of a pattern that leads to a greater deadline miss rate than 50% does not mean the actual miss rate is greater than 50% as well, since all other possible patterns and the related possibility must be considered. Furthermore, a deadline miss at time 15 will propagate into the next hyper-period[2], which complicates the calculation. Hence, to estimate the deadline miss rate in the displayed scenario, the aforementioned setting is deployed in the event-based simulator detailed in Chapter 3.4.3. The empirical results show that the deadline miss rate on average was 93.04% over 100 simulations where five million jobs of τ_2 were considered in each simulation.

Overall, this example shows that it is necessary to analyze more than just the first deadline miss of a task when considering the deadline miss rate if aborting jobs or rebooting the system after a deadline miss is not possible. *The probability of deadline misses and the deadline miss rate could be significantly different.*

5.1.2 Problem Definition

Suppose that a set of independent and preemptive tasks $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ are given in a uniprocessor system. Each task τ_i releases an infinite number of task instances, called jobs, under a minimum inter-arrival time constraint (or period) T_i, which specifies the minimum time between two consecutive job releases of τ_i. Each task

[2]The hyper-period is the least common multiple of all task periods.

is also associated with a relative deadline D_i. Therefore a job of task τ_i released at time t_a must be completed not later than the absolute deadline $t_a + D_i$ and the next job of task τ_i must be released exactly at (or not earlier than) $t_a + T_i$ for periodic (or sporadic) tasks. We consider: 1) *Implicit-deadline* task sets, i.e., $D_i = T_i \ \forall \tau_i \in \Gamma$, and 2) *Constrained-deadline* task sets, i.e., $D_i \leq T_i \ \forall \tau_i \in \Gamma$.

To model the execution of a job, we assume that the occurrence of soft errors can be modeled by a given probability $\mathbb{P}_i^A$, i.e., the probability is $\mathbb{P}_i^A$ that τ_i is executed abnormally. Depending upon the occurrence of soft-errors and the applied SIHFT techniques, the execution time of a job may differ. The probability of executing a job normally is thus $\mathbb{P}_i^N = 1 - \mathbb{P}_i^A$ for each job of τ_i. We assume that $\mathbb{P}_i^A$ is independent from previous errors and executions, as similar assumptions are used in [MC13; DGK+02; BKH+16].

Two distinct WCETs are assumed for each task τ_i. When no fault occurs during the execution of task τ_i and therefore error recovery is not necessary, the execution is considered to be a *normal* execution with a smaller WCET value, denoted as C_i^N. If a fault is detected in a job of task τ_i, the related job has a longer WCET denoted as C_i^A for potential error recovery, called an *abnormal* execution, i.e., $C_i^A \geq C_i^N \ \forall \tau_i$. The fault detection is assumed to perform perfectly and be done at predefined checkpoints or the end of a job execution, in which the incurred additional computation time is integrated into C_i^N.

With all these assumptions, for a specific task τ_k, the upper bound probability of deadline misses is defined as follows:

Definition 6 (Probability of deadline misses)**.** The Probability of Deadline Misses (DMP) of task τ_k, denoted by Φ_k, is an upper bound on the probability that a job of task τ_k is not finished before its (relative) deadline D_k. To be more precise, the DMP of τ_k is:

$$DMP_k = \max_j \left\{ \mathbb{P}(R_{k,j} > D_k) \right\}, \ j = 1, 2, \ldots \tag{5.1}$$

where $R_{k,j}$ is the response time of the j-th job of τ_k.

The first problem in this chapter focuses on how to derive a probabilistic guarantee for a specific task τ_k that calculates the upper bound Φ_k on the probability of deadline misses based on probabilistic WCETs. In addition to the probability of deadline misses, we also provide the analytical upper bound $\Phi_{k,\ell}$ on the probability of ℓ-consecutive deadline misses. The solution to this problem is presented in Section 5.2.

Secondly, for a given schedule of a sequence of jobs of τ_k, the deadline miss rate is formally defined as follows:

Definition 7 (Miss Rate)**.** The miss rate of a task $\tau_k \in \Gamma$ for a given schedule S is the number of jobs missing their relative deadline D_k in S divided by the number of released jobs of task τ_k in S.

The expected miss rate of τ_k is defined as:

Definition 8 (Expected Miss Rate)**.** The expected miss rate of a task $\tau_k \in \Gamma$, denoted by $\mathbb{E}_k$, is the probability that a job of τ_k misses its deadline.

Theoretically, the expected miss rate for a task τ_k can be determined by counting the number of jobs of τ_k that miss the deadline and the number of total releases in an infinitely long sequence of jobs, considering all tasks in Γ under the constraint given by Γ, the related scheduling algorithm, and the given fault rate. However, how to count the number of jobs missing their deadlines in an infinitely long sequence is obviously not possible in practice. Therefore, the objective here is to calculate a safe upper bound on the expected miss rate of a task τ_k, denoted as $\hat{\mathbb{E}}_k$. By definition, $\hat{\mathbb{E}}_k \geq \mathbb{E}_k$.

The second problem in this chapter focuses on how to over-approximate the expected miss rate $\hat{\mathbb{E}}_k$ of a specific task τ_k, which is done under the assumption that the system is never restarted when a deadline miss happens. In addition, all jobs are never aborted in this assumed system, i.e., if a job misses its deadline, the remaining part of the job still has to be executed before the next job of the task can start executing. As discussed in [BKH+16; KBC16], there are many reasons to apply this assumption in real contexts as long as the system safety is not jeopardized. The solution to this problem is presented in Section 5.4.

Although each task is assumed to have two distinct WCETs with their corresponding probabilities, this assumption is only used in the examples in Section 5.2.1, Section 5.4.1, and the evaluation in Section 5.5. The proposed approaches in this chapter are applicable for any general probabilistic distributions as long as the probabilistic WCETs are all independent of each other.

5.2 Analytical Upper Bound

As shown in [MC13], the DMP of a job is maximized when τ_k is released at its critical instant [LL73], i.e., τ_k is released together with a job of all higher priority tasks and all following jobs of those higher priority tasks are released as early as possible. This implies that the well-known TDA [LSD89] can be applied to determine the worst case response time of task τ_k, which is also an exact schedulability test with pseudo-polynomial runtime under the assumption that the schedulability of tasks in $hp(\tau_k)$ is already ensured by finding a time point t where the total workload released by tasks in $hep(\tau_k)$ is smaller than t. That is, if and only if

$$\exists\ t \text{ with } 0 < t \leq D_k, \qquad \text{such that} \qquad S_t = C_k + \sum_{\tau_i \in hp(\tau_k)} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq\ t \tag{5.2}$$

where S_t is the sum of the WCETs of these jobs in $hep(\tau_k)$ from time 0 to time t. Hence, if $S_t \leq t$ holds, task τ_k is schedulable under the fixed-priority scheduling algorithm, where C_k and C_i in the considered model of this dissertation can be C_k^A and C_i^A, respectively.

The probability that τ_k cannot finish within this interval is denoted accordingly with $\mathbb{P}(S_t > t)$. Since the TDA only needs to hold for one time point t with $0 < t \leq D_k$ to ensure that τ_k is schedulable, the probability that the test fails is upper bounded by the minimum probability among all time points at which the test could fail. Therefore, the probability of deadline misses Φ_k can be upper bounded by

$$\Phi_k = \min_{0<t\leq D_k} \mathbb{P}(S_t \geq t) \tag{5.3}$$

Instead of using convolution-based approaches proposed in the literature [BPK+18; MC13; BMC16; DGK+02] with significant computation times, we notice that concentration inequalities in the literature can be applied to obtain upper bounds on $\mathbb{P}(S_t \geq t)$ and then derive analytical bounds on the DMP. In the rest of this section, we summary three analytical approaches proposed in [KC17] and [BPK+18], and explain how such inequalities in the literature can be used.

To apply these inequalities as the backbones, the proposed approaches only look for $\mathbb{P}(S_t \geq t)$ instead of $\mathbb{P}(S_t > t)$, which is still an upper bound of $\mathbb{P}(S_t > t)$ by definition $\mathbb{P}(S_t \geq t) > \mathbb{P}(S_t > t)$.

5.2.1 Chernoff-Bound Approaches

To apply the Chernoff bound as the backbone, the MGF is used to specify the probability distribution of a random variable [MU05]. The MGF is defined as follows:

Definition 9 (Moment-Generating Function)**.** The moment-generating function of a random variable X_i is defined by the expected value of $e^{X_i \cdot s}$ for any non-negative real number s i.e., $\text{MGF}(X_i) = E[\exp(X_i \cdot s)]$.

For the specific case of the WCET distribution of a task τ_i, the MGF with respect to a given real number s is:

$$\text{MGF}_i(s) = \sum_{j=1}^{h_i} \exp(C_i^j \cdot s) \cdot \mathbb{P}_i^j \tag{5.4}$$

where h_i is the number of (but finite) possible values of execution time C_i^j, exp is the exponential function, i.e., $\exp(x) = e^x$, and $s > 0$ is a given real number[3]. For the considered task model in this dissertation, i.e., each task τ_i has two distinct WCET values C_i^A, C_i^N, Eq. (5.4) can be simplified as

$$\text{MGF}_i(s) = \exp(C_i^A \cdot s) \cdot \mathbb{P}_i^A + \exp(C_i^N \cdot s) \cdot (1 - \mathbb{P}_i^A) \tag{5.5}$$

[3]Usually t is often used instead of s when describing a MGF in the literature regarding probability theory and statistics. Since typically t denotes the time in real-time scheduling, s is used here for avoiding confusion.

which is specifically used in Section 5.5 to demonstrate the applicability of analytical approaches on fault-tolerance.

Since the execution times of the jobs are assumed to be all independent, the probability distribution of the sum of the execution time of these jobs can be defined as the multiplication of their MGFs. By independence, the distribution of the sum of the WCET of these jobs from time 0 to t is

$$\text{MGF}_{hep(\tau_k)}(s) = \prod_{\tau_i \in hep(\tau_k)} (\text{MGF}_i(s))^{\rho_{i,t}} \tag{5.6}$$

, where $\rho_{i,t}$ is the number of jobs released to interfere task τ_k up to time t.

By using Eq. (5.6) and the Chernoff bound, $\mathbb{P}(S_t \geq t)$ in Eq. (5.3) can be over-approximated as follows:

Lemma 3. Suppose that S_t is the sum of the execution times of the $\rho_{k,t} + \sum_{\tau_i \in hp(\tau_k)} \rho_{i,t}$ jobs in $hep(\tau_k)$ from time 0 to t. In this case, the probability of S_t is greater than or equal to t is

$$\mathbb{P}(S_t \geq t) \leq \min_{s>0} \left(\frac{\text{MGF}_{hep(\tau_k)}(s)}{\exp(s \cdot t)} \right) \tag{5.7}$$

Proof. From Lemma 2.9 in [Buc04] and Pages 63-65 in [MU05], for a random variable X defined by a moment generating function $\text{MGF}(s)$ for $s > 0$, the definition of the Chernoff bound is:

$$\mathbb{P}(X \geq t) \leq \text{MGF}(s)/\exp(s \cdot t), \forall s > 0 \tag{5.8}$$

Therefore, a safe upper bound on the probability that $\mathbb{P}(S_t \geq t)$ under the MGF defined in Eq. (5.6) is equivalent to Eq. (5.7). □

With Eq. (5.7), the upper bound on the probability that the total released workload from $hep(\tau_k)$ is not able to finish at time point t can be obtained. The Chernoff bound is in general pessimistic and there is no guarantee for the quality of the approximation (even if the optimal value for s is known). Nevertheless, an upper bound on $\mathbb{P}(S_t \geq t)$ can still be obtained by taking the minimum over any number of s values[4].

Based on Eq. (5.7), the following theorem show that how we can safely extend the TDA as Eq. (5.2) to a probabilistic version to calculate a safe upper bound on the probability of deadline misses:

Theorem 3. Given a set of constrained-deadline (or implicit-deadline) sporadic tasks Γ. 1) If the condition in Eq. (5.2) holds, then the probability of deadline misses of task τ_k is 0. 2) Otherwise, the probability of deadline misses (at least one) Φ_k can be upper bounded by Eq. (5.3), where $\mathbb{P}(S_t \geq t)$ is derived by taking the right-hand side from Eq. (5.7) under $\rho_{i,t} = \left\lceil \frac{t}{T_i} \right\rceil$ for each task τ_i in $hp(\tau_k)$ and $\rho_{k,t} = 1$.

[4]How to find out a specific s to derive the optimal result for the Chernoff bound is discussed in Section 5.3.

Proof. The first assertion directly holds based on the TDA [LSD89]. Since the given task set is schedulable, the probability of deadline misses must be 0 by definition. We only prove the second assertion. At first, we prove why testing t in the range $(0, D_k]$ for τ_k is sufficient.

Since the jobs of τ_k are only preempted by higher-priority jobs in $hp(\tau_k)$ and preempts any lower-priority jobs, we can safely remove any lower-priority jobs and only take $\tau_1, \ldots, \tau_k$ into consideration. Suppose a job of task τ_k is ready at time t' with completion time t_R, in which $t_R - t' > D_k$. Let t_{-1} be the latest instant before t_R, at which 1) either the processor idles at time t_{-1} or 2) all the jobs of task τ_k released *strictly* before t_{-1} have finished their executions. That is, from t_{-1} to t_R, the processor executes only the jobs of task τ_k and $hp(\tau_k)$ that are released after or at t_{-1}. Such a time point t_{-1} always exists, i.e., the starting time of the system.

Now, we remove all the jobs executed before t_{-1} from the schedule. The new schedule from t_{-1} to t_R is the same as the original schedule, in which only jobs arrived at or after t_{-1} are executed. It is possible that there are multiple jobs of task τ_k executed in the time interval $[t_{-1}, t_R)$. We consider two cases:

- **Case 1**, there is only one job of task τ_k executed in the time interval $[t_{-1}, t_R)$: We can move the release of the job of task τ_k from t' to t_{-1}. The response time of the job of task τ_k is not decreased.
- **Case 2**, there are at least two jobs of task τ_k executed in the time interval $[t_{-1}, t_R)$: By the definition that the schedule is busy for executing either task τ_k or $hp(\tau_k)$ in $[t_{-1}, t_R)$, the response time of the first job of task τ_k executed in this window must be greater than T_k. We can move the release time of this first job of task τ_k to t_{-1} as well. The response time of the first job of task τ_k in the time interval $[t_{-1}, t_R)$ is still greater than T_k.

In short, in both cases above, we can safely consider that task τ_k releases a job at time t_{-1}. In the first case, the deadline misses happen when the accumulated workload (sum of the requested execution time of the jobs released by τ_k and $hp(\tau_k)$) executed from t_{-1} to $t_{-1} + t$ is greater than t for any $0 < t \leq D_k$. In the second case, the deadline misses happen when the accumulated workload from t_{-1} to $t_{-1} + t$ is greater than t for any $0 < t \leq T_k$. By the assumption $D_k \leq T_k$, the probability that the accumulated workload executed from t_{-1} to $t_{-1} + t$ is greater than or equal to t for any $0 < t \leq D_k$ is a safe upper bound of the probability of deadline misses.

For notational brevity, let t_{-1} be 0. There are at most $\left\lceil \frac{t}{T_i} \right\rceil$ jobs of task τ_i and one job of task τ_k released from time 0 to time t for any $0 < t \leq D_k$. When a job of task τ_k misses its deadline, by the above analysis, we can safely take $\left\lceil \frac{t}{T_i} \right\rceil$ jobs of task τ_i and one job of task τ_k and evaluate the sum of their execution times up to time t. Consequently the condition in Eq. (5.3) provides an upper bound on the probability of deadline misses of task τ_k. □

However, there is an infinite number of points in the interval $(0, D_k]$ in Eq. (5.3), which is impossible to to test over in practice. The following lemma shows that it is sufficient to test only a pseudo-polynomial number of time points:

Lemma 4. Let $\mathbf{L}_k$ be a set of time interval lengths, where $\mathbf{L}_k = \{r \cdot T_i | \tau_i \in hp(\tau_k); r = 1, \ldots, \lfloor D_k/T_i \rfloor\} \cup \{D_k\}$. The upper bounded DMP Φ_k derived by using Eq. (5.3) is exactly the same as only testing the discretized points $t \in \mathbf{L}_k$, defined as follows:

$$\Phi_k = \min_{\{t \in \mathbf{L}_k\}} \mathbb{P}(S_t \geq t), \tag{5.9}$$

where $\mathbb{P}(S_t \geq t)$ is derived by taking the right-hand side from Eq. (5.7) under $\rho_{i,t} = \left\lceil \frac{t}{T_i} \right\rceil$ for each task τ_i in $hp(\tau_k)$ and $\rho_{k,t} = 1$.

Proof. Suppose for contradiction that the minimum $\mathbb{P}(S_t \geq t)$ (by using Eq. (5.3)) happens when $t = t'$ and t' lies in an interval (α, β), where α and β are two consecutive discretized points in $\mathbf{L}_k$, i.e., $\nexists (\gamma \in \mathbf{L}_k \text{ and } \gamma \in (\alpha, \beta))$. More specifically, $\left\lceil \frac{t'}{T_i} \right\rceil$ is the same as $\left\lceil \frac{\beta}{T_i} \right\rceil$ for any task $\tau_i \in hep(\tau_k)$. Therefore, for each task τ_i in $hep(\tau_k)$, we know that $\rho_{i,t'}$ is also the same as $\rho_{i,\beta}$. With this, $\text{MGF}_{hep(\tau_k)}(s)$ in Eq. (5.6) is exactly the same when $t = t'$ or $t = \beta$ for any $s > 0$. As a result, for any given $s > 0$, we have $\frac{\text{MGF}_{hep(\tau_k)}(s)}{exp(s \cdot t')} > \frac{\text{MGF}_{hep(\tau_k)}(s)}{exp(s \cdot \beta)}$ since $\beta > t'$, in which the contradiction is reached. □

To more efficiently analyze the DMP, we can select a few testing points in $\mathbf{L}_k$ and the minimum value among the probability in these points is still a safe upper bound on the probability of deadline misses. We here introduce a k-point DMP test which may trade off the quality of delivered results for the time complexity, which is motivated by Chen et al. in [CHL15] and Bini et al. in [BB04]. We define k selected points corresponding to the $k-1$ higher-priority tasks and task τ_k. At each time point t, we verify if the total released workload up to time t from $hep(\tau_k)$ can be finished. With Theorem 3, the proposed k-point DMP test is defined as follows:

Theorem 4. Given a set of constrained-deadline sporadic tasks Γ, the probability of deadline misses of task τ_k is upper bounded by $\hat{\Phi}_k$, defined as follows:

$$\hat{\Phi}_k = \min_{t \in \left\{ \lfloor \frac{D_k}{T_1} \rfloor T_1, \lfloor \frac{D_k}{T_2} \rfloor T_2, \ldots, D_k \right\} \setminus \{0\}} \mathbb{P}(S_t \geq t), \tag{5.10}$$

where $\mathbb{P}(S_t \geq t)$ can be derived from Eq. (5.7) by setting $\rho_{i,t} = \left\lceil \frac{t}{T_i} \right\rceil$ for each task τ_i in $hp(\tau_k)$ and $\rho_{k,t} = 1$.

Proof. Since these k selected points, i.e., $\lfloor \frac{D_k}{T_1} \rfloor T_1, \lfloor \frac{D_k}{T_2} \rfloor T_2, \ldots, \lfloor \frac{D_k}{T_{k-1}} \rfloor T_{k-1}, D_k$, lie in the range of $[0, D_k]$, it is sufficient to only test those (up to) k selected points (by removing 0). Since these k selected points are part of $\mathbf{L}_k = \{r \cdot T_i | \tau_i \in hp(\tau_k); r = 1, \ldots, \lfloor D_k/T_i \rfloor\} \cup \{D_k\}$ in Lemma 4, it is clear that $\hat{\Phi}_k \geq \Phi_k$. □

The following example illustrates how Theorem 4 works for calculating the DMP.

Example 2 (Example of Using Theorem 4)**.** Suppose a task set has three sporadic tasks:

$t \in \mathbf{L}_k$	10	20	30	40	45	50	60	70	75
$\arg_{s>0} \min(\mathbb{P}(S^t \geq t))$	5.4999	5.4799	3.72	0.6214	0.6358	4.9155	0.6483	0.711	0.7216
$\min_{s>0}(\mathbb{P}(S^t \geq t))$	1.0	1.0	1.0	0.1041	0.05551	1.0	0.02921	0.00049	0.00024

Table 5.1: Corresponding probabilities of deadline misses on all discretized points t in $\mathbf{L}_k$ gathered from Lemma 4.

- $\tau_1 : T_1 = D_1 = 10, C_1^N = 4, C_1^A = 6, \mathbb{P}_1^A = 10^{-5}$,
- $\tau_2 : T_2 = D_2 = 45, C_2^N = 10, C_2^A = 15, \mathbb{P}_2^A = 10^{-5}$,
- $\tau_3 : T_3 = D_3 = 75, C_3^N = 10, C_3^A = 30, \mathbb{P}_3^A = 10^{-6}$,

to be scheduled on a uniprocessor with the RM fixed-priority scheduling policy. In this example, we evaluate the probability of deadline misses of task τ_3, i.e., $k = 3$. At first three time points are selected accordingly: $t \in \{45, 70, 75\}$. The upper bound probability $\mathbb{P}(S_t \geq 45)$ is at most 0.05551 when s is around 0.6358: $[(\exp(6s)\cdot 10^{-5} + \exp(4s)\cdot(1-10^{-5}))^4 \times (\exp(15s)\cdot 10^{-5} + \exp(10s)\cdot(1-10^{-5})) \times (\exp(30s)\cdot 10^{-6} + \exp(10s)\cdot(1-10^{-6}))] / \exp(45s)$. For time point 70, the upper bound is 0.000492 when s is around 0.711. For time point 75, the upper bound is 0.00024 when s is around 0.721. Therefore, $\hat{\Phi}_k$ is set to 0.00024.

We also provide the results obtained from Lemma 4 in Table 5.1. In this example, we can observe that the minimum probability among all the time points t in $\mathbf{L}_k$ is 0.00024 while $t = 75$, which is the same as the delivered result from Theorem 4, i.e., $\Phi_k = \hat{\Phi}_k$ in this example. □

As a result, we can see testing k selected points is not necessarily worse than testing over all the time points t in $\mathbf{L}_k$ with respect to the quality of the derived results. Empirically, we did not observe any lose from Lemma 4 to k-point DMP test from the numerical evaluations (presented in Section 5.5).

Hoeffding's inequality and Bernstein's inequality

There are two more concentration inequalities, which are easier to compute than the Chernoff bound, and can be applied to derives the targeted probability that the sum of independent random variables exceeds a given value, e.g. $\mathbb{P}(S_t \geq t)$. The first one is the *Hoeffding's inequality*. For completeness, the original theorem is presented here:

Theorem 5 (Theorem 2 from [Hoe63])**.** Suppose that we are given M independent random variables, i.e., $X_1, X_2, \ldots, X_M$. Let $S = \sum_{i=1}^{M} X_i$, $\bar{X} = S/M$ and $\mu = \mathbb{E}[\bar{X}] = \mathbb{E}[S/M]$. If $a_i \leq X_i \leq b_i$, $i = 1, 2, \ldots, M$, then for $s > 0$,

$$\mathbb{P}(\bar{X} - \mu \geq s) \leq \exp\left(-\frac{2M^2 s^2}{\sum_{i=1}^{M} (b_i - a_i)^2}\right) \tag{5.11}$$

Let $s' = sM$, i.e, $s = s'/M$. *Hoeffding's inequality* can also be stated with respect to S:

$$\mathbb{P}(S - \mathbb{E}[S] \geq s') \leq \exp\left(-\frac{2s'^2}{\sum_{i=1}^{M} (b_i - a_i)^2}\right) \tag{5.12}$$

By adopting Theorem 5, we can derive the probability that the sum of the execution times of the jobs in $hep(\tau_k)$ from time 0 to time t is no less than t:

Theorem 6. Let a_i be $C_{i,1}$ and b_i be C_{i,h_i}. Suppose that S_t is the sum of the execution times of the $\rho_{k,t} + \sum_{\tau_i \in hp(\tau_k)} \rho_{i,t}$ jobs in $hep(\tau_k)$ released from time 0 to time t. Then,

$$\mathbb{P}(S_t \geq t) \leq \begin{cases} \exp\left(-\frac{2(t-\mathbb{E}[S_t])^2}{\sum_{\tau_i \in hep(\tau_k)} (b_i - a_i)^2 \rho_{i,t}}\right) & \text{if } t - \mathbb{E}[S_t] > 0 \\ 1 & \text{otherwise} \end{cases} \tag{5.13}$$

where $\rho_{i,t} = \left\lceil \frac{t}{T_i} \right\rceil$ and $\mathbb{E}[S_t] = \sum_{\tau_i \in hep(\tau_k)} (\sum_{j=1}^{h_i} C_{i,j} \mathbb{P}_i(j)) \cdot \rho_{i,t}$.

Proof. Since the execution time of a job of task τ_i is an independent random variable, there are in total $\rho_{i,t}$ independent random variables with the same distribution function upper bounded by $C_{i,h}$ and lower bounded by $C_{i,1}$ for each $\tau_i \in hep(\tau_k)$. With Eq. (5.12) and $s' = t - \mathbb{E}[S_t]$, we directly get:

$$\mathbb{P}(S_t \geq t) = \mathbb{P}(S_t - \mathbb{E}[S_t] \geq t - \mathbb{E}[S_t]) \leq \exp\left(-\frac{2(t-\mathbb{E}[S_t])^2}{\sum_{\tau_i \in hep(\tau_k)} (b_i - a_i)^2 \rho_{i,t}}\right) \tag{5.14}$$

when $s' > 0$. Otherwise, i.e., when $s' \leq 0$, we use the safe bound $\mathbb{P}(S_t \geq t) \leq 1$. □

The Chernoff bound and the related inequality by Hoeffding and Azuma can be generalized by the *Bernstein's inequality*. The original corollary is also stated here:

Theorem 7 (Corollary 7.31 from [FR13])**.** Suppose that we are given L independent random variables, i.e., $X_1, X_2, \ldots, X_L$, each with zero mean, such that $|X_i| \leq K$ almost surely for $i = 1, 2, \ldots, L$ and some constant $K > 0$. Let $S = \sum_{i=1}^{L} X_i$. Furthermore, assume that $\mathbb{E}[X_i^2] \leq \theta_i^2$ for a constant $\theta_i > 0$. Then for $s > 0$,

$$\mathbb{P}(S \geq s) \leq \exp\left(-\frac{s^2/2}{\sum_{i=1}^{L} \theta_i^2 + Ks/3}\right) \tag{5.15}$$

The proof can be found in [FR13]. Note, however, that the result in [FR13] is stated for the two-sided inequality, i.e., as upper bound on $\mathbb{P}(|S| \geq s)$. Here, the one-sided result, which is a direct consequence of the proof in [FR13] (page 198), is tighter.

Hence, the following upper bound can be derived:

Theorem 8. Suppose that the sum of the execution times of all $L = \rho_{k,t} + \sum_{\tau_i \in hp(\tau_k)} \rho_{i,t}$ jobs is S_t. Let $K = \max_{\tau_i \in hep(\tau_k)} C_{i,h_i} - \mathbb{E}[C_i]$ be the centralized WCET of any job, where $\mathbb{E}[C_i] = \sum_{j=1}^{h_i} \mathbb{P}_i(j) C_{i,j}$ is the expected execution time of a job of task τ_i. Then,

$$\mathbb{P}(S_t \geq t) \leq \begin{cases} \exp\left(-\frac{(t-\mathbb{E}[S_t])^2/2}{\sum_{\tau_i \in hep(\tau_k)} \mathbb{V}[C_i] \rho_{i,t} + K(t-\mathbb{E}[S_t])/3}\right) & \text{if } t - \mathbb{E}[S_t] > 0 \\ 1 & \text{otherwise} \end{cases} \tag{5.16}$$

for any $t > 0$, where $\rho_{i,t} = \left\lceil \frac{t}{T_i} \right\rceil$ and $\mathbb{E}[S_t] = \sum_{\tau_i \in hep(\tau_k)} (\sum_{j=1}^{h_i} C_{i,j} \mathbb{P}_i(j)) \rho_{i,t}$.

Proof. Since for each task $\tau_i \in hep(\tau_k)$ the execution time of a job of task τ_i is an independent random variable, there are in total $\rho_{i,t}$ independent random variables with the same distribution function. Suppose that C_l is a random variable representing the execution time of a job of task τ_i and let $Y_l = C_l - \mathbb{E}[C_i] = C_l - \sum_{j=1}^{h_i} C_{i,j}\mathbb{P}_i(j)$ denote its centralized execution time. Since the expected execution time of a job is fully determined by its corresponding task, we have $\mathbb{E}[C_l] = \mathbb{E}[C_i]$.

Hereinafter, we explain why we adopt $\mathbb{V}[C_i]$ instead of θ_i^2 as known from Theorem 7. Consider Eq. (5.15) with $S = \sum_{l=1}^{M} Y_l$. The exact variance $\mathbb{V}[Y_l] = \mathbb{E}[Y_l^2] - \mathbb{E}[Y_l]^2 = \mathbb{E}[Y_l^2]$ is unknown and hence some loose upper bound θ^2 must be considered in most applications of Bernstein's inequality, like stated in Theorem 7. Here, the probabilities of the different execution modes are given numerically, i.e., $\mathbb{P}_i(j)$ for $C_{i,j}$. Hence, for an arbitrary but fixed task τ_i with h_i different execution modes, this results in

$$\mathbb{V}[Y_l] = \sum_{j=1}^{h_i} \mathbb{P}_i(j) \left(C_{i,j} - \mathbb{E}[C_i]\right)^2 = \sum_{j=1}^{h_i} \mathbb{P}_i(j) \left(C_{i,j}^2 - 2C_{i,j}\mathbb{E}[C_i] + \mathbb{E}[C_i]^2\right)$$
$$= \sum_{j=1}^{h_i} \mathbb{P}_i(j) C_{i,j}^2 - \sum_{j=1}^{h_i} \mathbb{P}_i(j) 2C_{i,j}\mathbb{E}[C_i] + \sum_{j=1}^{h_i} \mathbb{P}_i(j)\mathbb{E}[C_i]^2 = \mathbb{E}[C_i^2] - \mathbb{E}[C_i]^2 = \mathbb{V}[C_i] \tag{5.17}$$

i.e., $\mathbb{V}[Y_l] = \mathbb{V}[C_i]$, which can be computed exactly in time $\mathcal{O}(h_i)$. Instead of imposing an upper bound θ^2, we can invoke the tightest version of Theorem 7 by using the exact variance.

Since $\mathbb{E}[Y_l] = 0$ and $\forall 1 \leq l \leq M : Y_l \leq K$, we can invoke Theorem 7 with $s = t - \mathbb{E}[S_t]$. When $s \leq 0$, we use a safe bound $\mathbb{P}(S_t \geq t) \leq 1$. When $s > 0$, Eq. (5.15) can be rewritten as

$$\mathbb{P}\left(\sum_{l=1}^{M} Y_l \geq t - \mathbb{E}[S_t]\right) \leq \exp\left(-\frac{(t - \mathbb{E}[S_t])^2/2}{\sum_{l=1}^{M} \mathbb{V}[Y_l] + K(t - \mathbb{E}[S_t])/3}\right) \tag{5.18}$$

Finally, observing that $\sum_{l=1}^{M} Y_l = S_t - \mathbb{E}[S_t]$ and $\sum_{l=1}^{M} \mathbb{V}[Y_l] = \sum_{\tau_i \in hep(\tau_k)} \mathbb{V}[C_i]\rho_{i,t}$ (from Eq. (5.17)) completes the proof. □

The implementation of three analytical bounds, i.e., Lemma 3, Theorem 6, and Theorem 8, are all released in combination with the event-based simulator on [Kua18].

5.2.2 Consecutive Deadline Misses

We also study how to handle more general cases for the upper bound on the probability of ℓ-consecutive deadline misses. For the rest of this section, we reform the notation of the DMP from Φ_k to $\Phi_{k,\ell}$ for the probability of ℓ-consecutive deadline misses. While ℓ is 1, we define the probability $\Phi_{k,1}$ as Φ_k delivered by Theorem 3, i.e., $\Phi_{k,1} = \Phi_k$.

The following theorem shows that how to extend Eq. (5.3) and recursively obtain a safe upper bound on the probability of ℓ-consecutive deadline misses:

Theorem 9. Given a set of constrained-deadline sporadic tasks Γ. Suppose that

$$\Phi^{\theta}_{k,w} = \min_{0<t\leq(w-1)T_k+D_k} \mathbb{P}(S^t \geq t) \tag{5.19}$$

where $\mathbb{P}(S^t \geq t)$ can be derived from Eq. (5.7) by setting $\rho_{i,t} = \left\lceil \frac{t}{T_i} \right\rceil$ for each task τ_i in $hep(\tau_k)$. That is, $\Phi^{\theta}_{k,w}$ is the upper bound on the probability of w-consecutive deadline misses when the processor executes at least w (consecutively released) jobs of task τ_k without any idling. For notational brevity, let $\Phi_{k,0}$ be 1. The probability of ℓ-consecutive deadline misses of task τ_k is upper bounded by $\Phi_{k,\ell}$, defined as follows:

$$\Phi_{k,\ell} = \max\left\{\Phi^{\theta}_{k,w} \cdot \Phi_{k,\ell-w} | w \in \{1,2,\dots\ell\}\right\} \tag{5.20}$$

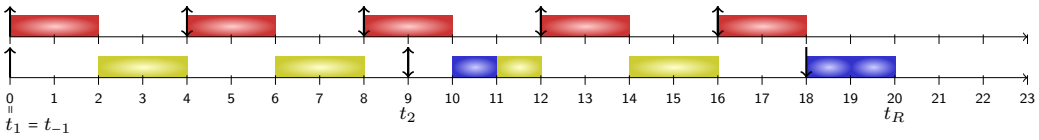
Proof. We prove this theorem by constructing $\Phi_{k,j}$ from $j = 1,2,\dots,\ell$ sequentially. When j is 1, the upper bound $\Phi_{k,1}$ is equal to Φ_k and can be derived by using Theorem 3 or 4. Therefore, suppose that $\Phi_{k,j}$ for $j \in \{1,2,\dots,\ell-1\}$ is already calculated by the previous steps. For a preemptive fixed-priority schedule, removing tasks with priority lower than τ_k does not change the schedule of task τ_k. As a result, we only consider $hep(\tau_k)$ in the proof. To have ℓ-consecutive jobs of task τ_k with deadline misses, there must be at least ℓ consecutively released jobs of task τ_k missing t heir deadlines. Let J_ℓ be a job of task τ_k in which its previous $\ell-1$ jobs, $J_1, J_2, \dots, J_{\ell-1}$, released by task τ_k all miss their deadlines.

Let t_j be the arrival time of job J_j released by task τ_k. Let t_R be the completion time of job J_ℓ. Since task τ_k is a sporadic task with a minimum inter-arrival time T_i, by definition, we have $t_\ell - t_1 \geq (\ell-1)T_k$ and $t_R - t_\ell > D_k$. That is, $t_R - t_1 > (\ell-1)T_k + D_k$. Let t_{-1} be the latest instant before t_R, at which either the processor idles at time t_{-1}, or all the jobs of task τ_k before t_{-1} have finished their executions. Suppose that there are w^* jobs of task τ_k released after or at time t_{-1}.

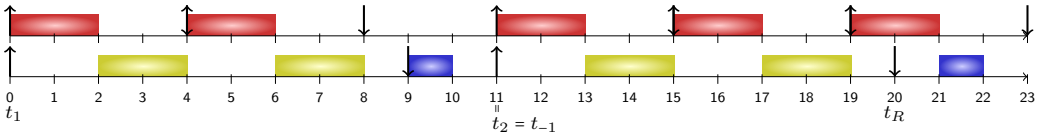
We consider two different cases in this interval $[t_{-1}, t_R)$:

1. **Case 1 - if** $t_1 \geq t_{-1}$: This implies that $w^* \geq \ell$ and the processor is busy from time t_{-1} to time t_R executing the jobs released at or after t_{-1}. Similar to the proof of Theorem 3, we can remove all the jobs executed before t_{-1} and set the release time of the first job of τ_k released in this interval to time t_{-1} in the schedule.[5] Similarly, we can also advance the subsequent jobs of τ_k to release at time $t_{-1}+T_k, t_{-1}+2T_k, \dots$. This adjustment does not decrease the response times of these consecutively released jobs of task τ_k. Therefore, all of these w^* jobs of task τ_k still miss their deadlines. With a similar argument to the one made in the proof of Theorem 3, the processor is busy executing the *periodically* released workload from time t_{-1} to time $t_{-1} + (\ell-1)T_k + D_k \leq t_{-1} + (w^*-1)T_k + D_k$. Hence, the upper bound on the probability of this case is $\Phi^{\theta}_{k,\ell}$.
2. **Case 2 - if** $t_1 < t_{-1}$: This implies that $w^* < \ell$ and the processor is busy from time t_{-1} to time t_R executing the jobs released at or after t_{-1}. Therefore, we

[5] The first job of task τ_k released at or after t_{-1} may not be J_1.



(a) Case 1: $t_1 \geq t_{-1}$



(b) Case 2: $t_1 < t_2 = t_{-1}$

Figure 5.2: Task τ_1 and τ_2 are implicit-deadline sporadic tasks with $C_1^A = 2, T_1 = D_1 = 4, C_2^A = 5, T_2 = D_2 = 9$. The blue marked blocks are overrunning executions.

know that from time t_1 to time t_{-1}, there must be at least $\ell - w^*$ consecutive jobs of task τ_k with deadline misses and probability upper bounded by $\Phi_{k,(\ell-w^*)}$. We now only have to evaluate the probability of the w^* consecutive deadline misses of task τ_k from time t_{-1} to time t_R, which is upper bounded by Φ^{θ}_{k,w^*}, as an identical scenario to Case 1. Therefore, the upper bound on the probability of this case is hence $\Phi^{\theta}_{k,w^*} \cdot \Phi_{k,(\ell-w^*)}$.

If w^* is known, one of these two cases defines the upper bound on the probability of ℓ-consecutive deadline misses of task τ_k. However, even though w^* is unknown, we can iterate all the possible values from 1 to ℓ. Therefore, the upper bound on the probability of ℓ-consecutive deadline misses can be found by Eq. (5.20). □

Figure 5.2 illustrates the two cases in the proof of Theorem 9. Suppose that τ_2 is the targeted task τ_k in Theorem 9. As shown in Figure 5.2a, the execution of the second job released at time 9 is pushed by the overrun of the first job, i.e., the first blue block. Therefore, the analyzed window should cover the time interval $[t_1, t_R)$. In Figure 5.2b, there are only two jobs of τ_2 with an idle instant, but in fact the second job can be followed by the other $w^* - 1$ jobs consecutively without any idle instants. We can find that the analyzed schedule of the jobs finished before t_2 and the jobs released after t_2, can be individual.

With Theorem 9, we can obtain $\Phi_{k,\ell}$ for the upper bound on the probability of ℓ-consecutive deadline misses. To avoid testing all time points in the interval $(0, (w-1)T_k + D_k]$ in Eq. (5.19), we can again apply the same strategy as in Lemma 4 to generate a pseudo-polynomial number of time points $\mathbf{L}_k$ to test. That is, $\mathbf{L}_k = \{r \cdot T_i | \tau_i \in hp(\tau_k); r = 1, \ldots, \lfloor((\ell-1)T_k + D_k)/T_i\rfloor\} \cup \{(w-1)T_k + D_k | w = 1, \ldots, \ell\}$. To efficiently analyze the probability $\Phi_{k,\ell}$ of ℓ-consecutive deadline misses, we use a similar strategy to select $k \cdot \ell$ testing points $\hat{\mathbf{L}}_k$ to derive $\hat{\Phi}_{k,\ell}$, which can be similarly proved as Theorem 4, i.e., $\hat{\Phi}_{k,\ell} \geq \Phi_{k,\ell}$. That is, $\hat{\mathbf{L}}_k = \{(w-1)T_k + D_k | w = 1, \ldots, \ell\} \cup \{r \cdot T_i | \tau_i \in hp(\tau_k); r = \lfloor D_k/T_i\rfloor, \ldots, \lfloor((\ell-1)T_k + D_k)/T_i\rfloor\}$.

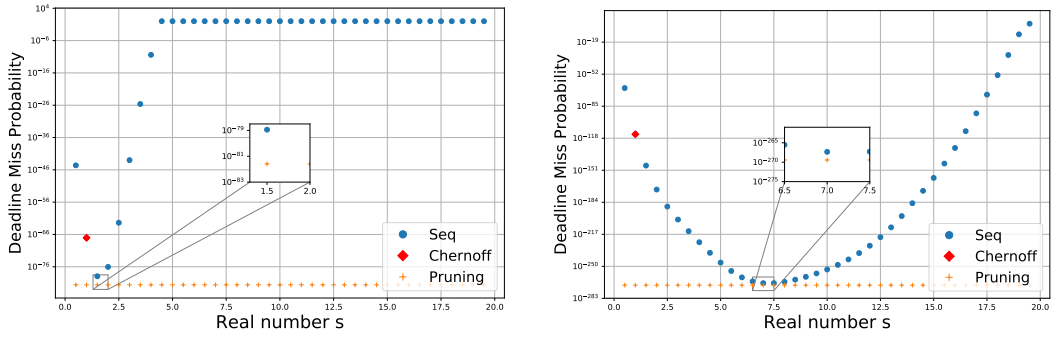
To the best of our knowledge, this approach is the first result regarding to the probability of ℓ-consecutive deadline misses. Furthermore, it is applied as a backbone to derive the deadline miss rate in the next section.

5.3 Finding Optimal s for Chernoff-Bound

Despite the Chernoff bound being an over-approximation without non-trivial analytical guarantees for the quality of approximation, the quality of approximation varies with the choice of s. Hence, in order to optimize the quality of approximation, it is beneficial to find the smallest Chernoff bound efficiently, based on all possible s values. The following example demonstrates the differences of the derived DMP from the Chernoff bound over different s and the task-level convolution-based approach:

Example 3 (Differences between Chernoff-Bound and Convolution-Based Approach)**.** Consider a real-time system with a set of sporadic tasks with 25 tasks, i.e., $\Gamma = \{\tau_1, \tau_2, \cdots, \tau_{25}\}$, on a uniprocessor. We use the UUniFast method [BB05] to synthesize sporadic implicit-deadline task sets with a given normal mode utilization of 60%, setting $\mathbb{P}_i^A = 10^{-4}$ and $C_i^A = 1.83 \cdot C_i^N$ for all tasks. The tasks are scheduled according to RM preemptive scheduling, i.e., tasks with shorter periods have higher priority.

For these task sets, we used the Chernoff bound approach to calculate the DMP considering a fixed value $s = 1$, denoted by **Chernoff**, and the task-level convolution-based approach in [BPK+18] with the proposed runtime optimization that prunes out unnecessary states, denoted by **Pruning**. Furthermore, we iteratively calculated the **Chernoff** bound for different values of s with a step size 0.5, denoted as **Seq**.



(a) Set 0 **(b)** Set 1

Figure 5.3: Derived results comparison: The red diamonds represent the obtained upper bound on the probability of deadline misses by setting $s = 1$ with the Chernoff bound approach. The blue curves are drawn by iterating over different s with step 0.5 by using the Chernoff bound approach. The bottom orange lines are the derived results from the task-level convolution-based approach in [BPK+18].

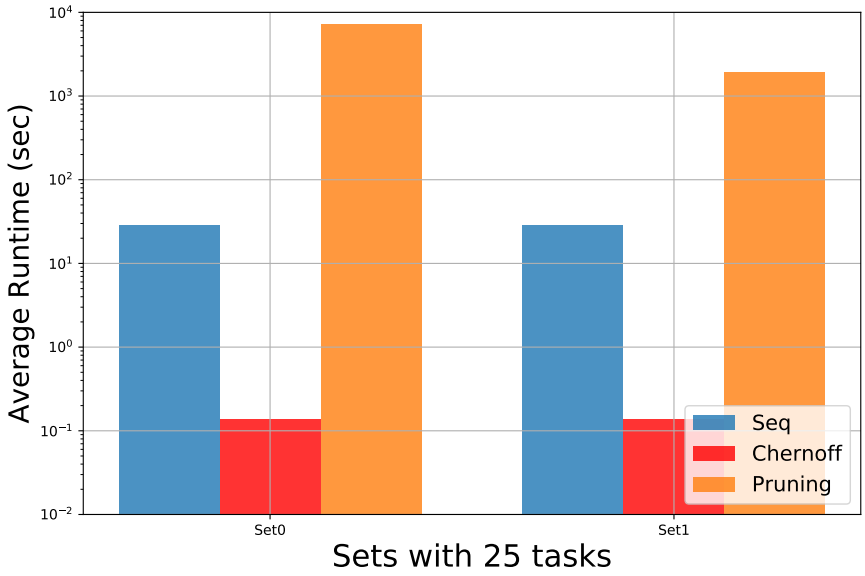


Figure 5.4: Runtime comparison: The average required runtime with different state-of-the-arts for the selected task sets in the motivational example.

Figure 5.3 displays these results and shows that using **Chernoff** with a fixed value can lead to a large gap compared to the DMP for **Pruning**. This also holds true if only a small interval of possible values for s can be considered, i.e., in [KC17] $s \in (0, 1]$ and in [BPK+18] $s \in (0, 3]$ was considered to achieve a good runtime. When **Chernoff** is used iteratively for multiple real number s to find out a suitable s, the gap to **Pruning** may be reasonably small as **Seq** shows. Regarding the required runtime for analyses, shown in Figure 5.4, **Pruning** is orders of magnitude slower than **Chernoff**. However, the required runtime for finding a suitable s by iteratively as in **Seq** highly depends upon the step size and the considered interval. Unfortunately, the results in Figure 5.3 suggest 1) that the actual value of s differs largely based on the considered task set, and 2) that being slightly off from the best value may lead to a large difference from the value obtained by **Pruning**. However, Figure 5.3 also suggests that for a given task set the Chernoff bound is a convex function with respect to s. □

In this section, we show that finding the smallest Chernoff bound based on all possible s values in fact poses a convex optimization problem and is thus efficiently solvable. This allows to increase the precision of the Chernoff bound approach, since a wider range of possible s values can be covered. This is due to the fact that the convex property allows to search the possible interval of s values more efficiently. Hence, the reduced runtime directly leads to a more precise DMP estimation. Furthermore, the task-level convolution-based approach has a runtime complexity that is exponential in the number of considered tasks for each point in time, while the Chernoff bound has a runtime complexity that is linear with respect to both the number of tasks and

the number of values that are considered for s. Therefore, the Chernoff bound is the only method to determine the deadline miss probability for really large task sets, e.g., 1000 tasks, while still resulting in a reasonable approximation quality.

5.3.1 Optimization Problem

As stated in Section 5.2.1, the Chernoff bound holds for any non-negative real-value s, and thus poses an optimization problem to find the smallest upper bound. In this section, we explain how to efficiently compute the smallest upper bound of each task's deadline-miss probability, using the Chernoff bound approach and to evaluate the precision loss compared to the convolution-based approach. That is, the following optimization problem transformed from Eq. (5.9) must be solved

$$\Pr(S_t \geq t) \leq \inf_{s_k > 0} \left\{ \left(\prod_{\tau_i \in hep(\tau_k)} \text{MGF}_i(s_k)^{\rho_{i,t}} \right) \cdot e^{-s_k \cdot t} \right\} \tag{5.21}$$

where

$$\text{MGF}_i(s_k) = \sum_{j=1}^{h_i} e^{C_i^j \cdot s_k} \cdot \mathbb{P}_i^j \tag{5.22}$$

This means, for each task τ_k in the task set, a non-negative real value s_j must be identified that minimizes Eq. (5.21) for some given time t and a given set of all higher-priority tasks $\tau_1, \tau_2, \ldots, \tau_{k-1}$. In the following, we show that this optimization problem shown in Eq. (5.21) is log-convex. It thus exhibits a unique minimum and is efficiently solvable by various numerical algorithms.

Theorem 10 (Boyd [BV04])**.** Let $y \in Y$ then if a function $f(x, y)$ is log-convex in x for each $y \in Y$ and $f(x, y) \geq 0$, the function $g(x) = \int_Y f(x, y)\ dy$ is log-convex.

Lemma 5. The moment-generating function of a task τ_i

$$\text{MGF}_i = \int_{-\infty}^{\infty} e^{u \cdot s} \cdot \mathbb{P}_i(u)\ du, \tag{5.23}$$

for a given probability density function $\mathbb{P}_i(u)$ and any $s \in \mathbb{R}^+$, is log-convex.

Proof. By definition of a probability density function, $P_i(u) \geq 0$ for any $u \in \mathbb{R}$ and thus satisfies the conditions stated in Theorem 10. Since the logarithm of the integrand, i.e., $s \cdot u + \ln(\mathbb{P}_i(u))$, is linear in s for any $u \in \mathbb{R}$, it is convex. Therefore, by the arguments of Theorem 10, MGF_i is log-convex. □

Theorem 11. The moment-generating function of the cumulative execution time of a given number of job-releases is log-convex.

Proof. By the i.i.d assumption, we know that the moment-generating function of the cumulative execution time of a given number of job-releases can be given as the multiplication of the individual moment-generating functions of each job instance, i.e., $\prod_{\tau_i \in hep(\tau_k)} \text{MGF}_i$ for j job releases. By the property of the logarithm function, the

logarithm of the cumulative moment-generating function, i.e., $ln(\prod_{\tau_i \in hep(\tau_k)} \text{MGF}_i)$ is equivalent to $\sum_{\tau_i \in hep(\tau_k)} ln(\text{MGF}_i)$. Since convexity is closed with respect to addition and Lemma 5, we know that the moment-generating function of the cumulative execution time of a given number of job-releases is log-convex. □

In order to minimize Eq. (5.21), we use the common approach to minimize the logarithm of the equation instead. Since the logarithm is strictly monotonically increasing, the minimum will be the same for both equations. In conclusion, for each task τ_k, we solve the following convex optimization problem

$$\inf_{s_k > 0} \left(\sum_{\tau_i \in hep(\tau_k)} \lceil t/T_i \rceil \cdot ln(\text{MGF}_i(s_k)) \right) - s_k \cdot t \tag{5.24}$$

where t is given from a finite set of values $\mathbf{L}_k$ like k-points derived by Theorem 4. The probability that task τ_k misses its deadline is thus upper bounded by

$$\Phi_k = \min_{t \in \mathbf{L}_k} \left\{ \inf_{s_k > 0} \left\{ \left(\sum_{\tau_i \in hep(\tau_k)} \lceil t/T_i \rceil \cdot ln(\text{MGF}_i(s_k)) \right) - s_k \cdot t \right\} \right\}. \tag{5.25}$$

Since this optimization problem is a set of finitely many convex optimization problems, it can be efficiently and unequivocally solved by $|\mathbf{L}_k|$ binary searches.

Numerical Issues

With respect to an implementation of the above optimization problem, the floating-point arithmetic is of special concern due to overflow and underflow problems, since the sum of exponential functions in MGF_i that may lead to over or underflow if not handled properly.

There are two types of floating-point arithmetics that can be used, namely either finite-precision with hardware support (by default), or arbitrary-precision with software supports, e.g., the *mpmath* library in Python [Joh+13].

In the former case, the computation may suffer from over- and underflow problems since $C_i^j \cdot s$ varies with the parameter range of s. In the latter case, the number of digits that can be used for numeric presentation is only limited by the available memory of the computing system. Note that truncation and approximation errors are unavoidable in both arithmetics due to the limitations of binary representation.

Although arbitrary-precision arithmetic is considerably slower than finite-precision arithmetic due to the incurred software overhead, we will adopt it to evaluate the proposed approach to avoid any over- and underflow problems in Section 5.5.

5.4 Deadline-Miss Rate

According to Definition 8, the expected miss rate is the probability of a job τ_k missing its deadline. Nevertheless, it is different to the probability of deadline misses discussed in the previous section, since the assumed system is never restarted or abort any

job missing its deadline. Ideally, the expected miss rate for a specific task τ_k can be determined by counting the number of jobs of τ_k that miss their deadlines and the number of total releases in an infinitely long sequence of jobs. However, it is obviously not practical to get an infinite number of released jobs to measure.

Therefore, we propose to use *busy intervals* to partition deadline misses and describe the probability for different exclusive events (see Section 5.4.1). Based on the above probabilistic arguments, it is possible to derive an upper bound for the expected deadline miss rate with at most $\mathbb{J}$ consecutive jobs missing their deadlines (see Section 5.4.2).

5.4.1 Partition Into Busy Intervals

At first the definition of busy intervals we use for partitioning deadline misses is:

Definition 10 (Busy Interval of τ_k)**.** *An interval $[t_a, t_b]$ is a busy interval of task τ_k, if no job of τ_k presents in the system right before t_a, a job of τ_k arrives at t_a, a job of τ_k finishes at time t_b, no further job of τ_k or a task in $hp(\tau_k)$ is in the system right before t_b, and between t_a and t_b only jobs of τ_k or of jobs in $hp(\tau_k)$ are executed.*

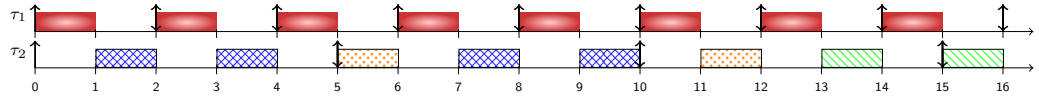
Please note that a busy interval for τ_k also ends at time t_b if a job of τ_k finishes at time t_b and one (or more) job of tasks in $hp(\tau_k)$ and/or of τ_k itself arrive at t_b, if no job of τ_k beside the one finishing at t_b is in the system right before t_b.

Considering constrained- and implicit-deadline task sets, a busy interval of τ_k can end due to different cases that are illustrated in Figure 5.5. Let τ_1 be defined by $C_1^N = C_1^A = 1$, $T_1 = 2$, and $D_1 = 2$. Red boxes represent the execution of τ_1. For τ_2 assume $C_2^N = 1$, $C_2^A = 3$, $T_2 = 5$, and $D_2 = 5$ if it is not declared differently. Blue boxes with crosshatch patterns represent abnormal execution before the deadline of the related job, orange boxes with dot patterns represent abnormal execution after the deadline, and green boxes with slash patterns represent normal execution.

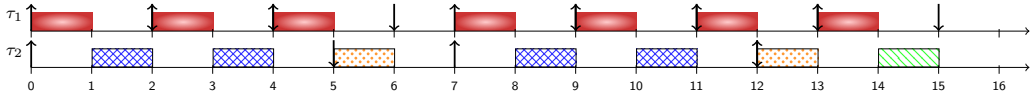
(1) Periodic tasks, implicit-deadline (Figure 5.5a): Busy intervals of τ_2 must always end with a job of τ_2 meeting its deadline. The first two jobs miss their deadlines, the third job meets its deadline at 14.
(2) Sporadic tasks, implicit-deadlines (Figure 5.5b): In this case busy intervals can also end by the processor running idle with respect to $hep(\tau_2)$ as happening at time 6.
(3) Periodic tasks, constrained-deadline (Figure 5.5c): Let $C_2^A = 5$, $T_2 = 8, D_2 = 5$. Similar to the previous case, a busy interval for periodic tasks with constrained deadlines can finish with a job meeting its deadline as happening at $t = 12$.

Depending on the number of jobs of τ_k that miss their deadlines, the busy intervals of τ_k can be partitioned as:

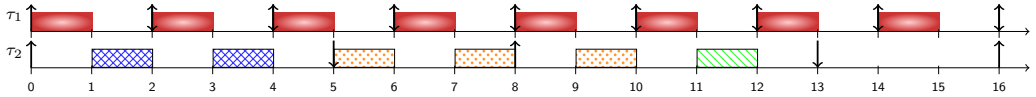
Definition 11 (I_j Busy Interval of τ_k)**.** *A busy interval of τ_k is an I_j busy interval, if the first j jobs of τ_k in the busy interval miss their deadlines.*



(a) Periodic, implicit-deadline task sets: An I_2 interval of τ_2 with two consecutive deadline misses (at $t = 5$ and $t = 10$), followed by a job meeting its deadline at $t = 14$. At $t = 15$, a new busy interval of τ_2 starts.



(b) Sporadic, implicit-deadline task sets: Two I_1 intervals of τ_2, both ending with idle time at $t = 6$ and $t = 15$, respectively. While the first busy interval finishes with idle time, the second finishes with a job of τ_2 meeting its deadline.



(c) Periodic, constrained-deadline task sets: One I_1 interval ending at $t = 12$. It ends after the second job of τ_2 finishes at $t = 12$, meeting its deadline.

Figure 5.5: The possible scenarios for busy intervals. Red boxes represent the execution of τ_1. For τ_2, blue boxes with crosshatch patterns represent abnormal execution before the deadline of the related job, orange boxes with dot patterns represent abnormal execution after the deadline, and green boxes with slash patterns represent normal executions. The upward arrows represent the arrival time for each job, whereas the downward arrows represent the deadline for each job.

For this definition it does not matter, if the busy interval ends with a job of τ_k meeting its deadline or by the processor running idle with respect to tasks in $hep(\tau_k)$. Note that an I_0 busy interval does not contain any deadline misses. The probability of the occurrence for I_j is denoted as $\psi(I_j)$.

Theorem 12. For a given sequence of a sufficiently large number of jobs of τ_k, the jobs can be partitioned into busy intervals of τ_k with at most $\mathbb{J}$ jobs missing their deadlines consecutively. All these busy intervals are independent from each other. Therefore, the sum of the probabilities $\psi(I_j)$ from $j = 0$ to $\mathbb{J}$ is one, i.e., $\sum_{j=0}^{\mathbb{J}} \psi(I_j) = 1$.

Proof. As each busy interval either ends with a job of τ_k meeting its deadline or the processor running idle with respect to tasks in $hep(\tau_k)$, all the busy intervals are independent. If a job misses its deadline, it must be part of a busy interval of type I_j with at least one deadline miss, i.e., $j \geq 1$, with a known probability $\psi(I_j)$, since $D_k \leq T_k$. Otherwise, the probability that a job is meeting its deadline is $\psi(I_0)$. Since the intervals are independent, they are the probabilities for their occurrences. Therefore, $\sum_{j=0}^{\mathbb{J}} \psi(I_j) = 1$. □

5.4.2 Expected Miss Rate

With at most $\mathbb{J}$ jobs consecutively missing their deadlines, now it is possible to describe a given schedule with probabilistic arguments in I_j busy interval manner. According to Definition 8, the expected deadline miss rate $\mathbb{E}_k$ with at most $\mathbb{J}$ jobs consecutively missing their deadlines can be obtained by the following equation:

$$\begin{aligned}\mathbb{E}_k &= \frac{\text{Expected number of deadline misses}}{\text{Expected number of released jobs}} \\ &= \frac{\sum_{j=1}^{\mathbb{J}} \psi(I_j) \cdot j}{\sum_{j=1}^{\mathbb{J}} \psi(I_j) \cdot j + \psi(I_0) \cdot 1}\end{aligned} \tag{5.26}$$

where $\psi(I_j)$ is the probability of the occurrence for I_j for task τ_k for any possible schedule. The numerator of Eq. (5.26), i.e., $\sum_{j=1}^{\mathbb{J}} \psi(I_j) \cdot j$, is the expected number of jobs missing their deadlines among all the possible busy intervals. The denominator of Eq. (5.26), i.e., $\sum_{j=1}^{\mathbb{J}} \psi(I_j) \cdot j + \psi(I_0) \cdot 1$ is the number of sampled jobs (may meet or miss their deadlines) over the given schedule, in which the left part is the number of jobs (missing deadlines) in each I_j busy interval, whereas the right part is the number of a job that must meet its deadline.

However, calculating the probability $\psi(I_j)$ precisely is still an unsolved problem to the best of our knowledge. Alternatively, we adopt the upper bound on the probability of ℓ-consecutive deadline misses $\Phi_{k,\ell}$ presented in Section 5.2.2 to derive the upper bound of $\psi(I_j)$ for over-approximating Eq. (5.26). While we explicitly analyze the miss rate for one task τ_k, the proposed approaches can be applied to each task in any given task sets.

Upper Bound of $\psi(I_j)$

Let $\Phi_{k,j}$ be a safe upper bound on the probability that j (or more) consecutive jobs of τ_k miss their deadline. By definition, for $j \geq 1$, the probability $\psi(I_j)$ for an interval I_j with *exactly* j-consecutive deadline misses must be upper bounded by $\Phi_{k,j}$, i.e., $\psi(I_j) \leq \Phi_{k,j}$. Moreover, as the sequence of $\Phi_{k,j}$ is non-increasing for $j \geq 1$, $\Phi_{k,l}$ must be larger than or equal to the probability $\psi(I_j)$ for an interval I_j with *exactly* j-consecutive deadline misses if $l < j$. For instance, $\Phi_{k,1} \geq \psi(I_2)$.

Several general approaches to calculate $\Phi_{k,j}$ are known from the literature. Job-level *Convolution-based* methods like in [MEP04; MC13] directly enumerate the WCET state space with the associated probabilities. Considering the jobs in non-decreasing order of arrival time, they convolute the current state of the system associated with a vector of possible states, i.e., possible total WCETs and related probability, with the current release jobs. By repeating this procedure until D_k is reached, all released jobs are convoluted, and the probability that the worst case response time is greater than D_k (at least one deadline miss) can be derived accordingly. In [BPK+18] a task-level convolution was proposed that evaluates the resulting deadline miss probability for a set of time points individually. Although these approaches focus on the probability of

one deadline miss, i.e., $\Phi_{k,1}$, they can be extended to multiple deadline misses. For ℓ-consecutive deadline misses, instead of repeating until D_k, the procedure is repeated until $(\ell-1)T_k + D_k$ is reached.

The approaches presented in [MEP04; MC13; KC17; BPK+18] all calculate the probability of deadline misses based on the probability that the workload S_t over a given interval of length t is larger than t. To be more precise, they evaluate if $\mathbb{P}(S_t \geq t)$ for certain t values of interest, i.e., the deadline of τ_k and the release times of all jobs of higher priority tasks. If these probabilities are known for some or all of these values of interest, we can directly apply Theorem 9 proposed in Section 5.2.2 to derive a safe upper bound on the probability for ℓ-consecutive deadline misses.

Please note that while $\Phi_{k,\ell}$ as the upper bound of $\psi(I_j)$ can be calculated by all methods mentioned above, the job-level convolution-based analyses suffers from state explosion due to the large number of jobs and hence are not practicably applicable as shown in [BPK+18] and the evaluation later on. Hence, either the task-level convolution presented by von der Brüggen et al. in [BPK+18] or the analytical approach proposed in Section 5.2.2 should be applied.

Approximation of the Expected Miss Rate

To approximate Eq. (5.26), here we assume that the safe bound on the probability for one deadline miss is greater than 0, i.e., $\Phi_{k,1} > 0$. Otherwise, the expected deadline miss rate is trivially zero, i.e., $\mathbb{E}_k = 0$. A safe upper bound on the expected miss rate $\hat{\mathbb{E}}_k$ can be obtained by the following theorem:

Theorem 13 (Approximation of the Deadline-Miss Rate)**.** Suppose that we are given a schedule of a set of constrained-deadline (or implicit-deadline) sporadic tasks Γ where the largest number of consecutive deadline misses of τ_k in this schedule is $\mathbb{J}$. An upper bound on the expected deadline miss rate of task τ_k can be calculated as

$$\hat{\mathbb{E}}_k = \frac{\sum_{j=1}^{\mathbb{J}} \Phi_{k,j} \cdot j}{\sum_{j=1}^{\mathbb{J}} \Phi_{k,j} \cdot j + (1 - \Phi_{k,1})} \tag{5.27}$$

where $\Phi_{k,1}$ and $\Phi_{k,j}$ are derived by Theorem 9.

Proof. By the above arguments for $\psi(I_j)$, we know $\psi(I_j) \leq \Phi_{k,j}$. For any possible schedule, the probability $\psi(I_0)$ that there is no job missing its deadline must be at least $1 - \Phi_{k,1}$, i.e., $\psi(I_0) \geq 1 - \Phi_{k,1}$, and $\sum_{j=1}^{\mathbb{J}} \psi(I_j) \cdot j > 0$, which means that

$$\frac{\psi(I_0)}{\sum_{j=1}^{\mathbb{J}} \psi(I_j) \cdot j} \geq \frac{1 - \Phi_{k,1}}{\sum_{j=1}^{\mathbb{J}} \Phi_{k,j} \cdot j} \tag{5.28}$$

Therefore,

$$\mathbb{E}_k = \frac{\sum_{j=1}^{\mathbb{J}} \psi(I_j) \cdot j}{\sum_{j=1}^{\mathbb{J}} \psi(I_j) \cdot j + \psi(I_0)}$$

$$= \frac{1}{1 + \frac{\psi(I_0)}{\sum_{j=1}^{\mathbb{J}} \psi(I_j) \cdot j}} \leq \frac{1}{1 + \frac{1-\Phi_{k,1}}{\sum_{j=1}^{\mathbb{J}} \Phi_{k,j} \cdot j}}$$
$$\leq \frac{\sum_{j=1}^{\mathbb{J}} \Phi_{k,j} \cdot j}{\sum_{j=1}^{\mathbb{J}} \Phi_{k,j} \cdot j + (1 - \Phi_{k,1})}$$
$$\leq \hat{\mathbb{E}}_k \tag{5.29}$$

This concludes that $\mathbb{E}_k$ must be upper bounded by $\hat{\mathbb{E}}_k$, i.e., $\mathbb{E}_k \leq \hat{\mathbb{E}}_k$. □

The following example illustrates how Eq. (5.26) and Eq. (5.27) can be used to calculate the expected miss rate.

Example 4 (Example of Expected Miss Rate)**.** Suppose that the probabilities of I_j are given as: $\psi(I_0) = 0.99$, $\psi(I_1) = 0$, and $\psi(I_2) = 0.01$. With Eq. (5.26), we get the expected miss rate:

$$\frac{0.01 \cdot 2}{0.01 \cdot 2 + 0.99} = 0.0198$$

Let the safe bounds of them be given as follows: $\Phi_{k,1} = 0.05$, $\Phi_{k,2} = 0.02$, and $\Phi_{k,3} = 0$. With Eq. (5.27), a safe upper bound on the expected miss rate can be derived as:

$$\frac{1}{1 + \frac{0.95}{0.05+0.02 \cdot 2}} = 0.0865$$

□

5.4.3 Threshold $\mathbb{J}'$ and Time Complexity

Since in Eq. (5.27) the maximum number of jobs with consecutive deadline misses $\mathbb{J}$ could be extremely large or even infinite, an additional approximation is required to bound the summation in Eq. (5.27). Let $S = \sum_{j=1}^{\infty} \Phi_{k,j} \cdot j$. We can use a threshold $\mathbb{J}'$ to simplify the procedure. Let $r_j = \frac{(j+1)\Phi_{k,j+1}}{j\Phi_{k,j}}$. Assume there is a given $\mathbb{J}'$, such that $r_{\mathbb{J}'}$ is always larger than the other r_j, where $\mathbb{J}' < j$ and $0 < r_{\mathbb{J}'} < 1$. A safe upper bound on $\hat{\mathbb{E}}_k$ can be obtained by the following theorem:

Theorem 14. Suppose that we are given an index $\mathbb{J}'$ such that $r_{\mathbb{J}'}$ is always larger than any other r_j if $0 < \mathbb{J}' < j$. Let $\hat{S} = \sum_{j=1}^{(\mathbb{J}'-1)} j\Phi_{k,j} + \frac{\Phi_{k,\mathbb{J}'}}{1-r_{\mathbb{J}'}}$. An upper bound on the expected deadline miss rate of task τ_k can be calculated as

$$\hat{\mathbb{E}}_k^* = \frac{\hat{S}}{\hat{S} + 1 - \Phi_{k,1}} \tag{5.30}$$

Proof. Comparing to Eq. (5.27), Eq. (5.30) only replaces S with $\hat{S}$. Therefore we prove $\hat{S} \geq S$ as follows:

$$S = \sum_{j=1}^{\infty} \Phi_{k,j} \cdot j$$

$$
\begin{aligned}
&= \sum_{j=1}^{(\mathbb{J}'-1)} j\Phi_{k,j} + \sum_{z=\mathbb{J}'}^{\infty} z\Phi_{k,z} \\
&= \sum_{j=1}^{(\mathbb{J}'-1)} j\Phi_{k,j} + \Phi_{k,\mathbb{J}'} + r_1\Phi_{k,\mathbb{J}'} + r_1 r_2 \Phi_{k,\mathbb{J}'} \dots \\
&\leq \sum_{j=1}^{(\mathbb{J}'-1)} j\Phi_{k,j} + \sum_{z=0}^{\infty} (r_{\mathbb{J}'})^z \Phi_{k,\mathbb{J}'} \qquad (5.31)
\end{aligned}
$$

As $\sum_{z=0}^{\infty}(r_{\mathbb{J}'})^z\Phi_{k,\mathbb{J}'}$ is an infinite series with a common ratio $r_{\mathbb{J}'}$, in which $0 < r_{\mathbb{J}'} < 1$, we can calculate Eq. (5.31) from the finite sum formula:

$$
\hat{S} = \sum_{j=1}^{(\mathbb{J}'-1)} j\Phi_{k,j} + \frac{\Phi_{k,\mathbb{J}'}}{1 - r_{\mathbb{J}'}}
$$

Since $\hat{S} \geq S$, it follows directly that $\hat{\mathbb{E}}_k^* \geq \hat{\mathbb{E}}_k$. □

The complexity of this approximation mainly comes from $\hat{S}$, which is dominated by the calculation of $j \cdot \Phi_{k,j}$ for $j = 1, 2, \dots, \mathbb{J}'$. According to Eq. (5.20), we need to calculate $\Phi_{k,w}^{\theta}$ in every iteration and use the derived $\Phi_{k,\ell-w}$ from the previous iterations. The space complexity to record all values $\Phi_{k,w}^{\theta}$ for $w = 1, 2, \dots, \mathbb{J}'$ is $O(\mathbb{J}')$. Suppose that the time complexity to determine if $\mathbb{P}(S_t \geq t)$ for a given single time point t is $O(L)$. For each $\Phi_{k,w}^{\theta}$, the time complexity is $O(D_k \cdot L)$ with discretized time points t from 0 to D_k. Since $\Phi_{k,\ell}$ can be calculated by the derived $\Phi_{k,w}^{\theta}$ immediately, thus, the time complexity of $\hat{S}$ is $O(\mathbb{J}' \cdot D_k \cdot L)$ (if we only look at $\Phi_{k,\mathbb{J}'}$).

5.5 Experimental Evaluation

This section presents two different piles of experimental evaluations for the proposed approaches in this chapter as follows:

1. We evaluated the approximation quality and the runtime of the deadline-miss probability computation of the analytical bound approach. As shown in [BPK+18], Hoeffding's and Bernstein's inequalities are several orders of magnitude faster than the Chernoff bound approach, whereas the error of them is large, i.e., by several orders of magnitude. Therefore we selectively only evaluated the Chernoff bound approach in Theorem 3 and Theorem 4, denoted as **Chernoff** and **Chernoff-K** respectively, compared to the traditional job-level convolution-based approach proposed in [MC13], denoted **CPRTA**.
2. We evaluated the efficiency and the pessimism of the proposed analytical approach in Section 5.4.2. To calculate the upper bound of deadline miss rate, we used two different ways to determine $\Phi_{k,j}$, namely the Chernoff bound approach with k selected points in Theorem 4, denoted as **Chernoff**[6], and the

[6]The scripts were downloaded from https://github.com/kuanhsunchen/EPST/ on July. 23 in 2018.

tighter task-level convolution-based approach presented by von der Brüggen et al. in [BPK+18], denoted as **CON**. Moreover, we compared to the results derived from the event-driven simulator, introduced in Section 3.4.3, labeled as **SIM**.

Please note that the convolution-based approach in [MC13], i.e., **CPRTA**, leads to identical results as the task-level convolution-based method, i.e., **CON**, proposed in [BPK+18] while it has (in general) a larger runtime. However, while we proposed the Chernoff bound approach, the state-of-the-art was still the job-level convolution-based method proposed in [MC13]. Therefore the first evaluation here keeps as it was to demonstrate the novelty of our contribution at that moment.

5.5.1 Experimental Setup

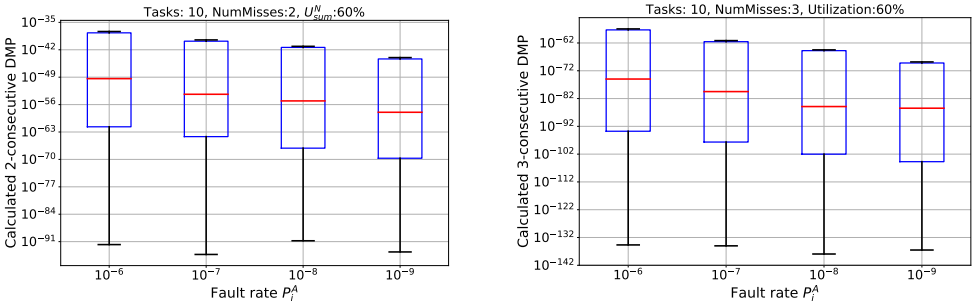
For the evaluation, we implemented the probabilistic schedulability test with the Chernoff bound approach on Linux kernel 3.13.0 with Python 2.7. The adopted machine had an Intel Core i7-4770 and 8GB DDR3 RAM. To find the s with the minimal probability in Eq. (5.7), we used SciPy library [JOP+01] and searched multiple s in (0, 1]. The complete scripts are all available at [Kua17]. Please note that the proposed analyses should be carefully implemented, as the considered probabilistic values are very small. Due to the lack of precision in floating-point calculation, the terms in the commutative operations should be pre-sorted to avoid inconsistent results.

We synthesized randomized implicit-deadline task sets with a given utilization value U^N_{sum}, i.e., $U^N_{\text{sum}} = \sum_{\tau_i \in \Gamma} U^N_i$ according to the UUniFast method [BB05], where U^N_i is defined as C^N_i/T_i. For each task set, the task periods were generated by a log-uniform distribution with two orders of magnitude, i.e., [1-100]. We adopted time-demand analysis [LSD89] to ensure the schedulability when all tasks are always executed normally, and discarded those task sets that were anyway not schedulable. To simulate a recovery mechanism, we set the error detection costs to 20% of the task execution time and assumed a complete re-execution if a fault is detected, i.e., C^A_i was set to $\frac{2.2}{1.2} \approx 1.83 \cdot C^N_i$ for all tasks $\tau_i \in \Gamma$[7]. With the generated utilization U^N_i, the normal execution time C^N_i was set to $U^N_i \cdot T_i$.

In the first evaluation, the considered cardinalities of the task sets were: 10, 20, and 30 tasks. For each given fault rate $\mathbb{P}^A_i$, we recorded 100 synthesized task sets. After observing that the results among these three cardinalities are similar, we select to show the case for 10 tasks to demonstrate the comparisons. For the second evaluation, task sets with a cardinality of 2, 5 and 10 were considered in this evaluation. We used an identical fault rate P^A_i for all tasks, and only present the miss rates of the lowest-priority tasks.

For the configuration of the simulator, since we consider tasks with their specified minimum inter-arrival times (i.e., they are sporadic), it is unnecessary to release a higher-priority job if there is no unfinished job of a task under analysis. In order to

[7] Here we assume there is at most one error to be handled for each job of tasks.



(a) DMP of 2-consecutive deadline misses **(b)** DMP of 3-consecutive deadline misses

Figure 5.6: Calculated DMPs with $U^N_{\text{sum}} = 60.0\%$, varying fault rates and ℓ .

fairly compare with our analytical approach considering the worst case, we enforced a worse release pattern in the simulator so that the estimated miss rate is closer to the worst case. Namely, if the lowest-priority task does not have any unfinished job at time t, a higher-priority task does not release any job even if its minimum inter-arrival time allows it to release a job at time t already. Instead, the next release of those higher-priority tasks is postponed to the point in time when the next job of τ_k is released.

5.5.2 Evaluation of Deadline-Miss Probability

In this evaluation, we present the maximum DMP $\hat{\Phi}_{i,\ell}$ among all the tasks with the given ℓ, i.e., $\tau_i \in \Gamma$. For each fault rate, we recorded 100 calculated DMP and reported the medians (red lines), the interquartile range of the sample (the width of the boxes), the maximums (top lines), and the minimums (bottom lines) with the box plots, where a base-10 log scale is used for the Y-axis.

Firstly we compare the results derived by **Chernoff** and **Chernoff-K** to evaluate the approximation by testing only k time points. Interestingly, the results derived from both approaches are almost identical. Namely, the probability $\hat{\Phi}_{i,\ell}$ of deadline misses delivered by **Chernoff-K** is always the same as $\Phi_{i,\ell}$ delivered by **Chernoff**. Although we know that testing a pseudo-polynomial number of time points may give us a tighter upper bound on the deadline-miss probability, our experiments empirically support that it may be sufficient to test only k-points derived by **Chernoff-K** to obtain the upper bound on the probability efficiently. For the real valued parameter s adopted in **Chernoff** and **Chernoff-K**, we search s only in the range of $(0, 1]$.

Figure 5.6 presents the relationship for the probability of $\{2, 3\}$-consecutive deadline misses, in which their interquartiles are all close to the maximums. The medians start to downgrade when the fault rate is decreased. Moreover, the ranges between the maximums and minimums are slightly changed when the fault rate goes down.

Methods	Cardinality	5 tasks	10 tasks	20 tasks	30 tasks
CPRTA	Avg. Time (sec)	7.4812	-	-	-
	Successful Runs	98/100	0/100	0/100	0/100
Chernoff	Avg. Time (sec)	0.1406	0.4855	1.6738	2.7545
	Successful Runs	100/100	100/100	100/100	100 /100
Chernoff-K	Avg. Time (sec)	0.0418	0.1253	0.4760	0.7917
	Successful Runs	100/100	100/100	100/100	100/100

Table 5.2: Analysis time need for 100 synthesized task sets per configuration.

Chernoff-Bound against Job-Level Convolution-Based Approaches

In order to evaluate the pessimism and the efficiency, we compared the proposed approaches, i.e., **Chernoff** and **Chernoff-K**, with the job-level convolution-based approach proposed in [MC13] without any further approximation, denoted **CPRTA**. We used the released scripts in MATLAB and only changed the input of the simulation by using the task generator described in Section 3.4.[8] Specifically, the randomized period of tasks in this comparison is uniformly distributed between $[1, 50]$. Comparing to typically using $[1, 100]$ in the literature, this is unfortunately needed to reduce the number of iterations in **CPRTA**.

Table 5.2 first presents the required analysis times for different analyses. We test over 100 task sets for each configuration and we set at most 10 minutes as the timeout threshold in each task sets. Unfortunately, **CPRTA** is still not able to derive the DMP for $10, 20$ and 30 tasks after we adjusted the randomized distribution of periods to $[1, 50]$. Without setting the timeout threshold, we also use 6 computers in our local cluster to derive the results by using **CPRTA** for 12 hours for task sets with 10 tasks. To the end, however, none of the **CPRTA** analyses is able to finish. The run time reported in [MC13] was 140 seconds in average for task sets with 16 tasks. However, the convolution-based approaches for testing the generated task sets in [MC13] were usually quite easy to finish. When we used the widely-accepted UUniFast method for generating the task sets, we note that the convolution takes much longer time to compute. Thus, such convolution-based approaches would be very time consuming and are not suitable for large task sets.

Moreover, we compare the derived DMPs of **CPRTA** and **Chernoff-K** under different U^N_{sum} for the lowest priority task in each set. In Figure 5.7, the results in the left-hand side are derived under $U^N_{\text{sum}} = 60\%$, whereas the results in the right-hand side are derived under $U^N_{\text{sum}} = 70\%$. As shown in Figure 5.7, besides the extreme cases, our method is a bit pessimistic (the lower the tighter). However, if the pessimism of sufficient tests is acceptable especially under a large number of tasks, the proposed approaches can efficiently derive the upper bound on the probability of deadline misses

[8] The scripts were downloaded from `https://who.rocq.inria.fr/Dorin.Maxim/` on Jan. 24 in 2017. The modified scripts and the input generated by UUniFast method can be found in [Kua17].

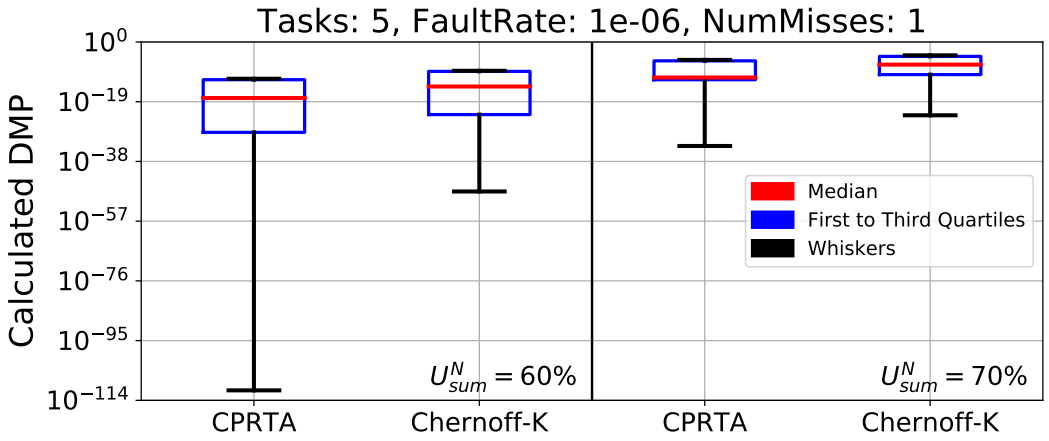


Figure 5.7: DMPs under different approaches and varying U^N_{sum}.

instead of unnecessarily deriving the whole response time distribution.

Job-level Convolution with Re-Sampling against Chernoff-K

As shown in the previous subsection, we can see that **CPRTA** without any approximation is very time consuming even with 10 tasks. In [MC13], the re-sampling technique by manually introducing a threshold of the valid number of data points was shown to improve the needed calculation time of **CPRTA** by orders of magnitudes faster. However, with the task sets generated by applying the UUniFast method, the results derived from **CPRTA** with the re-sampling technique, denoted as **CPRTA-resample**, may be worse than the results derived from **Chernoff-K** even for 10 tasks as shown in Figure 5.8. Although for 10 tasks using re-sampling in **CPRTA** with a threshold set to 100 indeed reduces the calculation time, i.e., around 1 second in average, the derived results are all worse than our analysis in medians and the interquartile ranges of the sample as shown in Figure 5.8. Especially when $U^N_{\text{sum}} = 70\%$, the DMPs derived from **CPRTA** with re-sampling are really closed to 1. For 20 tasks and above, we can not obtain any result by using re-sampling with a threshold set to 100 within 6 hours. With threshold 1000, **CPRTA** is not able to finish even with $U^N_{\text{sum}} = 60\%$ in 6 hours. With threshold 50, the calculation time of **CPRTA** is extremely fast, but the results are all close to 1 even under utilization $U^N_{\text{sum}} = 60\%$.

After all, we know that **CPRTA** with re-sampling could provide tighter results with a higher threshold but require much more time to calculate the probability. Conversely, **CPRTA** with re-sampling could need less time for calculation with a lower threshold but provide looser results with respect to the probability of the deadline-miss rate. If using convolution-based approaches anyway cannot avoid to use approximations like the re-sampling technique to reduce the time complexity, how to properly approximate the convolution for the balance between the tightness and

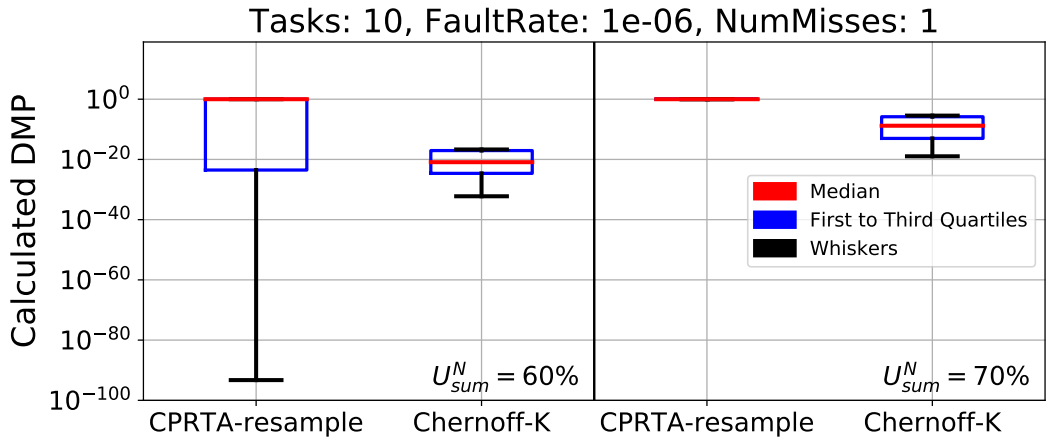


Figure 5.8: Comparison between **Chernoff-K** and **CPRTA-resample** with a threshold set to 100, varying U^N_{sum}.

the calculation time is another considerable issue, which is the essential problem that needs to be solved for the convolution-based approached proposed in the literature.

Task-level Convolution-based Approach against Chernoff bound Approach

In this evaluation, we considered the following approaches: **Chernoff** referring to the Chernoff bound based approach that uses a golden-section search to find an optimal s value with only k time points, i.e., we rename **Chernoff-K** as **Chernoff** for the rest of evaluations, and **Pruning** referring to the task-level convolution-based approach with the pruning technique described in [BPK+18]. We compare the computed deadline-miss probabilities of the lowest-priority tasks and the required runtime by **Chernoff** and **Pruning**. Further, we used task sets with a varying number of tasks, i.e., $\{10, 15, 20, 25\}$. Due to the high runtime required by **Pruning**, we created a varying number of task sets depending on the number of tasks, i.e., 100 sets with 10 tasks, 50 sets with 15 tasks, 25 sets with 20 tasks, and 5 sets with 25 tasks. We considered two normal-mode utilizations, namely 50% and 70%.

Fig. 5.9 and Fig. 5.10 show that the average computation time used for the Chernoff bounds is 1 – 3 magnitudes faster than **Pruning**. Additionally, unlike **Pruning**, the runtime of **Chernoff** is insensitive to the task set utilization. With a larger number of tasks per task set, i.e., 100 tasks, **Pruning** cannot derive any results even over 24 hours, whereas **Chernoff** finished the computations in average 507.5 sec over 5 sets.

With respect to the approximation quality, Fig. 5.11 displays the calculated deadline-miss probabilities of the lowest-priority tasks in each analyzed task set consisting of 20 tasks each with cumulative utilization of 50%. As expected, due to the lack of an analytical bound on approximation performance of **Chernoff**, the difference between the two methods can be arbitrarily large. Moreover, the differences are larger for task sets where the deadline-miss probability is already very low. By contrast, in the cases where the deadline-miss probability is higher, e.g., 10^{-3} to 10^{-1},

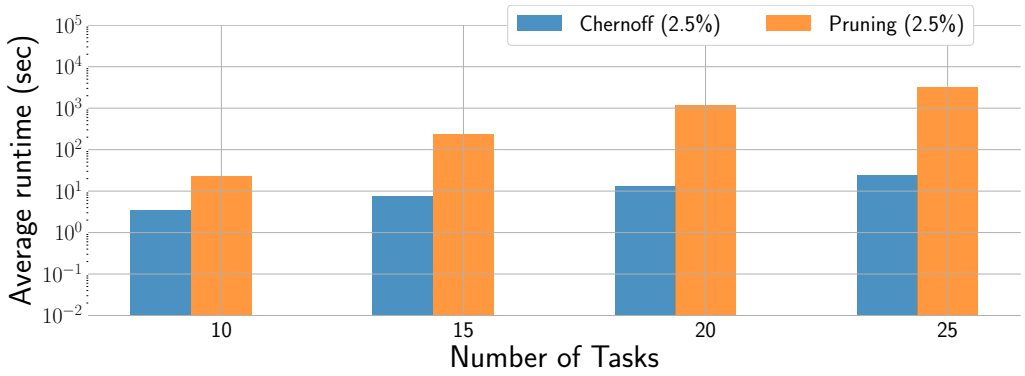


Figure 5.9: Average runtime time for the Chernoff and **Pruning** method for soft-error implicit-deadline task sets with 50% utilization for different number of tasks per task set. The soft-error probability is set to 2.5%.

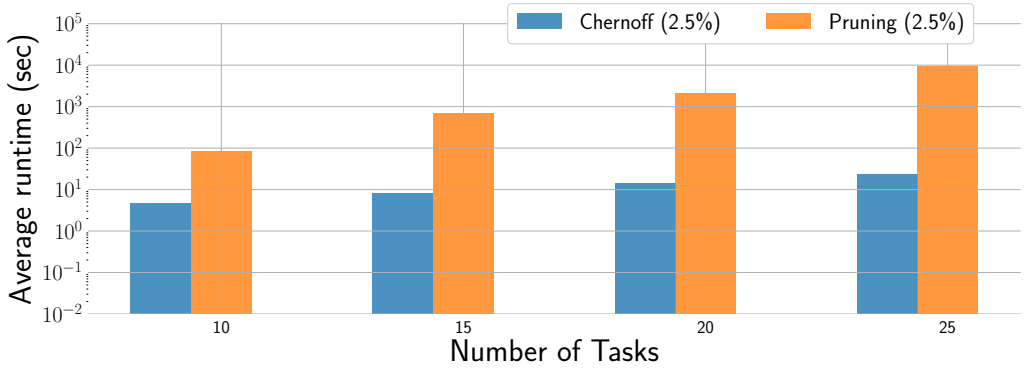


Figure 5.10: Average runtime time for the Chernoff and **Pruning** method for soft-error implicit-deadline task sets with 70% utilization for different number of tasks per task set. The soft-error probability is set to 2.5%.

the differences are relatively smaller. Fig. 5.11 also shows that the optimal value of s depends on the specific settings of the task set, e.g., the utilization, which empirically shows that testing only a specific range is not enough.

5.5.3 Evaluation of Deadline-Miss Rate

Since the calculation of $\Phi_{k,j}$ mainly determines the required analysis time and the precision for the upper bound of the expected miss rate, we first present the performances of two applicable state-of-the-art techniques, i.e., **Chernoff** and **CON**, over different value j of calculating $\Phi_{k,j}$.

Trends of $\Phi_{k,j}$ from CON and Chernoff

For a cardinality 10 tasks, we reported values of $\Phi_{k,j}$ for j up to 6 with 5 random task sets. For a cardinality 5 tasks, 10 random task sets were analyzed up to 5. From

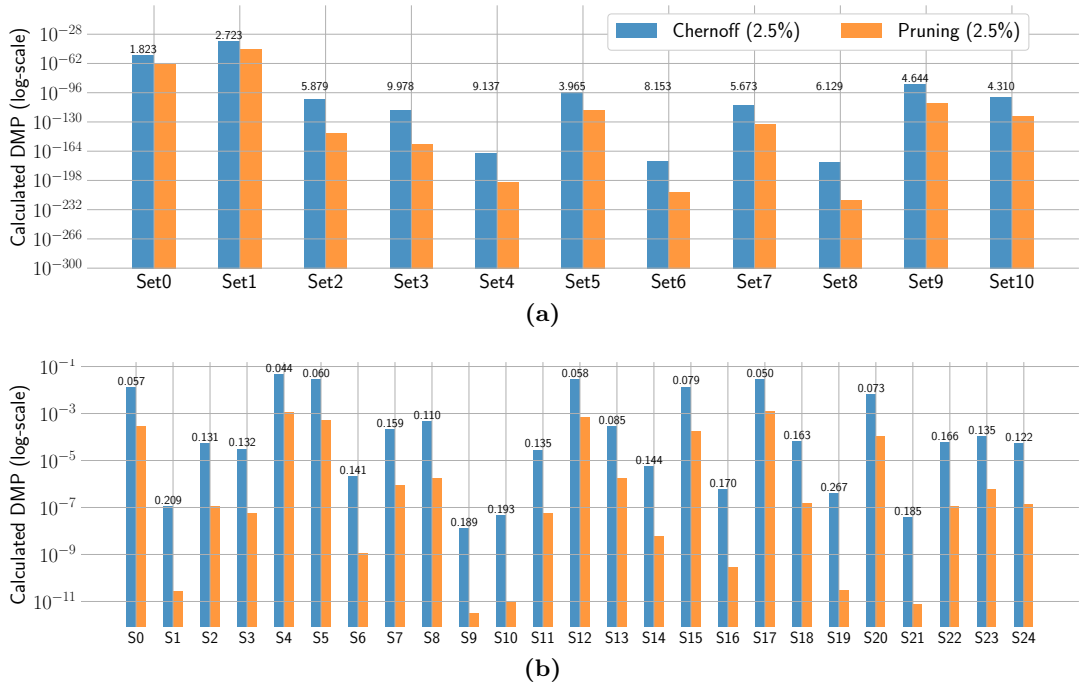


Figure 5.11: Approximation quality for the Chernoff and **Pruning** method for soft-error recovering implicit-deadline task sets with 50% Figure a) and 70% Figure b) utilization for different task sets. The bars are annotated with the computed optimal value of s.

Fig 5.12, we can observe that **Chernoff** requires significantly less amount of analysis time than **CON**. When the cardinality of the task sets increases from 5 to 10, we can see that for $j = 5$, calculating $\Phi_{k,j}$ by **CON** requires almost 6000 seconds in average, which is in practice hard to be applied. This is due to the large number of jobs involved in the interval of analysis since the time complexity of **CON** is roughly the number of jobs of a task to the power of the number of tasks.

In Figure 5.13, we compare the calculated expected miss rates. The y-axis here is the ratio between the derived values over two different approaches, in which the higher the ratio, the bigger the differences. We can see that if j is larger, the differences of the derived values are bigger. When the cardinality of the task sets is big like 10, the differences between two approaches are not that significant. Generally, when j increases, $\Phi_{k,j}$ decreases drastically. When we consider the calculation of S, we can expect that the value contributed from $j \cdot \Phi_{k,j}$ for bigger j becomes negligible very soon. This observation supports our approximation in Section 5.4.3. Since the derived upper bound of the expected miss rate is mainly dominated by those $\Phi_{k,j}$ with small j, such insignificant $\Phi_{k,j}$ values resulting from a large j and, hence, requiring significant analysis time can be sensibly ignored. Therefore, we set $\mathbb{J}'$ to 4 for $\hat{S}$ in the rest of evaluations to efficiently over-approximate the expected deadline miss rate $\hat{\mathbb{E}}_k$ for all the given task sets.

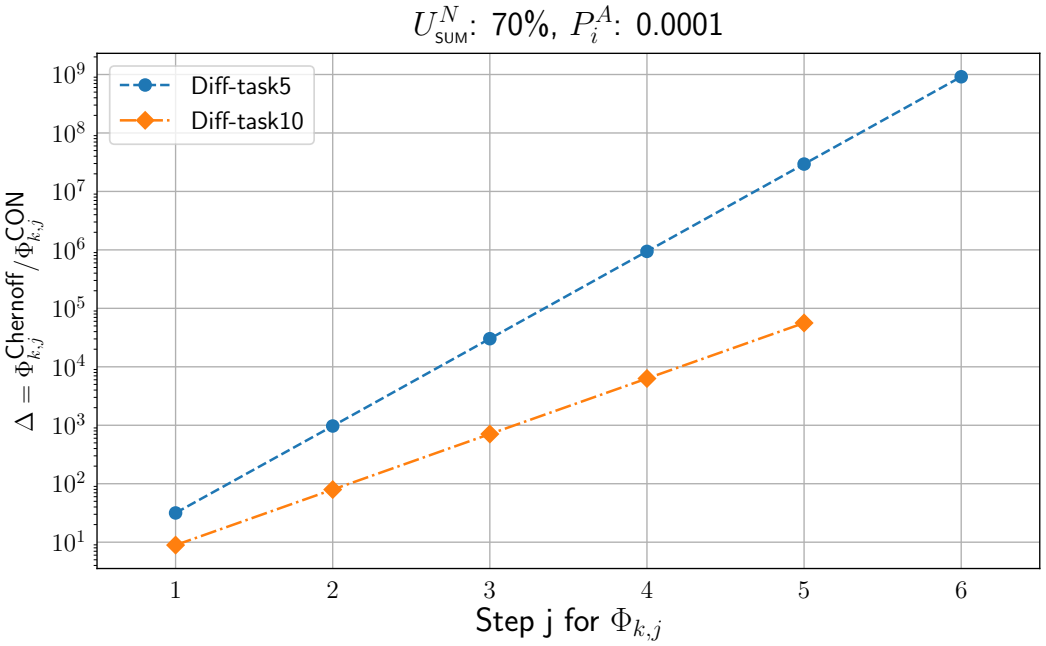


Figure 5.12: Trends of $\Phi_{k,j}$ with respect to different j and different cardinalities.

Comparison among SIM, CON and Chernoff

To evaluate the efficiency of our analysis, we compared the derived upper bounds from **CON** and **Chernoff** to the estimated deadline miss rate from the event-driven simulator **SIM** introduced in Section 3.4.3. The considered cardinality of the task sets is 2 in this evaluation, since deriving $\Phi_{k,j}$ using **SIM** to simulate the miss rates is really time consuming, i.e., it requires 338.84 seconds in average. The simulator stopped its simulation when two-million jobs from the lowest priority task finished their executions. We show the results for 20 different task sets where the expected miss rate derived from **SIM** was greater than 10^{-5} in Figure 5.14. The restriction regarding the displayed task sets is taken to increase the readability of the figure.

In Figure 5.14, the bounds derived from **CON** and **Chernoff** are generally higher than the miss rates of **SIM**. However, we can observe that from some sets like $\{S0, S4, S6, S7, S12, ...\}$, the expected miss rate derived by **SIM** is higher than **CON**. The main reason is that the number of tested jobs is not enough. This also empirically supports that deploying simulations to estimate the deadline miss rate is not practical.

Although the bounds derived from **CON** in our setting are relatively close to the simulated miss rate from **SIM**, **CON** does not scale that well with respect to the number of tasks in the system, especially when calculating $\hat{S}$. The bounds derived from **Chernoff** are generally greater than the estimated bound from **CON** by two orders of magnitude. However, it is more scalable than **CON** and **SIM**. Hence, the trade-off between precision and required time has to be carefully considered when

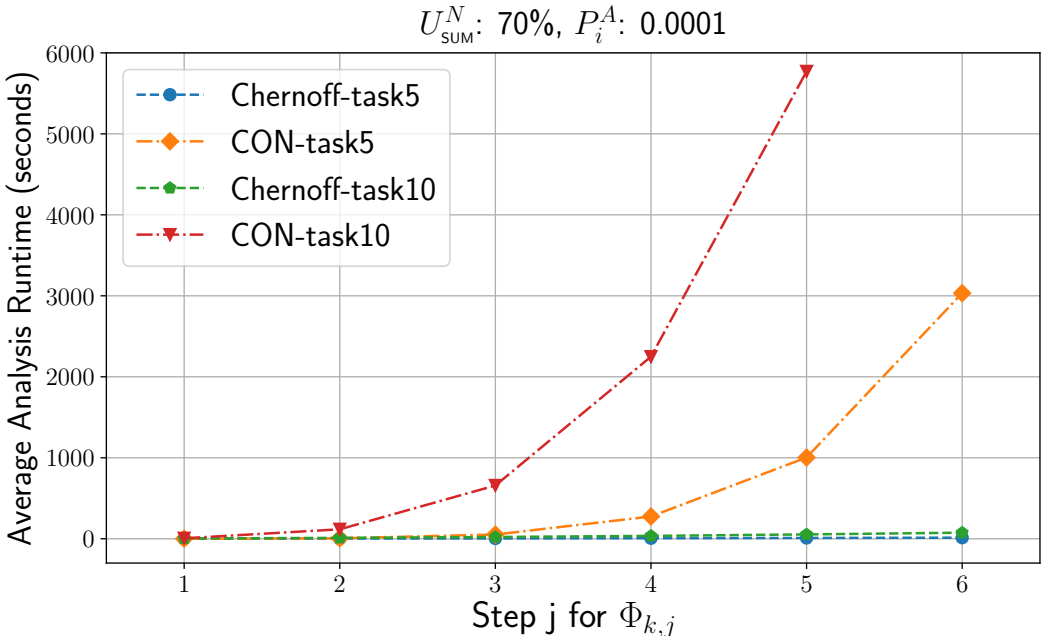


Figure 5.13: Average analysis runtime for calculating $\Phi_{k,j}$ with respect to different j and different cardinalities.

applying our analytical approach. If the error resulting from the pessimism of the Chernoff bound is acceptable, **Chernoff** is still a good choice.

Expected Miss Rate with More Tasks

Since the variation of the derived miss rates is significant even under the same utilization setting and fault rate, we choose box plots to present the results, i.e., shwoing the medians (red lines), the interquartile range of the sample (the width of the boxes), the maximums (top lines), and the minimums (bottom lines). We recorded the miss rates of 100 synthesized task sets for each configuration. Unfortunately, **CON** could not obtain $\hat{\mathbb{E}}_k$ in an acceptable amount of time even with $\mathbb{J}' = 4$, i.e., 30 minutes per task set, whereas **Chernoff** could obtain the result for each task set in, on average, 2 minutes. Therefore, we only present the results $\hat{\mathbb{E}}^*_k$ while $\Phi_{k,j}$ are derived by **Chernoff**, i.e., Chen and Chen's method [KC17], for the cardinality with 10 tasks. As shown in Figure 5.15, naturally the derived bounds are less if the fault rates are lower; the derived bounds are higher if U^N_{sum} is higher. The trends of results derived using different values of U^N_{sum} are similar to the results presented in Figure 5.15.

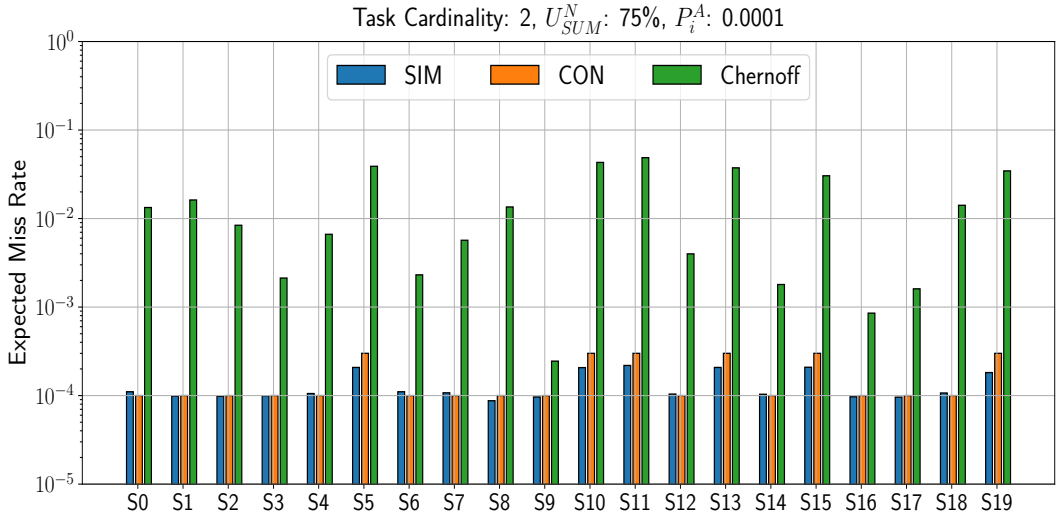
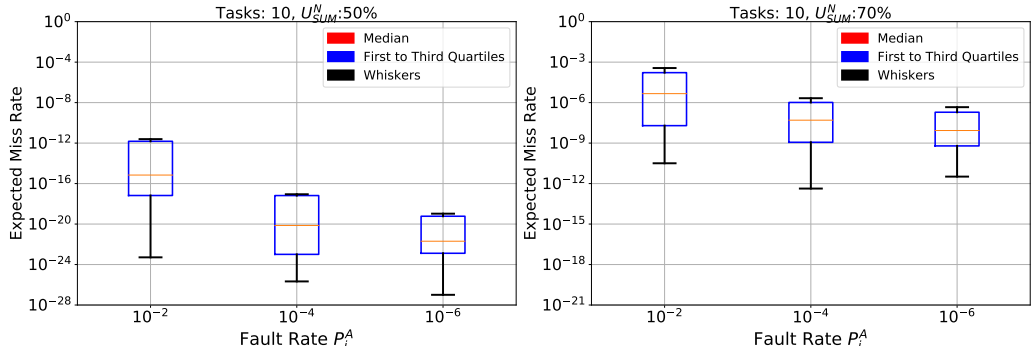


Figure 5.14: Expected miss rates with 20 random task sets.



(a) U^N_{sum} is 50% under different P^A_i. **(b)** U^N_{sum} is 70% under different P^A_i.

Figure 5.15: Expected Miss Rate $\hat{\mathbb{E}}_k$ derived from **Chernoff** for 10 tasks under different U^N_{sum} and different fault rates P^A_i.

5.6 Summary

As soft real-time systems tolerate and occasionally accept certain deadline misses, the *probability of deadline-miss* and the *deadline-miss rate* both are important performance indicators to evaluate the proposed analyses, scheduling algorithms, etc. This chapter firstly presented analytical bound approaches in Section 5.2.1, which can safely over-approximate the probability of deadline misses. It could be shown that the computations are substantially faster than the state-of-the-art convolution-based approaches. Furthermore, it could be shown that in the cases where the cut-off deadline miss probability of interest is in the region of 10^{-3} to 10^{-1}, the differences in approximation quality are reasonable in light of the runtime improvement. To the

best of our knowledge, this is the first work adopting concentration inequalities to efficiently derive the probability of deadline-misses.

Secondly, this chapter provided an analytical approach in Section 5.4.2, which can efficiently derive a safe upper bound on the expected deadline miss rate, based on the analytical bound approaches proposed in this chapter or the state-of-the-art convolution-based approaches in the literature. The experimental results show the applicability and the efficiency of our approaches, evaluating the derived expected miss rate, the precision and the required runtime over different backbones, and the estimated miss rate from simulations. To the best of our knowledge, this is the first study analyzing the expected deadline miss rate in general task and scheduling models.

Overall, the proposed approaches in this chapter provided some new trails, compared to the existing techniques in the literature, by which the system designer is allowed to efficiently provide probabilistic arguments for the probability of deadline-miss and the deadline-miss rate. The studied model and results are aligned with the design requirement for safety-critical systems. For example, verifying if the probability as a threshold to reboot the system for resolving consecutive deadline misses is acceptable or not. All related scripts are available on [Kua19] and [Kua18].

CHAPTER 6

Reliability Optimization for Multi-Cores Task Mapping

Contents

6.1 Overview

Instead of considering uniprocessor systems in Chapter 4 and 5, in this chapter, we study how to achieve resource-efficient reliability on multi-cores systems. In such

systems, a commonly adopted soft error mitigation technique is Redundant Multi-Threading (RMT) that achieves error detection and recovery through redundant thread executions on different cores for an application. Towards this, both need to be handled in this study, 1) which tasks should be assigned to which cores, namely *task mapping problem*, and 2) which reliable levels should tasks activate, namely *task redundancy level problem*.

In this chapter, we consider how to adopt RMT to improve the system reliability on two different system models. The first one is that the individual cores on the considered multi-core systems exhibit different frequencies due to process variations [BDM02], aging effects [SGH+14], and performance heterogeneous (micro-) architecture designs [ARM13]. For instance, *process variations* may lead to significant frequency variations (e.g., up to 30% [BDM02]). Another source of performance heterogeneity is iso-ISA performance-heterogeneous cores, e.g., ARM big.LITTLE architecture [ARM13] and Kumar et al. [KFJ+03]. In the state-of-the-art, i.e., dTune [RKS+14b], a greedy mapping approach is adopted to match reliability-critical tasks onto high-frequency cores. However such a greedy approach lacks effectiveness, since the number of high-frequency cores in the considered system might not be sufficient. In addition, the reliability degradation of executing tasks on low-frequency cores also needs to be considered for all tasks at the same time. Under core-to-core frequency variations, there may be scenarios as shown in the motivational example (see Section 6.1.1), where assigning the reliability-critical task to the high-frequency core is not reliability-wise beneficial, since the timeliness of the overall system also needs to be thoughtfully satisfied.

Secondly, instead of solely adopting Chip-level Redundant Multi-threading (CRT)-Triple Modular Redundancy (TMR) (or not) and one-by-one mapping, we consider that tasks can be executed in multiple different redundancy levels while satisfying the given hard real-time constraints under multi-tasking. Similar scenarios are in general considered in [BCM+13; DKV13; IPE+12], i.e., the objective of these papers is to schedule tasks with real-time constraints while concerning fault detection and tolerance features. However, these papers consider frame-based task systems. Even in such a simplified setting, the problem explored in this chapter is still very different from the problems explored in [BCM+13; DKV13; IPE+12]. In [BCM+13; DKV13] the energy efficiency and mean-time to failure under lifetime constraints is optimized. In [IPE+12] the authors present a runtime scheduling strategy to adapt the system to the occurrence of faults at runtime to reduce the overhead due to fault tolerance. Those algorithms are not applicable to the problem studied in this chapter.

Motivated by the above issues, in this chapter, we present several optimization approaches to tackle the task mapping problem and the task redundancy level problem under different system models respectively. These approaches aim to optimize the overall system reliability[1], e.g., the sum of reliability penalties or the maximum relia-

[1]Whenever we mention the system reliability, the considered metric is from the penalty perspective. If the reliability penalty is high, the system is less reliable. If the reliability penalty is low, the system is more reliable. Details can be found in Section 6.1.2.

bility penalty, while satisfying given hard real-time constraints. In the first model, we introduce the reliability-driven task mapping technique determining task redundancy levels, i.e., task execution with or without RMT, and task allocation decisions, i.e., mapping (redundant) tasks on many-cores with heterogeneous performance characteristics. In the second model, we consider different RMT levels on a multi-core system, i.e., Double Modular Redundancy (DMR)/TMR with Simultaneous Redundant Multi-threading (SRT), CRT, and Mixed Redundant Threading (MRT), and tackle the reliability optimization problem by adopting Federated Scheduling [LCA+14] and R-BOUND-MP-NFR [AJ03] as the backbone. We propose several dynamic programming algorithms to optimize the system reliability while guaranteeing its schedulability. Throughout the simulation results, we can see that the proposed approaches significantly improve the system reliability and even the system schedulability compared to the state-of-the-art techniques.

6.1.1 Motivational Example

This section presents two examples to illustrate the motivations of this chapter: The first example demonstrates a scenario that the greedy mapping approach is not good enough even only CRT-TMR is considered; the second example shows why the MRT-TMR can balance the usage of resource with respect to the number of cores and the execution time.

Greedy Mapping is Not Good Enough!

Here we provide a motivational example to explain why the greedy mapping, e.g., used in dTune [RKS+14b], is not good enough with respect to the overall reliability. For simplicity, we only present the motivational example for tasks by *single version* with given redundancy levels. Suppose that we are given three tasks, i.e., τ_1, τ_2, and τ_3. Task τ_1 has the CRT-TMR requirement but the others have no redundancy. Five cores are sorted by their frequencies beforehand, i.e., c_j is faster than c_{j+1}. Now we consider the task mapping problem to assign the cores to three tasks for minimizing overall penalty, where the reliability penalty of tasks on each core are defined in Table 6.1a. As shown in Figure 1.4 in Chapter 1, the penalty value of Table 6.1a for τ_1 depends on the frequency of lowest-frequency core in the assigned core group. Please note, we denote the penalty value as ∞ to show the infeasible mapping that violates the tolerance of deadline miss rate. With the above setting, there are four available assignments with different penalty values as illustrated in Table 6.1b.

In the above example, we can check all the possible mappings to obtain the optimal result that will be 0.42 while the miss rate constraint is not violated. In this example, τ_1 cannot adopt core c_5 for RMT activation, since the tolerable miss rate will be violated. By using the greedy mapping to assign the tasks and cores, overall penalty of mappings is 0.6. RMT-activated task τ_1 uses core group $\{c_1, c_2, c_3\}$ for the minimal communication overhead, and the reliability-wise critical task, i.e., τ_3, acquires the higher-frequency core c_4 among the rest of cores. Moreover, if the

Penalty value	c_1	c_2	c_3	c_4	c_5
CRT-TMR $- \tau_1$	ε	ε	ε	ε	∞
$\phi - \tau_2$	0.10	0.15	0.20	0.25	0.30
$\phi - \tau_3$	0.24	0.26	0.28	0.30	0.32

(a) Tasks reliability penalty on each core.

Mappings	Total penalty
$\tau_1 \rightarrow \text{TMR}(c_3, c_4, c_5), \tau_2 \rightarrow \phi(c_2), \tau_3 \rightarrow \phi(c_1)$	∞
$\tau_1 \rightarrow \text{TMR}(c_1, c_2, c_3), \tau_2 \rightarrow \phi(c_5), \tau_3 \rightarrow \phi(c_4)$	0.6
$\tau_1 \rightarrow \text{TMR}(c_2, c_3, c_4), \tau_2 \rightarrow \phi(c_5), \tau_3 \rightarrow \phi(c_1)$	0.54
$\tau_1 \rightarrow \text{TMR}(c_2, c_3, c_4), \tau_2 \rightarrow \phi(c_1), \tau_3 \rightarrow \phi(c_5)$	0.42

(b) Possible mappings and overall penalty.

Table 6.1: Possible mappings with corresponding reliability penalty values.

allocation of CRT-TMR task τ_1 is not assigned properly, overall penalty of mappings may be even worse in this example, i.e. ∞.

Overall, we can see that the reliability-wise critical task is a suboptimal choice without considering the total benefit of system. Moreover, the miss rate constraint for each task should also be considered to ensure the feasibility of task mapping as well. As a consequence, it is clear that such a task mapping problem requires better strategies, since the straightforward exhaustive search is obviously not feasible in practice with the expected high time complexity. It is not difficult to see that, if the reliability penalty of task increases non-linearly, it may lead to a result which is even worse than the proportional setting of this motivational example.

Why Mixed Redundant Threading is Desired?

Consider that we want to activate CRT-TMR to enable detection and correction for soft errors. In the literature, either SRT [MKR02; KCK+16] or CRT [RM00; VPC02; RKS+14b; KCK+16] is used to provide TMR but these techniques are not combined. This means, the task and the two replicas are either executed sequentially on the same core (SRT) or on three different cores (CRT). The proposed mixture of SRT and CRT, called Mixed Redundant Threading (MRT), can balance the number of used cores and the additional computation as MRT enables TMR with two available cores. As already demonstrated in Figure 1.6, adopting MRT can exploit the available cores more efficiently to improve the reliability under the given time constraints.

Suppose we can only use four homogeneous cores to execute two tasks τ_1 and τ_2 and that the corresponding reliability penalties under different RMT levels for τ_1 and τ_2 are given as shown in Table 6.2a. The penalties of task τ_1 and τ_2 without any redundancy (ϕ) are R_1 and R_2, respectively. The penalty values of τ_1 and τ_2 for SRT are ∞ due to deadline misses caused by the additional computing time. Δ

Penalty Value	ϕ	SRT	MRT	CRT
τ_1	R_1	∞	$\varepsilon + \Delta$	ε
τ_2	R_2	∞	$\varepsilon + \Delta$	ε

(a) Tasks reliability penalty on different levels

Mappings	Total Penalty	Max Penalty
SRT(τ_1) + $\phi(\tau_2)$ and SRT(τ_2) + $\phi(\tau_1)$	∞	∞
$\phi(\tau_1) + \phi(\tau_2)$	$R_1 + R_2$	$\max\{R_1, R_2\}$
CRT(τ_1) + $\phi(\tau_2)$ and CRT(τ_2) + $\phi(\tau_1)$	$R_1 + \varepsilon$ or $R_2 + \varepsilon$	$\max\{R_1 + \varepsilon, R_2 + \varepsilon\}$
MRT(τ_1)+MRT(τ_2)	$2(\varepsilon + \Delta)$	$\varepsilon + \Delta$

(b) Possible mappings and overall penalty

Table 6.2: Corresponding penalty values and possible mappings.

is the reliability penalty induced by MRT and ε is the negligible reliability penalty of CRT-TMR. Typically Δ and ε should be much smaller than R_1 and R_2, i.e., $R \gg \Delta > \varepsilon$, $R \in \{R_1, R_2\}$. As shown in Table 6.2b, the mappings with two SRT-TMRs and two CRT-TMRs are not feasible. The mapping with one CRT-TMR does not provide the optimal reliability penalty under the given reliability penalty metrics, as only one task can be protected by CRT. However, using MRT for both tasks allows them to have redundant executions concurrently, by which the system penalties in both metrics are lower than the CRT mapping, if the reliability penalty Δ is much smaller than R_1 and R_2. Eventually, this example shows that MRT provides an additional option for balancing the usage of cores and for reliability optimization.

6.1.2 Problem Definition

Assume we are given a multi-core processor $\mathbb{C}$ with $\mathbb{M}$ ISA-compatible homogeneous Reduced Instruction Set Computing (RISC) cores, and a set of tasks Γ with multiple versions. In this chapter, we directly use the same objective metric of reliability (-timing) penalty as in [RKS+14b; RTK+13] as a part of the linear combination for functional and timing reliabilities. We first define how we quantify the reliability of tasks, then the studied problems in this chapter are separately defined later on.

To quantify the reliability of tasks, we assume that the reliability penalty of each task level $\tau_{i,j}$ is given as $R_{i,j}$. When consider the performance heterogeneous among different cores, we assume the task reliability is given by a mapping function $R(\tau_{i,k}, c_m)$ that indicates the reliability penalty of task version $\tau_{i,k}$ on core c_m. $R_{i,j}$ and $R(\tau_{i,k}, c_m)$ describe the probability that a fault during the execution of level $\tau_{i,j}$ leads to a visible error when executing (on core c_m). The probability of an error for each instruction is estimated by using Instruction Vulnerability Index and Function Vulnerability Index metrics proposed in [RSK+11; RSH12]. The task vulnerability can

be characterized/estimated by the composition of its instructions. The task level with a lower vulnerability has a smaller reliability penalty, i.e., it has a better reliability.

Reliability Optimization via Perfect Matching

In this problem, we further assume the number of cores M must be greater than or equal to the number of tasks N. Each RISC core has only one single thread. Due to the performance variance, e.g., process variations [HBD+13; RTG+13; HGM12; HM08] and architectural design, each core c_i has its own frequency denoted as f_i. For notational brevity, we index the M cores in a non-decreasing order of current frequencies, i.e., $f_{max} = f_1 \geq f_2 \ldots \geq f_M = f_{min}$.

The studied problem can be divided into two sub-problems:

- **Task Mapping:** Given the redundancy levels θ and the tolerable miss rate constraints ρ_Γ, we consider how to *select the executing version $\tau_{i,k}$ and allocate the cores with corresponding frequency for each task τ_i, so that the overall reliability penalty Ψ_Γ is minimized.* For this sub-problem, we classify the given redundancy levels into two classes and propose two algorithms to minimize the overall reliability penalty in Section 6.2.
- **Redundancy Level Adaptation:** The objective is to *determine the task redundancy levels θ without violating the deadline miss rate.* Without checking all the combinations, we propose an Iterative Level Adaptation (ILA) to efficiently determine the redundancy levels of tasks with our mapping approaches so that the overall reliability penalty is minimized (See Section 6.2.3).

The above approaches are first presented with the assumption that there is no data dependencies and communication among the tasks. Therefore, the considered task mapping only affects the execution time of tasks. After addressing the simpler problem ideally, we consider how to enhance our system model to incorporate the overhead of execution time for the data dependencies and communication in Section 6.2.4.

The objective function is defined in the following:

Definition 12 (Overall Reliability-Timing Penalty Ψ_Γ)**.** The overall reliability penalty of task set Γ, denoted by Ψ_Γ, is given by $\Psi_\Gamma = \sum_{\tau_i \in \Gamma} R(\tau_{i,k}, c_m)$ under the miss rate constraint ρ, where $R(\tau_{i,k}, c_m)$ is the reliability penalty of task version $\tau_{i,k}$ executing on core c_m.

Reliability Optimization with Multi-Tasking

Under the same system setting, in this problem, we alternatively assume that the Worst-Case Execution Time (WCET) of tasks can be given. Without including deadline miss rates, the objective function is thus different. Instead of taking the tolerable deadline miss rate into consideration, it is redefined as follows:

Definition 13 (Overall Reliability Penalty $\Psi_\Gamma(\theta)$)**.** The overall reliability penalty of task set Γ is $\Psi_\Gamma(\theta) = \sum_{\tau_i \in \Gamma} R_{i,\theta_i}$, where R_{i,θ_i} is the reliability penalty of task τ_i

executing at redundancy level θ_i and the set θ contains the redundancy levels θ_i of all tasks $\tau_i \in \Gamma$.

Please note that, the main focus of this work is not the schedulability problem. For a given task set, the objective here is that the system reliability should be increased as much as possible by activating TMR for some (or at best all) tasks. We assume that the activation of TMR for a task τ_i has a different impact on the system reliability for each task, i.e, a reliability penalty is given for each task that is smaller when TMR is activated and larger when TMR is not activated. The total system reliability can be defined by any applicable metric, e.g., the sum of reliability penalty or the maximum reliability penalty, and the goal is to minimize the systems reliability penalty under the given metric.

Please also note that the proposed approaches are applicable to any system reliability metrics, if the optimization can be solved by a dynamic programming algorithm, i.e., $R_{i,j}$ can be set to any reliability penalty.

6.2 Reliability Optimization via Perfect Matching

In this section, we first present our task mapping approaches under the assumption that the redundancy levels for all the tasks are already known beforehand, i.e., θ is given. With given redundancy levels, the task mapping problems can be classified to two different cases, i.e. Homogeneous Redundancy Levels and Heterogeneous Redundancy levels. Afterwards we deal with the problem how to decide the redundancy levels of tasks together with the proposed mapping approaches. Please note that we assume a set of independent tasks first and relax this assumption in Section 6.2.4 later on.

6.2.1 Homogeneous Redundancy Levels

In this section, we show that Hungarian Algorithm (HA) [Kuh55] can be the subroutine of our approach to solve the case that all the tasks require a homogeneous redundancy level in time complexity $O(N^3)$. There exists two cases: either all the tasks are executed in the CRT-TMR level, or none of them requires RMT. We will focus on the former case, and explain how to cope with the latter case at the end of this section.

For the completeness, we link both cases to the well-known Minimum Weight Perfect Bipartite Matching (MWPBM) problem. In the MWPBM problem, there is a bipartite graph $\mathcal{G} = (\mathbb{V}, \mathbb{E})$ with two disjoint subsets $\mathbb{X} \subseteq \mathbb{V}$ and $\mathbb{Y} \subseteq \mathbb{V}$, where $\mathbb{E}$ is the set of edges between $\mathbb{X}$ and $\mathbb{Y}$. Each edge e in $\mathbb{E}$ is associated with a weight $w(e)$. The MWPBM problem is to find out a perfect matching of maximum weight where the weight of matching $\mathbb{M}$ is given by $w(\mathbb{M}) = \sum_{e \in \mathbb{M}} w(e)$. We use the terms $\mathbb{X}$, $\mathbb{Y}$, *and* $w(\mathbb{M})$ to refer to the MWPBM problem; the terms $\mathbb{C}$, Γ, *and* Ψ_Γ to refer to our problem, where $\Psi_\Gamma = \sum_{\tau_i \in \Gamma} R(\tau_{i,k}, c_m)$. The way we use the bipartite graph is defined as the following:

Definition 14 (Bipartite Graph for Task Mapping)**.** To build a bipartite graph $\mathcal{G} = (\mathbb{V}, \mathbb{E})$, we consider the tasks in Γ as the nodes in subset $\mathbb{X}$, and the cores in $\mathbb{C}$ as the nodes in subset $\mathbb{Y}$, by which $\mathbb{X} \cup \mathbb{Y} = \mathbb{V}$, and $\mathbb{X} \cap \mathbb{Y} = \varnothing$. With the version selection table ψ, each weight of edge $e \in \mathbb{E}$ can be referred to a corresponding entry $\psi(\tau_i, c_m)$ which connects two specific nodes, i.e., τ_i in $\mathbb{X}$ and c_m in $\mathbb{Y}$.

CRT-TMR for All Tasks In this case, each task needs three cores to execute in the CRT-TMR level. Although all the tasks have the same demanded number of cores for the redundancy, the way we group and assign the cores will affect their deadline miss rate. We observe that the execution time of each task in the CRT-TMR level relies on the lowest frequency core in its assigned core group. Therefore, to increase the feasibility of core grouping for the following task mapping procedure, an optimal grouping of cores should have the maximal summation of frequencies from each lowest frequency cores among all the groups. The following theorem shows that the optimal core grouping can be obtained by grouping every three cores $\{c_i, c_{i+1}, c_{i+2}\}, \forall i = 1, 4, 7, \ldots, (\delta_\theta - 2)$ consecutively.

Theorem 15 (Optimal Core Grouping)**.** Given a set of cores $\mathbb{C}$ with variation, grouping every three adjacent cores $\{c_i, c_{i+1}, c_{i+2}\}, \forall i = 1, 4, 7, \ldots, (\delta_\theta - 2)$ may obtain the optimal grouping which has the maximal summation of frequencies from each lowest frequency cores among all the groups.

Proof. First of all, it is not difficult to see that the first δ_θ high-frequency cores are definitely used in an optimal solution and formed into N groups, in which each group has 3 cores. Suppose that there is an optimal grouping solution, in which c_1 is in group $\mathbb{G}'_1$, c_2 is in group $\mathbb{G}'_2$, and c_3 is in group $\mathbb{G}'_3$. We only consider the case that $\mathbb{G}'_1 \neq \mathbb{G}'_2 \neq \mathbb{G}'_3$, as the other cases are simpler than this case. Let c_i, c_j, c_k be the lowest-frequency cores in each of these three groups. Without loss of generality, we index these three cores such that $i < j < k$, in which $i \geq 4$ by definition. Therefore, c_k can be in any of the three groups in this index rule and the summation of frequencies among these groups is $f_i + f_j + f_k$.

Now, we (1) swap c_2 in $\mathbb{G}'_2$ and the second fast core in group $\mathbb{G}'_1$ and (2) swap c_3 in $\mathbb{G}'_3$ and the lowest-frequency core in group $\mathbb{G}'_1$. These three groups now are called $\mathbb{G}^*_1, \mathbb{G}^*_2, \mathbb{G}^*_3$. The lowest-frequency core in $\mathbb{G}^*_1$ is c_3, which does not have lower frequency than the c_i. Moreover, after swapping, either the lowest-frequency core in $\mathbb{G}^*_2$, or the lowest-frequency core in $\mathbb{G}^*_3$ is core c_k. So, there are two cases:

- **Case 1**: lowest-frequency core in $\mathbb{G}^*_2$ is c_k: This implies that the lowest-frequency core of $\mathbb{G}'_2$ is also c_k, due to the fact that the swapping procedure does not change the lowest-frequency core in group $\mathbb{G}'_2$. If core c_i is in $\mathbb{G}'_3$ and core c_j is in $\mathbb{G}'_1$, then after swapping the lowest-frequency core of $\mathbb{G}^*_3$ is c_j. Similarly, if core c_i is in $\mathbb{G}'_1$ and core c_j is in $\mathbb{G}'_3$, then after swapping the lowest-frequency core of $\mathbb{G}^*_3$ remains as c_j. We illustrate these two scenarios in Figure 6.1, where the first and second scenarios of the three groups $\mathbb{G}'_1, \mathbb{G}'_2, \mathbb{G}'_3$ are in Figure 6.1a

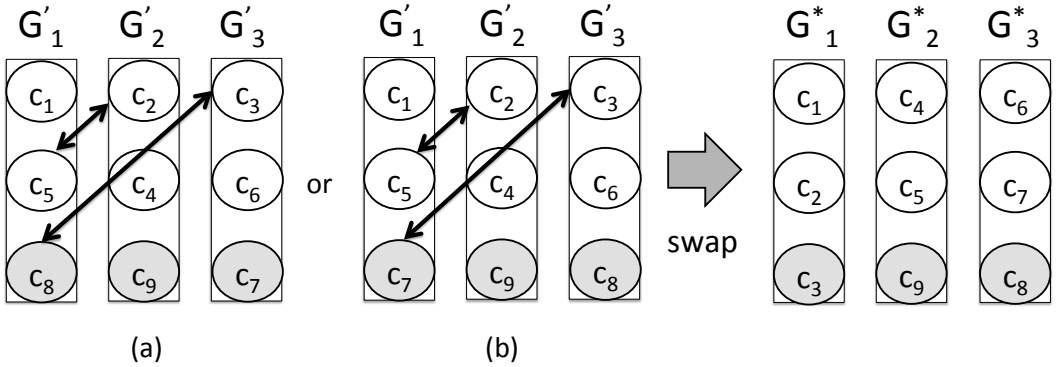


Figure 6.1: Example of two swapping scenarios in case 1 in the proof of Theorem 15. In this example, c_i is c_7, c_j is c_8, and c_k is c_9. After swapping, the frequencies of cores are increased from $f_7 + f_8 + f_9$ to $f_3 + f_8 + f_9$.

and Figure 6.1b, respectively. Therefore, for such a case, we show that the lowest-frequency core in $\mathbb{G}_3^*$ is c_j.

- **Case 2**: lowest-frequency core in $\mathbb{G}_3^*$ is c_k: For such a case, with the similar procedure as in Case 1, we can show that the lowest-frequency core in $\mathbb{G}_2^*$ is c_j.

As a result, the grouping $\mathbb{G}_1^*, \mathbb{G}_2^*, \mathbb{G}_3^*$ has higher total frequency (with respect to the lowest-frequency cores in the three groups) where $f_3 + f_j + f_k < f_i + f_j + f_k$ and $\mathbb{G}_1^* = \{c_1, c_2, c_3\}$. If we continue swapping, we can eliminate all differences between the grouping $\mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_3'$ and the consecutive core grouping $\{c_i, c_{i+1}, c_{i+2}\}, \forall i = 1, 4, 7, \ldots, (\delta_\theta - 2)$ without decreasing the total frequency of the solution. Therefore, we reach the conclusion. □

Based on Theorem 15, the task mapping problem can be transformed equivalently to the MWPBM problem with N tasks and N groups, $\{c_i, c_{i+1}, c_{i+2}\}, \forall i = 1, 4, 7, \ldots, (3N - 2)$. Therefore, we can construct the corresponding bipartite graph and adopt HA as the subroutine to find an optimal assignment with the minimal overall reliability penalty while all the miss rate constraints are satisfied. According to the preprocessing, we can ensure that if a task is not feasible to execute on a certain core, the value of corresponding entry in table ψ must be infinity. With the above setting, it is clear that if Ψ_Γ is infinity, there is no feasible assignment with such an input set of tasks Γ and cores $\mathbb{C}$. *Up to here, it should be also clear how to handle the case when none of the tasks require RMT execution. That is also a perfect matching problem by assigning N tasks to N cores.*

As a result, we summarize our approaches as Algorithm 2 to handle both cases (all tasks are executed in the CRT-TMR level or none of the tasks are protected). If the tasks require the RMT execution, we prepare a core group list List_c^* to record the cores for each group with the optimal grouping of cores by Theorem 15 (line 5). If none of the tasks require RMT execution, we choose the first δ_θ cores from $\mathbb{C}$ and record in List_c^* (line 8). Then, the corresponding bipartite graph $\mathcal{G}$ can be constructed

Algorithm 2 Homogeneous redundancy levels

Input: set of tasks Γ; selected redundancy levels for tasks θ; set of cores $\mathbb{C}$; best versions table ψ;
Output: mapping $\mathbb{M}$ with the set of selected versions;
1: $\text{List}_c^* \leftarrow \varnothing$;
2: **if** all redundancy levels are activated in CRT-TMR **then**
3: **for** $c_j \in \mathbb{C}, j \leftarrow 1, 4, \ldots, \delta_{\theta-2}$ **do**
4: //Assign the core grouping by Theorem 15
5: $\text{List}_c^* \cup \mathbb{G}_j = \{c_j, c_{(j+1)}, c_{(j+2)}\}$;
6: **else if** all redundancy levels are ϕ **then**
7: $\text{List}_c^* \leftarrow \mathbb{C}.\text{head}(\delta_\theta)$;
8: $\mathcal{G} \leftarrow$ build Bipartite Graph with Γ, List_c^*, and ψ;
9: $\mathbb{M} \leftarrow$ find the mapping by HungarianAlgorithm($\mathcal{G}$);
10: **if** Ψ_Γ is ∞ **then**
11: return **FAIL**

by Γ, List_c^*, and ψ by Def. 14. With the bipartite graph $\mathcal{G}$, we can find the minimum weight bipartite perfect matching and assign the tasks and cores with mapping $\mathbb{M}$ by HA and the bipartite graph including the information of possible mappings (line 11). In particular, if the total weight of mapping is infinity, we know that there is no feasible assignment to satisfy the miss rate constraint (lines 12-14).

According to the perfect matching property, the preprocessing, and the definition of $\mathcal{G}$, we can ensure that Algorithm 2 delivers a feasible mapping $\mathbb{M}$ for tasks and cores, where each core only appears once in a specific core group while the miss rate constraint is satisfied. The time complexity is dominated by HA with $2N$ nodes, i.e. $O(N^3)$.

6.2.2 Heterogeneous Redundancy Levels

In this section, we consider each task has an arbitrary redundant level requirement θ_i in the task mapping problem, i.e., $|\Gamma_\phi| \neq n$ and $|\Gamma_{\text{TMR}}| \neq n$. For such a case, the approach in Section 6.2.1 by reducing the assignment problem to the MWPBM problem is no longer applicable, *since the bipartite graph cannot be built due to the unknown properties of core grouping in optimal solutions.*

However, we observe that it is beneficial to assign the cores of CRT-TMR tasks before ϕ tasks, as the CRT-TMR tasks are fully protected with a negligible reliability penalty ε. No matter which cores are assigned to CRT-TMR tasks, their reliability penalty is always negligible. In order to supply more resilient cores in terms of performance for the ϕ tasks, the frequencies of assigned cores for CRT-TMR tasks should be as *low* as possible. Therefore, we propose our approach for this heterogeneous case, which consists of two parts: assigning CRT-TMR tasks and assigning ϕ tasks. Algorithm 3 presents the pseudo code for the two portions of tasks assignment with heterogeneous redundancy levels.

Assigning CRT-TMR tasks First of all, s_i denotes the resilience of CRT-TMR task by the *lowest acceptable core frequency* of task τ_i as Eq.(6.1) (line 4 in Algorithm 3)

$$s_i = \arg_{1 \le j \le \ \delta_\theta} \{\psi(\tau_i, c_j) \neq \infty, \text{and } \psi(\tau_i, c_{j+1}) = \infty\}, \tag{6.1}$$

where $\psi(\tau_i, c_{\delta_\theta+1})$ is set to a dummy version with ∞ penalty for notational brevity. To maximize the number of tasks satisfying their deadline, the assignment of CRT-TMR tasks should start from the most resilient task, which accepts the lowest frequency core among all the redundant cores.

To find out the most resilient tasks, we sort all the CRT-TMR tasks by a non-increasing order of c_{s_i}'s speed, and re-index them by the sorted list, in which ties are broken arbitrarily (line 6). The assigning procedure starts from the most resilient task τ_k, which has the maximum index s_k of cores in $\mathbb{C}$, with the lowest-frequency group $\mathbb{G}_k$, where $\mathbb{G}_k = \{c_{s_k-2}, c_{s_k-1}, c_{s_k} | s_k \geq 3\}$ (line 8). Then, we exclude the cores of $\mathbb{G}_k$ from $\mathbb{C}$ and consider the next resilience-wise task τ_{k-1} (line 9). For task τ_i, s_i should be the lowest frequency core among the rest of cores, if the original s_i is assigned to the task already (lines 11-13). By repeating the above procedure, the frequencies of assigned cores will be as low as possible which satisfies the minimal requirement of core frequency for each CRT-TMR task.

Assigning ϕ tasks After assigning the CRT-TMR tasks, the rest of cores and ϕ tasks can be transformed to MWPBM problem as Section 6.2.1. As a result, we can make a bipartite graph $\mathcal{G}$ (line 18) by Def. 14 and finish a perfect matching $\mathbb{M}$ by HA with the minimal Ψ_Γ as the optimal result (line 19). If the procedure cannot find a feasible mapping, the algorithm returns that there is no feasible solution (lines 20-22). With the CRT-TMR tasks assignment and the perfect matching property, we can ensure that the mapping assignment $\mathbb{M}$ derived by Algorithm 3 is feasible for tasks and cores, where each core is only assigned to one unique task while all the miss rate constraints in ρ_Γ are satisfied. The time complexity is similar as Algorithm 2, which is dominated by the HA, i.e., $O(N^3)$.

The solution derived from Algorithm 3 can be proved to be *optimal* in terms of overall reliability penalty if there exists a feasible solution for the input. If we try to handle TMR and ϕ tasks concurrently, there is no efficient way to decide the core grouping beforehand. However, as the reliability penalty of TMR tasks are negligible, we can reach the optimality as shown in the following theorem.

Theorem 16 (Optimality of Algorithm 3)**.** Given a set of cores $\mathbb{C}$, a set of tasks Γ, heterogeneous redundancy levels of tasks θ, and tolerable deadline miss rates ρ_Γ. Algorithm 3 provides a feasible task mapping with the minimal overall reliability penalty under heterogeneous redundancy levels θ.

Algorithm 3 Heterogeneous redundancy levels

Input: set of tasks Γ; selected redundancy levels for tasks θ; set of cores $\mathbb{C}$; best versions table ψ;
Output: mapping $\mathbb{M}$ with the set of selected versions;

```
//assigning TMR tasks
List*_c ← ℂ; List_L ← Γ_TMR;
for each τ_i ∈ List_L do
    find out s_i based on Eq.(6.1);
sort List_L by s_i and re-index them;
//τ_k has the maximum index as s_k
𝕄 ← assign 𝔾_k = {c_(s_k−2), c_(s_k−1), c_sk | s_k ≥ 3} to τ_k;
remove the cores of 𝔾_k from List*_c;
for each τ_i ∈ List_L, i = (k−1), (k−2), ..., 1 do
    if s_i > s_(i+1) − 3 then
        s_i ← s_(i+1) − 3;
    𝕄 ← assign τ_i with 𝔾_i = {c_(s_i−2), c_(s_i−1), c_si};
    remove the cores of 𝔾_i from List*_c;
//assigning φ tasks
𝒢 ← build Bipartite Graph with Γ_φ, List*_c, and ψ;
𝕄 ← find the mapping by HungarianAlgorithm(𝒢);
if Ψ_Γ is ∞ then
    return FAIL;
```

Proof. As the reliability penalty of the CRT-TMR level is assumed to be a negligible value ε in our model, the overall reliability Ψ_Γ under heterogeneous redundancy levels can be reformulated from Def. 12 to:

$$\Psi_\Gamma = \sum_{\tau_i \in \Gamma_\phi} R(\tau_{i,k}, c_m), c_m \in \mathbb{C}. \tag{6.2}$$

According to Eq.(6.2), we can observe that the overall reliability Ψ_Γ fully relies on the frequencies of assigned cores for ϕ tasks, if all CRT-TMR tasks are feasible to execute with their assigned cores. Therefore, we know that the derived task mapping will be an optimal mapping if the assigned cores of ϕ tasks have the maximal summation of frequencies to obtain the minimal overall reliability penalty.

As the candidate cores for ϕ tasks are the remaining cores after mapping the CRT-TMR tasks, the assigned cores for CRT-TMR tasks must have the lowest summation frequencies (for those lowest frequency cores in each group) to let the remaining cores have the maximal summation frequencies. The core grouping in Algorithm 3 groups every three adjacent low-frequency cores, which guarantees the optimal feasibility for CRT-TMR tasks by Theorem 15 and the lowest frequencies among the candidate cores. In the following, we will prove that the assignment for CRT-TMR tasks in Algorithm 3 based on the above grouping which can find out an optimal mapping such that the rest of cores for ϕ tasks have the maximal summation of frequencies.

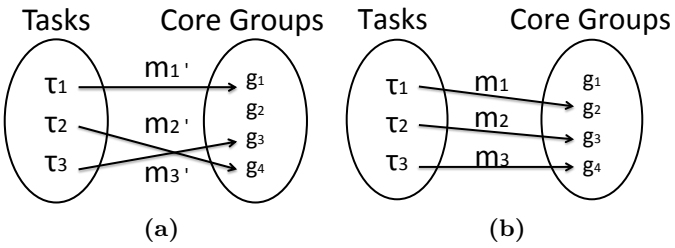


Figure 6.2: Example of task mappings for RMT tasks and core groups in the proof of Theorem 16, in which (a) is an optimal solution and (b) is derived by Algorithm 3.

Assume there is an optimal task mapping between CRT-TMR tasks and core groups as Figure 6.2(a), and the mapping Figure 6.2(b) is the result of Algorithm 3, in which the core groups are sorted by their lowest frequency core, i.e., $g_1 > g_2 > g_3 > g_4$. Then there are two cases:

- **Case 1:** There are two consecutive mappings in a different order in Figure 6.2(a) than they are in Figure 6.2(b): For such a case, we swap the order for these two consecutive mappings, i.e., m'_2 and m'_3, and they become m_2 and m_3. After the swapping procedure, the total frequency of remaining cores will be the same, as the assigned cores for CRT-TMR tasks are not changed.
- **Case 2:** There is an element of Figure 6.2(a) not in Figure 6.2(b) and an element of Figure 6.2(b) not in Figure 6.2(a): We swap g_1 and g_2 for the element of mapping m'_1, and now m'_1 becomes m_1. As the rest of the cores are changed from group g_1 to g_2, we know that the total frequency of rest of the cores is not less than before i.e, $g_2 < g_1$.

As a result, the swapping procedure shows that the derived mapping is no worse than before. The differences between Figure 6.2(a) and Figure 6.2(b) are eliminated without worsening the total frequency of the solution. We know that the derived mapping is as good as any optimal solution in which the rest of cores have the maximal total frequency. As the optimality of HA has been proved in [Kuh55], the derived task mapping must be optimal with the minimal overall reliability penalty. As a consequence, we reach the conclusion that the derived task mapping by Algorithm 3 is optimal. □

6.2.3 Redundancy Level Adaption

Until now, the assumption was that the redundancy levels of tasks θ are given. In this section, we present an ILA which determines the redundancy levels of tasks with the proposed mapping approaches in Section 6.2.2.

To minimize the overall reliability penalty, it is beneficial to execute as many tasks as possible in the CRT-TMR level. However, which tasks should execute in the

CRT-TMR level is not that trivial to determine. Some tasks may suffer from their higher vulnerability, whereas some of tasks may suffer from their tighter deadline miss rate.

Intuitively, deploying the CRT-TMR execution for the task with the highest reliability penalty is a reasonable way to decrease the overall reliability penalty as the greedy approach in [RKS+14b]. However, the task with the "highest reliability penalty" is only relative to a specific core frequency, e.g., on the highest frequency core. If we greedily execute this task in the CRT-TMR level, all the possible task mappings for the rest of tasks may even lead to an inferior overall system reliability. In addition, we are not able to know how a task is vulnerable under the core grouping, as the core grouping for the CRT-TMR execution is still unknown at this moment. Since checking all the combinations of redundancy levels may not be possible, here we propose an iterative approach exploiting our task mapping approaches as the subroutine to guarantee the feasibility and efficiency of redundancy levels.

Algorithm 4 presents the pseudo-code of level adaptation. It first adopts Theorem 15 to find out the mapping between the tasks and cores for the initial case that none of the tasks requires the CRT-TMR level, which helps us find out a reasonable reference to upgrade the redundancy levels (lines 2-3). Then, the following procedure is repeated until there is no more improvement. At first, we consider the task with the maximal reliability penalty in the current mapping solution (line 7). The objective is to upgrade one more task from level ϕ to CRT-TMR with the current solution for the reliability improvement. For this task, we greedily upgrade its redundancy level by picking up two unused cores that can satisfy the miss rate constraint of the task. If the upgrade is feasible, we can adopt Algorithm 3 to find out the next mapping $\mathbb{M}$ (lines 9-10); otherwise, we rollback the infeasible upgrade (line 12). Then, we continue the procedure finding the next high penalty task for upgrading from level ϕ to CRT-TMR (line 14). When there is no more improvement and all the tasks are considered, we can terminate the iterative procedure.

Algorithm 4 may deliver a feasible mapping $\mathbb{M}$ and minimize the system reliability penalty Ψ_Γ with as many as possible CRT-TMR tasks. The time complexity is only scaled by the number of tasks N, which is still applicable to be used online.

6.2.4 Communication and Data Dependency

In this section, we present how to deal with the communication and data dependency among the tasks when we considering the task mapping problem on a multi-core processor. We consider the communication fabric with the most popular deterministic routing algorithm, i.e., XY routing (proven to be deadlock-free) [MB06], on the most common topology, i.e., 2-Dimension (2D) mesh. Since the assumed multi-core processor only has a single thread per core, to parallelize the execution of dependent tasks and utilize all the redundant cores concurrently, one way is to adopt the well-known technique, i.e., software pipelining, to dispatch the dependencies into different pipeline stages, where the dependent inputs of tasks can be transformed by the

Algorithm 4 Levels adaptation and task mapping

Input: set of tasks Γ; set of cores $\mathbb{C}$; best versions table ψ;
Output: mapping $\mathbb{M}$ with the set of selected versions;
1: //mapping the first case
2: Vector$_\theta$ ← assign all redundancy levels of tasks as ϕ;
3: $\mathbb{M}$ ← apply Algorithm 2 with Γ and $\mathbb{C}$ to find out the mapping;
4: **if** Ψ_Γ is ∞ **then**
5: return **FAIL**;
6: find out task τ_h with the highest penalty in mapping $\mathbb{M}$;
7: **for each** $\tau_h, \tau_h \in \Gamma_\phi$ **do**
8: θ_h ← assign the redundancy level of task τ_h as TMR;
9: $\mathbb{M}$ ← apply Algorithm 3 to find out the task mapping;
10: **if** Ψ_Γ is ∞ **then**
11: restore θ_h to ϕ
12: check the next τ_h in mapping $\mathbb{M}$;

predecessors before the execution of next pipeline stage. With the software pipelining, we can consider the task mapping with the data dependencies on all the redundant cores concurrently. We assume that the communication overhead is significantly less than the computation overhead, so that the system reliability may not be dramatically changed.

For the given task graph $\mathbb{G}$, we prepare π_i to denote whether task τ_i has a predecessor: π_i is equal to 1 if task τ_i has a predecessor; otherwise, π_i is 0. Although all the dependent tasks can execute at the same time with software pipeline, the communication of dependent pipelines has to be considered with the allocation of assigned cores. As shown in Figure 6.3(a), the different allocation of assigned cores, may lead to a different communication distance. In addition, the CRT-TMR execution also has internal communication among the redundant threads for the majority-voting mechanism. As shown in Figure 6.3(b), the allocation of cores dedicated for CRT-TMR executions has to be considered for avoiding any unnecessary performance penalty.

In general, the communication overhead can be estimated by considering the data size, the required cycles per hop, and the distance of communication. However, the realistic distance of communications can only be calculated after the allocation of the tasks and cores is done. To mitigate the uncertainty, we propose to estimate the overhead with the maximal distance on the 2D mesh to cover the worst execution scenario. As shown in Figure 6.4, if a task is assigned to core c_i in a 3×3 mesh topology, the maximal distance will be 4 hops as c_i to c_k. If it is assigned to the pipeline on core c_j, the maximal distance will be 2 hops as c_j to c_k. Therefore, by applying XY routing, the maximal distance of communication on core c_m, denoted by q_m can be calculated statically.

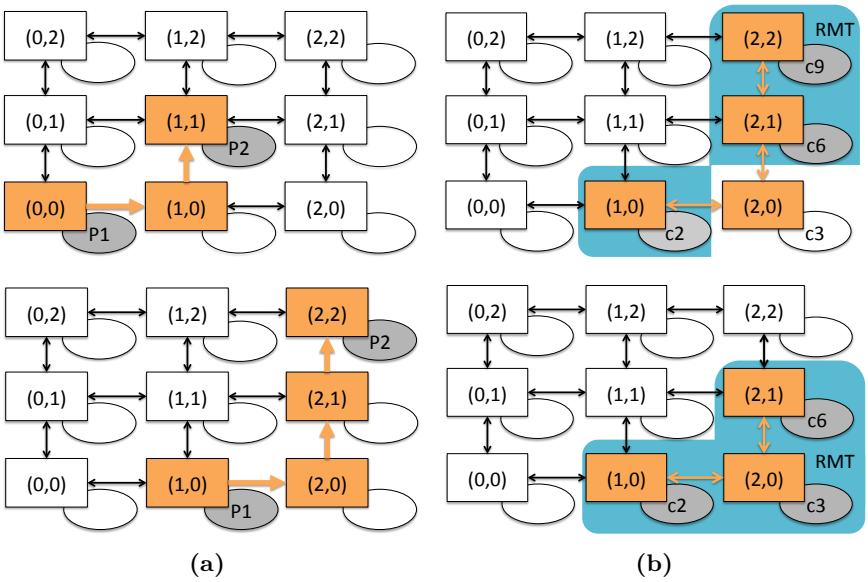


Figure 6.3: The communication on 2D mesh topology with XY routing. In (a), pipelines P_1 and P_2 have the communication. In (b), RMT adopts $\{c_2, c_6, c_9\}$ and $\{c_2, c_3, c_6\}$, respectively.

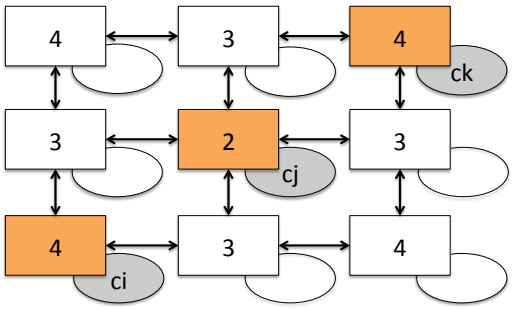


Figure 6.4: Example of the maximal distance/hops estimation in a 3x3 mesh. The maximal distance will be 4 if the task is assigned to c_i. It can be bounded by 2, if it assigned to c_j.

For the communication of dependent pipelines, the internal communication time overhead can be estimated by the input data size of task and the maximal distance q_m of assigned core c_m of pipeline as the following:

$$\Delta_{\text{in}}(\tau_i, c_m) = q_m \times inputData\_size(\tau_i) \tag{6.3}$$

Similarly, since the majority-voting mechanism has to wait for all the output data of the redundant threads, the internal communication in RMT can be estimated by

the output data size of task and the maximal distance q_m of assigned core c_m as the following:

$$\Delta_{\text{out}}(\tau_i, c_m) = q_m \times outputData\_size(\tau_i) \tag{6.4}$$

Assume the data transfer spends μ cycles per hop. Since the communications prolong the execution time of tasks, the interval between the release and deadline is reduced by all the possible communications as the following:

$$D'_i = D_i - \mu \cdot (\theta_i \cdot \Delta_{\text{out}}(\tau_i, c_m) + \pi_i \cdot \Delta_{\text{in}}(\tau_i, c_m)) \tag{6.5}$$

As a consequence, the deadline miss rate of task as Eq.(3.1) with the pipeline on core c_m should be reformulated as Eq.(6.6):

$$P_{\text{dm}}(\tau_{i,k}, c_m) = 1 - C_{i,k,m}(D'_i) \tag{6.6}$$

With the reformulated deadline miss rate Eq.(6.6), we can incorporate the communication overhead into our proposed approaches. Please note that the applicability is not limited to XY routing. The approximation can be easily extended for the other deterministic routing algorithms by changing Eq.(6.3) and (6.4) accordingly.

If none of the tasks requires RMT execution, Algorithm 2 is still optimal to the task mapping with the data dependency due to the optimality of Theorem 15 and HA for the worst case. However, if the compatibility of cores is arbitrary to each task, Theorem 15 can not hold any more, in which the feasibility and frequencies do not have the absolute relation. Even if a core has the highest frequency among the others, it may not be suitable for the task which has a significant communication overhead. As the cores' positions also affect the feasibility of mappings, it is not good enough to determine the assignment sequence only by the frequencies of cores.

Algorithm 5 takes the above issues into consideration and solves the task mapping problem with the communication and data dependencies. At first, we have to reformulate the deadline miss rate of task versions in the best versions table ψ with Eq.(6.6) (line 2), which can be done in the preprocessing. To present the impact of communication, we denote the number of available cores for task τ_i as a_i, which can be obtained by calculating the number of feasible entries in the reformulated best version table ψ (line 3). Then, we sort the tasks with the corresponding a_i by a non-increasing order. (line 4). The assignment starts from the task with the minimal number of available cores and assign three lower frequency cores among the available cores (lines 5-12). If the task cannot be satisfied by the remaining available cores, it is clear that there is no further feasible solution (lines 9-11). By checking all the RMT tasks, all the assigned cores and TMR tasks are excluded as the procedure in Section 6.2.2. As the rest of tasks are only ϕ tasks in Γ, we follow the same procedure as Algorithm 3 to build up the bipartite graph and find out a perfect matching $\mathbb{M}$ by HA with the minimal Ψ_Γ (lines 14-18). Please note that, in case of dependent tasks where the predecessor output has soft errors, we assume that the errors can be recovered by task re-execution before it is served to its dependent task, which is similar to *dTune* [RKS+14b].

Algorithm 5 Task mapping with data dependency

Input: set of tasks Γ; set of cores $\mathbb{C}$; best versions table ψ;
Output: mapping $\mathbb{M}$ with the set of selected versions;
1: $\text{List}_L \leftarrow \Gamma_{\text{TMR}}$, $\text{List}_c \leftarrow \mathbb{C}$;
2: reformulate best versions table ψ with Eq.(6.6);
3: calculate the number of available cores a_i for Γ_{TMR};
4: sort and re-index List_L by a_i;
5: **for each** $\tau_i \in \text{List}_L, i = 1, 2, \ldots, k$ **do**
6: //τ_k has the minimal number of available cores
7: assign three lower frequency cores to τ_i in List_c;
8: remove the assigned cores from List_c;
9: **if** τ_i is not able to activate RMT **then**
10: return **FAIL**;
11: //assigning ϕ tasks
12: $\mathcal{G} \leftarrow$ build Bipartite Graph with Γ_ϕ, List_c, and ψ;
13: $\mathbb{M} \leftarrow$ find the mapping by HungarianAlgorithm($\mathcal{G}$);
14: **if** Ψ_Γ is ∞ **then**
15: return **FAIL**;

Consequently, we can find out a reasonable task mapping $\mathbb{M}$ by using Algorithm 5 and the reformulated versions table ψ. The time complexity is the same as Algorithm 3, i.e., $O(N^3)$. The evaluation of the proposed approaches in this Section 6.2 is in Section 6.4.1

6.3 Reliability Optimization with Multi-Tasking

In this section, we first give a short, general overview of Federated Scheduling *(FS)* [LCA+14] that includes a short example, which is used to tackle the studied problem. Afterwards, we present several dynamic programming algorithms to optimize the system reliability while guaranteeing the system schedulability. The evaluation for the proposed approaches in this section is presented in Section 6.4.2 eventually.

6.3.1 Federated Scheduling

In FS, the tasks in Γ are partitioned into subsets that are scheduled individually on a multi-core system with M homogeneous cores. To simplify the presentation, we here assume that the execution levels of all tasks have been determined, i.e., θ_i is given $\forall \tau_i \in \Gamma$. Obviously a schedule for Γ on M uniform frequency cores can only exist, if the following two necessary conditions are met:

- The sum of all task utilizations is not greater than the number of processors, i.e., $\sum_{\tau_i \in \Gamma} u_{i,\theta_i} \leq M$.
- No task has a critical path greater than its period, i.e., $\forall \tau_i \in \Gamma$, $L_{i,\theta_i} \leq T_i = D_i$.

Federated Scheduling [LCA+14] partitions the tasks into two disjoint subsets according to their utilization: τ_{BIG} contains all high-utilization tasks, i.e., $u_{i,j} \geq 1$, and τ_{LITTLE} contains all low-utilization tasks, i.e., $u_{i,j} < 1$. We denote the executing levels of tasks in τ_{BIG} with θ_{BIG} and the executing redundancy levels of tasks in τ_{LITTLE} as θ_{LITTLE}. First, the number of cores necessary to schedule τ_{BIG} is determined while τ_{LITTLE} will be scheduled on the remaining cores if possible. Please note that we present *FS* in the general case here. Some remarks regarding our studied problem and some properties that arise due to the structure of that problem are given in the next section.

- **High-utilization tasks (τ_i in τ_{BIG}):** For each task $\tau_i \in \tau_{\text{BIG}}$, the parallel sub-tasks are scheduled on cores dedicated to the task, called list scheduling in the literature [Gra66], by any work-conserving parallel scheduler. A work-conserving list scheduler is a scheduler that never lets a core idle if there is any sub-task ready to be executed. As shown in Theorem 2 in [LCA+14], the required number of dedicated cores H_{i,θ_i} for τ_{i,θ_i} is at most:

$$H_{i,\theta_i} = \left\lceil \frac{C_{i,\theta_i} - L_{i,\theta_i}}{T_i - L_{i,\theta_i}} \right\rceil \tag{6.7}$$

  For the task set τ_{BIG}, we denote the sum of dedicated cores as $H_{\text{BIG}} = \sum_{\tau_i \in \tau_{\text{BIG}}} H_{i,\theta_i}$.
- **Low-utilization tasks (τ_i in τ_{LITTLE}):** We adopt R-BOUND-MP-NFR, developed by Andersson et al. [AJ03], to schedule the tasks in τ_{LITTLE} on the number of remaining cores $H_{\text{LITTLE}} = M - H_{\text{BIG}}$. On each core of H_{LITTLE} Rate-Monotonic (RM) priority assignment is used. According to Theorem 7 in [AJ03], R-BOUND-MP-NFR feasibly schedules the tasks in τ_{LITTLE} using partitioned scheduling if

$$\sum_{\tau_j \in \tau_{\text{LITTLE}}} u_{j,\theta_j} \leq H_{\text{LITTLE}}/2 \tag{6.8}$$

This directly leads to the following sufficient schedulability test. A prove is therefore omitted.

Lemma 6. FS can schedule the tasks $\tau_{\text{BIG}} \cup \tau_{\text{LITTLE}}$ in Γ on M cores, if the following condition holds:

$$\sum_{\tau_i \in \tau_{\text{BIG}}} H_{i,\theta_i} + \sum_{\tau_j \in \tau_{\text{LITTLE}}} u_{j,\theta_j} \cdot 2 \leq M \tag{6.9}$$

Example 5 (Example for Federated Scheduling)**.** We assume the redundancy level of the tasks to be given and thus drop the level indexes for C_i and L_i. Consider five tasks to be scheduled on 7 processors using FS as follows:

- $(T_1 = 10, C_1 = 12, L_1 = 5)$ and $u_1 = 1.2$
- $(T_2 = 2, C_2 = 4, L_2 = 1)$ and $u_2 = 2$
- $(T_3 = 20, C_3 = 2, L_3 = 1)$ and $u_3 = 0.1$

- $(T_4 = 30, C_4 = 6, L_4 = 3)$ and $u_4 = 0.2$
- $(T_5 = 12, C_5 = 6, L_5 = 4)$ and $u_5 = 0.5$.

In FS the tasks are first classified into τ_{BIG} and τ_{LITTLE} according to their utilization. Therefore, $\tau_{\text{BIG}} = \{\tau_1, \tau_2\}$ and $\tau_{\text{LITTLE}} = \{\tau_3, \tau_4, \tau_5\}$. We determine the number of dedicated cores for each task $\tau_i \in \tau_{\text{BIG}}$, i.e., $H_1 = \left\lceil \frac{12-5}{10-5} \right\rceil = 2$ and $H_2 = \left\lceil \frac{4-1}{2-1} \right\rceil = 3$.

Since $H_{\text{BIG}} = 2 + 3 = 5$, $H_{\text{LITTLE}} = M - H_{\text{BIG}} = 7 - 5 = 2$. According to Theorem 7 in [AJ03], the tasks in τ_{LITTLE} are schedulable as Eq. (6.8) holds, i.e., $(0.1 + 0.2 + 0.5) \leq 2/2$. The tasks are sorted in an ascending order of their periods, i.e., $\{\tau_5, \tau_3, \tau_4\}$, and assigned to the remaining 2 cores in this order according to R-BOUND-MP-NFR. As a result, τ_5 is assigned to core 1, and $\{\tau_3, \tau_4\}$ are assigned to core 2.

6.3.2 Redundancy Level Selection

In this section we show how the overall reliability penalty can be optimized, assuming that we can choose from different given redundancy levels for each task. Obviously it is possible to determine the optimal selection of redundancy levels by checking all possible combinations of redundancy levels and core assignments, and choosing the combination that yields the minimum reliability penalty while satisfying the timing constraints. However this straightforward method has exponential time complexity. Instead, we propose a dynamic programming algorithm to determine the optimal redundancy level for each task while satisfying the feasibility under FS. We start by calculating the possible solution for $\{\tau_1\}$, use those results to calculate the possible solution for $\{\tau_1, \tau_2\}$, use those results to calculate the possible solution for $\{\tau_1, \tau_2, \tau_3\}$ and so on until we calculate the possible solution for $\{\tau_1, \tau_2, \ldots, \tau_N\}$. From the results for $\{\tau_1, \tau_2, \ldots, \tau_N\}$ we choose the one with the minimum reliability penalty.

When calculating possible solutions for $\{\tau_1, \tau_2, \ldots, \tau_i\}$, those solutions depend on:

- The selection of the redundancy levels θ_j for the tasks $\tau_j \in \{\tau_1, \tau_2, \ldots, \tau_{i-1}\}$.
- The number of total required cores $m = \sum_{j=1}^{i-1} H_{j,\theta_j}$ for $\tau_j \in \{\tau_1, \tau_2, \ldots, \tau_{i-1}\}$ where $u_{j,\theta_j} \geq 1$, i.e., τ_j that are in H_{BIG} for there selected redundancy level θ_j.
- The sum of utilizations $k = \sum_{j=1}^{i-1} u_{j,\theta_j}$ for tasks $\tau_j \in \{\tau_1, \tau_2, \ldots, \tau_{i-1}\}$ with $u_{j,\theta_j} < 1$, i.e., τ_j that are in H_{LITTLE} for there selected redundancy level θ_j.
- The chosen redundancy level θ_i and the resulting increase of either m or k.

The necessary values are stored in two 3-dimensional tables G and j^* to record the sub-optimal reliability values and the selected redundancy levels, respectively. This means, $G(i, m, k)$ stores the minimum reliability penalty for the first i tasks, using m cores for tasks in H_{BIG} and with a total utilization of k for tasks in H_{LITTLE}, while $j^*(i, m, k)$ stores the selected redundancy level for task i. Using these values, the chosen redundancy levels for all other tasks can be traced back step by step. Those calculations have to be done for all possible combinations of m and k, i.e., m is an integer with $0 \leq m \leq M$ and k is in the range of $[0, 0.5 \cdot M]$.

When building $G(i,m,k)$ and $j^*(i,m,k)$, we assume $u_{i,j}$ and $H_{i,j}$ to be given for all redundancy levels. If this is not the case, they can be calculated in a preprocessing step.

In the initial step, i.e., when we only consider τ_1, for given values of m and k we calculate $j^*(1,m,k)$ as

$$j^*(1,m,k) = \arg\min_{j\in\{1,2,\ldots,K_1\}} \begin{cases} R_{1,j} & \text{if } u_{1,j} \geq 1 \text{ and } m \geq H_{1,j} \\ R_{1,j} & \text{if } u_{1,j} < 1 \text{ and } k \geq u_{1,j} \\ \infty & otherwise \end{cases} \tag{6.10}$$

leading to minimum reliability penalties of

$$G(1,m,k) = R_{1,j^*(1,m,k)}. \tag{6.11}$$

For the following steps we calculate the values of $G(i,m,k)$ assuming $G(i-1,m,k)$ to be given, where $i = 2,3,\ldots,N$. This means, when we select level j for τ_i, we know that the minimum reliability penalty for task $\tau_1,\tau_2,\ldots,\tau_{i-1}$ has been calculated and stored in

- $G(i-1,m-H_{i,j},k)$ when $u_{i,j} \geq 1$, or
- $G(i-1,m,k-u_{i,j})$ when $u_{i,j} < 1$.

Let $P_j(i,m,k)$ be the resulting reliability penalty for the selection of level j for task τ_i, defined as:

$$P_j(i,m,k) \begin{cases} R_{i,j} + G(i-1,m-H_{i,j},k) & \text{if } u_{i,j} \geq 1 \text{ and } m \geq H_{i,j} \\ R_{i,j} + G(i-1,m,k-u_{i,j}) & \text{if } u_{i,j} < 1 \text{ and } k \geq u_{i,j} \\ \infty & otherwise \end{cases} \tag{6.12}$$

Suppose that j_i^* is the j which minimizes $P_j(i,m,k)$ for given values of m and k. This means we know that

$$G(i,m,k) = P_{j_i^*}(i,m,k) \tag{6.13}$$

and $j^*(i,m,k)$ is j_i^*. This calculations have to be done for all $i = 2,3,\ldots,n$, all integer m with $0 \leq m \leq M$, and all utilization values k in the range of $[0, 0.5 \cdot M]$.

The pseudo-code of the presented level selection can be found in Algorithm 6, using a scaling factor ω for the utilization values, i.e., the third dimension of the table. This is necessary to upper bound the number of entries in the 3-dimensional tables and thus bounding the time needed to construct those tables. Obviously, the number of values we have to consider for the first dimension is the number of tasks N while for the second dimension we only have to consider up to M integer values. Note, that the maximum number of dedicated cores is bounded by 3 for each task.[2] However,

[2] In the general case (explained in Section 6.3.1) the number of cores needed for the execution of a single task can be arbitrary large, depending on the relation of $C_{i,\theta_i} - L_{i,\theta_i}$ to $T_i - L_{i,\theta_i}$. However, the number of processors needed to activate CRT-TMR is bounded by 3 due to the assumption that $L_{i,\theta_i} \leq T_i$ and at most 3 instances are executed in parallel. If the fine-grained selection in Section 6.3.3 is used, still none of the sequential stages will be executed more then 3 times in parallel and thus the bound of 3 still holds.

Algorithm 6 Offline table construction

Input: N tasks, M cores, ω scale unit
Output: j^* level selection table

for $m \leftarrow 0, \ldots, M$ **do**
  for $k \leftarrow 0, \ldots, \left\lceil \frac{0.5M}{\omega} \right\rceil$ **do**
    if $m + 2k \cdot \omega > M$ **then**
      $j^*(1, m, k) \leftarrow \infty$
      $G(1, m, k) \leftarrow \infty$
    else
      calculate $j^*(1, m, k)$ and $G(1, m, k)$ by using Equations ((6.10)) and ((6.11))
for $i \leftarrow 2, 3, \ldots, N$ **do**
  for $m \leftarrow 0, \ldots, M$ **do**
    for $k \leftarrow 0, \ldots, \left\lceil \frac{0.5M}{\omega} \right\rceil$ **do**
      if $m + 2k \cdot \omega > M$ **then**
        $j^*(i, m, k) \leftarrow \infty$
        $G(i, m, k) \leftarrow \infty$
      else
        for each $j \in$ possible redundancy levels **do**
          calculate P_j by using Equation ((6.12))
        $j^*(i, m, k) \leftarrow \arg\min_{j=1,2,\ldots,K_i} P_j$
        $G(i, m, k) \leftarrow P_{j^*(i,m,k)}$

if the utilization values are not discretized, we would have to consider an infinity number of values for the third dimension. Therefore, we discretize all utilization values based on a scale unit ω, i.e., $0 < \omega \leq 1$ and all values of $u_{i,j}$ are replaced with $u_{i,j}/\omega$. Under the assumption that all scaled utilization values u_i/ω are integers, our dynamic programming approaches in Section 6.3.2 and Section 6.3.3 find the solution with the minimized reliability penalty that is possible when Federated Scheduling [LCA+14] is used and the schedulability of the task set under a selection of execution levels is tested based on the sufficient schedulability test in Lemma 6.

Theorem 17. Let ω with $0 < \omega \leq 1$ be given and let $\frac{u_{i,j}}{\omega}$ be an integer $\forall u_{i,j}$. For all $i \in \{1, 2, \ldots, N\}$, $m \in \{1, 2, \ldots, M\}$, and $k \in \left\{1, 2, \ldots, \frac{0.5M}{\omega}\right\}$, Eq. (6.10), Eq. (6.11), Eq. (6.12), and Eq. (6.13) compute the optimal task redundancy level selection $j^*(i, m, k)$ and the optimal solution overall reliability penalty $G(i, m, k)$ achievable under FS when the sufficient schedulability test in Lemma 6 is used.

Proof. This can be proved using mathematical induction:

Base case ($i = 1$): Eq. (6.10) calculates the optimal level for each m and k and Eq. (6.11) calculates the resulting minimal reliability penalty, stored in $j^*(1, m, k)$ and $G(1, m, k)$, respectively. Thus $G(1, m, k)$ and $j^*(1, m, k)$ are optimal.

Inductive step ($i \geq 2$): Assume that $G(i-1, m, k)$ and $j^*(i-1, m, k)$ are optimal for the sub-problem considering the first $i - 1$ tasks for all values of $m \leq M$ and k, i.e., $G(i-1, m, k)$ stores the minimal reliability penalty value and the selected version j^* of τ_{i-1} is stored in $j^*(i-1, m, k)$ for each m and k. Suppose for contradiction

that $G(i,m,k)$ is not optimal. This means, that at least one of the level selections for $\tau_1, \ldots, \tau_i$ is not optimal. For each version of τ_i the penalty $P_j(i,m,k)$ for a given m and k can be calculated by adding the reliability penalty $R_{i,j}$ to either $G(i-1, m-H_{i,j}, k)$ for a version where $u_{i,j} \geq 1$ or $G(i-1,m,k-u_{i,j})$ if $k \geq u_{i,j}$. If the version is not applicable, i.e., $m < H_{i,j}$ for versions with $u_{i,j} \geq 1$ or $k < u_{i,j}/\omega$ for versions with $u_{i,j} < 1$, $P_j(i,m,k)$ is set to ∞. As we take the version j^* that minimizes $P_j(i,m,k)$ we know that $G(i,m,k)$ and $j^*(i,m,k)$ are calculated correctly based on $G(i-1,m,k)$ and $j^*(i-1,m,k)$. Therefore, if $G(i,m,k)$ or $j^*(i,m,k)$ is wrong for any combination of m and k, at least one of the previously selected $i-1$ task levels is not optimal, which contradicts the induction hypothesis. □

After the tables G and j^* are calculated, the minimum value stored in $G(N,m,k)$ is the minimum penalty value. We denote this position by m_N^* and k_N^*. The redundancy level of θ_N can be found in the related entry of table j^*, i.e., at $j(N, m_N^*, k_N^*)$. From this value, we can easily trace back the redundancy levels selected for $\tau_{N-1}, \tau_{N-2}, \ldots, \tau_1$ iteratively, i.e., if the utilization of the selected level $u_{N,\theta_N} < 1$, for τ_{N-1} the selected version is stored at $j^*(N-1, m^*, k^* - \frac{u_{N,\theta_N}}{\omega})$, otherwise it is the version at $j^*(N-1, m^* - H_{N,\theta_N}, k^*)$, and so on. As k is defined as $M/(2*\omega)$, the time complexity of Algorithm 6 is $O((\sum_{i=1}^{N}|K_i|)\cdot M^2/\omega)$ and the space complexity is $O(NM^2/\omega)$.

If the scaling factor that is necessary to ensure that all utilization values are integers is too small, the number of entries that have to be considered in the table would be too large. To avoid this, a ceiling function can be used when calculating the utilization values for a given scale unit ω, i.e, all values of $u_{i,j}$ are replaced with $\lceil u_{i,j}/\omega \rceil$. This leads to a trade-off between the accuracy of our dynamic programming approach on one hand and the space and the time complexity on the other hand.

That the dynamic programming finds the optimal solution under FS using the schedulability test in Lemma 6 implicates that better reliability penalties can be achieved if other scheduling approaches or tighter schedulability tests are used. This also means that in some cases other scheduling strategies can perform better as shown in Section 6.4.2. However, our approach in general is not limited to FS and the schedulability test in Lemma 6. It can be applied for other strategies and tests by reformulating Eq.(6.10), Eq.(6.11), Eq.(6.12), and Eq.(6.13) accordingly if the sub-optimality to construct an optimal solution to schedule the first i tasks on m cores can be achieved by referring to the optimal schedules of the first $i-1$ tasks on m' processors (with $m' \leq m$) in a similar manner.

One specific example is to adopt semi-partitioned scheduling instead of partitioned scheduling for both the tasks in τ_{BIG} and τ_{LITTLE}. Rate-Monotonic Scheduling with Task Splitting (RM-TS) as proposed in [GSY+12] can be applied for the tasks in τ_{LITTLE} under FS. In this case, Eq.(6.8) should be reformulated as

$$\sum_{\tau_j \epsilon \tau_{\text{LITTLE}}} u_{j,\theta_j} \leq H_{\text{LITTLE}} \cdot \frac{2\Theta(\Gamma)}{1+\Theta(\Gamma)} \tag{6.14}$$

where $\Theta(\Gamma) = N(2^{1/N} - 1)$. Eq. (6.14) directly leads to the following sufficient test. A prove is therefore omitted.

Lemma 7. FS can schedule the tasks $\tau_{\text{BIG}} \cup \tau_{\text{LITTLE}}$ in Γ on M cores, if the following condition holds:

$$\sum_{\tau_i \in \tau_{\text{BIG}}} H_{i,\theta_i} + \sum_{\tau_j \in \tau_{\text{LITTLE}}} u_{j,\theta_j} \cdot \frac{1 + \Theta(\Gamma)}{2\Theta(\Gamma)} \leq M. \tag{6.15}$$

Please note that Eq.(6.10), Eq.(6.11), Eq.(6.12), and Eq.(6.13) should be reformulated accordingly. The corresponding results derived by using FS with Eq.(6.14) and Eq.(6.15) are also presented in Section 6.4.2 for completeness.

For the tasks in τ_{BIG} it is possible that nearly 50% of the utilization of 3 cores is wasted when CRT is used, i.e., when one activation of a task has a utilization slightly bigger than 0.5.[3] In this case using MRT-TMR directly is not possible as two complete activations of the task cannot be placed on the same core. However, using MRT-TMR under semi-partitioned scheduling is possible as long as the total utilization of 3 activations is below 2, i.e, by starting the original on core 1 and one replica on core 2, preempting the replica on core 2 after 50% of the replica is executed, starting the second replica on core 2 and finishing the previously preempted first replica on core 1 after the original task is finished.

In addition, using MRT-TMR for tasks in τ_{BIG} may also result in unbalanced utilization of CPUs and the waste of nearly 50% of the utilization of two CPUs in a similar scenario. This utilization could be used by other tasks. However, balancing the utilization of CPUs is not the main focus of this work. Our goal is to explain the approach in general and to show its effectiveness. Therefore we use well known techniques, i.e., FS and list scheduling.

6.3.3 Fine-Grained Selection

In this section we extend our approach from selecting among given redundancy levels to a more fine-grained approach where different stages of the task execution can be hardened individually. For each given task τ_i, we assume that it has S_i sequential stages, e.g., a function or a basic block, that can be hardened by redundancy individually as in the fork-join task model adopted by Axer et al. [AQN+13]. This means, we can decide whether we run a stage of a task with TMR, DMR, or without any redundancy. If a stage is executed with redundancy, the task execution is forked at the beginning of this stage and joined at the end of this stage.

If task τ_i's stage s is executed in the redundancy level $\theta_{i,s} \in \{\phi, \text{DMR}, \text{TMR}\}$ we assume its critical-path length, WCET, and reliability penalty are all given with corresponding mapping functions, i.e., $L_i(s, \theta)$, $C_i(s, \theta_{i,s})$, and $\mathbb{R}_i(s, \theta_{i,s})$, respectively, thus its utilization $\mathbb{U}_i(s, \theta_{i,s})$ is $C_i(s, \theta_{i,s})/T_i$. Our objective is to select the redundancy level $\theta_{i,s}$ for each task's stage that minimize the overall reliability penalty while satisfying the given timing constraints.

[3] We neglect the workload due to the synchronization in this example as it only adds additional complexity in the description without adding any insight.

Algorithm 7 Preprocessing for utilization

Input: N tasks
Output: Utilization-grained table $\hat{U}$
1: **for** $i \leftarrow 1, 2, \ldots, N$ **do**
2: **for** $k \leftarrow 0, 1, \ldots, 1/\omega$ **do**
3: $s_i^u(1,k) \leftarrow$ by using Eq.(6.16)
4: $q_i^u(1,k) \leftarrow \mathbb{R}_i(1, s_i^u(1,k))$
5: **for** $s \leftarrow 2, 3, \ldots, S_i$ **do**
6: calculate $s_i^u(s,k)$ by using Eq.(6.17)
7: $q_i^u(s,k) \leftarrow \mathbb{R}_i(s,j) + q_i^u(s-1, k - \mathbb{U}_i(s,j))$
8: $\hat{U}(i,k) \leftarrow q_i^u(S_i,k)$
9: $\mathbb{K}(i,k) \leftarrow$ backtrack with table s_i^u and $\mathbb{U}_i$

Definition 15 (Overall Reliability Penalty Ψ'_Γ)**.** The overall reliability penalty of task set Γ, denoted by Ψ'_Γ, is the sum of the tasks' reliability penalties given by $\Psi'_\Gamma = \sum_{\tau_i \in \Gamma} R_{i,\theta_i}$, where R_{i,θ_i} is the reliability penalty of task τ_i for a given selection $\theta_i = \{\theta_{i,1}, \theta_{i,2}, \ldots, \theta_{i,s}\}$ of stage redundancy levels.

Preprocessing

To make the final scheduling/task partition decision, the number of cores for τ_{BIG} and utilization for τ_{LITTLE} are both required. We prepare two reference tables $\hat{U}$ and $\hat{C}$ to record the optimal reliability penalty under given resource constraints and refer to both tables to obtain the best redundancy for each task stage. When for a stage s of τ_i a redundancy level $\theta_{i,s}$ is selected, we have to consider two possibilities: 1) $\tau_i \in \tau_{\text{LITTLE}}$ and the constraint is the utilization of the task, or 2) $\tau_i \in \tau_{\text{BIG}}$ and the constraint is the number of dedicated cores.

Utilization Demand Table For the first case, we prepare a table $\hat{U}(i,k)$ to record the optimal reliability penalty of τ_i with given utilization k and a corresponding cost table $\mathbb{K}(i,k)$ for recording the exact required utilization of $\hat{U}(i,k)$. Again we use ω to scale the utilization values and assume that all the scaled utilization values u_i/ω are integers. To find the optimal reliability penalty in $\hat{U}(i,k)$, we use a stage-wise dynamic programming algorithm. Therefore, we construct two 2-dimensional tables $s_i^u(s,k)$ and $q_i^u(s,k)$. The first dimension saves the considered stages and therefore is in the range $[1, S_i]$ while the second dimension depends on the corresponding utilization values k in the range of $[0, 1/\omega]$. In each stage s, all levels $j \in \{\phi, \text{DMR}, \text{TMR}\}$ are considered with their corresponding utilization $\mathbb{U}_i(s,j)$ and reliability penalty $\mathbb{R}_i(s,j)$. The pseudo code of the preprocessing is provided in Algorithm 7. For the first stage, we find the level $j^\star_{i,1}$ with the minimal reliability penalty for each k in $[0, 1/\omega]$ and record $j^\star_{i,1}$ in the $s_i^u(1,k)$ entry:

$$s_i^u(1,k) = \arg \min_{j \in \{\phi, \text{DMR}, \text{TMR}\}} \begin{cases} \mathbb{R}_i(1,j) & \text{if } k \geq \mathbb{U}_i(1,j) \\ \infty & otherwise \end{cases} \tag{6.16}$$

Algorithm 8 Preprocessing for cores

Input: N tasks, M cores
Output: Cores-grained table $\hat{C}$

```
for i ← 1, 2, ..., N do
    calculate ξ_i^max and l_i^max with all stages in TMR
    for s ← 1, 2, ..., S_i do
        for ξ ← 0, 1, ..., ξ_i^max do
            for l ← 0, 1, ..., l_i^max do
                calculate s_i^c(s, ξ, l) by using Eq.(6.18) and Eq.(6.19)
                q_i^c(s, ξ, l) ← R_i(s, s_i^c(s, ξ, l))
    Ĉ(i, 0) ← ∞
    H(i, 0) ← ∞
    for m ← 1, ..., M do
        Ĉ(i, m) ← by using Eq.(6.20)
        H(i, m) ← by ⌈(ξ' − l')/(T_i − l')⌉ with corresponding ξ' and l'
```

The corresponding reliability penalty $\mathbb{R}_i(1, s_i^u(1,k))$ is recorded in $q_i^u(1,k)$. For the following stage $s = 2, 3, \ldots, S_i$

$$s_i^u(s,k) = \arg \min_{j \in \{\phi, \text{DMR}, \text{TMR}\}} \begin{cases} \mathbb{R}_i(s,j) + q_i^u(s-1, k - \mathbb{U}_i(s,j)) & \text{if } k \geq \mathbb{U}_i(s,j) \\ \infty & otherwise \end{cases} \tag{6.17}$$

After all the entries in table q_i^u are calculated, we can find the optimal selection for each task with utilization k and record the exact utilization demand (scaled up by ω) in $\mathbb{K}(i,k)$. Both the time and the space complexity of Algorithm 7 are $O((\sum_{i=1}^{N} S_i) \cdot N/\omega)$.

Core Demand Table For the second case, we prepare a core-level table $\hat{C}(\tau_i, m)$ to record the optimal reliability penalty and record the number of required cores in table $\mathbb{H}(i,m)$. Similarly, we prepare a stage-wise table $s_i^c(s,\xi,l)$ to find the stage redundancy $j \in \{\phi, \text{DMR}, \text{TMR}\}$ with the minimal reliability penalty under the critical length l and the worst-case execution time ξ constraints, where $\xi \geq l \geq L_i(s, s_i^c(s,\xi,l))$. As each task τ_i in τ_{BIG} is assigned to $\left\lceil \frac{C_i - L_i}{T_i - L_i} \right\rceil$ cores in FS, we intend to find the optimal redundancy selection for each stage s by which the sum of the critical length l and the total execution time ξ among all the stages does not exceed L_i and C_i, respectively. The pseudo code is shown in Algorithm 8. Here we calculate the maximal critical length $l_i^{\max}$ and the maximal total execution time $\xi_i^{\max}$ by profiling each task τ_i's critical length $\sum_{s=1}^{S_i} L_i(s, \theta_{i,s})$ and total execution time $\sum_{s=1}^{S_i} C_i(s, \theta_{i,s})$ with TMR, respectively, on all its stages. We start from the first stage with the minimal reliability penalty and record the redundancy level in $s_i^c(1,\xi,l)$:

$$s_i^c(1,\xi,l) = \arg \min_{j \in \{\phi, \text{DMR}, \text{TMR}\}}$$

$$\begin{cases} \mathbb{R}_i(1,j) & \text{if } \xi \geq C_i(1,j) \text{ and } l \geq L_i(1,j) \\ \infty & otherwise \end{cases} \tag{6.18}$$

Its corresponding $R_i(1, s_i^c(1,\xi,l))$ reliability penalty is recorded in $q_i^c(1,\xi,l)$. For the following stage $s = 2, 3, \ldots, S_i$

$$\begin{aligned} s_i^c(s,\xi,l) &= \arg \min_{j \in \{\phi, \text{DMR}, \text{TMR}\}} \\ &\begin{cases} \mathbb{R}_i(s,j) + \mathbb{R}_i^t & \text{if } \xi \geq C_i(s,j) \text{ and } l \geq L_i(s,j) \\ \infty & otherwise \end{cases} \end{aligned} \tag{6.19}$$

where $\mathbb{R}_i^t = q_i^c(s-1, \xi - C_i(s,j), l - L_i(s,j))$ and the reliability penalty is recorded in $q_i^c(s,\xi,l)$. After all the entries in q_i^c are calculated, we can find a certain combination of ξ' and l' under the condition that $\xi \geq l \geq L_{i,\phi}$ to obtain the minimal reliability penalty $\hat{C}(i,m)$, defined as:

$$\hat{C}(i,m) = \min \begin{cases} q_i^c(S_i,\xi,l) & \text{if } m \geq \left\lceil \frac{\xi - l}{T_i - l} \right\rceil \text{ and } \xi \geq l \geq L_{i,\phi} \\ \infty & otherwise \end{cases} \tag{6.20}$$

where $L_{i,\phi}$ is the critical length of task τ_i without any redundancy. For each m and i combination, those ξ' and l' are recorded to $\xi_i^m = \xi'$ and $l_i^m = l'$. The time complexity of Algorithm 8 is $O((\sum_{i=1}^{N} S_i) \cdot \xi l)$ while the space complexity is $O(NM\xi l)$.

Selecting and Scheduling

Using the reference tables $\hat{U}$ and $\hat{C}$, our fine-grained approach builds two 3-dimensional tables $j^*(i,m,k)$ and $G(i,m,k)$ to record the sub-optimal selections of task τ_i and the resulting penalty values, respectively, for m dedicated cores and utilization, presented as pseudo-code in Algorithm 9. We use the tables $\hat{U}$ and $\hat{C}$ to find the selection with the minimal reliability penalty between the fine grained versions that are in τ_{BIG} or τ_{LITTLE}, i.e., with utilization > 1 and utilization ≤ 1, respectively. Again, all the possible combinations of utilization value k and number of available cores m have to be checked for all tasks, i.e., $0 \leq k \leq 1/\omega$ and $1 \leq m \leq M$.

For the first task τ_1 (lines 1-5 in Algorithm 9) the minimum reliability penalty for each m and each k can be calculated as:

$$G(1,m,k) = \min\left\{\hat{U}(1,k), \hat{C}(1,m)\right\} \tag{6.21}$$

For the other tasks, i.e., τ_i with $i > 1$ (lines 6-26 in Algo. 9), for each combination of m and k we need to consider all possible k' with $0 \leq k' \leq k$ to select the best achievable penalty when a selection θ_i with $\mathbb{U}_i(s,\theta_{i,s}) \leq 1$ is chosen and all possible m' with $1 \leq m' \leq m$ to select the best achievable penalty when a selection θ_i with $\mathbb{U}_i(s,\theta_{i,s}) > 1$ is chosen.

As a result, the best possible selection $P^*_{\text{LITTLE}}(i,m,k)$ for a solution with $\mathbb{U}_i(s,\theta_{i,s}) \leq 1$ can be found as:

$$P^*_{\text{LITTLE}}(i,m,k) = \min_{0 \leq k' \leq k} \left\{\hat{U}(i,k') + G(i-1, m, k - \mathbb{K}(i,k'))\right\} \tag{6.22}$$

Algorithm 9 Fine-grained table construction

Input: N tasks, M cores, $\hat{U}$ and $\hat{C}$ fine-grained tables;

for $m \leftarrow 0, \ldots, M$ **do**
  for $k \leftarrow 0, \ldots, \lceil \frac{0.5M}{\omega} \rceil$ **do**
    $G(1, m, k) \leftarrow \min\{\hat{U}(1, k), \hat{C}(1, m)\}$
    $j^*(1, m, k) \leftarrow \theta_{1,m,k}$
for $i \leftarrow 2, 3, 4, \ldots, N$ **do**
  for $m \leftarrow 0, \ldots, M$ **do**
    for $k \leftarrow 0, \ldots, \lceil \frac{0.5M}{\omega} \rceil$ **do**
      if $m + 2k \cdot \omega > M$ **then**
        $G(i, m, k) \leftarrow \infty$;
      else
        $P^*_{\text{BIG}}(i, m, k) \leftarrow \infty$
        for $m' \leftarrow 0, \ldots, m$ **do**
          $P(i, m', k) = \hat{C}(i, m') + G(i-1, m - \mathbb{H}(i, m'), k)$
          $P^*_{\text{BIG}}(i, m, k) = \min\{P^*_{\text{BIG}}(i, m, k), P(i, m', k)\}$
        $P^*_{\text{LITTLE}}(i, m, k) \leftarrow \infty$
        for $k' \leftarrow 0, \ldots, k$ **do**
          $P(i, m, k') = \hat{U}(i, k') + G(i-1, m, k - \mathbb{K}(i, k'))$
          $P^*_{\text{LITTLE}}(i, m, k) = \min\{P^*_{\text{LITTLE}}(i, m, k), P(i, m, k')\}$
        $G(i, m, k) = \min\{P^*_{\text{BIG}}(i, m, k), P^*_{\text{LITTLE}}(i, m, k)\}$
        $j^*(i, m, k) \leftarrow \theta_{i,m,k}$

The best possible selection $P^*_{\text{BIG}}(i, m, k)$ for a solution with $\mathbb{U}_i(s, \theta_{i,s}) > 1$ can be found as:

$$P^*_{\text{BIG}}(i, m, k) = \min_{1 \le m' \le m} \left\{ \hat{C}(i, m') + G(i-1, m - \mathbb{H}(i, m'), k) \right\} \tag{6.23}$$

Therefore, for τ_i with $i > 1$ the best possible selection for m and k is:

$$G(i, m, k) = \min\{P^*_{\text{BIG}}(i, m, k), P^*_{\text{LITTLE}}(i, m, k)\} \tag{6.24}$$

Please note, that $P^*_{\text{BIG}}(i, m, k)$ and $P^*_{\text{LITTLE}}(i, m, k)$ and therefore $G(i, m, k)$ may be ∞ for some values of m and k. The number of dedicated cores or the utilization of the chosen reliability selection θ^*_i is stored in $j^*(i, m, k)$.

Afterwards, the table $G(i, m, k)$ contains the optimal reliability values for each combination of i, m, and k. It contains entries with value ∞ if the condition of the schedulability test in Lemma 6 does not hold, i.e., $m + 2k \cdot \omega > M$. Note, that some other values may be ∞ as well, if the combination of the number of processors m and the utilization value k is too small to schedule any selection redundancy stages, i.e, if $m = 0$ and the sum of the utilizations of the tasks versions with no redundancy at all is larger than k. We search for the entry $G(N, m^*, k^*)$ with the minimal reliability penalty. Based on the related entry in $j^*(N, m^*, k^*)$ we know the chosen selection θ^*_N and can backtrack to the entry in $j^*(N-1, m, k)$ etc. The time and space complexity of Algorithm 9 both are $O(NM^2/\omega)$. To schedule tasks in Γ with their redundancy selection the tasks are classified in τ_{BIG} and τ_{LITTLE} and scheduled using FS.

6.4 Experimental Evaluation

To evaluate the performance of the proposed approaches in this chapter, we use the same setting as **dTune** [RKS+14b]. The details of the experimental framework are introduced in Chapter 3. The proposed approaches in Section 6.2 and Section 6.3 are evaluated in Section 6.4.1 and Section 6.4.2, respectively.

6.4.1 Evaluation for Task Mapping Approaches

At first, we evaluated the proposed mapping approach as Algorithm 3, the level adaptation approach as Algorithm 4, and Algorithm 5 with the generated reliability penalty value, different redundancy levels, and cores performance heterogeneity.

In total, we generate 128 different redundancy levels for the above 7 functions, i.e., 2^7, to test our approaches and the greedy mapping approaches used in *dTune*. Depending upon the performance heterogeneity, the infeasible scenarios in the evaluation are excluded. To simulate the performance heterogeneity of cores, the evaluation is performed by three different scenarios with variations ω on 8×8 cores as follows:

- **Grouping Frequency Levels**: Such scenarios are for evaluating architectures with heterogeneous performance, e.g., ARM big.LITTLE architecture [ARM13]. We evaluate four different frequency levels in a multi-core processor. We assume the performance variation is ω, where the cores are with frequencies f_1, $(1-\omega)f_1$, $(1-2\omega)f_1$, and $(1-3\omega)f_1$.
- **Uniform distribution**: Based on the variation model of [RTG+13], we uniformly generate the frequencies of cores from the highest one f_1 to the lowest one f_M to consider process variations.
- **Normal distribution**: The various frequencies of cores are normally distributed/generated [GM08] in the range of $(0,1]\cdot f_1$ with the mean $1-\omega$ and standard deviation $\sigma = 0.05$ to consider process variations. As it is possible that the normal distribution has a random variable greater than 1 or less than 0, we take such cases to the corresponding boundary conditions.

Considering real-word scenarios on performance variations, we only evaluate our proposed approaches while ω is up to 30% [BDM02]. Figure 3.5(b) shows the example variation scenarios under normal distribution. For simplicity of presentation, we set all the individual miss rate $\forall \rho_i \in \rho_\Gamma$ with the same rate ρ.

For each configuration of core frequencies, we generate 500 different processors with different variations, and report the average results. As we are not aware of any other state-of-the-art related works, we normalize our results to the greedy mapping and compare the efficiency with the same set of task versions and core configurations for fairness, in which the normalized ratio is calculated as ϕ_γ of the resulting solution divided by ϕ_γ of the greedy mapping. By definition, the lower normalized penalty ratio is better.

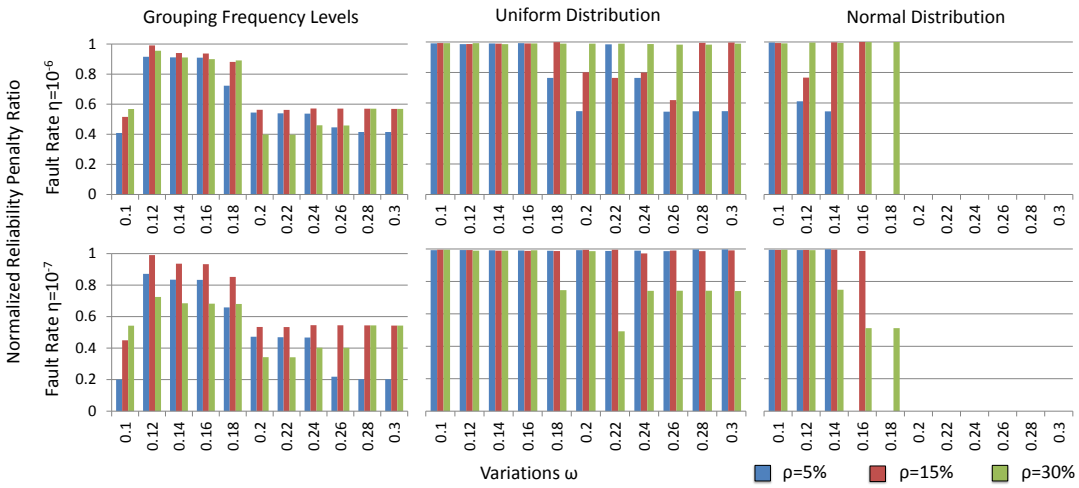


Figure 6.5: Overall reliability penalty ratio by normalizing the results of proposed approaches to the greedy mapping strategy. It shows the evaluation results under two different fault rates, i.e., 10^{-6} and 10^{-7}.

Evaluational Results for Task Mapping

In these simulations, we evaluate our mapping approach with all the possible redundancy levels. Each bar in the presented figures is obtained by averaging the reliability results through these 128 different redundancy levels. Since the greedy mapping cannot guarantee the feasibility of the task mapping, it may be possible that the greedy mapping is not a feasible one to meet the miss rate constraint.

Simulation without Data Dependency Figure 6.5 shows the evaluation results under two different fault rates, i.e., 10^{-6} and 10^{-7}. Overall, we can observe that our approach outperforms the greedy mapping approach, and the average improvement is around 20% among all the cases. In particular, the improvement can be up to 80% (0.2 in the bar plot) when the fault rate is 10^{-7} under Grouping Frequency Levels. In such scenarios, the reliability penalties may play a minor role, whereas the greater penalties of timing constraint violations dominate the value of the penalty function. Moreover, when the variations of performance among the cores are higher, our approach is typically more effective than the greedy mapping approach. It is because our approach prevents the severe degradation of reliability, in which the cores are not grouped properly. Please note, if the design constraints are too strict, none of the approaches can deliver a feasible solution. Even if there exists a feasible solution, there is no space to further improve the reliability among different approaches.

In case the difference of frequencies between different grouping levels is large enough, the greedy mapping approach may suffer from the sequential assignment of cores, in which the task with RMT mode may have severe performance degradation

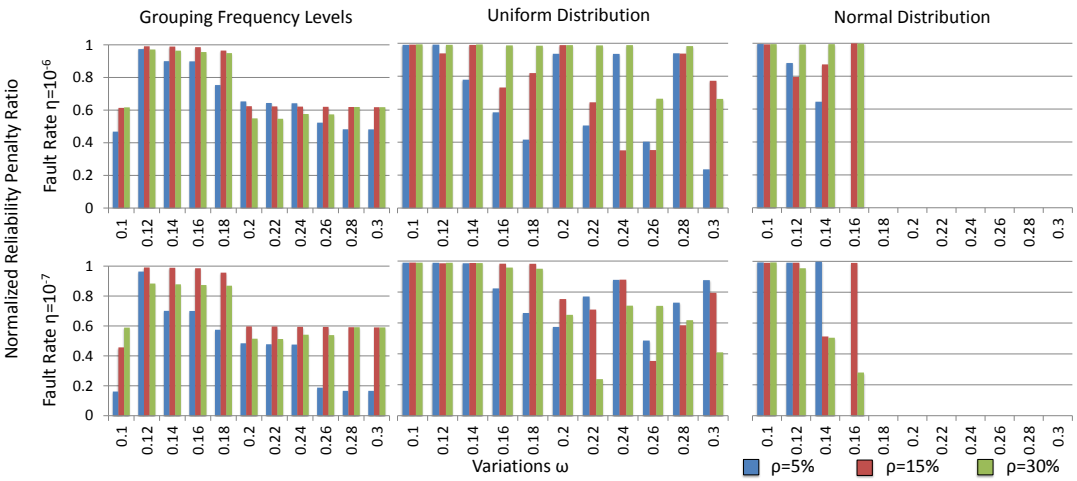


Figure 6.6: Overall reliability penalty ratio for the complicated application under two different fault rates 10^{-6} and 10^{-7}.

due to the domination of its lowest-frequency core in the majority-voting. In particular, the most improvement can reach up to 80% when $\omega = 0.3$, $\eta = 10^{-7}$, and $\rho_\Gamma = 5\%$.

Interestingly, we can observe that the improvement is not significant under the scenario of uniform distribution. Among all the possible combinations, the differences of overall reliability penalty between both approaches are not significant, since the frequencies of the cores are degraded smoothly. Nevertheless, our task mapping approaches can still perform well in some cases. For example, in the lower fault rate as 10^{-7}, the improvement can be up to 31%, when the tolerable miss rate is higher, i.e, $\rho_\Gamma = 30\%$.

In the scenario of normal distribution, some of the results with the severe performance variations, i.e., $\omega \geq 0.16$, have no feasible solutions in the simulation. Due to the lack of high-frequency cores, most of the redundancy levels cannot be satisfied, in which most of the cores are degraded as the middle-frequency under normal distribution. When the tolerable miss rate is strict with the lower fault rate, i.e., $\eta = 10^{-7}$ and $\rho = 15\%$, the results in our simulations depend upon the timeliness of task mapping, in which the improvement is less because of the negligible differences of feasible mappings. Among all the feasible constraints, our proposed approach outperforms the greedy mapping approach under both fault rates.

Fig 6.6 presents the comparison results under different fault rates for a more complicated application scenario. We construct this application by using the functions selected from MiBench as mentioned previously, and duplicate the functions to increase the demand for cores. Based on 14 functions in the complicated application, we examine 2^{14} different redundancy levels, and present the overall reliability penalty ratio in average. As a result, we know that our approach is still applicable and outperforms the greedy approach in case the application is more complicated.

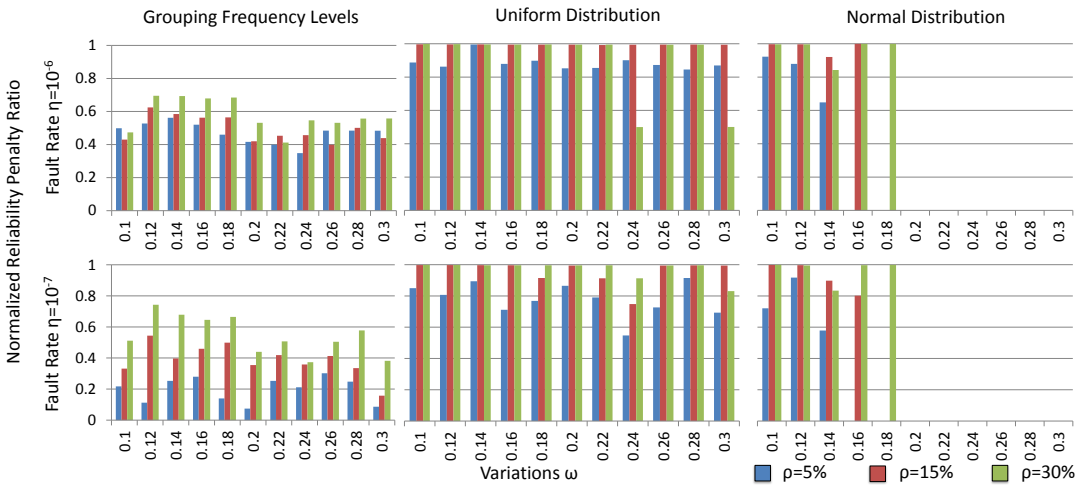


Figure 6.7: Evaluation of the reliability penalty ratio with the communication overhead under different fault rates.

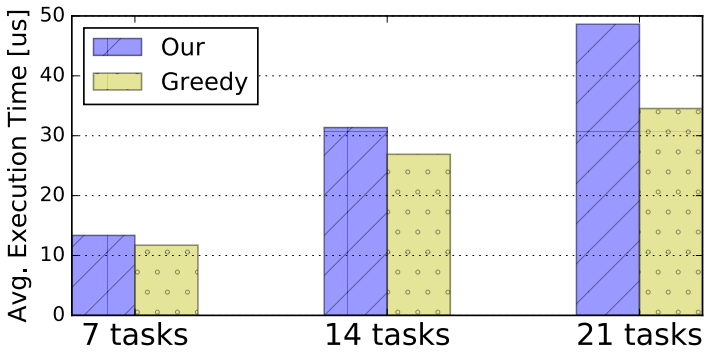


Figure 6.8: Overhead between our approach and greedy mapping.

Simulation with Communication Overhead By applying the communication model presented in Section 6.2.4, we reformulate the deadline miss rate of tasks in the preprocessing and adopt Algorithm 5, to obtain the simulation results in Figure 6.7. At first, we can observe that the trends of results are similar as the previous case (without data dependency). Since the overhead of communication increases the hardness of meeting deadline, the feasible versions of tasks are reduced greatly, in which most of the tasks have a few choices to utilize the different frequencies of cores. However, the reliability improvement among all the different mappings is generally more than the case without dependency consideration.

Required Analysis Time To compare the required analysis time, here we report the average execution time for the experiments reported in Figure 6.5 and Figure 6.6. As shown in Figure 6.8, we can see that if the input number is as small as 7 tasks, our method can still be efficient. However, when the input number is increased to 21

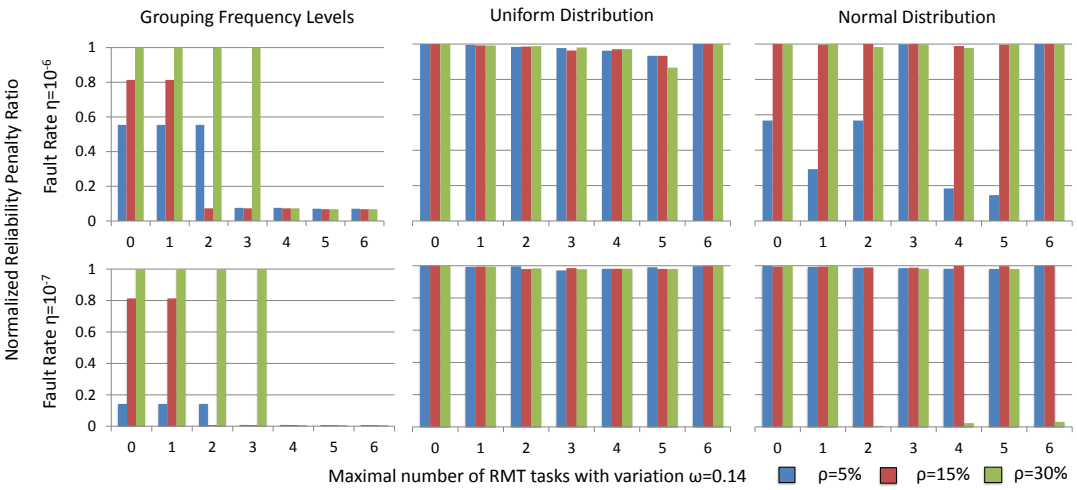


Figure 6.9: Evaluation of the resulting redundancy levels with variation $\omega = 0.14$ under different fault rates.

tasks, the overhead difference is more obvious. As the greedy mapping sorts the tasks by the reliability penalties, the execution time is mainly dominated by the sorting algorithm. However, the time complexity of our approach is $O(N^3)$. When there are many tasks, some approximations are required to trade the execution time for the efficiency.

Evaluational Results for Levels Adaptation

Here are the evaluational results when the redundancy levels of tasks are not determined beforehand. To evaluate the effectiveness, we adopt Algorithm 4 to derive the redundancy levels with different maximal numbers of RMT tasks, in which the maximal number of RMT tasks is determined by the number of available cores. Here we present the evaluation under different fault rates 10^{-6} and 10^{-7} with variation $\omega = 0.14$. As we are not aware of any other approaches of redundancy levels adaptation, we show the normalized reliability penalty ratios with our mapping approach.

As shown in Figure 6.9, we can see that the trends in the charts with the derived redundancy levels still follow our observation in the previous evaluational results. If the frequency variation among the cores is not negligible as in the case of grouping frequency levels, the proposed mapping approach perform well most of time. If the frequencies of cores are degraded smoothly as the case of uniform distribution, the improvement of overall reliability penalty is not significant. In the case of normal distribution, the improvement is still significant with the derived redundancy levels when the tolerable miss rate is tighter, i.e., $\rho = 5\%$.

We also compare the resulting redundancy levels with the optimal redundancy levels for seven tasks, which is obtained by a brute-force search with a factorial timing complexity. We observe that the task mapping and redundancy levels derived by

our proposed approaches are equal to the optimal task mapping under the optimal redundancy levels.

6.4.2 Evaluation for Multi-Tasking Approaches

For evaluating the proposed optimization approaches in Section 6.3, we analyzed the performance of both the coarse-grained and the fine-grained approaches compared to the greedy approach used in **dTune** [RKS+14b] with respect to the number of feasible configurations and by comparing the best resulting reliability penalties among those feasible configurations, using real tasks from an embedded benchmark.

For each redundancy level, we determined the WCET by a large number of measurements and estimated the reliability penalty values under two different fault rates η, i.e., $\eta = 10^{-6}$ and 10^{-7} $fault/cycles$, as adopted in [HWZ06; LDV+04]. Similarly to [HWZ06; LDV+04], the reliability penalty for each function/task is estimated using the same metrics as in [RSK+11; RCK+16]. The time overhead of the synchronization for DMR and TMR redundancy levels are integrated into the total execution time $C_{i,j}$ and the critical-path length $L_{i,j}$ for level $\tau_{i,j}$. From those values we generated the tasks period using two different approaches:

- **Random Periods:** We randomly drew T_i in the range of $[1.65 \cdot C_{i,\phi}, (1.65 + \rho) \cdot C_{i,\phi}]$ using a uniform distribution, i.e., analyzing task sets with smaller / larger periods compared to the execution time and therefore larger / smaller utilization values. The ρ values we used in the experiments were 0.6, 1.2, and 1.8, leading to upper bounds of 2.25, 2.85, and 3.45, respectively, for the periods compared to the execution time. We generated 500 task sets for each ρ to get a sufficient sample size.
- **Given Total Utilization:** We applied the UUnifast-Discard method proposed by Davis et al. [DB11] to generate task sets of size N with a given total utilization U^* for the execution without any redundancy, i.e., $U^* = \sum_{\tau_i \in \Gamma} u_{i,\phi}$. Each task τ_i is assigned with a utilization value $u_{i,\phi}$ and the period is $T_i = C_{i,\phi}/u_{i,\phi}$, i.e., the execution time of the non-RMT level divided by the assigned utilization from the UUnifast-Discard method. We evaluated using three different values of U^*, i.e., 250%, 300%, and 350% creating 500 task sets for each utilization value.

As **dTune** only can decide to activate CRT-TMR or not, it needs at least 9 cores to activate CRT-TMR for one task and 21 cores to activate CRT-TMR for all tasks. Therefore, we evaluated the range of 9 to 21 cores.

For the setting of U^*, we choose the range of 250% to 350% due to the following reasons: (1) When U^* is larger than 350%, the average utilization of a task is more than 50%. Therefore most tasks cannot activate SRT-TMR or MRT-TMR due to the additional computations for replicas and synchronization. (2) If the total utilization is less than 250%, the average task utilization is below 33% and therefore most tasks can activate SRT-TMR.

We implement our dynamic programming approaches with C++, i.e., Algorithm 6 and Algorithm 9, using an Intel Core i7-4770 with 16GB DDR3 RAM for the evaluations. The derived reliability penalties of Algorithm 6 and Algorithm 9 with Lemma 6 (partitioned scheduling) and Lemma 7 (semi-partitioned scheduling) are compared with **dTune** [RKS+14b] by evaluating the number of feasible configurations and the derived overall reliability penalties $\Psi_\Gamma(\theta)$. The greedy approach adopted in **dTune** [RKS+14b] works as follows:

- The tasks are sorted by their reliability penalty $R_{i,\phi}$.
- The $\lfloor \frac{M-N}{2} \rfloor$ tasks with the highest reliability penalty are selected to activate CRT-TMR.
- The remaining tasks are executed in NON-RMT.

Although the task mapping approaches introduced in Section 6.2 outperform the greedy approach in [RKS+14b] under variation-performance multi-cores, the approaches from both papers will perform the same in the studied problem that all the cores in the considered system are homogeneous.

Evaluation of the Coarse-Grained Approach

To show that MRT provides additional possibilities for reliability optimization we adopted a similar comparison as in [KCK+16]. We restrict the coarse-grained approach to only use TMR with partition scheduling and semi-partition scheduling, as **dTune** can only choose between CRT-TMR and no redundancy as well. For each of the seven tasks, TMR can be activated individually, leading to $2^7 = 128$ configurations. We report the number of feasible configurations for both the coarse-grained approach (Algorithm 6) and **dTune** [RKS+14b]. The average analysis duration for one run of the coarse-grained approach considering 7 tasks was 0.22 seconds. As results under different fault rates η are similar, we only present the results for $\eta = 10^{-6}$.

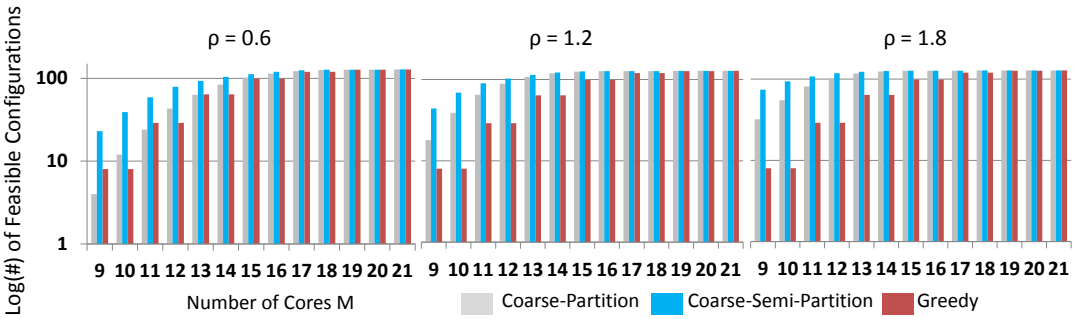


Figure 6.10: Number of feasible configurations for different ρ.

Figure 6.10 shows the number of feasible configurations with respect to ρ and the number of cores M. A log scale with base-10 is used for the Y-axis to improve the readability. The number of feasible configurations using the coarse-grained approach is generally not less than for the greedy approach. Some exceptions are the cases

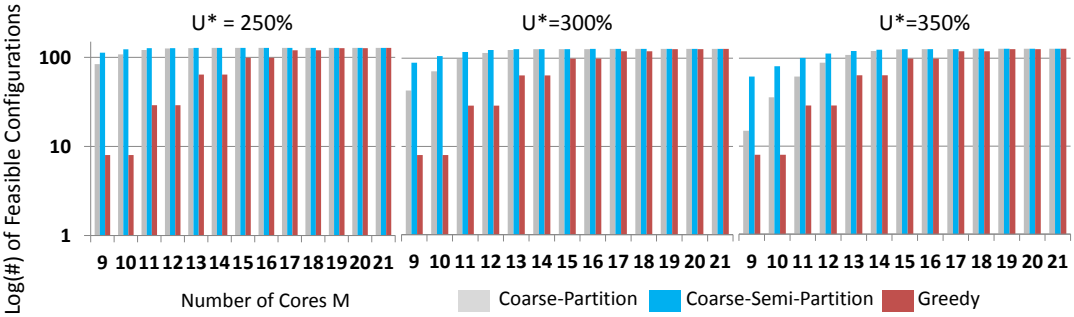


Figure 6.11: Number of feasible configurations for different U^*.

using partitioned-scheduling under tight constraints, e.g., 9 cores and $\rho = 0.6$. This is due to the limitation of the sufficient bound of the partitioned-scheduling. As all tasks without redundancy are by construction in τ_{LITTLE} and the sufficient bound only accepts tasks with a total utilization of $\sum_{\tau_i \in \tau_{\text{LITTLE}}} u_{i,\phi} \leq M/2$ some task sets are barely schedulable without any redundancy. When the parameter ρ is increased, i.e., the utilization of tasks are relatively lower, the coarse-grained approach adopts SRT/MRT-TMR to exploit the spare utilization of cores, while the greedy approach can only decide if CRT-TMR should be activated or not without any adaptation.

As shown in Figure 6.11, for different U^* settings, the number of feasible configurations using the coarse-grained approach are generally superior to the greedy approach when the number of available cores is less than 18. However, we can see that our approach does not always outperform the greedy approach if the number of available cores is sufficient to activate CRT-TMR for many tasks. For example, when the number of available cores is 21 and U^* is close to 350%, the coarse-grained approach does not outperform the greedy approach. However, these results strongly support our claim that using MRT is able to increase the schedulability for given selections of TMR.

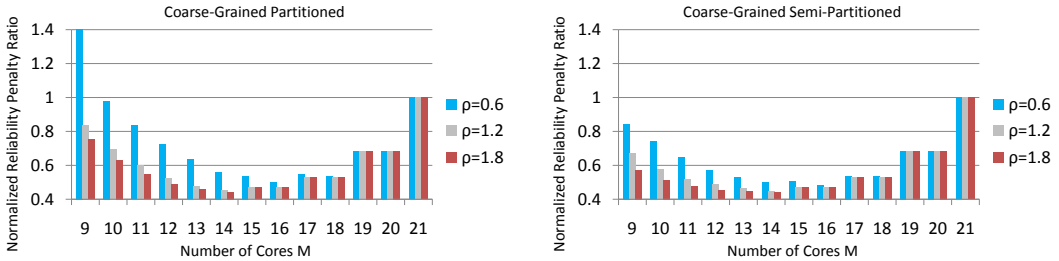


Figure 6.12: Comparison of the coarse-grained approach and the greedy approach under different ρ.

Figure 6.12 shows the normalized ratio of overall system reliability, which is calculated as Ψ_Γ of the resulting solution divided by Ψ_Γ of the greedy approach. To generate the reliability penalties, $\eta = 10^{-6}$ was used. Instead of using the given

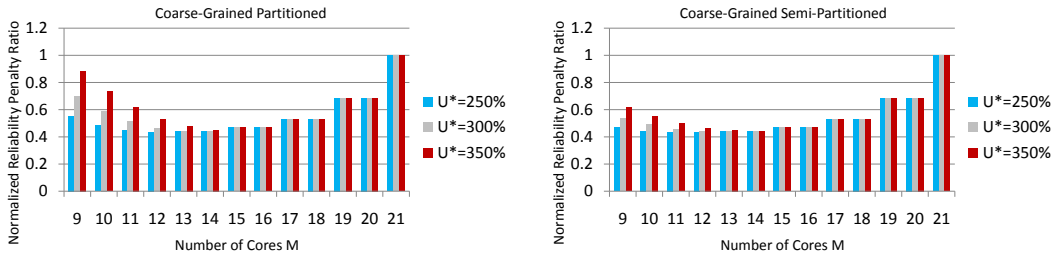


Figure 6.13: Comparison of the coarse-grained approach and the greedy approach under different U^*.

redundancy configurations in the previous evaluation, the greedy approach decides which task can be executed with CRT-TMR. Our coarse-grained approach determines the executed redundancy level of each tasks. For the sake of fairness, we only compare those task sets where both approaches are able to provide feasible mappings. By the definition of the penalty value, lower values are better. Generally, most of the results derived by the coarse-grained approach are better than the greedy approach results. However, when M is 9, we can see that our approach cannot outperform the greedy approach when partitioned scheduling is used in the coarse-grained approach. From the previous evaluation we know that our approach can find some feasible mappings, but the best one it can find is limited by the sufficient condition of FS and in this case the coarse-grained approach can barely execute the tasks without redundancy. For 12 up to 20 cores the coarse-grained approach achieves a significantly better reliability penalty than the greedy approach. For the normalized reliability penalty ratio displayed in Figure 6.13, we observe that the gain by using the coarse-grained approach is larger when the overall utilization is smaller. If semi-partitioned scheduling is used, the gain is higher when the number of available cores is in the range between 9 to 11 especially for $U^* = 350$. Overall, as our approach can fully exploit the available cores with SRT or MRT rather than solely using CRT, it specifically performs well when the number of available cores is in the most interesting region, i.e., TMR can be activated for some tasks but not for most/all.

Evaluation of the Fine-Grained Approach

We compare the results of the fine-grained approach (Algorithm 9) with the coarse-grained approach to show the possible benefit if the stage redundancy can be determined arbitrarily. We show the comparison to the coarse-grained approach here. Please note that the fine-grained approach performs as least as good as the coarse-grained approach. Since the latter one can only harden the whole task, it is always one option for the former one as well.

Figure 6.14 and Figure 6.15 show the average system reliability values achieved by the fine-grained approach over 500 runs, normalized by the values achieved by the coarse-grained approach. The number of task stages S_i was drawn uniformly

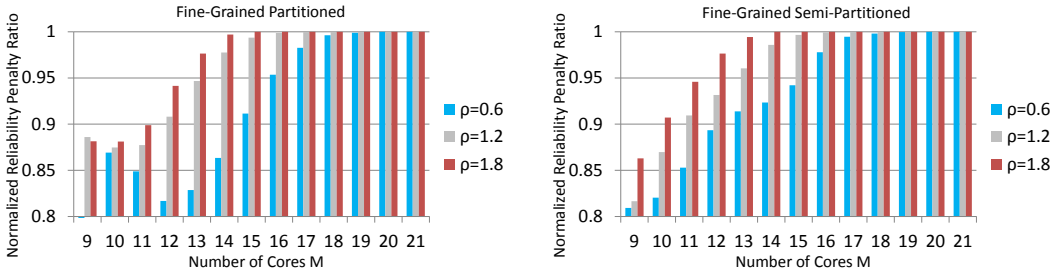


Figure 6.14: Comparison of the fine-grained approach and the coarse-grained approach under different ρ.

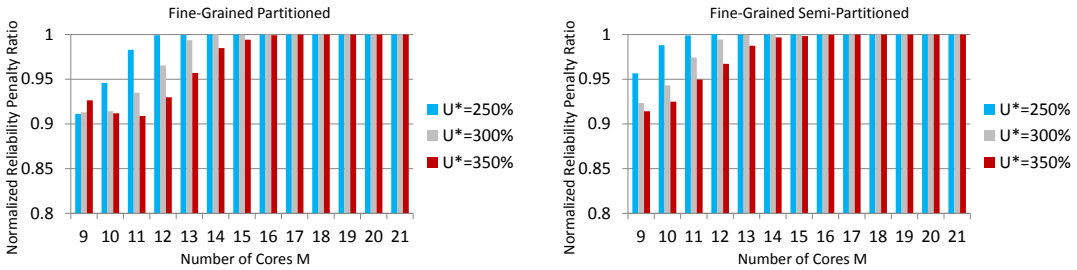


Figure 6.15: Comparison of the fine-grained approach and the coarse-grained approach under different U^*.

distributed in the range of 2 to 5. Generally, the more stages a task has, the more flexibility for using redundancy it has. Please note, that when S_i is 1 for each task τ_i, the fine-grained approach is the same as the coarse-grained approach. The average analysis duration for one run of the fine-grained approach with 7 tasks was 3.8 seconds.

In Figure 6.14, we see that the fine-grained approach improves the results of the coarse-grained approach more when ρ is smaller. While the coarse-grained approach with partitioned scheduling suffers from the small number of available cores which does not allow to harden many complete tasks if the utilization is high, i.e., $M = 9$ and $\rho = 0.6$, the fine-grained approach can harden some stages of the tasks with redundancy. In this case, the normalized ratio between the fine-grained and the coarse-grained approaches is 0.6292, but the bar is not shown due to the scale of Y-axis. However, we choose to use the scale from 0.8 to 1.0 as it improves the readability for the other settings. Under semi-partitioned scheduling, the gain of the fine-grained approach is not as large as under the coarse-grained approach, since the coarse-grained approach can activate some redundancy already for some tasks due to a larger sufficient bound in Lemma 7.

In Figure 6.15, we can observe that the benefit of using the fine-grained approach is not as large as in Figure 6.14, especially under semi-partitioned scheduling. The reason is that if the coarse-grained approach can already activate TMR for most complete

tasks, the fine-grained approach can only perform as good as the coarse-grained approach for those tasks.

6.5 Summary

Using Redundant Multithreading (RMT) for error detection and recovery is a prominent technique to mitigate soft-error effects in multi-core systems. Simultaneous Redundant Threading (SRT) on the same core or Chip-level Redundant Multithreading (CRT) on different cores can be adopted to implement RMT. This chapter studies how to achieve resource-efficient reliability on multi-core systems by using RMT and redundant cores while considering the given resource constraints.

At first we introduce reliability-driven mapping techniques to allocate the tasks onto a multi-core processor by taking application vulnerability and performance heterogeneity into consideration. We show that a special case of the studied problem with homogeneous redundancy levels is equivalent to the MWPBM problem, and an approach is developed to optimally handle heterogeneous redundancy levels. To consider communication and data dependencies, we provide a viable way to estimate the transfer overhead with a deterministic routing algorithm and show how it enhances our proposed mapping approaches. Our evaluations show that the proposed approaches may improve greedy method drastically when the frequency variation among the cores is not negligible. For different scenarios of chip frequency variation maps, the overall improvement may result in average in 20% less reliability penalty, while guaranteeing all tasks meet their given deadline miss rate constraints

For the implementation, the interactions between the compiler and the operating system are required. The reliable compilation helps us exploit the resilience of tasks with varying execution time and vulnerabilities The proposed approaches needs to be implemented in the scheduler to determine the task mapping and the redundancy levels. Since the time complexity of proposed approaches are polynomial time based, they can be adopted, either for on-line reconfiguration due to aging-induced effects or process variations, or off-line configuration due to heterogeneous architectures pursuing the dependable application design.

Secondly, this chapter also exploits redundant cores to mitigate soft-error effects by using RMT reasonably with multi-tasking. We provide software synthesis methodologies for real-time embedded system designers to efficiently exploit mixed redundancy techniques to decrease the system reliability penalty while satisfying the timing constraints in multi-core systems. We provide a combination of CRT and SRT called MRT to achieve these goals. In addition, we provide two reliability optimization approaches to decrease the system reliability penalty with different granularity while scheduling the hard real-time tasks on multi-cores. The results show that the proposed approaches are generally better than the greedy approach in terms of reliability when the number of available cores is limited to activate CRT-TMR for tasks. Furthermore, since the fine-grained approach has more flexibility to harden

tasks in stage-level, the decrease of the system reliability penalty is at least as good as for the coarse-grained approach. When the resources are more limited, the benefit of adopting the fine-grained approach is more significant. While our approaches already perform better than the greedy approach in most cases if partitioned scheduling is used for the tasks in τ_{LITTLE}, using semi-partitioned scheduling can increase the gain even further.

To the best of our knowledge, this work provides the first solid foundation for using mixed redundancy techniques in multi-core systems. The methodologies are not limited to the mixture of two redundancy levels but also applicable to multiple redundancy levels up to the designer's choice. However, our study is limited to implicit-deadline real-time tasks under federated scheduling in homogeneous multi-core systems. It has been shown by Chen [Che16a] that FS does not perform well for constrained- and arbitrary-deadline real-time task systems. Nevertheless, the proposed MRT can directly be used for other scheduling strategies while the applicability of the proposed optimization techniques depends on the scheduling strategy that is used.

CHAPTER 7

Overrun Handling in Real-Time Operating System

Contents

7.1 Overview

In soft real-time systems like in Chapter 5 or Mixed-Criticality Systems [Ves07], deadline misses (we call this behavior as *overrun* of tasks in this chapter) are not absolutely forbidden but require a proper reaction from the designed system to handle such cases as discussed in Chapter 5. Therefore, a Real Time Operating System (RTOS) and platforms, e.g., FreeRTOS [Rea16], Litmus-RT [Bra06], and Real-Time Executive for Multiprocessor Systems (RTEMS) [RTE13], for running such real-time systems should ensure that the system still behaves as expected if such overrun situations occur. Namely, the task jobs should still be released according to the given pattern and all task jobs are still released even if another task job is still in the system at the moment the next task job would be released according to the

pattern. This makes sure that the system behavior is predictable even with very rare overrun situations.

It is usually assumed that multiple task jobs of a task are executed in a First-Come-First-Serve (FCFS) manner. Thus, it is sufficient to release the second task job at the moment the first task job finishes, assuming that the first task job finishes after the time point at which the second task job would have been released according to a strictly periodic pattern. However, during our work on *Systems with Dynamic Real-Time Guarantees* [BCH+16], we ran simulations with RTEMS (*Real-Time Executive for Multiprocessor Systems*) [RTE13] to analyze the impact of transient faults during the execution of a task job. We assumed those faults to happen randomly under a given fault rate and wanted to analyze the impact of the fault rate on the system behavior. Interestingly, we discovered that RTEMS (version 4.11) did not behave as expected when these faults result in a task missing its deadline within these analyzes.

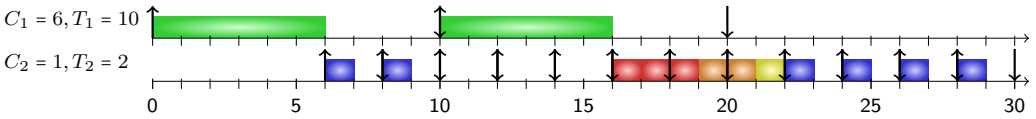
In this chapter, we focus on explaining the problems that arise from the original implementation of RTEMS when overruns take place and provides a solution that tackles these problems. Two major problems are discussed and solved: On one hand, missing the deadline of a task job leads to a shift of the release pattern of the task. On the other hand, if the deadline was missed by more than one period, the task job that should have been released during this period was never released. We extended the current release of the RTEMS source code to tackle these two problems. In addition we shortly explain the case study we performed for the paper *Systems with Dynamic Real-Time Guarantees in Uncertain and Faulty Execution Environments* [BCH+16] in Section 7.4, which was the motivation why we discovered those problems and enhanced the implementation of RTEMS.

7.2 Original Design in RTEMS

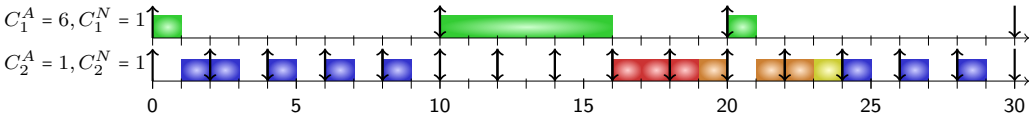
In this section, we describe the original design of overrun handling mechanism in RTEMS [RTE13] and present how to integrate the enhancement perfectly in the latest RTEMS release. A motivational example is provided to demonstrate the differences between the original design and the enhanced version of overrun handling. In addition to the proper overrun handling mechanism, we also provide a useful helper function, which is detailed at the end of this section.

7.2.1 Task Model

In this chapter we consider n independent periodic real-time tasks $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ in a uniprocessor system. Each task τ_i releases an infinite number of jobs (also called task jobs) under the given period (temporal) constraint T_i. As we assume the tasks to be strictly periodic this means if at time θ_a a job of task τ_i arrives, the next job of the task must arrive at $\theta_a + T_i$. The relative deadline of task τ_i is D_i, i.e., a task job



(a) The overrun handling with our enhancement in RTEMS. The postponed jobs due to the execution of τ_1 are marked red, the jobs postponed due to the execution of previous jobs of τ_2 that are not executed in the right period are marked orange. The yellow job is postponed due to the orange job in the same period but can still finish its execution on time.



(b) The overrun handling with our enhancement for dynamic real-time guarantees in RTEMS. Due to error recovery routine, task τ_1 runs in the second period with the abnormal mode, i.e., $C_1^A = 6$. The postponed jobs due to the execution of τ_1 are marked red, the jobs postponed due to the execution of previous jobs of τ_2 that are not executed in the right period are marked orange. The yellow job is postponed due to the orange job in the same period but can still finish its execution on time.

Figure 7.1: An example illustrates the enhanced overrun handling mechanism and how does it work for dynamic real-time guarantees [BCH+16].

released at θ_a must be finished before $\theta_a + D_i$. We mainly consider implicit-deadline task sets in this chapter, i.e., $D_i = T_i \ \forall \tau_i$.

7.2.2 Original Design

In the considered version 4.11 of RTEMS[1], the scheduler uses a virtual table of function pointers to hook scheduler-specific code and the thread management. In this chapter, we limit our attention on the fixed-priority scheduler in the RTEMS implementation. Although the related functions have a prefix *rate-monotonic* and is called RMS in RTEMS, the scheduler can be used for any fixed-priority scheduling, by which the priorities of tasks can be set by the system designer. For the sake of clarity, we will remove the prefix when we mention the functions of the scheduler in the rest of paper. The primary source code is located in the following files in the SuperCore (cpukit/score):

- cpukit/rtems/src/ratemonperiod.c
- cpukit/rtems/src/ratemontimeout.c
- cpukit/rtems/include/rtems/rtems/ratemon.h
- cpukit/rtems/include/rtems/rtems/ratemonimpl.h

The rate-monotonic manager in RTEMS is responsible for handling the periodicity of the tasks. Based on it, this chapter mainly redefines the behavior of the `Period()`

[1]When this work was published, version 4.11 was the latest version. The proposed enhancement here is included in the latest version 5.1 as well.

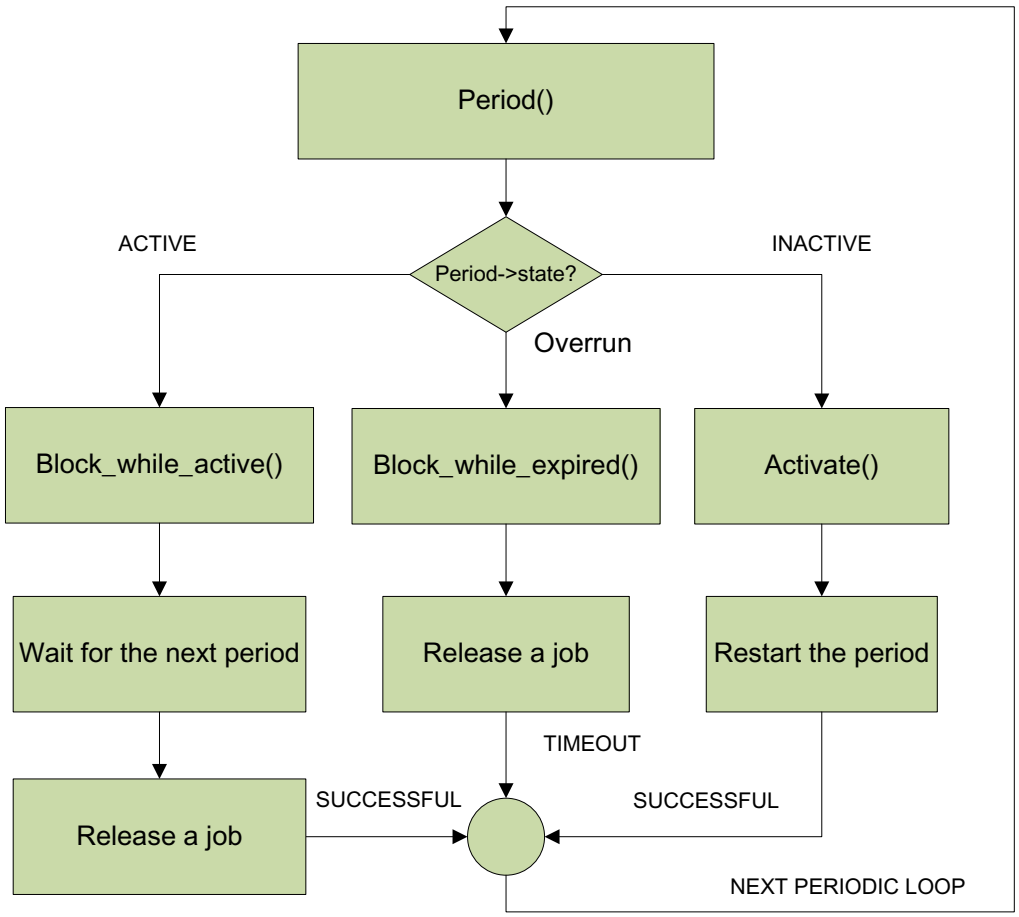


Figure 7.2: Flowchart of the RMS manager in RTEMS.

function and the `Timeout()` function in `ratemonperiod.c` and `ratemontimeout.c`, respectively. As mentioned in [BS14], for each task, its periodicity is implemented by using a timer to track its period.

To implement a periodic task, the system designer should create a periodic timer and implement the task body with a loop that calls `Period()` at the beginning to initialize the corresponding timer with the current system tick plus its period. Every time the task finishes its task body, i.e., at the end of the loop, `Period()` is called to setup the next iteration and immediately checks if the period is depleted yet. If the task finishes before its next period, it immediately goes to sleep (or suspends) until its period elapses, at which time its timer fires, wake the task to continue the loop body, and so on so forth. In the original design in RTEMS, if the watchdog notices that the deadline of a job expires but the job has not finished yet, i.e., a period timeout of a task takes place but the task body has not been finished, RTEMS marks the period state as `EXPIRED` and does nothing more. The next time `Period()` is called at the beginning of the loop, i.e., when the expired task is finished, the `Block_while_expired()` function records the marked state of the expired period

to update the system statistic routine and releases the next job immediately while updating the timer with the current system tick plus its period. Figure 7.2 illustrates the flowchart of the RMS manager. However, such an overrun handling mechanism is not able to keep the periodicity of the task, since the system tick at which the delayed task job finishes is normally not at an integer multiple of the period of the task. Also this moment does not necessarily have to happen during or at the end of the first period after the deadline expired, but could be in any period after the originally expired one. In such a scenario, all the postponed jobs that would have been initialized in the time interval between the expired deadline and the newly released task job is created are just gone.

7.3 Enhancement

After discussing the original design, we now explain how we can enhance the original implementation to handle the deadline overrun correctly. Based on the previous observations, the main ideas of our enhancement can be summarized as follows:

- correcting the deadline assignment errors using a watchdog, i.e., keep the periodicity of the tasks,
- tracking the number of postponed jobs, and
- providing two modes of job releasing depending on the situation, i.e., normal and postponed release.

The flowchart is provided in Figure 7.3, in which the light background blocks are involved in the enhancement. In the rest of this subsection, we explain more details about our implementation. Please note that the use of the word *deadline* in the rest of section is referring to the deadline of the watchdog timer. Since the arrival time of the tasks' jobs is exactly the deadline of the watchdog timer in RMS, the periodicity of tasks should be fixed from the deadline assignment of the watchdog.

Correcting the deadline assignment

To keep the correct deadline assignment after a job misses its deadline, we add an additional variable called `latest_deadline` in the `Control` structure that records the latest deadline assigned by the period watchdog. This variable is used to call an additional function named `Renew_deadline()` in the `Timeout()` function, making sure that the watchdog updates the timer to the next absolute deadline. The next deadline will be recorded in the variable `latest_deadline`. Due to this enhancement, the watchdog updates the timer correctly rather than doing nothing in the original design. The recorded deadline in the variable `latest_deadline` is prepared for the next call of the `Timeout()` function, allowing to set the deadline to the sum of latest deadline and period for the next deadline assignment directly.

Tracking the number of postponed jobs

In addition to the correct deadline assignment, we also implemented a tracking mechanism for the number of the postponed jobs to ensure that the postponed jobs are correctly released. To deploy this idea, we add an additional variable in the `Control` structure called `postponed_jobs` and initialize it while the periodic timer is created. When the watchdog detects a deadline miss (a period timeout while the related task is not finished), this variable is increased by one immediately in the `Timeout()` function. Conversely, every time a postponed job is released by `Release_postponedjob()`, this variable is decreased by one immediately.

Two modes of job releasing

As we mentioned before, normally the execution of periodic tasks is correct, if there is no deadline miss. When there is an overrun, the watchdog of the expired period detects such an overrun and the next call of `Period()` will immediately release one job and set the expired period state back to the normal state by using the function `Block_while_expired()`. In the original facility, the above behavior is handled by the function `Release_job()`. In the enhancement, we replaced it with an additional function `Release_postponedjob()` and let the watchdog handle the deadline assignment individually. We implemented it similar to the original job releasing routine, but it does not assign the new deadline to the timer while releasing a postponed job. Since they already missed their deadline and are tracked by the watchdog in the `Timeout()` function, it is meaningless to assign an already expired deadline to the watchdog. Every time a postponed job is released, the variable `postponed_jobs` is decreased by one. When the variable `postponed_jobs` is not 0, the scheduler is in the **postponed** mode.

Based on the original design in RTEMS, every time `Period()` is called at the beginning of the loop, it checks the state of the current period. When the state is EXPIRED, in fact the period might be expired many times already. In this block, now it calls the enhanced `Block_while_expired()` to release the postponed jobs without assigning a new deadline. We added one more condition when checking if the state of the period is ACTIVE, in which the task is blocked and waits for the next period. The additional condition checks if the variable `postponed_jobs` is greater than 0, which means the scheduler is still in the **postponed** mode. Otherwise, when the variable `postponed_jobs` is 0, the scheduler is in the **normal** mode and the job release is the same as it was in the original design.

Example with Enhancement

We use the same example as before in Section 1.2.4 to illustrate the effect of our enhancement in Figure 7.1a, which is now matching the expectation of most applications and researches. At time 10 τ_1 is executed, as it has higher priority than τ_2, which leads to three expired periods of τ_2. This means, the watchdog increases the

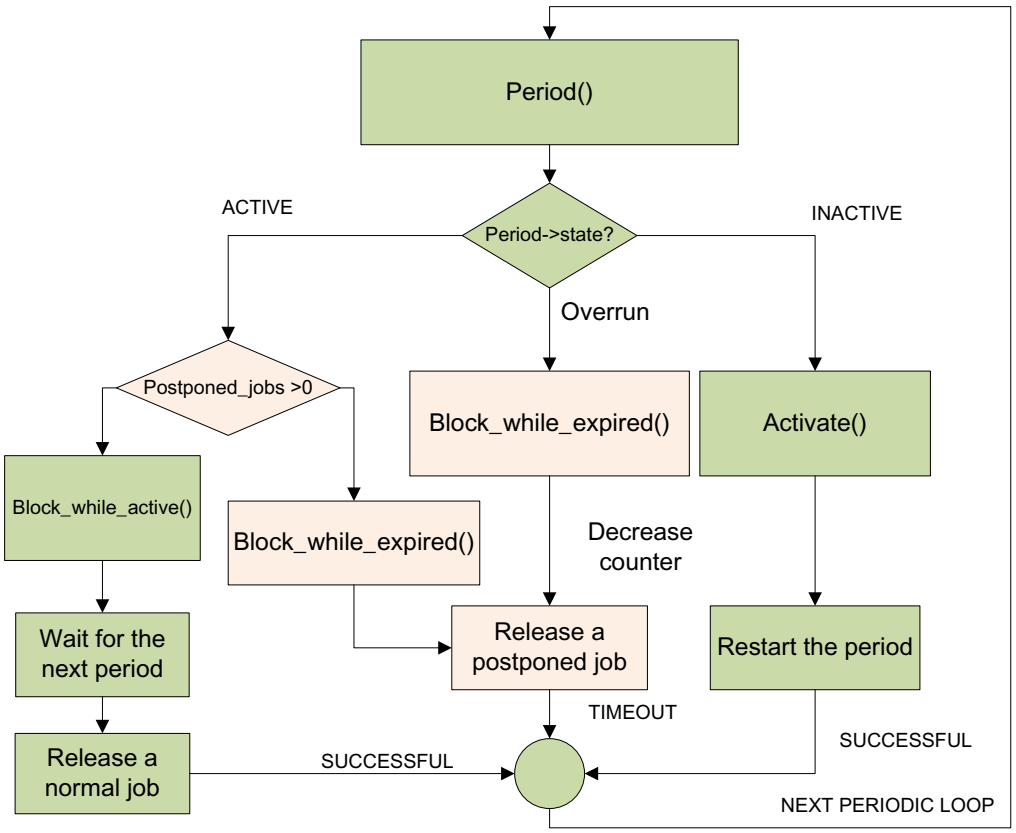


Figure 7.3: Flowchart of the enhanced RMS manager. The light background blocks are involved in the enhancement.

variable `postponed_jobs` to 3. When task τ_2 can be executed, it starts to release the postponed jobs in the postponed mode. Moreover, the three postponed jobs of τ_2 that should be released at time units 10, 12, and 14 due to the execution of τ_1, marked red, are released at time units 16, 17, and 18, respectively. However, the period from 16 to 18 and from 18 to 20 also expired while executing the three previously postponed jobs and the job postponed at 16, marked orange. At 20 the job postponed at 18 (marked orange as well) is released and is finished at 21, at which time the job postponed at 20 is released (yellow) and finishes at 22. After all the postponed jobs are finished, the release of τ_2 turns back according to the original pattern again.

We provide an additional example in Figure 7.1b as well to demonstrate how the enhancement works for dynamic real-time guarantees [BCH+16]. Detailed notation can be found in Section 7.4 or in [BCH+16]. Suppose that task τ_1 requires a full timing guarantee with the abnormal execution time $C_1^A = 6$, and the normal execution time $C_1^N = 1$. Task τ_2 is a timing tolerable task with $C_2^A = C_2^N = 1$. In Figure 7.1b, the second job of task τ_1 needs 6 time units for its execution time. We can see that after all the postponed jobs of task τ_2 are finished at 24, the release of task τ_2 turns back according to the original periodic pattern again.

7.3.1 Helper Function for Online-Monitoring

In the enhancement, we also provided a helper function called `Postponed_num()` to return the number of postponed jobs with the current period ID of tasks as function input. According to the expected behavior, the number of postponed jobs is only increased by the watchdog of the corresponding period; it is only decreased by the routine of postponed job releasing in RMS manager. This helper function is especially useful for on-line admission control and the system monitor design in terms of scheduling. For example, it is already used in [BCH+16] for the system state analysis, where the overhead of the enhancement is negligible in our evaluation (see Section 7.4).

7.3.2 Arbitrary Deadline and Non-Rebooting Policy

Up to this point, we have presented how to handle the overrun for implicit- ($D_i = T_i \ \forall \tau_i \in \Gamma$) or constrained-deadline ($D_i \leq T_i \ \forall \tau_i \in \Gamma$) task sets properly with our enhancement based on the original scheduler design in RTEMS. In the arbitrary-deadline task model, no general relation between D_i and T_i exists. Especially, for some tasks $D_i > T_i$ is possible. However, due to the limitation of the original design in the fixed-priority scheduler, the arbitrary-deadline task model is not supported yet in RTEMS as well. Since the deadline detection is originally embedded in the routine of the task periodicity, the deadline is expected to be less than or equal to their period without any overrun. However, this expectation is only true for implicit-deadline and constraint-deadline task models and applications where all deadlines are met.

From the schedulability analysis aspect, the detection of deadline misses should be separated as an individual feature. One potential solution is to set the deadline of a task explicitly as the input parameter while the period is initialized and update the deadline accordingly while every job is released. The detection routine for deadline misses should be revised for recording the number of deadline misses rather than the number of periods expired. Based on our enhancement, this solution could make the fixed-priority scheduler of RTEMS support more general real-time task models.

In addition, the enhancement proposed in this chapter is also necessary to the considered system model in the previous Chapter that all jobs are never aborted or the system is never *rebooted* while a job misses its deadline. Since the remaining part of the job still has to be finished fully before the next job of the same task can execute, the proposed enhancement can handle the postponed released jobs correctly while keeping the periodicities for all the tasks.

7.4 Case Study: Dynamic Real-Time Guarantees

The need for the presented enhanced implementation for RTEMS was discovered during the work on the paper *Systems with Dynamic Real-Time Guarantees in Uncertain and Faulty Execution Environments* [BCH+16]. A *System with Dynamic*

Real-Time Guarantees can be used to analyze and schedule systems where some tasks have two different Worst-Case Execution Time (WCET); a shorter WCET C_i^N for executions that happens more often (called normal execution), and a longer WCET C_i^A for some rare special cases (called abnormal execution). The general idea is that not only all tasks in the system normally need to fulfill strict timing guarantees but also in some special cases, i.e., a number of tasks with abnormal execution happen in a short period of time, for some not so important tasks (called *timing tolerable* tasks) rare deadline misses are tolerable while for the more important tasks (called *timing strict* tasks) deadline misses are allowed under no circumstances. In *Systems with Dynamic Real-Time Guarantees* fixed-priority scheduling is used.

A *System with Dynamic Real-Time Guarantees* assumes that at the beginning of a task's execution it is not possible to determine if the task is executed in a normal or an abnormal mode, i.e., abnormal executions happen randomly and do not follow a strict pattern. This is the case when we look at the fault tolerance enhanced to handle soft errors, i.e., the consequences of transient faults of the computing hardware or the memory subsystem, by using software-based solutions, e.g., re-execution [MD11] or checkpointing [ELS+13]. Such faults can either happen for each individual task job with low probability or they can happen as a burst that affects (nearly) all tasks over a small time window. In both cases it would not be sensible to in general assume the longer C_i^N in the analysis if those faults occur rarely and some deadline misses can be tolerated, as it would lead to over dimensioning the system. The idea of *Systems with Dynamic Real-Time Guarantees* is to give *full timing guarantees*, i.e., all tasks meet all their deadlines, if tasks are executed normally, and maybe downgrade this guarantees to *limited timing guarantees* if some tasks are executed in the abnormal mode. When *limited timing guarantees* are given, only the *timing strict* tasks are guaranteed to meet their deadlines while the *timing tolerable* tasks may miss some deadlines but still bounded tardiness for these tasks is guaranteed. In addition, *Systems with Dynamic Real-Time Guarantees* provide a system monitor that analyzes if *full timing guarantees* can be given for all tasks, i.e., all tasks will meet their deadline if no faults occur, or if only *limited timing guarantees* can be given for some *timing tolerable* tasks. To determine this for each *timing tolerable* tasks an over estimation of the busy period is calculated, summing up the current carry-in workload by jobs with higher or identical priority and the workload created in the future under the assumptions of 1) a worst-case release pattern and 2) that no further faults occur. For details see Section 6 of [BCH+16].

Similar behavior occurs in Mixed-Criticality Systems [Ves07] which have two modes, a high- and a low-criticality mode where tasks have a longer execution time in the high-criticality mode. It is often assumed that low-criticality tasks can be abandoned when the system switches from the low-criticality to high-criticality mode to ensure that the high-criticality tasks will still meet their deadlines. However, this assumption has been criticized recently [ENN+15; EN16; BD16]. The problem is tackled when a *System with Dynamic Real-Time Guarantees* is used, as the low-

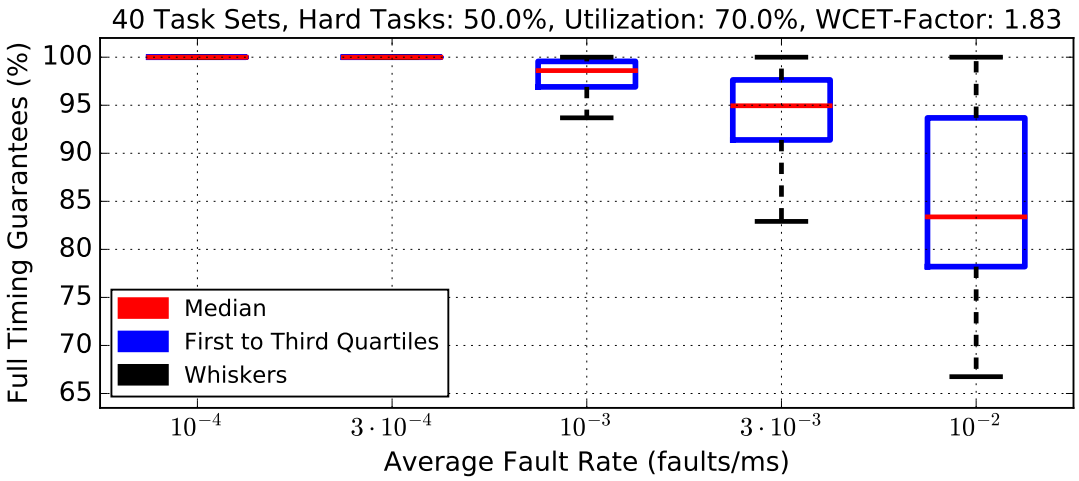


Figure 7.4: Percentage of Time where `Full Timing Guarantees` can be given for task sets with utilization 70% in the normal mode under different fault rates.

criticality tasks are seen as *timing tolerable* tasks and the system gives *limited timing guarantees*, i.e., guarantees bounded tardiness instead of abandoning the task.

To analyze the behavior of *Systems with Dynamic Real-Time Guarantees*, RTEMS on a Quick Emulator (QEMU) emulator was used, where the number of cores was set to 1 for the emulation. It was assumed that transient faults happen randomly, i.e., a given rate of faults per millisecond, and at the moment a task job finished its normal execution a random draw, based on the probability of faults per millisecond and C_i^N, determined if the execution was prolonged to run up to C_i^A or not. The system monitor was used to determine the amount of time when only *limited timing guarantees* could be provided.

As the *timing tolerable* tasks are not abandoned those tasks may miss their deadlines. However, they should be executed after the deadline as the result may still be useful. In addition, it may happen that not only one job of a task misses the deadline but also that the execution of the task may be postponed for more than one period. In these cases more than one job of a task may be ready to execute at a given time, another situation previously not covered in RTEMS. The enhancement presented in Section 7.3 was necessary to ensure that the release pattern was still correct when a task missed its deadline. The number of the postponed releases was determined using the helper function. The system monitor framework also adopts the helper function when it calculates the carry-in workload for each task.

The scheduling algorithm in [BCH+16] can only schedule 44.4% of the task sets when the task sets are randomly generated with 10 tasks, 50% of these tasks are randomly chosen to be *timing strict* tasks, the total utilization in the normal mode is 70%, and $C_i^A = 1.83 \cdot C_i^N \forall i$, i.e., the total utilization in the abnormal mode is $\approx 128.1\%$. We used 40 of those randomly created tasks sets under the given setting that

are schedulable according to the scheduling algorithm in [BCH+16] if the bounded tardiness condition for the *timing tolerable* tasks is dropped. Obviously, for utilization $> 100\%$ in the abnormal mode, bounded tardiness for the *timing tolerable* tasks can only be guaranteed if the fault rate is not too high as a high fault rate will lead to more overruns.

We let the system simulate one hour per run under different fault rates for each of those task sets, i.e., on average 10^{-4}, $3 \cdot 10^{-4}$, $10^{-3}, 3 \cdot 10^{-3}$ and 10^{-2} faults per millisecond (f/ms). For each executed task job we decided if the job was faulty or not by a random draw. The results, i.e., the percentage of time the system was running with *full timing guarantees*, are shown in Figure 7.4 (which is Figure 8 in [BCH+16]). The median of those 40 sets is colored red. The blue box represents the interval from the first to the third quartile, while the black whiskers show the minimum and maximum of all of the data.

7.5 Summary

The demand of overrun handling has emerged and it is widely used in practical and theoretical systems, e.g., in the design of soft real-time systems, Mixed-Criticality systems [Ves07], *Systems with Dynamic Real-Time Guarantees* [BCH+16], and the system model without rebooting/aborting policy for deadline misses. RTOSs are not only used in embedded real-time systems but also useful for the simulation and validation of those systems. Therefore, a RTOS should ensure that the system can behave properly as expected in the literature when such overrun situations occur.

In this chapter, we introduce how to enhance the fixed-priority scheduler in the released version 4.11 of RTEMS with a generally expected overrun handling mechanism. The provided enhancement was accepted on January 30th in 2017 as stated in the RTEMS report ticket #2795 [Che16b] for version 4.11 and is inherited by the latest version 5.1. In addition to the proposed enhancement, we also prepare the corresponding test-suites on uniprocessors obeying the standard validation and the coding convention in RTEMS. At the end, we thank Mr. Huan-Fui Lee for his assistance on deploying the case study.

CHAPTER 8

Conclusion and Outlook

Contents

In the preceding chapters of this dissertation, it has been shown that the usage of Software-Implemented Hardware Fault Tolerance (SIHFT) techniques in uniprocessor or multi-core systems poses a number of novel challenges with regards to satisfying and proving real-time guarantees. For instance, the classical non-probabilistic approaches proposed in the literature may be too pessimistic and thus lead to over-provisioned systems. Towards this, several published and peer-reviewed analyses and scheduling algorithms have been presented in this dissertation that are beneficial to designing and verifying resource-efficient real-time systems with respect to fault-tolerance and reliability requirements.

8.1 Summary

Firstly, the concepts of SIHFT techniques have been introduced, which are applied in software systems to mitigate the effects of transient faults in hardware without additional hardware support. Further, this dissertation was giving an overview of real-time systems, and the prominent techniques to verify the timeliness of safety critical systems.

The studied problems in this dissertation are summarized as follows: In Chapter 4, we have studied how to reduce the analytic pessimism while deploying SIHFT techniques in hard real-time systems. We have shown that the inherent robustness, i.e., safety margins and noise tolerance in control applications can be modeled as (m, k) robustness requirements and be exploited to avoid over-provisioning when applying software-based error-handling approaches against soft-errors. Motivated by

this observation, we have presented a lightweight off-line static pattern scheduling, which uses (m, k) patterns to determine when to execute a reliable version, and a dynamic compensation algorithm for optimizations during runtime. The dynamic compensation is based on static pattern scheduling to monitor the current *tolerance status* and determine the necessary instant in time to execute a reliable version subsequently.

Moreover, in Chapter 5 we have studied how to efficiently analyze and compute the probability of deadline misses and miss rates incurred by the overheads of potential error recovery mechanisms in systems with soft real-time constraints. We have shown that the probability of deadline misses can be safely over-approximated by using analytical bound approaches. Further, it was shown that the differences in approximation quality, comparing to the state-of-the-art, are reasonable in light of the runtime improvement. Based on the proposed approximation approach and accounting for consecutive deadline misses, we have presented that a safe upper bound on the expected deadline misses rate can be derived efficiently.

Furthermore, in Chapter 6 we have studied how to optimize the system reliability via Redundant Multi-Threading (RMT) on multi-core systems whilst satisfying given real-time constraints. We have proposed to solve reliability-aware task-mapping problems based on the Minimum Weight Perfect Bipartite Matching (MWPBM) problem to improve the overall system reliability. In addition, we have developed several dynamic programming approaches to decide redundancy levels for each task whilst adopting Federated Scheduling (FS) to handle multi-tasking.

Finally, in Chapter 7 we have proposed an enhancement for implementations in a Real Time Operating System (RTOS) to guarantee strict periodicity for potential task overruns due to hardware transient faults. We have introduced how to enhance the implementation of the fixed-priority scheduler in Real-Time Executive for Multiprocessor Systems (RTEMS) with a overrun handling mechanism that provides the aforementioned periodicity. Additionally, the proposed enhancement has been reviewed and was accepted on January 30th in 2017 and is included in the latest version of released source code.

8.2 Ongoing and Future Work

Tolerance-Aware Analysis and Scheduling: The rational behind the proposed approaches in Chapter 4, motivated us to consider similar scenarios in different applications and domains, e.g., wireless communication with limited bandwidths, or network communication systems with limited reliable channels. For example, the reliable transmission of messages are typically costly and thus the number of reliable transmission channels is limited.

How to efficiently monitor the *tolerance status* and improve the correct transmission in such systems, i.e., resource-aware scheduling, is a foreseeable challenge.

Investigating Impacts of Different System Models: The considered scheduling models in this dissertation are solely focused on fixed-priority preemptive scheduling policy. Thus, researching and transferring our proposed results for different scheduling and task models, e.g., arbitrary deadlines, non-preemptive scheduling policy, and dynamic priority scheduling is an open problem. Furthermore, the costs of using SIHFT techniques in this dissertation is only considered by their time overheads and is assumed to perform always correctly. In future work, we would like to consider different models of SIHFT techniques and analyze their respective impacts.

Extension of RTOS and Open Source: As described in Chapter 7, we have noticed that the implementations of task models and overrun-handling in RTOSs may not fully conform to what is premised and believed in the literature. Therefore, making them consistent with the assumptions made in the analyses is mandatory and an important future work. Moreover, the implementations of the proposed approaches, presented in this dissertation, are released and publicly available. In order to further improve the applicability of the released scripts, we would like to extend our contributions, e.g., the event-based simulator mentioned in Chapter 2, to consider multi-core system models and dynamic priority assignment policies.

Appendices

List of Figures

List of Tables

List of Algorithms

Acronyms

ACS	Attitude Control System 2
CRT	Chip-level Redundant Multi-threading 5, 9–12, 23, 31, 100–103, 105–113, 119, 122, 132–135, 137
DC	Dynamic Compensation 46, 47, 49, 50, 52, 53, 58, 60
DM	Deadline-Monotonic 19, 43
DMP	Probability of Deadline Misses 66–68, 71, 72, 74, 77, 78, 88–90, 158, 161
DMR	Double Modular Redundancy 9–11, 23, 31, 101, 122, 132
E-pattern	Evenly Distributed Pattern 21, 22, 53, 58, 59
ECC	Error-Correcting Code 2
EDF	Earliest-Deadline-First 20
ESA	European Space Agency 14, 60
ETCS	Electronic Throttle Control System 2
FCFS	First-Come-First-Serve 12, 140
FS	Federated Scheduling 12, 24, 116–118, 120–122, 124, 126, 135, 138, 152
HA	Hungarian Algorithm 11, 12, 105, 107–109, 111, 115
ICF	Incorrect Computation Faults 4
ILA	Iterative Level Adaptation 11, 12, 104, 111

IRU	Inertial Reference Unit 2
JAXA	Japan Aerospace Exploration Agency 2
L&L	Liu and Layland 20, 43
LLF	Least-Laxity-First 20
MGF	Moment Generating Function 62, 68, 69
MRT	Mixed Redundant Threading 10–12, 31, 101–103, 122, 132–135, 137, 138
MWPBM	Minimum Weight Perfect Bipartite Matching 105, 107–109, 137, 152
NMR	N-Modular Redundancy 23
NOC	Network on Chip 32, 33
OSEK	Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen 25, 33
PST	Probabilistic Schedulability Tests 8, 9, 14
QEMU	Quick Emulator 148
R-pattern	Deep Red Pattern 21, 53, 55, 58, 59
RISC	Reduced Instruction Set Computing 32, 103, 104
RM	Rate-Monotonic 19, 34, 43, 72, 77, 117
RMT	Redundant Multi-Threading 4, 5, 9–12, 14, 17, 23, 32, 100–102, 105, 107, 111, 114–116, 128, 131, 133, 137, 152, 157, 158
RTEMS	Real-Time Executive for Multiprocessor Systems vii, 12–15, 19, 24, 37, 38, 60, 139–142, 144, 146, 148, 149, 152, 159
RTOS	Real Time Operating System 19, 33, 139, 149, 152, 153
RW	Reaction Wheel 2
SDC	Silent Data Corruption 28

SEU	Single Event Upset 3, 5, 28, 42
SIHFT	Software-Implemented Hardware Fault Tolerance vii, 2–5, 7, 8, 12, 21, 29, 30, 40, 41, 62, 66, 151, 153
SOCIS	Summer of Code in Space 14, 60
SRE	Static Pattern-Based Reliable Execution 44, 50, 52, 53, 56–60
SRT	Simultaneous Redundant Multithreading 3, 5, 9–12, 23, 31, 58, 101–103, 132, 134, 135, 137
TDA	Time-Demand Analysis 20, 43, 67–70
TDMA	Time Division Multiple Access 19
TMR	Triple Modular Redundancy 5, 9–11, 23, 31, 32, 100–103, 105–113, 115, 119, 122, 124, 132–137, 157
VDX	Vehicle Distributed eXecutive 25
WCET	Worst-Case Execution Time 3, 4, 20, 28–32, 38, 43, 58, 62, 65–69, 73, 83, 104, 122, 132, 147

Bibliography

[AE13] P. Axer and R. Ernst. “Stochastic response-time guarantee for non-preemptive, fixed-priority scheduling under errors”. In: *Design Automation Conference (DAC)*. 05/2013, pp. 1–7 (Cited on pages 8, 18, 22).

[AJ03] B. Andersson and J. Jonsson. “The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%”. In: *Real-Time Systems, 2003. Proceedings. 15th Euromicro Conference on*. 2003, pp. 33–40 (Cited on pages 101, 117 sq.).

[Ale16] Alexandra Witze. *"Software error doomed Japanese Hitomi spacecraft"*. 2016. URL: https://www.nature.com/news/software-error-doomed-japanese-hitomi-spacecraft-1.19835 (visited on 05/14/2018) (Cited on page 2).

[Alt13] Altera Corporation. *White paper: Introduction to single-event upsets*. https://www.altera.com/en_US/pdfs/literature/wp/wp-01206-introduction-single-event-upsets.pdf. 09/2013 (Cited on pages 3, 5, 42).

[AQN+13] P. Axer, S. Quinton, M. Neukirchner, R. Ernst, B. Döbel, and H. Härtig. “Response-Time Analysis of Parallel Fork-Join Workloads with Real-Time Constraints”. In: *Real-Time Systems (ECRTS), 25th Euromicro Conference on*. 2013, pp. 215–224 (Cited on pages 23 sq., 122).

[ARB+05] R. Azevedo, S. Rigo, M. Bartholomeu, G. Araujo, C. C. de Araujo, and E. Barros. “The ArchC Architecture Description Language and Tools”. In: *International Journal of Parallel Programming* (2005) (Cited on page 36).

[ARM13] ARM. *big.LITTLE Technology: The Future of Mobile*. 2013. URL: http://www.arm.com/files/pdf/big_LITTLE_Technology_the_Futue_of_Mobile.pdf (Cited on pages 5, 9, 100, 127).

[Bau05a] R. C. Baumann. “Radiation-induced soft errors in advanced semiconductor technologies”. In: *IEEE Transactions on Device and Materials Reliability* 5.3 (09/2005), pp. 305–316. ISSN: 1530-4388. DOI: 10.1109/TDMR.2005.853449 (Cited on page 62).

[Bau05b] R. Baumann. “Soft Errors in Advanced Computer Systems”. In: *IEEE Des. Test* 22.3 (05/2005), pp. 258–266. ISSN: 0740-7475. DOI: 10.1109/MDT.2005.69. URL: http://dx.doi.org/10.1109/MDT.2005.69 (Cited on page 2).

[BB04] E. Bini and G. C. Buttazzo. “Schedulability analysis of periodic fixed priority systems”. In: *IEEE Transactions on Computers* 53.11 (11/2004), pp. 1462–1473. ISSN: 0018-9340. DOI: 10.1109/TC.2004.103 (Cited on page 71).

[BB05] E. Bini and G. C. Buttazzo. "Measuring the Performance of Schedulability Tests". In: *Real-Time Systems* 30.1-2 (2005), pp. 129–154. DOI: 10.1007/s11241-005-0507-9. URL: https://doi.org/10.1007/s11241-005-0507-9 (Cited on pages 33 sq., 58, 77, 87).

[BBC+07] B. B. Brandenburg, A. D. Block, J. M. Calandrino, U. Devi, H. Leontyev, and J. H. Anderson. *LITMUS RT: A Status Report*. 2007 (Cited on page 24).

[BBL+03] S. A. Brandt, S. Banachowski, C. Lin, and T. Bisson. "Dynamic integrated scheduling of hard real-time, soft real-time, and non-real-time processes". In: *Real-Time Systems Symposium (RTSS)*. 2003 (Cited on page 23).

[BCH+16] 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 Environment". In: *Real-Time Systems Symposium, 2016. Proceedings., 37th.* 2016 (Cited on pages 140 sq., 145–149).

[BCM+13] C. Bolchini, M. Carminati, A. Miele, A. Das, A. Kumar, and B. Veeravalli. "Run-time mapping for reliable many-cores based on energy/performance tradeoffs". In: *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*. 10/2013, pp. 58–64 (Cited on page 100).

[BD16] A. Burns and R. Davis. *Mixed criticality systems-a review*. Tech. rep. 7th edition. University of York, 2016 (Cited on page 147).

[BDM02] K. Bowman, S. Duvall, and J. Meindl. "Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration". In: *Solid-State Circuits, IEEE Journal of* 37.2 (2002), pp. 183–190 (Cited on pages 5, 9, 100, 127).

[BDP96] A. Burns, R. Davis, and S. Punnekkat. "Feasibility analysis of fault-tolerant real-time task sets". In: *Proceedings of the Eighth Euromicro Workshop on Real-Time Systems*. 06/1996, pp. 29–33. DOI: 10.1109/EMWRTS.1996.557785 (Cited on page 4).

[BF93] R. W. Butler and G. B. Finelli. "The infeasibility of quantifying the reliability of life-critical real-time software". In: *IEEE Transactions on Software Engineering* 19.1 (01/1993), pp. 3–12. ISSN: 0098-5589. DOI: 10.1109/32.210303 (Cited on page 63).

[BKH+16] von der Brüggen Georg, **Kuan-Hsun Chen**, W.-H. Huang, and J.-J. Chen. "Systems with Dynamic Real-Time Guarantees in Uncertain and Faulty Execution Environments". In: *IEEE Real-Time Systems Symposium (RTSS)*. 11/2016, pp. 303–314. DOI: 10.1109/RTSS.2016.037 (Cited on pages 12, 19, 30, 37 sq., 62, 66 sq.).

[BLA+05] G. Buttazzo, G. Lipari, L. Abeni, and M. Caccamo. *Soft Real-Time Systems: Predictability vs. Efficiency (Series in Computer Science)*. Plenum Publishing Co., 2005. ISBN: 0387237011 (Cited on page 18).

[BMC16] S. Ben-Amor, D. Maxim, and L. Cucu-Grosjean. "Schedulability Analysis of Dependent Probabilistic Real-time Tasks". In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems (RTNS)*. 2016 (Cited on pages 8, 18, 22, 62, 64, 68).

[Bor05] S. Borkar. "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation". In: *IEEE Micro* 25.6 (11/2005), pp. 10–16. ISSN: 0272-1732. DOI: 10.1109/MM.2005.110 (Cited on page 2).

[BPK+18] G. v. d. Brüggen, N. Piatkowski, **Kuan-Hsun Chen**, J.-J. Chen, and K. Morik. "Efficiently Approximating the Probability of Deadline Misses in Real-Time Systems". In: *2018 30th EUROMICRO Conference on Real-Time Systems (ECRTS)*. 07/2018 (Cited on pages 8, 15, 22, 62 sqq., 68, 77 sq., 83 sq., 86 sq., 91).

[Bra06] B. Brandenburg. *LITMUS^{RT}: Linux Testbed for Multiprocessor Scheduling in Real-Time Systems*. 2006. URL: http://www.litmus-rt.org/ (Cited on pages 24, 139).

[BS14] G. Bloom and J. Sherrill. "Scheduling and Thread Management with RTEMS". In: *SIGBED Rev.* 11.1 (02/2014), pp. 20–25. ISSN: 1551-3688. DOI: 10.1145/2597457.2597459. URL: http://doi.acm.org/10.1145/2597457.2597459 (Cited on page 142).

[BS15] T. Bund and F. Slomka. "Sensitivity Analysis of Dropped Samples for Performance-Oriented Controller Design". In: *Real-Time Distributed Computing (ISORC), 2015 IEEE 18th International Symposium on*. 04/2015, pp. 244–251. DOI: 10.1109/ISORC.2015.16 (Cited on pages 21, 40, 43, 52 sq.).

[BSS13] C. Borchert, H. Schirmeier, and O. Spinczyk. "Generative software-based memory error detection and correction for operating system data structures". In: *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 06/2013, pp. 1–12. DOI: 10.1109/DSN.2013.6575308 (Cited on page 3).

[Buc04] J. Bucklew. *Introduction to Rare Event Simulation*. 1st. New York, NY, USA: Springer-Verlag, 2004. ISBN: 9781475740783 (Cited on page 69).

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004 (Cited on page 79).

[CBS10] CBS News / AP. *"Toyota "unintended acceleration" has killed 89"*. May 2010. URL: http://www.cbsnews.com/news/toyota-unintended-acceleration-has-killed-89/ (visited on 05/10/2018) (Cited on page 2).

[CCK+06] G. Chen, G. Chen, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin. "Object duplication for improving reliability". In: *Asia and South Pacific Conference on Design Automation, 2006.* 01/2006. DOI: 10.1109/ASPDAC.2006.1594672 (Cited on page 3).

[Che16a] J.-J. Chen. "Federated scheduling admits no constant speedup factors for constrained-deadline DAG task systems". In: *Real-Time Systems* 52.6 (2016), pp. 833–838 (Cited on page 138).

[Che16b] K.-H. Chen. *#2795 ticket: Overrun Handling for general real-time models*. http://devel.rtems.org/ticket/2795. 2016. URL: http://devel.rtems.org/ticket/2795 (Cited on page 149).

[Chi13] T. Chikamasa. *nxtOSEK*. 2013. URL: http://lejos-osek.sourceforge.net/ (Cited on page 33).

[CHL15] J.-J. Chen, W.-H. Huang, and C. Liu. "k2U: A General Framework from k-Point Effective Schedulability Analysis to Utilization-Based Tests". In: *Real-Time Systems Symposium, 2015 IEEE*. 12/2015, pp. 107–118. DOI: 10.1109/RTSS.2015.18 (Cited on page 71).

[Con02] C. Constantinescu. "Impact of deep submicron technology on dependability of VLSI circuits". In: *Proceedings International Conference on Dependable Systems and Networks*. 2002, pp. 205–209. DOI: 10.1109/DSN.2002.1028901 (Cited on page 2).

[Con03] C. Constantinescu. "Trends and Challenges in VLSI Circuit Reliability". In: *IEEE Micro* 23.4 (07/2003), pp. 14–19. ISSN: 0272-1732. DOI: 10.1109/MM.2003.1225959. URL: http://dx.doi.org/10.1109/MM.2003.1225959 (Cited on page 2).

[CRA06] J. Chang, G. A. Reis, and D. I. August. "Automatic Instruction-Level Software-Only Recovery". In: *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*. 06/2006, pp. 83–92. DOI: 10.1109/DSN.2006.15 (Cited on page 29).

[DB09] R. I. Davis and A. Burns. "Priority Assignment for Global Fixed Priority Pre-Emptive Scheduling in Multiprocessor Real-Time Systems". In: *2009 30th IEEE Real-Time Systems Symposium*. 12/2009, pp. 398–409. DOI: 10.1109/RTSS.2009.31 (Cited on page 33).

[DB11] R. I. Davis and A. Burns. "Improved Priority Assignment for Global Fixed Priority Pre-emptive Scheduling in Multiprocessor Real-time Systems". In: *Real-Time Syst.* 47.1 (01/2011), pp. 1–40 (Cited on page 132).

[DC19] R. Davis and L. Cucu-Grosjean. "A Survey of Probabilistic Schedulability Analysis Techniques for Real-Time Systems". In: *Leibniz Transactions on Embedded Systems* 6.1 (2019), 04–1-04:53. ISSN: 2199-2002. DOI: 10.4230/LITES-v006-i001-a004. URL: https://ojs.dagstuhl.de/index.php/lites/article/view/LITES-v006-i001-a004 (Cited on page 22).

[DGK+02] J. L. Diaz, D. F. Garcia, K. Kim, C.-G. Lee, L. L. Bello, J. M. Lopez, S. L. Min, and O. Mirabella. "Stochastic analysis of periodic real-time systems". In: *Real-Time Systems Symposium, 23rd IEEE*. 2002, pp. 289–300 (Cited on pages 8, 18, 22, 62, 64, 66, 68).

[DKV13] A. Das, A. Kumar, and B. Veeravalli. "Reliability-driven task mapping for lifetime extension of networks-on-chip based multiprocessor systems". In: *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*. 03/2013, pp. 689–694 (Cited on page 100).

[DVA+11] S. Dighe, S. R. Vangal, P. Aseron, S. Kumar, T. Jacob, K. A. Bowman, J. Howard, J. Tschanz, V. Erraguntla, N. Borkar, V. K. De, and S. Borkar. "Within-Die Variation-Aware Dynamic-Voltage-Frequency-Scaling With Optimal Core Allocation and Thread Hopping for the 80-Core TeraFLOPS Processor". In: *IEEE Journal of Solid-State Circuits* 46.1 (01/2011), pp. 184–193 (Cited on page 9).

[DW11] A. Dixit and A. Wood. "The impact of new technology on soft error rates". In: *2011 International Reliability Physics Symposium*. 04/2011, 5B.4.1–5B.4.7. DOI: 10.1109/IRPS.2011.5784522 (Cited on page 2).

[DZB08] R. I. Davis, A. Zabos, and A. Burns. "Efficient Exact Schedulability Tests for Fixed Priority Real-Time Systems". In: *IEEE Transactions on Computers* 57.9 (09/2008), pp. 1261–1276. ISSN: 0018-9340. DOI: 10.1109/TC.2008.66 (Cited on pages 34, 58).

[ELS+13] I. P. Egwutuoha, D. Levy, B. Selic, and S. Chen. "A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems". In: *The Journal of Supercomputing* 65.3 (09/2013), pp. 1302–1326. ISSN: 1573-0484. DOI: 10.1007/s11227-013-0884-0. URL: https://doi.org/10.1007/s11227-013-0884-0 (Cited on pages 4, 12, 62, 147).

[EN16] R. Ernst and M. D. Natale. "Mixed Criticality Systems - A History of Misconceptions?" In: *IEEE Design & Test* 33.5 (2016), pp. 65–74. DOI: 10.1109/MDAT.2016.2594790. URL: http://dx.doi.org/10.1109/MDAT.2016.2594790 (Cited on page 147).

[ENN+15] A. Esper, G. Nelissen, V. Nélis, and E. Tovar. "How Realistic is the Mixed-criticality Real-time System Model?" In: *RTNS*. 2015, pp. 139–148 (Cited on page 147).

[ESD10] P. Emberson, R. Stafford, and R. Davis. "Techniques For The Synthesis Of Multiprocessor Tasksets". In: *WATERS workshop at the Euromicro Conference on Real-Time Systems*. 7/2010, pp. 6–11 (Cited on page 33).

[ESH+11] M. Engel, F. Schmoll, A. Heinig, and P. Marwedel. "Unreliable yet Useful – Reliability Annotations for Data in Cyber-Physical Systems". In: *Proceedings of the 2011 Workshop on Software Language Engineering for Cyber-physical Systems (WS4C)*. Berlin / Germany, 10/2011 (Cited on page 29).

[Evi16] Evidence Srl. *ERIKA Enterprise*. 2016. URL: http://www.erika-enterprise.com/ (Cited on pages 24 sq.).

[FGI09] I. Finocchi, F. Grandoni, and G. F. Italiano. "Optimal resilient sorting and searching in the presence of memory faults". In: *Theoretical Computer Science* 410.44 (2009). Automata, Languages and Programming (ICALP 2006), pp. 4457–4470. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2009.07.026. URL: http://www.sciencedirect.com/science/article/pii/S0304397509005003 (Cited on page 6).

[FR13] S. Foucart and H. Rauhut. *A Mathematical Introduction to Compressive Sensing*. Springer New York, 2013. DOI: 10.1007/978-0-8176-4948-7. URL: https://doi.org/10.1007/978-0-8176-4948-7 (Cited on pages 62, 73).

[Gar00] P. Gargini. "The International Technology Roadmap for Semiconductors (ITRS): "Past, present and future"". In: *GaAs IC Symposium. IEEE Gallium Arsenide Integrated Circuits Symposium. 22nd Annual Technical Digest 2000. (Cat. No.00CH37084)*. 11/2000, pp. 3–5. DOI: 10.1109/GAAS.2000.906261 (Cited on page 1).

[GBL+00] P. Gai, E. Bini, G. Lipari, M. D. Natale, and L. Abeni. "Architecture For A Portable Open Source Real Time Kernel Environment". In: *In Proceedings of the Second Real-Time Linux Workshop and Hand's on Real-Time Linux Tutorial*. 2000 (Cited on page 25).

[GDD19] S. Ghosh, S. Dey, and P. Dasgupta. "Synthesizing Performance-Aware (m, k)-Firm Control Execution Patterns Under Dropped Samples". In: *2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)*. 01/2019, pp. 1–6. DOI: 10.1109/VLSID.2019.00019 (Cited on pages 21, 43).

[GM08] S. Garg and D. Marculescu. "System-level Throughput Analysis for Process Variation Aware Multiple Voltage-frequency Island Designs". In: *ACM Trans. Des. Autom. Electron. Syst.* 13.4 (2008), 59:1–59:25 (Cited on page 127).

[GNB18] A. Gujarati, M. Nasri, and B. B. Brandenburg. "Quantifying the Resiliency of Fail-Operational Real-Time Networked Control Systems". In: *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*. Ed. by S. Altmeyer. Vol. 106. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 16:1–16:24. ISBN: 978-3-95977-075-0. DOI: 10.4230/LIPIcs.ECRTS.2018.16. URL: http://drops.dagstuhl.de/opus/volltexte/2018/8988 (Cited on page 43).

[Gra66] R. L. Graham. "Bounds for Certain Multiprocessing Anomalies". In: *Bell System Technical Journal* 45.9 (1966), pp. 1563–1581. ISSN: 1538-7305. DOI: 10.1002/j.1538-7305.1966.tb01709.x (Cited on page 117).

[GRE+01] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. "MiBench: A Free, Commercially Representative Embedded Benchmark Suite". In: *WWC-4. IEEE International Workshop*. 2001 (Cited on page 36).

[GSY+12] N. Guan, M. Stigge, W. Yi, and G. Yu. "Parametric Utilization Bounds for Fixed-Priority Multiprocessor Scheduling". In: *IEEE 26th International Parallel and Distributed Processing Symposium*. 05/2012, pp. 261–272 (Cited on page 121).

[HAN12] S. K. S. Hari, S. V. Adve, and H. Naeimi. "Low-cost program-level detectors for reducing silent data corruptions". In: *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*. 06/2012, pp. 1–12. DOI: 10.1109/DSN.2012.6263960 (Cited on page 3).

[HBD+13] J. Henkel, L. Bauer, N. Dutt, P. Gupta, S. Nassif, M. Shafique, M. Tahoori, and N. Wehn. "Reliable On-Chip Systems in the Nano-Era: Lessons Learnt and Future Trends". In: *DAC*. 2013 (Cited on page 104).

[HGM12] S. Herbert, S. Garg, and D. Marculescu. "Exploiting Process Variability in Voltage/Frequency Control". In: *IEEE Trans. Very Large Scale Integr. Syst.* (2012) (Cited on page 104).

[Hil00] M. Hiller. "Executable assertions for detecting data errors in embedded control systems". In: *Proceeding International Conference on Dependable Systems and Networks. DSN 2000*. 2000, pp. 24–33. DOI: 10.1109/ICDSN.2000.857510 (Cited on page 3).

[HJS+06] L. He, S. A. Jarvis, D. P. Spooner, H. Jiang, D. N. Dillenberger, and G. R. Nudd. "Allocating non-real-time and soft real-time jobs in multiclusters". In: *IEEE Transactions on Parallel and Distributed Systems* 17.2 (2006), pp. 99–112 (Cited on page 23).

[HJS02] M. Hiller, A. Jhumka, and N. Suri. "On the placement of software mechanisms for detection of data errors". In: *Proceedings International Conference on Dependable Systems and Networks*. 2002, pp. 135–144. DOI: 10.1109/DSN.2002.1028894 (Cited on page 3).

[HLD+15] M. Hoffmann, F. Lukas, C. Dietrich, and D. Lohmann. "dOSEK: the design and implementation of a dependability-oriented static embedded kernel". In: *21st IEEE Real-Time and Embedded Technology and Applications Symposium*. 04/2015, pp. 259–270. DOI: 10.1109/RTAS.2015.7108449 (Cited on pages 32, 36).

[HM08] S. Herbert and D. Marculescu. "Characterizing Chip-multiprocessor Variability-tolerance". In: *DAC*. Anaheim, California, 2008, pp. 313–318 (Cited on page 104).

[Hoe63] W. Hoeffding. "Probability Inequalities for Sums of Bounded Random Variables". In: *Journal of the American Statistical Association* 58.301 (1963), pp. 13–30. ISSN: 01621459. URL: http://www.jstor.org/stable/2282952 (Cited on pages 62, 72).

[HSJ08] E. Henriksson, H. Sandberg, and K. H. Johansson. "Predictive compensation for communication outages in networked control systems". In: *47th IEEE Conference on Decision and Control*. 12/2008, pp. 2063–2068. DOI: 10.1109/CDC.2008.4739306 (Cited on pages 21, 40, 52 sq.).

[HWZ06] J. Hu, S. Wang, and S. Ziavras. "In-Register Duplication: Exploiting Narrow-Width Value for Improving Register File Reliability". In: *DSN*. 2006 (Cited on pages 31, 37, 132).

[Inta] Intel. *Intel Single-chip Cloud Computer*. URL: http://techresearch.intel.com/ProjectDetails.aspx?Id=1 (Cited on page 4).

[Intb] Intel. *Intel Xeon PHI™Product Family*. URL: http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html (Cited on page 4).

[Int00] International Organization for Standardization (ISO). "Iso/fdis26262: Road vehicles - functional safety". In: 2000 (Cited on page 2).

[Int10] International Electrotechnical Commission (IEC). "Functional safety of electrical / electronic / programmable electronic safety-related systems ed2.0". In: 2010 (Cited on page 2).

[IPE+12] V. Izosimov, P. Pop, P. Eles, and Z. Peng. "Scheduling and Optimization of Fault-Tolerant Embedded Systems with Transparency/Performance Trade-Offs". In: (05/2012), pp. 261–272 (Cited on page 100).

[ITR] ITRS. *System drivers, 2011*. URL: http://www.itrs.net (Cited on page 4).

[Jap16] Japan Aerospace Exploration Agency (JAXA). *Supplemental Handout on the Operation Plan of the X-ray Astronomy Satellite ASTRO-H (Hitomi)*. Tech. rep. 04/28/2016. URL: http://global.jaxa.jp/press/2016/04/files/20160428_hitomi.pdf (visited on 03/09/2017) (Cited on page 2).

[Joh+13] F. Johansson et al. *mpmath: a Python library for arbitrary-precision floating-point arithmetic (version 0.18)*. http://mpmath.org/. 12/2013 (Cited on page 80).

[JOP+01] E. Jones, T. Oliphant, P. Peterson, et al. *SciPy: Open source scientific tools for Python*. [Online; accessed <today>]. 2001–. URL: http://www.scipy.org/ (Cited on page 87).

[Jun13] Junko Yoshida. *"Toyota case: Single bit flip that killed"*. 2013. URL: http://www.eetimes.com/document.asp?doc_id=1319903 (visited on 05/10/2018) (Cited on page 2).

[KBC+16] **Kuan-Hsun Chen**, B. Bönninghoff, J.-J. Chen, and P. Marwedel. "Compensate or Ignore? Meeting Control Robustness Requirements Through Adaptive Soft-error Handling". In: *Proceedings of the 17th ACM SIGPLAN/SIGBED Conference on Languages, Compilers, Tools, and Theory for Embedded Systems*. LCTES 2016. Santa Barbara, CA, USA: ACM, 2016, pp. 82–91. DOI: 10.1145/2907950.2907952 (Cited on pages 14, 40).

[KBC16] **Kuan-Hsun Chen**, G. von der Brüggen, and J.-J. Chen. "Overrun Handling for Mixed-Criticality Support in RTEMS". In: *Workshop on Mixed Criticality Systems*. Proceedings of WMC 2016. Porto, Portugal, 11/2016 (Cited on pages 13, 15, 62, 67).

[KBC18a] **Kuan-Hsun Chen**, G. v. d. Brüggen, and J.-J. Chen. "Analysis of Deadline Miss Rates for Uniprocessor Fixed-Priority Scheduling". In: *2018 24th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. 08/2018 (Cited on pages 15, 19, 63).

[KBC18b] **Kuan-Hsun Chen**, G. v. der Bruggen, and J.-J. Chen. "Reliability Optimization on Multi-Core Systems with Multi-Tasking and Redundant Multi-Threading". In: *IEEE Transactions on Computers* 67.4 (04/2018), pp. 484–497. DOI: 10.1109/TC.2017.2769044 (Cited on page 15).

[KC17] **Kuan-Hsun Chen** and J.-J. Chen. "Probabilistic schedulability tests for uniprocessor fixed-priority scheduling under soft errors". In: *12th IEEE International Symposium on Industrial Embedded Systems (SIES)*. 06/2017, pp. 1–8 (Cited on pages 8, 14, 62 sq., 68, 78, 84, 95).

[KCK+15] **Kuan-Hsun Chen**, J.-J. Chen, F. Kriebel, S. Rehman, M. Shafique, and J. Henkel. "Reliability-Aware Task Mapping on Many-Cores with Performance Heterogeneity". In: *1st International Workshop on Resiliency in Embedded Electronic Systems*. 2015 (Cited on page 15).

[KCK+16] **Kuan-Hsun Chen**, J.-J. Chen, F. Kriebel, S. Rehman, M. Shafique, and J. Henkel. "Task Mapping for Redundant Multithreading in Multi-Cores with Reliability and Performance Heterogeneity". In: *IEEE Transactions on Computers* 65.11 (11/2016), pp. 3441–3455. DOI: 10.1109/TC.2016.2532862 (Cited on pages 5, 15, 23, 102, 133).

[KFJ+03] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. "Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction". In: *MICRO*. 2003 (Cited on page 100).

[KG94] B. Kao and H. Garcia-Molina. "Subtask deadline assignment for complex distributed soft real-time tasks". In: *14th International Conference on Distributed Computing Systems*. 1994 (Cited on page 23).

[KGC+12] P. Kumar, D. Goswami, S. Chakraborty, A. Annaswamy, K. Lampka, and L. Thiele. "A hybrid approach to cyber-physical systems verification". In: *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*. 06/2012, pp. 688–696. DOI: 10.1145/2228360.2228484 (Cited on pages 21, 40, 43, 52 sq.).

[Kil76] J. S. Kilby. "Invention of the integrated circuit". In: *IEEE Transactions on Electron Devices* 23.7 (07/1976), pp. 648–654. ISSN: 0018-9383. DOI: 10.1109/T-ED.1976.18467 (Cited on page 1).

[KKP+15] J. Kwon, K. W. Kim, S. Paik, J. Lee, and C. G. Lee. "Multicore scheduling of parallel real-time tasks with multiple parallelization options". In: *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2015 IEEE*. 2015, pp. 232–244 (Cited on pages 23 sq.).

[Koo14] P. Koopman. *A case study of Toyota unintended acceleration and software safety*. 09/2014. URL: https://users.ece.cmu.edu/~koopman/pubs/koopman14_toyota_ua_slides.pdf (Cited on page 2).

[KS95] G. Koren and D. Shasha. "Skip-Over: algorithms and complexity for overloaded systems that allow skips". In: *Proceedings 16th IEEE Real-Time Systems Symposium*. 12/1995, pp. 110–117. DOI: 10.1109/REAL.1995.495201 (Cited on pages 21, 44, 53).

[Kua17] **Kuan-Hsun Chen**. *Efficient Probabilistic Schedulability Test*. 2017. URL: https://github.com/kuanhsunchen/EPST (Cited on pages 87, 89).

[Kua18] **Kuan-Hsun Chen**. *Event-based Miss Rate Simulator*. 2018. URL: https://github.com/tu-dortmund-ls12-rt/MissRateSimulator/ (Cited on pages 34, 74, 97).

[Kua19] **Kuan-Hsun Chen, Niklas Ueter**. *Analytical Approaches for Deadline-Miss Probability*. 2019. URL: https://github.com/tu-dortmund-ls12-rt/AnalyticalDMP (Cited on page 97).

[KUB+19] **Kuan-Hsun Chen**, N. Ueter, G. von der Bruggen, and J.-J. Chen. "Efficient Computation of Deadline-Miss Probability and Potential Pitfalls". In: *Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy, March 25-29, 2019*. 03/2019. DOI: 10.23919/DATE.2019.8714908 (Cited on page 63).

[Kuh55] H. W. Kuhn. "The Hungarian Method for the assignment problem". In: *Naval Research Logistics Quarterly* 2 (1955), pp. 83–97 (Cited on pages 105, 111).

[LB05] C. Lin and S. A. Brandt. "Improving soft real-time performance through better slack reclaiming". In: *Real-Time Systems Symposium (RTSS)*. 2005 (Cited on page 23).

[LCA+14] J. Li, J. J. Chen, K. Agrawal, C. Lu, C. Gill, and A. Saifullah. "Analysis of Federated and Global Scheduling for Parallel Real-Time Tasks". In: *Real-Time Systems (ECRTS), 26th Euromicro Conference on*. 2014, pp. 85–96 (Cited on pages 23 sq., 31, 101, 116 sq., 120).

[LCP+09] G. Lyle, S. Chen, K. Pattabiraman, Z. Kalbarczyk, and R. Iyer. "An end-to-end approach for the automatic derivation of application-aware error detectors". In: *2009 IEEE/IFIP International Conference on Dependable Systems Networks*. 06/2009, pp. 584–589. DOI: 10.1109/DSN.2009.5270291 (Cited on page 3).

[LDV+04] L. Li, V. Degalahal, N. Vijaykrishnan, M. Kandemir, and M. Irwin. "Soft Error and Energy Consumption Interactions: A Data Cache Perspective". In: *ISLPED*. 2004 (Cited on pages 37, 132).

[Leg] Lego Inc. *Lego Mindstorms*. URL: http://www.lego.com/en-us/mindstorms/ (Cited on page 33).

[LKA04] J. Leung, L. Kelly, and J. H. Anderson. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Boca Raton, FL, USA: CRC Press, Inc., 2004. ISBN: 1584883979 (Cited on page 19).

[LL73] C. L. Liu and J. W. Layland. "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment". In: *J. ACM* 20.1 (01/1973), pp. 46–61. ISSN: 0004-5411. DOI: 10.1145/321738.321743. URL: http://doi.acm.org/10.1145/321738.321743 (Cited on pages 20, 43, 67).

[LSD89] J. Lehoczky, L. Sha, and Y. Ding. "The rate monotonic scheduling algorithm: exact characterization and average case behavior". In: *Real Time Systems Symposium, 1989., Proceedings.* 12/1989, pp. 166–171. DOI: 10.1109/REAL.1989.63567 (Cited on pages 20, 67, 70, 87).

[LV62] R. E. Lyons and W. Vanderkulk. "The Use of Triple-modular Redundancy to Improve Computer Reliability". In: *IBM J. Res. Dev.* 6.2 (04/1962), pp. 200–209. ISSN: 0018-8646. DOI: 10.1147/rd.62.0200. URL: http://dx.doi.org/10.1147/rd.62.0200 (Cited on page 23).

[LWW+02] M. N. Lovellette, K. S. Wood, D. L. Wood, J. H. Beall, P. P. Shirvani, N. Oh, and E. J. McCluskey. "Strategies for fault-tolerant, space-based computing: Lessons learned from the ARGOS testbed". In: *Proceedings, IEEE Aerospace Conference*. Vol. 5. 2002. DOI: 10.1109/AERO.2002.1035377 (Cited on page 2).

[MB06] G. D. Micheli and L. Benini. "Networks on Chips: Technology and Tools." In: Elsevier Science, 2006 (Cited on page 112).

[MC13] D. Maxim and L. Cucu-Grosjean. "Response Time Analysis for Fixed-Priority Tasks with Multiple Probabilistic Parameters". In: *Real-Time Systems Symposium (RTSS), IEEE 34th*. 2013, pp. 224–235 (Cited on pages 8, 18, 22, 62, 64, 66 sqq., 83 sq., 86 sq., 89 sq.).

[MC96] A. K. Mok and D. Chen. "A multiframe model for real-time tasks". In: *Real-Time Systems Symposium, 1996., 17th IEEE*. 12/1996, pp. 22–29. DOI: 10.1109/REAL.1996.563696 (Cited on page 45).

[MC97] A. K. Mok and D. Chen. "A Multiframe Model for Real-Time Tasks". In: *IEEE Trans. Software Eng.* 23.10 (1997), pp. 635–645 (Cited on page 43).

[MD11] F. Many and D. Doose. "Scheduling Analysis under Fault Bursts". In: *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*. 04/2011, pp. 113–122. DOI: 10.1109/RTAS.2011.19 (Cited on pages 4, 12, 62, 147).

[MEP04] S. Manolache, P. Eles, and Z. Peng. "Schedulability Analysis of Applications with Stochastic Task Execution Times". In: *ACM Trans. Embed. Comput. Syst.* 3.4 (11/2004), pp. 706–735. ISSN: 1539-9087. DOI: 10.1145/1027794.1027797. URL: http://doi.acm.org/10.1145/1027794.1027797 (Cited on pages 8, 23, 83 sq.).

[MEP08] S. Manolache, P. Eles, and Z. Peng. "Task Mapping and Priority Assignment for Soft Real-time Applications Under Deadline Miss Ratio Constraints". In: *ACM Trans. Embed. Comput. Syst.* (2008) (Cited on page 23).

[Mik17] Mikail Yayla and Gedare Bloom and Kuan-Hsun Chen and Joel Sherill. *Software-based Fault Tolerance: Adding Fault Tolerance Code in the Rate Monotonic Scheduler*. 2017. URL: https://devel.rtems.org/wiki/SOCIS/2017 (Cited on pages 14, 60).

[MKR02] S. S. Mukherjee, M. Kontz, and S. K. Reinhardt. "Detailed Design and Evaluation of Redundant Multithreading Alternatives". In: *ISCA*. 2002, pp. 99–110 (Cited on pages 9, 23, 31, 102).

[MU05] M. Mitzenmacher and E. Upfal. *Probability and computing - randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005. ISBN: 978-0-521-83540-4 (Cited on pages 68 sq.).

[Muk08] S. Mukherjee. *Architecture Design for Soft Errors*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008. ISBN: 9780080558325, 9780123695291 (Cited on page 3).

[MWE+03] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin. "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor". In: *IEEE/ACM MICRO*. 2003 (Cited on page 36).

[NQ06] L. Niu and G. Quan. "Energy minimization for real-time systems with (m,k)-guarantee". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 14.7 (07/2006), pp. 717–729. ISSN: 1063-8210. DOI: 10.1109/TVLSI.2006.878337 (Cited on pages 21, 44, 52).

[NV03] B. Nicolescu and R. Velazco. "Detecting soft errors by a purely software approach: method, tools and experimental results". In: *2003 Design, Automation and Test in Europe Conference and Exhibition*. 2003, 57–62 suppl. DOI: 10.1109/DATE.2003.1253806 (Cited on page 3).

[NVS+02] B. Nicolescu, R. Velazco, M. Sonza-Reorda, M. Rebaudengo, and M. Violante. "A software fault tolerance method for safety-critical systems: effectiveness and drawbacks". In: *Integrated Circuits and Systems Design, 2002. Proceedings. 15th Symposium on*. 2002, pp. 101–106. DOI: 10.1109/SBCCI.2002.1137644 (Cited on page 54).

[NX06] V. Narayanan and Y. Xie. "Reliability concerns in embedded system designs". In: *Computer* 39.1 (01/2006), pp. 118–120. ISSN: 0018-9162. DOI: 10.1109/MC.2006.31 (Cited on page 2).

[OSE05] OSEK. *OSEK/VDX Operating System Manual*. Version 2.2.3. 02/2005. URL: https://www.irisa.fr/alf/downloads/puaut/TPNXT/images/os223.pdf (Cited on page 25).

[OSM02] N. Oh, P. P. Shirvani, and E. J. McCluskey. "Error detection by duplicated instructions in super-scalar processors". In: *IEEE Transactions on Reliability* 51.1 (03/2002), pp. 63–75. ISSN: 0018-9529. DOI: 10.1109/24.994913 (Cited on page 3).

[PGZ08] K. Pattabiraman, V. Grover, and B. G. Zorn. "Samurai: Protecting Critical Data in Unsafe Languages". In: *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*. Eurosys '08. Glasgow, Scotland UK: ACM, 2008, pp. 219–232. ISBN: 978-1-60558-013-5. DOI: 10.1145/1352592.1352616. URL: http://doi.acm.org/10.1145/1352592.1352616 (Cited on page 3).

[PM98] M. Pandya and M. Malek. "Minimum achievable utilization for fault-tolerant processing of periodic tasks". In: *IEEE Transactions on Computers* 47.10 (10/1998), pp. 1102–1112. ISSN: 0018-9340. DOI: 10.1109/12.729793 (Cited on page 4).

[Pra96] D. K. Pradhan, ed. *Fault-tolerant Computer System Design*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996. ISBN: 0-13-057887-8 (Cited on page 3).

[QH00] G. Quan and X. Hu. "Enhanced Fixed-priority Scheduling with (M,K)-firm Guarantee". In: *Proceedings of the 21st IEEE Conference on Real-time Systems Symposium*. RTSS'10. Orlando, Florida: IEEE Computer Society, 2000, pp. 79–88. ISBN: 0-7695-0900-2. URL: http://dl.acm.org/citation.cfm?id=1890629.1890640 (Cited on pages 21, 44, 52 sq.).

[Ram99] P. Ramanathan. "Overload Management in Real-Time Control Applications Using (M,K)-Firm Guarantee". In: *IEEE Trans. Parallel Distrib. Syst.* 10.6 (06/1999), pp. 549–559. ISSN: 1045-9219. DOI: 10.1109/71.774906. URL: http://dx.doi.org/10.1109/71.774906 (Cited on pages 18, 21, 40, 44, 46, 52 sq.).

[RCK+16] S. Rehman, **Chen, K. -H.**, F. Kriebel, A. Toma, M. Shafique, J. Chen, and J. Henkel. "Cross-Layer Software Dependability on Unreliable Hardware". In: *Computers, IEEE Transactions on* (2016) (Cited on pages 35 sqq., 132).

[RCV+05a] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, and D. I. August. "SWIFT: software implemented fault tolerance". In: *International Symposium on Code Generation and Optimization*. 03/2005, pp. 243–254. DOI: 10.1109/CGO.2005.34 (Cited on pages 3, 29).

[RCV+05b] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, D. I. August, and S. S. Mukherjee. "Software-controlled Fault Tolerance". In: *ACM Trans. Archit. Code Optim.* 2.4 (12/2005), pp. 366–396. ISSN: 1544-3566. DOI: 10.1145/1113841.1113843. URL: http://doi.acm.org/10.1145/1113841.1113843 (Cited on pages 3, 58).

[RE17] E. A. Rambo and R. Ernst. "Replica-Aware Co-Scheduling for Mixed-Criticality". In: *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*. Ed. by M. Bertogna. Vol. 76. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, 20:1–20:20. ISBN: 978-3-95977-037-8. DOI: 10.4230/LIPIcs.ECRTS.2017.20. URL: http://drops.dagstuhl.de/opus/volltexte/2017/7152 (Cited on pages 23 sq.).

[Rea16] Real Time Engineers Ltd. *FreeRTOS*. 2016. URL: http://www.freertos.org/ (Cited on pages 24, 139).

[Reh15] S. Rehman. "Reliable Software for Unreliable Hardware - A Cross-Layer Approach". In: *Ph.D. Thesis* (2015) (Cited on page 35).

[RKS+14a] S. Rehman, F. Kriebel, M. Shafique, and J. Henkel. "Reliability-Driven Software Transformations for Unreliable Hardware". In: *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* (2014) (Cited on page 36).

[RKS+14b] S. Rehman, F. Kriebel, D. Sun, M. Shafique, and J. Henkel. "dTune: Leveraging Reliable Code Generation for Adaptive Dependability Tuning under Process Variation and Aging-Induced Effects". In: *DAC*. 2014, pp. 1–6 (Cited on pages 5, 23, 100–103, 112, 115, 127, 132 sq.).

[RM00] S. K. Reinhardt and S. S. Mukherjee. "Transient Fault Detection via Simultaneous Multithreading". In: *ISCA*. 2000, pp. 25–36 (Cited on pages 3, 5, 9, 23, 31, 62, 102).

[Rot99] E. Rotenberg. "AR-SMT: a microarchitectural approach to fault tolerance in microprocessors". In: *Fault-Tolerant Computing*. 1999 (Cited on pages 5, 9).

[RSH12] S. Rehman, M. Shafique, and J. Henkel. "Instruction scheduling for reliability-aware compilation". In: *DAC*. 2012 (Cited on pages 36, 103).

[RSK+11] S. Rehman, M. Shafique, F. Kriebel, and J. Henkel. "Reliable software for unreliable hardware: embedded code generation aiming at reliability". In: *CODES+ISSS*. 2011 (Cited on pages 30 sq., 35 sqq., 103, 132).

[RSS+17] E. A. Rambo, C. Seitz, S. Saidi, and R. Ernst. "Designing Networks-on-Chip for High Assurance Real-Time Systems". In: *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*. 01/2017, pp. 185–194. DOI: 10.1109/PRDC.2017.32 (Cited on pages 32, 36).

[RTE13] RTEMS. *RTEMS: Real-Time executive for multiprocessor systems*. http://www.rtems.com/. 2013. URL: http://www.rtems.com/ (Cited on pages 37, 139 sq.).

[RTG+13] B. Raghunathan, Y. Turakhia, S. Garg, and D. Marculescu. "Cherry-picking: Exploiting Process Variations in Dark-silicon Homogeneous Chip Multi-processors". In: *DATE*. 2013, pp. 39–44 (Cited on pages 104, 127).

[RTK+13] S. Rehman, A. Toma, F. Kriebel, M. Shafique, J. J. Chen, and J. Henkel. "Reliable code generation and execution on unreliable hardware under joint functional and timing reliability considerations". In: *IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 04/2013, pp. 273–282. DOI: 10.1109/RTAS.2013.6531099 (Cited on page 103).

[SA05] A. Srinivasan and J. H. Anderson. "Fair scheduling of dynamic task systems on multiprocessors". In: *Journal of Systems and Software* 77.1 (2005). Parallel and distributed real-time systems, pp. 67–80 (Cited on page 24).

[SAL+08] A. Sundaram, A. Aakel, D. Lockhart, D. Thaker, and D. Franklin. "Efficient Fault Tolerance in Multi-media Applications Through Selective Instruction Replication". In: *Proceedings of the 2008 Workshop on Radiation Effects and Fault Tolerance in Nanometer Technologies*. WREFT '08. Ischia, Italy: ACM, 2008, pp. 339–346. ISBN: 978-1-60558-092-0. DOI: 10.1145/1366224.1366227. URL: http://doi.acm.org/10.1145/1366224.1366227 (Cited on page 29).

[SCT10] A. Schranzhofer, J.-J. Chen, and L. Thiele. "Timing Analysis for TDMA Arbitration in Resource Sharing Systems". In: *Proceedings of the 2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*. RTAS '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 215–224. ISBN: 978-0-7695-4001-6. DOI: 10.1109/RTAS.2010.24. URL: https://doi.org/10.1109/RTAS.2010.24 (Cited on page 19).

[SGF+06] J. Smolens, B. Gold, B. Falsafi, and J. Hoe. "Reunion: Complexity-Effective Multicore Redundancy". In: *MICRO-39. IEEE/ACM*. 2006 (Cited on pages 5, 9).

[SGH+14] M. Shafique, S. Garg, J. Henkel, and D. Marculescu. "The EDA Challenges in the Dark Silicon Era: Temperature, Reliability, and Variability Perspectives". In: *DAC*. 2014 (Cited on pages 5, 9, 100).

[SKK+02] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvisi. "Modeling the effect of technology trends on the soft error rate of combinational logic". In: *Proceedings International Conference on Dependable Systems and Networks*. 2002, pp. 389–398. DOI: 10.1109/DSN.2002.1028924 (Cited on page 2).

[Skl76] J. R. Sklaroff. "Redundancy Management Technique for Space Shuttle Computers". In: *IBM Journal of Research and Development* 20.1 (01/1976), pp. 20–28. ISSN: 0018-8646. DOI: 10.1147/rd.201.0020 (Cited on page 2).

[SLS+99] J. A. Stankovic, C. Lu, S. H. Son, and G. Tao. "The case for feedback control real-time scheduling". In: *Real-Time Systems, 1999. Proceedings of the 11th Euromicro Conference on*. 1999 (Cited on page 23).

[SRA+13] M. Shafique, S. Rehman, P. V. Aceituno, and J. Henkel. "Exploiting Program-level Masking and Error Propagation for Constrained Reliability Optimization". In: *DAC*. 2013, 17:1–17:9 (Cited on page 30).

[SS82] D. P. Siewiorek and R. S. Swarz. *The theory and practice of reliable system design*. Digital press, 1982 (Cited on page 3).

[SSF09] U. Schiffel, M. Süßkraut, and C. Fetzer. "AN-Encoding Compiler: Building Safety-Critical Systems with Commodity Hardware". In: *SAFECOMP '09: Proceedings of the 28th International Conference on Computer Safety, Reliability, and Security*. Hamburg, Germany: Springer-Verlag, 2009, pp. 283–296. ISBN: 978-3-642-04467-0. DOI: 10.1007/978-3-642-04468-7_23 (Cited on page 29).

[SWK+05] G. P. Saggese, N. J. Wang, Z. Kalbarczyk, S. J. Patel, and R. K. Iyer. "An Experimental Study of Soft Errors in Microprocessors". In: *IEEE Micro* 25.6 (2005), pp. 30–39 (Cited on page 36).

[TBE+15] B. Tanasa, U. D. Bordoloi, P. Eles, and Z. Peng. "Probabilistic Response Time and Joint Analysis of Periodic Tasks". In: *27th Euromicro Conference on Real-Time Systems*. 07/2015, pp. 235–246 (Cited on pages 8, 18, 22).

[Tes] N. Tesla. *Tesla Processor Family, 2013*. URL: http://www.nvidia.com (Cited on page 4).

[Til] Tilera Corporation. *Tile-GX Processor Family, 2013*. URL: http://www.tilera.com (Cited on page 4).

[VCT+99] R. Velazco, P. Cheynet, A. Tissot, J. Haussy, J. Lambert, and R. Ecoffet. "Evidences of SEU tolerance for digital implementations of artificial neural networks: one year MPTB flight results". In: *1999 Fifth European Conference on Radiation and Its Effects on Components and Systems. RADECS 99 (Cat. No.99TH8471)*. 1999, pp. 565–568. DOI: 10.1109/RADECS.1999.858648 (Cited on page 6).

[Ves07] S. Vestal. "Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance". In: *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS 2007), 3-6 December 2007, Tucson, Arizona, USA*. 2007, pp. 239–243. DOI: 10.1109/RTSS.2007.47. URL: https://doi.org/10.1109/RTSS.2007.47 (Cited on pages 139, 147, 149).

[VPC02] T. N. Vijaykumar, I. Pomeranz, and K. Cheng. "Transient-fault recovery using simultaneous multithreading". In: *Computer Architecture, 2002. Proceedings. 29th Annual International Symposium on*. 2002, pp. 87–98 (Cited on pages 23, 102).

[VZB+10] R. Vadlamani, J. Zhao, W. Burleson, and R. Tessier. "Multicore Soft Error Rate Stabilization Using Adaptive Dual Modular Redundancy". In: *Proceedings of the Conference on Design, Automation and Test in Europe*. DATE '10. Dresden, Germany: European Design and Automation Association, 2010, pp. 27–32. ISBN: 978-3-9810801-6-2. URL: http://dl.acm.org/citation.cfm?id=1870926.1870937 (Cited on page 23).

[WA08] F. Wang and V. D. Agrawal. "Single Event Upset: An Embedded Tutorial". In: *21st International Conference on VLSI Design (VLSID 2008)*. 01/2008, pp. 429–434 (Cited on pages 3, 5, 42).

[Wil06] J. Wilkinson. *Flux calculator*. http://www.seutest.com/cgi-bin/FluxCalculator.cgi. 2006. URL: http://www.seutest.com/cgi-bin/FluxCalculator.cgi (Cited on page 36).

[YKC18] M. Yayla, **Kuan-Hsun Chen**, and J.-J. Chen. "Fault Tolerance on Control Applications: Empirical Investigations of Impacts from Incorrect Calculations". In: *4th International Workshop On Emerging Ideas and Trends In Engineering of Cyber-Physical Systems (EITEC)*. Proceedings of EITEC 2018. Porto, Portugal, 04/2018 (Cited on pages 14, 40, 43).

[YYa10] Y.Yamamoto. *Two wheeled self-balancing R/C robot controlled with a HiTechnic Gyro Sensor*. 2010. URL: http://lejos-osek.sourceforge.net/nxtway%5C_gs.htm (Cited on pages 53, 55, 60).