Data Age Formulation and Analysis in Real-Time Embedded Systems - Fault tolerant and Thermal aware perspectives

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science in ***Computer Science and Engineering*** *by Research*

by

Sridhar Mallareddy
2018111021
sridhar.m@research.iiit.ac.in

International Institute of Information Technology
Hyderabad - 500 032, INDIA
June 2024

International Institute of Information Technology

Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled **"Data Age Formulation and Analysis in Real-Time Embedded Systems - Fault tolerant and Thermal aware perspectives"** by Sridhar Mallareddy, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date	Adviser: Prof. Deepak Gangadharan

To Amma, my strongest supporter and dearest friend

Acknowledgments

Completing this thesis has been a journey, one enriched by the guidance and support of so many incredible individuals. First and foremost, my deepest gratitude goes to Dr. Deepak Gangadharan, whose unwavering support shaped the very foundation of this dissertation. His mentorship not only fueled my research but also instilled in me the confidence to explore. I'll forever cherish the freedom he granted me to chart my own path, be it personal, professional or academic. Special thanks to Pavan Kondooru, whose collaborative spirit and tireless dedication were instrumental in bringing our ideas to fruition. Without his support, completing projects on time would have been an uphill battle. Also a heartfelt shoutout to Naren Akash, whose unwavering dedication to the pursuit of knowledge served as an inspiration from day one.

To SPRSH, especially Swati Naidu—thank you hardly seems enough. You guys have been a mother, a friend, a mentor, and so much more throughout this journey. Your wisdom, warmth, and unwavering support have been my guiding lights through the challenging and rewarding days. You nurtured my ambitions with your kindness and with your unfiltered insights. I am profoundly grateful for you in my life, shaping both who I am and the what I look forward to accomplish.

To my dearest friends—Srinath, Mohee, Arohi, Yash, Dolton, Sambhav, Erin, Mukund, Nikunj, Nishit, Raghav, and Rashmeet—your warmth and friendship have made my days colourful and lightened my burdens throughout college and my research journey. I am so lucky for all the laughter and reliable support you've given me. Each moment spent with you guys is a treasured memory, and I'm thankful for every one.

To my mom and dad, thank you for your constant support, sacrifices, and trust throughout my journey. You've always believed in me and provided me safe space, and for that, I am perpetually grateful. I am blessed with parents like you and you are the foundation of everything I do.

Abstract

Safety-critical real-time systems demand meticulous attention to both temporal requirements and fault tolerance mechanisms. This thesis addresses critical challenges in analyzing and optimizing system performance under the influence of transient faults and thermal constraints.

Our first work introduces an analytical framework for evaluating data age in real-time task chains augmented with checkpointing mechanisms. Transient faults pose significant threats to system predictability, and conventional fault tolerance methods such as checkpointing can exacerbate data age dynamics. Our framework offers a systematic approach to quantifying data age in both single-core and multi-core platforms, validated through extensive simulations. The results demonstrate that the data age bounds achieved are comparable to those of existing techniques, while the computational overhead remains minimal in relation to traditional methods. This balance enhances the predictability and reliability of safety-critical systems, ensuring efficient performance without sacrificing accuracy or increasing resource consumption significantly.

The second work focuses on the analysis to incorporate thermal constraints alongside data age considerations in task scheduling for safety-critical applications. By introducing a thermal-aware data age analysis framework, this work addresses the pressing need for simultaneous optimization of task periods to ensure both data age and processor thermal safety. Through systematic evaluations, we establish the efficacy of our approach in deriving schedulable task periods, thus mitigating risks associated with invalid data consumption and thermal violations in safety-critical systems. Results underscore the effectiveness of our framework in achieving optimal task scheduling while meeting stringent safety and thermal requirements.

Together, these contributions provide essential insights and methodologies to enhance the resilience and performance predictability of safety-critical real-time systems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Real-time systems underpin critical operations, from self-driving cars to medical devices. These systems rely on fresh data for accurate decision-making. However, ensuring the validity of data faces challenges like thermal constraints on processors and the influence of fault tolerance mechanisms.

This thesis investigates the interplay between these factors and data age. It explores how thermal thresholds in resource-constrained processors and the delays introduced by checkpointing (a fault tolerance technique) can impact data age. By analyzing these influences, the research aims to develop a novel data-age analysis framework for real-time systems. This framework will contribute to designing more robust systems that operate within acceptable data age boundaries, ultimately enhancing reliability and safety in critical applications.

1.1 Scope of the Problem

Real-time systems underpin a wide range of safety-critical applications, hence they require deterministic and timely data processing to guarantee correct functionality and ensure safety. *Data age*, a metric reflecting how long a data is in system from it's production time to last consumption by any job, plays a vital role in ensuring data validity within these systems. A *vaild data* is a data whose data age is under provided threshold or data age limit. Using outdated data can lead to erroneous decisions with potentially catastrophic consequences.

Traditional worst-case execution time (WCET) analysis, while crucial for ensuring system stability, can be overly conservative, leading to inefficiencies in real-time system design. As

real-time systems become increasingly complex, efficient data-age management strategies are crucial for guaranteeing data validity under various operating conditions.

Existing research on data age analysis primarily focuses on minimizing the end-to-end age of data within task chains, a pipeline of tasks which will be consuming the data. However, these methods often overlook additional factors that can significantly impact data validity. Here, we identify two key challenges that this thesis aims to address:

Fault Tolerance Mechanisms: Fault tolerance mechanisms, like checkpointing, are essential for ensuring system reliability in the face of potential hardware or software failures. However, these mechanisms can introduce additional delays into the system. For instance, checkpointing involves periodically saving the state of a task, which can add overhead and potentially increase data age. Existing data-age analysis approaches may not adequately account for the impact of fault tolerance mechanisms on data age.

Thermal Constraints: Processors in resource-constrained real-time systems have limitations in terms of heat dissipation. When thermal thresholds are reached, processors may throttle or slow down to manage heat. This performance degradation can directly impact task execution times, potentially increasing data age beyond acceptable limits. Traditional data age analysis methods often don't consider these thermal constraints, leading to potentially inaccurate age estimations.

This thesis aims to bridge the gap in existing research by proposing two novel data-age analysis frameworks that considers thermal constraints and the impact of fault tolerance mechanisms individually. This comprehensive approach can lead to more accurate data-age estimations, enabling the design of real-time systems that operate within acceptable data age boundaries while ensuring system reliability and performance.

1.2 Motivation

Real-time systems are the backbone of modern infrastructure, silently orchestrating critical tasks across diverse domains. From the self-driving cars navigating our streets to the intricate control systems within nuclear power plants, these systems demand guaranteed performance and utmost reliability. At the heart of this reliability lies *data age*, an important metric to define validity of data.

While crucial for ensuring system stability, WCET analysis can be overly conservative, leading to inefficiencies. As real-time systems become increasingly complex, with intricate task chains and stringent deadlines, there's a growing need for sophisticated data-age management strategies. "*Data age*", a metric reflecting the time elapsed since data generation, provides a more nuanced understanding of data validity compared to the binary "*fresh*" or "*stale*" classification imposed by using WCET during data age calculation.

Frameworks to calculate data age play a pivotal role in addressing the necessity of precision posed by real-time systems. These frameworks provide structured methodologies and tools that facilitate systematic analysis and optimization of system performance under diverse conditions. In scenarios where thermal constraints are in play, frameworks enable the precise calculation of data age, allowing system designers to strike an optimal balance between performance and thermal management. Similarly, in the presence of fault tolerance mechanisms, frameworks provide a structured approach for evaluating the effectiveness of these mechanisms in preserving system integrity and functionality. By formalizing the analysis process and providing guidelines for decision-making, frameworks empower engineers to make informed choices and design robust systems that can withstand unforeseen challenges. Without such frameworks, the ability to proactively manage data age and system resilience is severely compromised, potentially exposing critical systems to unforeseen vulnerabilities.

This thesis delves into two critical aspects that impact data validity in real-time systems: thermal constraints and the influence of fault tolerance mechanisms.

The first research work explores the relationship between data age and fault tolerance mechanisms. Imagine a high-frequency trading platform relying on real-time market data to make split-second investment decisions. Even a slightly older data past its data age limit can result in significant financial losses. To ensure system reliability and prevent errors in transactions, fault tolerance mechanisms like data replication are often employed. Data replication involves storing copies of critical data across multiple servers. While this approach enhances system robustness, it introduces additional processing overhead to maintain data consistency across replicas. These delays can potentially increase data age, causing the trading platform to base investment decisions on slightly outdated market information. Existing data-age analysis approaches may not adequately account for the impact of data replication on data age, potentially leading to suboptimal system design for high-frequency trading applications.

The second research work in this thesis focuses on the impact of thermal constraints on data age. Consider the scenario of an autonomous car relying on real-time sensor data for navigation. These sensors generate a constant stream of data on surrounding obstacles, traffic lights, and road conditions. The car's onboard computer needs to process this data swiftly and accurately to make critical decisions. However, processors in resource-constrained systems like autonomous vehicles have limitations in heat dissipation. During periods of intense processing or high ambient temperatures, processors may throttle or slow down to manage heat build-up. This thermal throttling directly impacts task execution times, potentially causing delays that can significantly increase data age. Existing data-age analysis methods often don't consider these thermal constraints, leading to inaccurate data-age estimations and potentially compromising the safety of autonomous vehicles.

By addressing both thermal constraints and fault tolerance mechanisms in data-age analysis, this thesis aims to bridge the gap in existing research. This comprehensive approach can lead to more accurate data-age estimations and enable the design of real-time systems that operate within acceptable data age boundaries, guaranteeing system reliability, performance, and ultimately, safety in critical real-world applications.

1.3 Scope of Thesis

Real-time systems are basis for a diverse range of safety-critical applications where consistent performance and high reliability are paramount. One of the principal challenges in these systems is maintaining the reliability of data, which is crucial for decision-making processes and system stability. Traditional approaches, such as ones using the worst-case execution time (WCET) analysis for data age calculation, often adopt overly conservative estimates that can compromise the efficiency of data age management.

This thesis aims to explore two of many factors that significantly impact data age: thermal constraints and fault tolerance mechanisms. Current methodologies lack a comprehensive framework that can accurately calculate data age while accounting for these factors.

This work aims to optimize system performance and enhance reliability without the unnecessary conservatism that characterizes existing approaches. Through this research, we seek to contribute

to the advancement of real-time system designs, enabling them to meet the rigorous demands of safety-critical applications more effectively.

1.3.1 Objectives of the Thesis

The primary objective of this thesis encompasses the development of a more precise and adaptable model to calculate data age, considering the dynamic effect on system due to thermal conditions and fault tolerance strategies. By providing more accurate estimations of data age within real-time systems, this framework will empower the design of systems that operate within acceptable data validity boundaries.

1.3.2 Specific Areas Covered

To achieve this objective, the thesis will explore the following key areas:

- **Fault Tolerance Mechanisms and Data Age:** In chapter 3, the thesis analyzes different fault tolerance mechanisms, such as checkpointing and re-execution. We explore how these mechanisms introduce delays that can potentially compromise data freshness by increasing data age.
- **Interference from Checkpointing:** In chapter 3, we investigate cases where checkpointing, as a fault tolerance strategy, introduces interference and delays in task execution. This exploration considers the resultant effects on data freshness and system reliability, particularly in environments with stringent real-time requirements.
- **Thermal Modeling:** In chapter 2, we delve into various thermal models for resource-constrained processors commonly used in real-time systems. This analysis provides a deeper understanding of how processor temperature, as well as ambient temperature, impacts performance.
- **Task Splitting for Thermal Management:** In chapter 4, an exhaustive examination is conducted on the use of task splitting techniques to manage thermal constraints. This area focuses on case studies where task splitting has been applied to maintain optimal processor temperatures, thereby minimizing thermal impacts on system performance and data age.

- **Comprehensive Data-Age Analysis Framework:** Leveraging the insights gained from thermal modeling and fault tolerance analysis, we develop a comprehensive data-age analysis framework. This framework incorporates the impact of both thermal constraints and fault tolerance mechanisms for more accurate data age estimations.
- **Framework Validation:** The proposed data-age analysis framework is rigorously validated through simulations. This validation process assesses the accuracy and effectiveness of the framework.

1.3.3 Approach

To tackle the challenge of maintaining data validity in real-time systems amidst thermal constraints and fault tolerance mechanisms, this thesis adopts a structured, multi-faceted methodology:

- **Literature review:** To lay the groundwork for this study, a comprehensive literature review was conducted. This review encompasses existing theories and empirical studies related to thermal modeling, data-age analysis limitations (focusing on thermal and fault tolerance), and fault tolerance mechanisms' impact on data age. The purpose is to identify gaps in current knowledge and to understand the broader context within which data age is impacted. This comprehensive review will inform the development of our novel data-age analysis framework.
- **Leveraging Existing Research:** Building on the established research in thermal modeling of processors, fault and fault tolerance models, and data-age analysis in real-time systems, this thesis will enhance and adapt these foundational insights to develop our own analytical framework.
- **Modeling and Analysis:** We will analyze the working of existing models, particularly their capacity to maintain safe operational parameters within set thermal thresholds and fault tolerance mechanisms. This analysis will help us understand the width impact on data age, laying the groundwork for refining and proposing targeted enhancements.
- **Framework Development:** Drawing on the insights gained from thermal-aware scheduling and fault tolerance analysis, we will develop a comprehensive data-age analysis framework.

This framework will incorporate the influence of techniques, such as checkpointing for fault tolerance and task splitting to manage thermal loads, quantifying their impact on data latency.

- **Framework Validation:** The proposed data-age analysis framework will be rigorously validated through simulations. Simulations will allow us to evaluate the framework's accuracy and effectiveness in estimating data age under diverse operating conditions, including varying thermal scenarios and different fault tolerance configurations.

  This approach ensures a deep and nuanced understanding of the variables affecting data age, facilitating the development of a robust framework suitable for optimizing data validity in real-time systems.

1.3.4 Limitations

This research, while detailed, recognizes certain limitations in its design and scope. Initially, the development of the data age framework for fault-tolerant systems is targeted at single-core and multi-core processor architectures. However, for systems that require thermal-aware considerations, the focus will be specifically on single-core processors. This delineation suggests a need for future adaptations of the proposed framework to include comprehensive support for various scheduling techniques, which are critical for optimizing both fault tolerance and thermal management in heterogeneous multi-core system.

Additionally, while the framework will incorporate certain thermal-aware resource management strategies, it may require further refinement to fully integrate an expanded array of such techniques, particularly for multi-core systems. The study will also consider a popular fault tolerance mechanism. Although this method is expected to cover broad number of applications of fault resilience, there exists a broader spectrum of fault tolerance strategies- like load balancing, sharding; that could impact system performance and data age. Subsequent investigations might need to explore these additional techniques to broaden the framework's applicability and enhance its efficacy in diverse real-time system environments.

1.4 Thesis Layout

The remaining chapters of this thesis will build upon the foundation laid out in this introduction.

- **Chapter 2: Theoretical Framework and Literature Review:** This chapter will delve deeper into existing research on data-age analysis, thermal modeling, and fault tolerance mechanisms in real-time systems. We will critically evaluate existing approaches and identify the gaps that this thesis aims to address.
- **Chapter 3: Checkpoint aware data age analysis of task chains:** this chapter focuses on modeling the overhead introduced by checkpointing, a crucial fault tolerance mechanism, and how it impacts data age calculations in real-time systems. It will explore methods for estimating checkpoint size and execution time based on task characteristics, and then integrate this information into the framework to account for the increased data age due to checkpointing intervals. The chapter also include results based on use of benchmark tasksets for automotive systems, representing real-world use case to analyze the impact of checkpointing on data age and draw insights for system design.
- **Chapter 4: Thermal aware data age analysis of task chains:** This chapter delves into the critical issue of thermal constraints and their influence on data-age estimations. It will work on specific thermal model for this research and how this model is integrated into the data-age analysis framework. The chapter will detail how the framework incorporates processor temperature and ambient temperature as an input parameter and utilizes the thermal model to predict potential throttling scenarios. It will then describe how these situations are controlled and how the framework adjusts task execution times based on predicted throttling frequencies and duration. Finally, the chapter will discuss the process of validating the thermal-aware data age analysis framework through simulations.
- **Chapter 5: Conclusion and Future Work:** This chapter will summarize the key findings of the thesis, discuss the limitations of the research, and propose potential directions for future work.

This structure ensures a logical flow of information, guiding the reader through the problem, the proposed solution, and the evaluation process. By addressing both thermal constraints

and fault tolerance mechanisms, this thesis aims to significantly advance the field of data-age analysis for real-time systems.

Chapter 2

Literature Review

2.1 Introduction

To lay the groundwork for this study, a comprehensive literature review was conducted which encapsulates the breadth of existing theories and studies related to data age, data freshness, and the intricate interplay of thermal and fault models within real-time systems. The objective of this review is twofold: to pinpoint gaps in current understanding and to contextualize the landscape within which data age analysis unfolds. In addition to synthesizing existing knowledge, this stage entails a meticulous examination of the factors influencing delays attributed to thermal thresholds and fault tolerance mechanisms. By laying this comprehensive groundwork, we set the stage for a nuanced exploration of data age dynamics in real-time systems, with implications for system reliability and performance optimization.

2.2 Data Age Analysis of Real-Time Systems

Data age serves as a pivotal metric for assessing data quality in various systems, including Web Servers [38] and Real-Time Databases [49, 29]. Notable studies such as [11, 42] provide comprehensive frameworks for understanding data age as a quality metric and its associated analysis. Feiertag et al. [17] were pioneers in defining end-to-end latency semantics, including data age, but their approach lacked consideration for the timing variations in task releases and execution times. Building on this, Dürr et al. [16] found that the maximum data age is bounded by the maximum reaction time, providing valuable insights into data age constraints. However, Davare et al. [15] and Kloda et al. [31] offered upper bounds for maximum data age, although

their estimates tended to be overly conservative. Kloda et al.'s analysis, in particular, relied on synchronous task releases and necessitated knowledge of worst-case response times, which limited its applicability. Günzel et al. [23] took a different approach by proposing an analysis that assumed tasks always execute at their worst-case times, thereby allowing them to derive bounds on reaction latency and data age for distributed systems. However, this method might not perform well in scenarios with task release jitter or execution-time variations.

Becker et al. [7] introduced a schedule-agnostic data age analysis, which, although inherently pessimistic as it didn't account for scheduling policy and task response times, marked a step forward. They later extended their approach to compute data age bounds with greater knowledge about the system, including response times and even the exact schedule[8]. More recently, Gohari et al. [22] contributed to this research landscape by providing an analysis for determining the maximum data age under non-preemptive scheduling. They employed a schedule abstraction graph, offering an alternative perspective on data age calculations.

In [49], the authors investigate the impact of update policies on data age in distributed real-time databases, proposing a novel update policy. Conversely, works like [9] and [50] explore preemptive scheduling policies and deferrable algorithms, respectively, to optimize data age, throughput, and delay performance. However, none of these studies address the modeling of real-time tasks or derive task period parameters to meet target data age requirements.

This subsection has provided an overview of existing research on data age analysis in real-time systems. The following sections will delve deeper into specific limitations and how our proposed framework addresses them.

2.3 Fault Tolerance Mechanisms in Real-Time Systems

Achieving fault tolerance in real-time systems is paramount, especially for safety-critical applications. For reliability of real-time systems, various efforts have been made to reduce error frequency at a circuit and architectural design level[48], but they cannot entirely eliminate errors, especially in Commercial Off-The-Shelf (COTS) processors commonly used in embedded systems. Our research can target transient errors that evade mechanisms like logic masking, temporal masking, and architectural masking. These faults can manifest as application crashes, control flow violations, or silent data corruption. Online error detection techniques are commonly

classified into two categories, as seen in [21]- (a) Embedded Error Detection (EED): These techniques encompass error detection methods that do not rely on redundant execution, such as watchdog time control flow checking and instruction signature checking. EED often introduces a performance overhead that can vary depending on the implementation and application. (b) Explicit Output Comparison (EOC): EOC relies on explicit redundancy by executing the same task multiple times and comparing their outputs, either through classic triple modular redundancy (TMR) or some simplified versions.

Researchers have explored ways to increase system reliability without adding expensive hardware. They've used techniques like checkpointing, which periodically saves the system's state to recover from errors [44]. Error recovery techniques, like checkpointing [45], are crucial for bringing systems back to an error-free state with minimal extra work. Rollback error recovery, especially common in embedded systems, helps restore the processor state with low overhead [40]. Some studies, like the one by Huang et al. [27], looked at redundancy in time-triggered systems, while others, like Pinello et al. [43], focused on replication in controlled environments. Burns et al. [40] delved into priority assignment and scheduling for fault-tolerant systems. Kim et al. [30] categorized tasks based on their fault tolerance needs and used replication strategies. More recent works addressed fault burst models [25], which deal with multiple transient faults occurring in a short time.

2.3.1 Limitations of Existing Work in Checkpointing and Data Age Analysis

The limitations of current research predominantly revolve around the inadequate consideration of dynamic conditions such as variability in task release and execution times. Methods like those proposed by Feiertag et al. [17] and Dürr et al. [16] do not adequately adapt to these factors, leading to inflexible and sometimes inaccurate management of data age. Moreover, the conservative estimates provided by Davare et al. [15] and Kloda et al. [31] often result in suboptimal system utilization. Additionally, approaches that disregard the effects of different scheduling policies, as seen in Becker et al. [7], fail to capture true system behavior, limiting the effectiveness of data age optimization efforts.

2.3.2 Our Contribution

Addressing some of the limitations, our research introduces a framework that incorporates a more granular analysis of task variability and system dynamics, providing a more accurate and less conservative assessment of data age. We propose a model that not only considers task release and execution time variations but also integrates these factors with checkpointing policies to better reflect the real operational conditions of fault tolerant embedded systems.

Following are the contributions detailed in Chapter 3:

- To the best of our knowledge, the work conducts the first comprehensive data age analysis for cause-effect chains in scenarios where data age and checkpointing as fault tolerance mechanisms coexist.
- Applied this analysis to multirate tasks, extending the understanding of fault tolerance in a broader range of real-time systems.
- Investigated both single-core and multi-core platforms, contributing insights into the performance implications of our approach across different computing environments.
- Introduced a calculation method for interference due to checkpoints, accounting for task execution possibilities both before and after their scheduled execution time.

2.4 Real-Time Systems and Thermal Constraints

Effective thermal management is paramount in advanced embedded real-time systems, such as vehicle controls and smartphones, which are built on high-performance computing platforms that exhibit rapidly growing power densities. Elevated temperatures on these platforms can substantially reduce their operational lifespan, undermine performance, and compromise safety, potentially leading to vehicle malfunctions or smartphone incidents. It is essential to maintain the processor's temperature beneath a critical maximum threshold while also adhering to all required timing constraints for applications.

Embedded real-time systems face two primary thermal management challenges: 1) the ambient temperature that can change dynamically, and 2) the power dissipation at the task level. These fluctuations in temperature present considerable obstacles in adhering to the necessary

timing constraints for applications. Specifically, the maximum computation power available fluctuates with changes in ambient temperature because a processor's temperature is influenced by its surroundings. Under such circumstances, a processor might breach its thermal limits if it undertakes a task that has an average power dissipation above a predefined threshold. Measures like reducing processor activity or decreasing its speed can be implemented to manage temperature, but these actions may result in missed deadlines for tasks or jobs, thereby breaching application timing requirements. This necessitates a flexible resource management strategy that accounts for the dynamic changes in ambient temperature and task-specific power dissipation to satisfy both thermal and timing constraints.

2.4.1 Thermal Management in Real-Time Systems

Extensive research has been conducted on thermal-aware scheduling in real-time environments [33]. Traditional methods often include techniques like DVFS scheduling [14, 18], idle-time scheduling [35, 5], and task scheduling [12] to reduce maximum temperatures while ensuring timing requirements are met. Approaches for worst-case temperature analysis [37, 46] have been developed to provide offline assurances that thermal constraints will be maintained. The notion of thermal utilization [1] has been introduced to quantify the thermal effects of periodic real-time tasks on processors. However, many of these solutions assume either a constant ambient temperature or uniform power dissipation that does not vary with the task. Though real-time thermal controllers that adjust task utilization [19] or frequency [20] to maintain a desired temperature have been developed, their effectiveness is often limited when it comes to preventing temperature spikes.

Research into thermal-aware real-time scheduling continues to evolve, focusing on balancing thermal and timing constraints in static conditions. DVFS scheduling adjusts a processor's voltage and frequency to optimize power consumption [4, 52], and manage peak temperatures while meeting set timing constraints on single-core systems [14, 47, 13].

Additionally, strategies that account for variations in task-level power dissipation aim to minimize temperature peaks [28, 2] or maximize operational throughput [51, 26] by alternating between tasks with higher and lower heat outputs. Such strategies are crucial for defining peak temperature thresholds that adhere to thermal constraints [37].

2.4.2 Limitations of Existing Works

While thermal management techniques exist for real-time systems, they primarily focus on schedulability and task deadline. These approaches often neglect the crucial aspect of data age. Integrating thermal considerations into data age analysis is essential for ensuring that real-time systems operate within acceptable data age boundaries, ultimately guaranteeing system reliability, safety, and performance. Despite advancements in thermal management for real-time systems, a gap exists in the context of data age analysis.

2.4.3 Our Contribution

To meet the need for reliable thermal aware real time systems, we first verify the significance of these factors using benchmark tasksets, then develop and validate a data age analysis framework that considers individual tasks' different power dissipation. Building upon the task-level thermal model, and using task splitting mechanism as thermal-aware resource management scheme we propose a novel data age calculation framework which can assist developing more reliable systems where scheduled tasks to meet both thermal and timing constraints.

Following are our contributions detailed in Chapter 4:

- Formulation of the $2-task$ scenario and the associated optimization problem to derive task periods under both end-to-end data freshness and thermal constraints.
- Extension of our formulation to accommodate an $n-task$ scenario, allowing for scalability and applicability to real-world systems.
- Evaluation of our theoretical framework using task chains derived from the *MiBench* automotive benchmark. Through this evaluation, we demonstrate variations in schedulability percentages under varying thermal and freshness constraints, providing insights into the practical implications of our approach.

2.5 Conclusion

This comprehensive literature review has explored the intricate dimensions of data age, data freshness, and their interdependencies with thermal and fault models within real-time systems. Our examination revealed significant gaps in the existing methodologies, particularly in terms

of accommodating dynamic system conditions and optimizing data age within the constraints of real-time operational demands. While substantial work has been done to understand and improve data age through various scheduling and fault tolerance mechanisms, there remains a pressing need for models that integrate more dynamic task handling and less conservative assumptions to better mirror actual system behaviors.

The synthesis of research on data age highlights its critical role in assessing system performance and emphasizes the need for more refined and adaptable strategies. Similarly, our review of fault tolerance mechanisms underscores the importance of robust error management in maintaining system reliability, particularly in safety-critical applications using COTS processors. The studies reviewed have laid a strong foundation for our research, which aims to address these gaps by introducing innovative approaches that more accurately reflect and enhance the real-time system operations.

In conclusion, this review not only encapsulates the current state of knowledge but also sets the stage for our subsequent contributions, which aspire to refine these models and propose solutions that are both practical and effective for real-time systems facing contemporary challenges. Our work will build on this foundation, aiming to advance the understanding of how data age can be optimized across various system configurations to improve overall reliability and performance.

Chapter 3

Checkpointing-Aware Data Age Analysis of Task Chains

In automotive systems, standards have been established to enable independent development, improve software compatibility, and meet reliability requirements in case of subsystem failures[16], e.g. AUTOSAR[3]. This approach allows tasks to access up-to-date data independently without coordination with other tasks. However, it introduces challenges related to system reliability, performance, and safety. One crucial metric that aids in navigating these challenges is *data age*. *Data age* refers to the amount of time that input data within a system influences the output value of that chain. In other words, when a specific output is generated, data age indicates how old the input data was for that output.

In complex systems, overlooking maximum data age can potentially lead to catastrophic outcomes. Moreover, the presence of faults and fault tolerance mechanisms can cause fluctuations in data age. Faults manifest in various forms – permanent, intermittent, and transient. Our research specifically considers transient faults or soft errors. These faults, induced by factors like electromagnetic radiation, temperature fluctuations, and electrical noise, potentially disrupt embedded systems [6][48].

Achieving fault tolerance in computer systems entails employing redundancy either in space or time[34]. In particular, for tolerating transient faults, the use of time redundancy is a cost-efficient means to achieve fault tolerance[34]. In case of time redundancy, whenever a fault leads to an error, and the error is detected, the faulty task is re-executed. A widely adopted technique is checkpointing [45]. This involves periodically saving the system's state to enable recovery and resume execution from a known fault-free state. However, introduction of fault detection and tolerance techniques comes with significant timing overhead, which can affect

deadlines and the data age. Consequently, it becomes crucial to **model the impact of fault tolerance mechanisms on data age** to ensure the safety and effectiveness of the system.

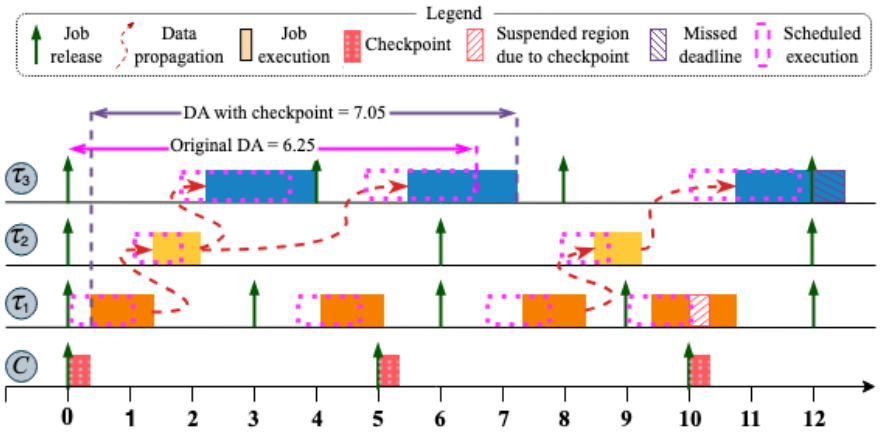


Figure 3.1 Effect of checkpointing on a three-task task chain when no fault is detected

To understand the impact of checkpointing, acting like an interrupt, consider a system with three data-dependent tasks, as shown in Fig. 3.1, denoted as τ_1, τ_2, τ_3, which are scheduled on a non-preemptive fixed-priority scheduler running on a single-core processor. Task τ_1 holds the highest priority, while τ_3 has the lowest. The tasks have execution times and periods as follows: τ_1 (1, 5), τ_2 (0.75, 6), and τ_3 (1.75, 4). These tasks form a task chain, where the output of the first job of τ_1 feeds into the first job of τ_2, and from τ_2 to the first and second job of τ_3. Similarly, the data produced by the third job of τ_1 is consumed by the third job of τ_3. Here, there is no release time delay for the jobs within the hyperperiod. The checkpointing period is set to 5 time units, with an overhead of 0.4 time units. We observed that checkpointing can significantly influence data age. For example, the data age for the first job of τ_1 increased from 6.25 to 7.05 time units. Moreover, the third job of τ_3 missed its deadline due to these changes.

In this chapter we describe our system model in Section 3.1. Section 3.2 discusses the data age analysis framework without checkpointing, forming our research's foundation. Moving on to Section 3.3, we analyze the impact of checkpoints and faults on the reachability of tasks in task chains. We then dive into the discussion of *interference* resulting from checkpointing in Section 3.4. Section 3.5 extends our framework to multi-core setups with shared global buffer. Finally, in Section 3.6, we present comprehensive experimental results that demonstrate the effects of fault tolerance on data age, while utilizing the proposed re-execution strategy.

3.1 System Model

In this work, we focus on a standard automotive application, denoted as S, which comprises a set of periodic tasks. The application may be mapped onto a single-core or homogeneous multi-core platform. We assume task scheduling follows a task-level fixed-priority (TLFP) policy and that jobs execute using run-to-completion semantics. Our work begins by establishing a data age calculation framework on a single-core platform. This framework lays the groundwork for our research as it involves understanding and predicting how fault tolerance mechanisms affect the data age. We then extend our framework for analyzing end-to-end data age on a multi-core platform.

3.1.1 Application model

The application is modeled as a set of n data-dependent periodic tasks, denoted as $\Gamma = \{\tau_1, \tau_2, \tau_3, ..., \tau_n\}$. Each task, τ_i, is described by the tuple $\{C_i, T_i, D_i, \psi_i\}$, representing worst-case execution time(WCET) C_i, period T_i, deadline D_i, and optional offset ψ_i. τ_1 has the highest priority, and τ_n has the lowest. All tasks have implicit deadlines, i.e., $T_i = D_i$. The j^{th} job of task τ_i is denoted as $\tau_{i,j}$. The hyperperiod (T_{HP}) is defined as the least common multiple of all task periods in Γ, i.e., $T_{HP} = LCM(T_i, \forall\ \tau_i \in \Gamma)$. Consequently, task τ_i can execute multiple jobs within one hyperperiod.

This application is represented by a directed acyclic graph (DAG) called the data propagation graph (DPG). The graph, denoted as $G(\Gamma, E)$, consists of nodes in set Γ, representing tasks, and edges in set E, representing data dependencies between tasks. An edge from τ_a to τ_b signifies that τ_a produces output data which gets consumed by τ_b, making τ_a a '*data producer*' and τ_b a '*data consumer*' of τ_a. As seen in Fig. 3.2, a system can have multiple producers and consumers.

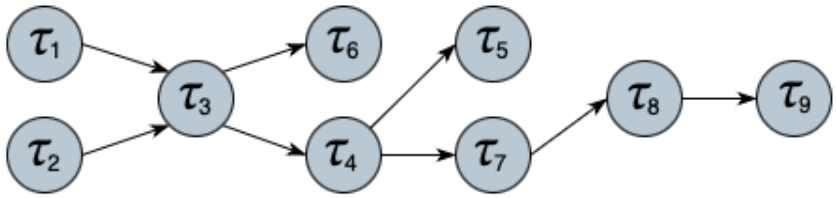


Figure 3.2 Representation of cause-effect chains as Directed Acyclic Graphs(DAG)

For a given task set Γ, we define a set of *cause-effect chains* Π, which may include forks and joins. To simplify our analysis, we break down these cause-effect chains with forks and joins into distinct sequential chains, called *task chains*. This set of task chains is denoted by A. In Figure 3.2, each possible path from the root to a leaf node represents a task chain, A_i, in set 'A'. A *chain instance*, denoted by α_j, refers to a partial order of jobs that appear in a schedule and follow the same execution order of tasks as task chain, A_i. Our analysis focuses on a single Observation Window OW, which is defined as the smallest time interval after which the task chains' pattern recurs. Additionally, OW must be an integer multiple of the hyperperiod T_{HP}. Note that multiple chain instances can exist over an OW. For a chain instance of α_i, the first data producer job will be called *source* and last consumer job, *sink* .

3.1.2 Data communication model

In this work, we assume the implicit communication model of AUTOSAR[3], which employs the read-execute-write semantics to enhance determinism. In implicit communication model, tasks retrieve all necessary inputs upon initiation and store local copies in their buffer memory. During execution, tasks access these local copies, and after execution, outputs are transferred to global shared data buffers, making them available for dependent tasks. Essentially, input and output values are read and written at deterministic points in time. The time required to safely access these shared buffers has been accounted for in the tasks' WCET. It is to be noted that the implicit communication model does not impose synchronization between data consumers and producers.

3.1.3 Fault and fault-tolerance model

3.1.3.1 Fault model

In this study, the fault model focuses on transient faults. These faults are short-lived and do not cause permanent damage. The study assumes a maximum of K faults occurring within a duration T_{HP}, and can randomly impact any core within the system. The most commonly used fault distribution model assumes these transient faults follow a Poisson distribution with a mean arrival rate λ representing their frequency [41].

$$P(N(t) = K) = (\lambda)^K \frac{e^{-\lambda}}{K!} \tag{3.1}$$

Here, Eqn. (3.1) provides the probability of a fault occurring during any given time interval t, where $N(t)$ represents the expected number of faults occurring during the interval T_{HP}, here equal to K.

3.1.3.2 Fault detection model

For single-core systems, the employed fault detection mechanisms primarily revolve around error detection codes and checksum verification. Error-correcting codes (ECC) in memory, parity checks, and checksums computed on critical system data structures aid in detecting faults. Deviations from expected checksum values or parity errors indicate potential faults.

In the context of homogeneous multi-core systems, each core is assumed to be identical and runs tasks in sync time. Tasks are statically pre-assigned to specific cores, meaning that once a task is assigned to a core, it does not migrate to other cores during runtime. All cores share a global shared memory, facilitating communication and coordination among tasks.

Fault detection in this multi-core environment involves inter-core consistency checks. This includes comparing checkpoints and error detection codes across all cores. The assumption of homogeneity ensures that the same fault-detection mechanisms are applied uniformly across all cores. Any inconsistencies detected during these inter-core checks indicate potential transient faults affecting the shared global memory or the execution state of tasks on individual cores.

3.1.3.3 Fault tolerance model

The fault tolerance strategy employed in this study relies on a checkpointing system that acts as an interrupt within a non-preemptive scheduled taskset. Periodic checkpoint creation captures the system state to recover from faults. To implement checkpointing, tasks are periodically interrupted at defined checkpoints, ensuring a consistent system state for recovery purposes. The implementation of checkpointing within the non-preemptive scheduling scheme introduces certain trade-offs and overheads. While providing enhanced fault tolerance, it might introduce increased latency or interruptions during task execution due to the need for checkpoint creation at specific intervals. For this work we only re-execute the job which got interrupted.

To illustrate this, consider the system shown in Fig. 3.5: after a fault is detected, $\tau_{2,3}$ is re-executed, taking the necessary input from the previous successful checkpoint. However, as part of our approach, the jobs $\tau_{2,2}$, $\tau_{3,2}$, and $\tau_{1,2}$ are not reexecuted. This design decision

optimizes the system's performance by avoiding the reexecution of jobs that may not have been affected by the detected faults, consequently reducing the overall computational overhead.

3.2 Data age calculation

In this section, we recapitulate the calculations of data propagation paths for systems with TLFP scheduling which provides a deterministic timing behavior. In such schedule, in the absence of faults, we can find the order of job execution during the design time. On the other hand, for dynamically scheduled systems it is often possible to compute the Worst-Case Response Time (WCRT), but the exact times when the tasks will be executed are not known. However, the earliest and latest time a task can be executing is known [7].

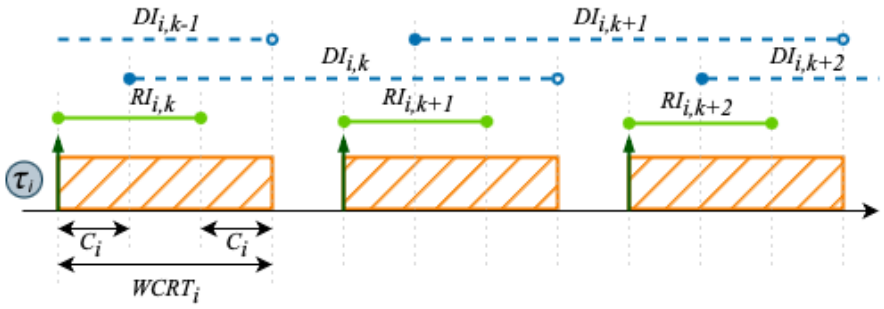


Figure 3.3 Read interval (RI) and Data interval (DI) when WCRT is known

Schedule agnostic reachability framework: The notions of *read interval* and *data interval* are key in deciding the valid span of communication between the jobs. For a job $\tau_{i,j}$ the *read interval*, $RI_{i,j}$, is defined as the interval starting from the earliest time a job can potentially read its input data, $R_{min}(\tau_{i,j})$, until the latest possible time a job can do so without violating its timing constraints, $R_{max}(\tau_{i,j})$. Similarly, the *data interval*, $DI_{i,j}$, is defined as the interval from the earliest time the output data of a job can be available, $D_{min}(\tau_{i,j})$, up to the latest time a successor job of the same task overwrites the data, $D_{max}(\tau_{i,j})$. Hence,the read interval $RI_{i,j}$ is the interval $[R_{min}(\tau_{i,j}), R_{max}(\tau_{i,j})]$, and the data interval is $[D_{min}(\tau_{i,j}), D_{max}(\tau_{i,j})]$. These concepts are depicted in Fig. 3.3 for jobs of a task τ_i. The expressions of RI and DI for different levels of known information are shown in Table 3.1 rows 2 to 4, as explained in Becker et al. [7]. Note that RI and DI are properties of a job subject to their schedule. For example, if tasks

	$R_{min}(\tau_{i,j})$	$R_{max}(\tau_{i,j})$	$D_{min}(\tau_{i,j})$	$D_{mam}(\tau_{i,j})$
No Knowledge	$\psi_i + (j-1) \cdot\ T_i$	$j \cdot T_i - C_i$	$R_{min}(\tau_{i,j}) + C_i$	$R_{max}(\tau_{i,j+1}) + C_i$
Exact Schedule	$start_{i,j}$	$R_{min}(\tau_{i,j})$	$end_{i,j}$	$end_{i,j+1}$
Known WCRT	$\psi_i + (j-1)C_i$	$R_{min}(\tau_{i,j}) + WCRT_i - C_i$	$R_{min}(\tau_{i,j}) + C_i$	$R_{max}(\tau_{i,j+1}) + Ci$
	Successful checkpoint			
Partial overlap	$start_{i,j} + \delta$	$start_{i,j} + \Delta - \delta$	$R_{min}(\tau_{i,j}) + C_i$	$R_{max}(\tau_{i,j+1}) + C_i$
Full overlap	$start_{i,j}$	$start_{i,j} + \Delta$	$R_{min}(\tau_{i,j}) + C_i + \Delta$	$R_{max}(\tau_{i,j+1}) + C_i$
	Fault occurrence			
Partial overlap	$start_{i,j} + \delta$	$start_{i,j} + \Delta - \delta$	$R_{min}(\tau_{i,j}) + C_i$	$R_{max}(\tau_{i,j+1}) + C_i$
Full overlap	$start_{i,j} + \delta + \Delta$	$start_{i,j} + C_i - \delta + \Delta$	$R_{min}(\tau_{i,j}) + C_i$	$R_{max}(\tau_{i,j+1}) + C_i$

Table 3.1 R_{min}, R_{max}, D_{min} and D_{max} expressions for data age calculation with different timing information

from same taskchain are scheduled on different cores, then RI and DI will be determined based on schedule these tasks will have on the cores they are mapped to.

Checking for reachability between jobs: As mentioned in [7], in order for a job $\tau_{k,l}$ to consume data of a job $\tau_{i,j}$, the data interval of $\tau_{i,j}$ must intersect with the read interval of $\tau_{k,l}$. If there is an overlap, the function $follows(\tau_{i,j}, \tau_{k,l})$ is defined to return *true*, irrespective of number of core(s):

$$follows(\tau_{i,j}, \tau_{k,l}) = \begin{cases} true, & \text{if } RI_{k,l} \cap DI_{i,j} \neq \phi \\ false, & \text{otherwise} \end{cases} \tag{3.2}$$

3.2.1 Determining plausible chain instances

A recursive function is used to calculate all possible chain instances of a task chain in the observation window, OW. This function takes in a job from the source node and finds the path(s) till it reaches a job of the sink node. Consequently, this needs to be done for all jobs of a source node of the task chain and of all task chains, within the OW. The function starts at the first (source) node of the task chain; for the initial job, all jobs of the second task of the chain are considered where $follows()$ returns true. To these nodes, a logical data path is

created from the initial job. The same principle is applied from each level of these nodes to find the plausible jobs of the next level node of the task chain. Once the last (sink) node is reached function returns and all paths that reached the sink node. These paths are plausible chain instances. A detailed explanation of the method is presented in [7].

3.2.2 Maximum Data age calculation

For a given chain instance, α_i, the maximum end-to-end delay and thus the data age can be computed as follows, where τ_{source} is a job of the source task of the chain instance, and τ_{sink} is a job of the sink task:

$$AgeMax(\alpha_i) = (R_{max}(\tau_{sink}) + C_{sink}) - R_{min}(\tau_{source}) \tag{3.3}$$

To compute the maximum data age for all possible paths in the system, $AgeMax()$ must be computed for all computed chain instances of all task chains. The maximum of these values is the maximum data age of the cause-effect chain.

3.3 Checkpoint-centric reachability framework

Adding checkpointing as an interrupt to TLFP scheduling brings a dynamic element into the usually predictable system. In this setup, task priorities stay the same, but checkpointing briefly interrupts tasks to save the system's state, causing a time overhead. With periodic interruptions for checkpointing, the WCET of tasks can grow because of the extra time needed and potential delays in task resumption after a fault. Therefore, introducing checkpointing in a system requires a careful analysis to understand how it affects worst-case response times, ensuring that timing requirements and system reliability are maintained. In this section, we explore the effects of checkpointing solely on the interrupted job in two scenarios: when a fault is detected during checkpointing and when no fault is detected. Later, in Section 3.4, we'll examine its effects on other jobs.

Our analysis begins by analyzing the schedule using both, the best-case execution time (BCET) and WCET, providing the earliest start time and worst finish times, $start_{i,j}$ and $end_{i,j}$ respectively, for each $\tau_{i,j}$ in the task set. Using this information, we adjust the RI and DI for the schedule-centric reachability framework (refer to the'Exact schedule' row in Table 3.1). Note

that this method is less pessimistic than the one that factors in checkpoint overhead in the WCET of all jobs.

Checkpointing instances can interact with the existing task schedule in three primary scenarios: *no overlap*, *partial overlap*, and *full overlap*, as illustrated in Fig. 3.4. Here, Δ represents the checkpoint overhead. In the partial overlap scenario, the checkpointing task initiates before the job and overlaps with its start time, requiring that the difference in start times between the checkpoint task and the job remains less than Δ.

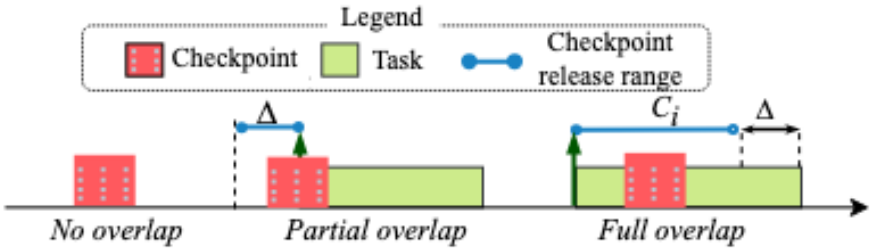


Figure 3.4 Checkpoint interrupt conditions

3.3.0.1 Reachability with no faults detected

When no faults are detected, a successful checkpoint is created. As illustrated in Fig. 3.4, only partial overlap and full overlap affect the RI and DI of the interrupted job. As seen in Table 3.1, for partial overlap case, the shift in RI can range from δ to $\Delta - \delta$, δ represents a very small amount of time. For the full overlap case, there is no change in the earliest read time, and the maximum shift in read time can be Δ. DI adjusts accordingly.

3.3.0.2 Reachability with faults detected

As observed from Table 3.1, a partially overlapping checkpoint interrupt behaves identically whether a fault is detected or not. In this case, the checkpoint can delay the start of job execution by a minimum of δ and a maximum of Δ.

For a checkpoint to cause full overlap, it has to start after the job begins executing, it can be at minimum δ and maximum $C_i - \delta$ from $start_{i,j}$. When a fault is detected, immediate job re-execution is triggered, incurring an additional overhead of Δ on the RI of the re-executed

task. It is essential to note that these updated read and data intervals do not account for interference caused by checkpoints.

3.4 Checkpointing interference

Using checkpointing mechanisms for fault tolerance or data backup introduces *interference*, which affects the interrupted job and subsequent jobs. *Interference* can be defined as variation in the schedule of a job due to the presence of an interrupt, such as a checkpoint. It can lead to disruptions in the system's data age. Fig. 3.5 provides an example of such interference. To address this, we propose a method to calculate the interference caused by checkpoint instances or similar events, like periodic interrupts.

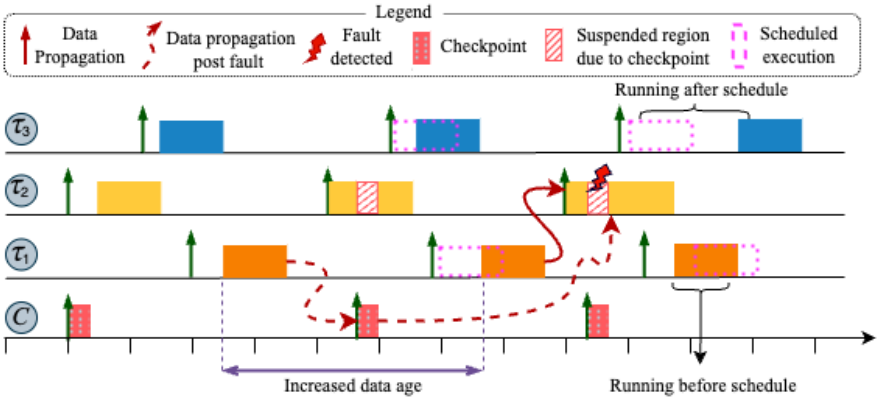


Figure 3.5 τ_1, τ_2, and τ_3 forms a task chain. (a) $\tau_{3,3}$ is executing after its scheduled start time (b) $\tau_{1,3}$ is executing before its scheduled start time (c) $\tau_{2,3}$ is re-executing due to fault detection. It is now taking data from $\tau_{1,1}$, instead of $\tau_{1,2}$. Due to this, there is an increase in the data age of the task chain

In a fixed priority scheduling system, interference quantifies the impact of a checkpointing task on the execution of subsequent jobs. Based on this we determine changes in the RI and DI of jobs. The effect of each checkpoint instance is evaluated individually to assess its influence on the system's overall schedule. After updating the RI and DI of the interrupted job from Table 3.1, we calculate interference factors for all the subsequent jobs that get affected due to the change in the interval of the interrupted job, considering both the upper and lower bounds of the RI (R_{min} and R_{max}) and update the corresponding DI (D_{min} and D_{max}) accordingly.

3.4.1 Updating R_{max}

In the system, prior to the introduction of checkpoints, the calculation of R_{max} for all jobs was performed, as detailed in Section 3.3. However, the incorporation of checkpoints necessitates an examination to determine if the R_{max} values should be updated. This process individually involves going through each job scheduled on the core to compute their respective R_{max} values.

The R_{max} calculation follows an iterative approach. For each job, an initial value of R_{max} is established, as outlined in Section 3.3. Subsequently, the maximum delay in RI is computed using *Delay function* and *ReExecution function*, as explained below. If the calculated R_{max} for a job is less than its previously assigned value, no update is made, and the process concludes. However, if it exceeds the previous value, the R_{max} is updated, and the verification process is reiterated, as shown in Algorithm 1 ($R_{max}Calculator$). This step is crucial as the updated R_{max} may result in new instances of checkpoints and re-executions in between. For instance, as depicted in Figure 3.1, subsequent to the alteration in RI for $\tau_{3,2}$ due to the first checkpoint, further adjustments are necessary to accommodate second checkpoint.

Definition 3.4.1. *The duration gap denotes the duration between the release of a job and its R_{max}. "gap" is a function of R_{max}.*

$$gap = R_{max}(\tau_{i,j}) - ReleaseTime(\tau_{i,j}) \tag{3.4}$$

For each job, the calculation involves determining the maximum possible delay it might encounter from its release until it begins execution. This delay comprises the cumulative effects of:

1. Being blocked by a lower-priority job
2. Awaiting the completion of higher-priority job
3. The presence of checkpoint instances during the *gap* period
4. Re-executions due to fault detection

The *Delay function* addresses factors 1, 2, and 3, while the *ReExecution function* handles factor 4. Further clarification of these functions is provided below.

Delay function: $Delay(\tau_{i,j})$ analytically calculates the sum of delay because of worst-case blocking due to lower priority jobs, the delay due to execution of higher priority jobs released during *gap* and the overhead due to the number of checkpoints released, as shown in Eqn. 3.5. Here, the checkpointing period is T_{ch}, 'l' represents tasks of lower priority, i.e. $n \geq l > i$, and 'h' denote tasks of higher priority, i.e. $1 \leq h < i$.

$$Delay(\tau_{i,j}) = \max_{\{i<l\leq n\}}(C_l) + \sum_{h=1}^{i-1} C_h \left\lceil \frac{gap}{T_h} \right\rceil + \Delta \left\lceil \frac{gap}{T_{ch}} \right\rceil \tag{3.5}$$

Algorithm 1 R_{max} calculator

Data: $ReleaseTime(\tau_{i,j}), R_{max}(\tau_{i,j})$

Result: *delay* or updated $R_{max}(\tau_{i,j})$

$k = 0$;

$gap_k = R_{max}(\tau_{i,j}) - ReleaseTime(\tau_{i,j})$;

$ReExecution(\tau_{i,j}, k)$\{ Returns re-execution delay\};

while *true* **do**

  if $R_{max}(\tau_{i,j}) + C_i > D_i$ **then**

    return Timing constraint violated;

  end

  $delay = Delay(\tau_{i,j}, k) + ReExecution(\tau_{i,j}, k)$;

  if $delay < gap_k$ **then**

    return $R_{max}(\tau_{i,j})$;

  else

    $gap_{k+1} = delay$;

  end

  $R_{max}(\tau_{i,j}) = ReleaseRime(\tau_{i,j}) + gap_{k+1}$;

  $k = k + 1$;

end

ReExecution function: This function computes the worst-case delay for a job attributed to re-executions. For a job, the maximum number of re-executions that can occur within its *gap* is equivalent to the number of checkpoints within that interval, denoted by n in eqn.(3.6). To determine the worst-case execution time of these re-executions, we compile a list of jobs released during the *gap* and identify those with the highest WCET. By focusing on these jobs, it is

ensured that any other potential re-executions won't take longer than these selected ones. The number of selected jobs matches the number of checkpoints within the *gap*. By considering re-executions at all checkpoints with jobs possessing the highest possible execution time within the *gap*, we effectively estimate the worst-case delay due to re-execution.

$$\begin{gathered}\text{ReExecutionDelay} = \sum_{i=1}^{n}(\text{RE}_i), \text{ where,} \\ \text{RE} = \left\{ C_i \;\middle|\; \text{Release}(\tau_{i,j}) \in \text{ gap, sorted by } C_i \text{ in descending order}, n = \left\lceil \frac{\text{gap}}{T_{ch}} \right\rceil \right\}[:n]\end{gathered} \tag{3.6}$$

In Algorithm 1, we add the outputs from $Delay()$ and $ReExecution()$ functions to $ReleaseTime(\tau_{i,j})$ to get the updated R_{max} for each job. After each calculation, we check if R_{max} has been updated. If it has, we go through another iteration of verification. Note that this approach considers worst case for reexecutions. Using this approach for systems that define a maximum number of faults in a hyperperiod or minimum time interval between faults, we can determine less pessimistic RI and DI.

3.4.2 Updating R_{min}

In Fig.3.5, we can see that the introduction of checkpointing can sometimes result in job execution starting earlier than expected. To account for this, we go through each checkpoint instance, as given in for loop of Algorithm 2 ($R_{min}Calculator$), to determine if R_{min} needs to be updated for the interrupted job and subsequently affected jobs. When a job gets interrupted by a checkpoint, it can either resume its execution or start a re-execution process. We begin by creating a list of higher-priority jobs, $JRLs$, that are released between the start and finish times of the interrupted job. As these are the jobs that will be in the queue when the current executing task finishes. We then iterate through this list from the job of highest priority to lowest, since the highest priority job will start executing once the interrupted job finishes execution. For the highest-priority job in this list, we update its R_{min} to the finish time of the current job being examined. This updated job becomes the new "current job", and the process repeats recursively to identify any additional jobs that could be released during the execution of the current job, provided they have higher priority than the interrupted job. These newly identified jobs are added to the list, and the process continues until the list is empty.

Algorithm 2 $R_{min}Calculator$

Data: $\tau_{i,j}, S_{i,j}, F_{i,j}, JRls[\,]$

$//R_{min}(\tau_{i,j})$ is R_{min} before checkpoint interrupt

// $S_{i,j}$ & $F_{i,j}$: Start and Finish time of $\tau_{i,j}$

Result: fin -Finish time of last affected job

// 'i' is also the priority of $\tau_{i,j}$

$R_{min}(\tau_{i,j}) = min(R_{min}(\tau_{i,j}), S_{i,j}))$;

$fin = F_{i,j}$;

forall $\tau_{p,q}$ *released between* $S_{i,j}$ & $F_{i,j}$ **do**

  if $p < i$ **then**

    $//p$ is of higher priority

    insert $\tau_{p,q}$ to the sorted JRls[ ];

    if *JRls[] is not empty* **then**

      //call the '$R_{min}Calculator$' for the highest priority job in JRls- Say $\tau_{a,b}$

      $fin = R_{min}Calculator(\tau_{a,b},\ fin,\ fin + C_a,\ JRls)$;

    end

  end

end

return fin

3.5 Multi-core CPU platform

In the previous sections, we established a theoretical framework to analyze how checkpointing affects TLFP scheduling in single-core real-time systems. We now extend our framework to multicore platforms.

3.5.0.1 Task mapping

In our research, we focus on a homogeneous multi-core CPU platform. Tasks are statically allocated to the cores, and they remain fixed without migration. This distribution is done randomly and WCET is obtained considering the distribution.

3.5.0.2 Communication model

Tasks across cores share a global memory buffer. Multi-core systems often rely on locks and semaphore techniques for synchronization. Having multiple cores accessing a single shared buffer system increases the memory access overhead. This extra overhead is assumed to be a constant γ. We get a total memory access overhead during checkpointing in multicore platform, Δ_{MC} as:

$$\Delta_{MC} = \Delta + \gamma \tag{3.7}$$

3.5.0.3 Fault model

In our study, we assume that faults are independently distributed among the cores of the system. To maintain synchronization, checkpointing is carried out simultaneously across all cores. We also consider a flexible fault detection model that accommodates various fault detection techniques. However, it's important to note that the recovery process exclusively occurs during checkpoint instances, ensuring synchronization across all cores. During the checkpointing process, if a fault is detected in any of the cores, a specific protocol is followed. The global shared memory is rolled back to the state captured by the previous checkpoint. Additionally, all the jobs that were interrupted at the time of the fault detection are re-executed. This rollback mechanism is employed because a fault occurring in one core or memory element can propagate to other cores due to the shared memory resource.

3.5.0.4 Interference

The calculation of RI and DI remains consistent for the multicore platform, as these properties are inherent to each job and depend on the scheduling on the core to which the job is assigned. Therefore, the algorithm used to calculate interference, which relies on these RI and DI values, also remains unchanged in this context. Note that the checkpointing overhead may vary in a multicore environment due to factors such as synchronization across cores.

3.6 Evaluation

In this section, we comprehensively evaluate our proposed approach for analyzing the end-to-end delay of a fault-tolerant system under varying levels of system information availability.

Our experiments focus on (a) demonstrating the importance and reliability of the proposed framework, (b) assessing the accuracy of derived bounds of RI and DI, and (c) measuring the computation time of the framework in presence of faults. We conducted experiments on an Intel i5-11300H processor with 8GB of RAM, performing sequential computations without parallel processing.

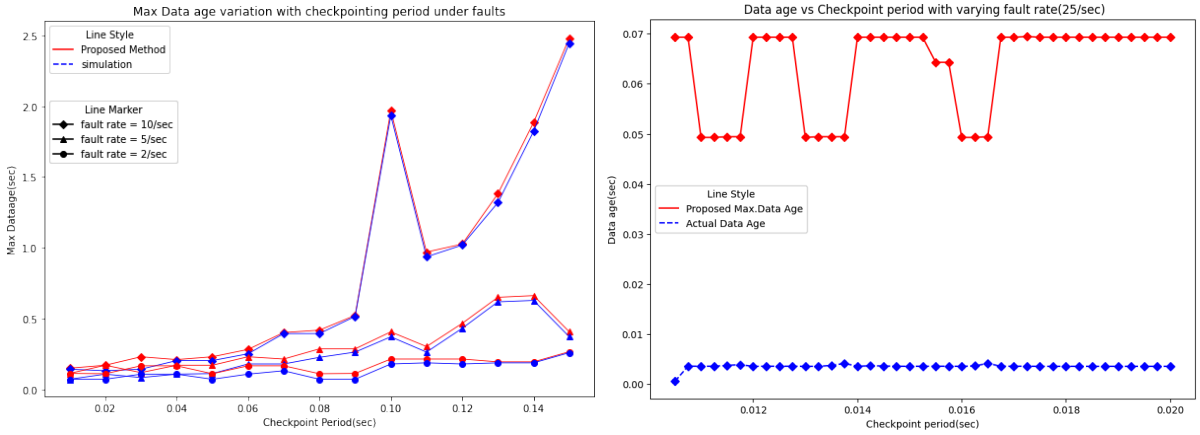
Experimental Setup: This comprehensive application of our approach encompasses both real-world automotive benchmarks[3] and synthetically generated tasksets, providing a robust evaluation across diverse contexts. All tasks are periodic tasks with implicit deadlines, and their priorities are decided based on the rate monotonic (RM) scheduling policy. For tasksets generated using [3], the periods of all tasks are automotive-specific, and they are drawn at random from the set $\{1, 2, 5, 10, 20, 50, 100, 200, 1000\}$ with an associated probability mentioned in [3]. These periods are semi-harmonic in nature, that means while there is a harmonic relationship between task periods, it might not follow a strictly regular harmonic sequence, there could be deviations in the periods that do not strictly adhere to the harmonic progression. The WCET C_i of each task τ_i in the task set is deduced based on its period T_i and utilization U_i generated by the UUniFast approach from [10]. Although our framework considers release time offset, we assume zero offsets for experimental simplicity. The OW for data propagation paths is set to six times the hyperperiod, i.e. $6 * T_{HP}$. The average-case execution time (ACET), and the WCET for τ_i is generated according to [3].

To create random data-propagation graphs, we assign edges from each task to other tasks, ensuring no cycles are formed. We limit the maximum branching from the node and the maximum number of inputs into the nodes to 3. The maximum depth of the task graph (i.e. task chain length) is set to 8.

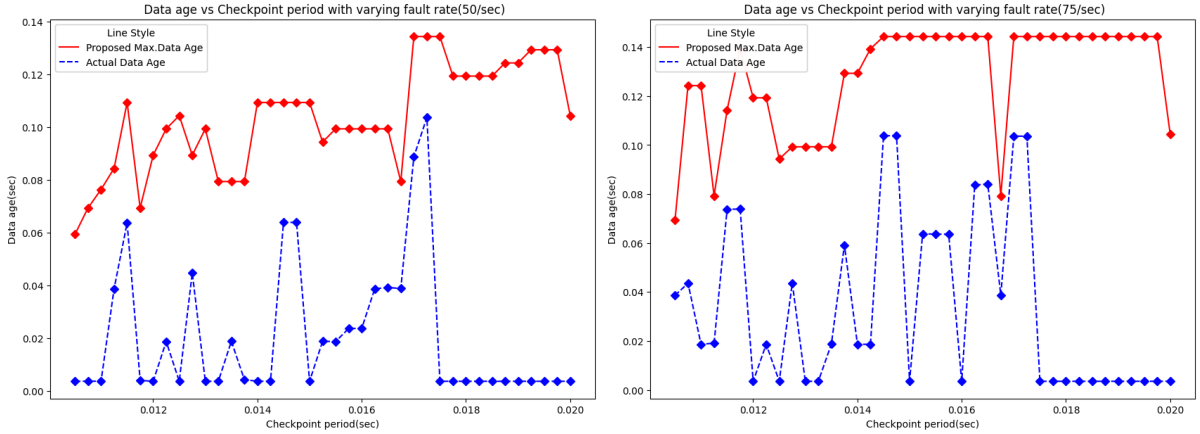
In subsequent sections and figures, the term 'proposed analysis' refers to a system implementing our proposed solution and calculating data age through our approach. On the other hand, 'actual data age' signifies the real-time duration taken by data, starting from its initial consumption to the conclusion of the job where it is last utilized. This distinction clarifies the specific contexts and metrics employed throughout our analysis.

3.6.1 Single core processor

Experiment 1 (Fault Rate Variation): In this experiment, we delve into the impact of checkpointing periods on data age within a real-time system across varying fault rates. Our system comprises 14 tasks, with task chain lengths spanning from 3 to 8.



(a) Synthetic taskset with varying fault rates (b) AUTOSAR taskset with 25 faults/second



(c) AUTOSAR taskset with 50 faults/second (d) AUTOSAR taskset with 75 faults/second

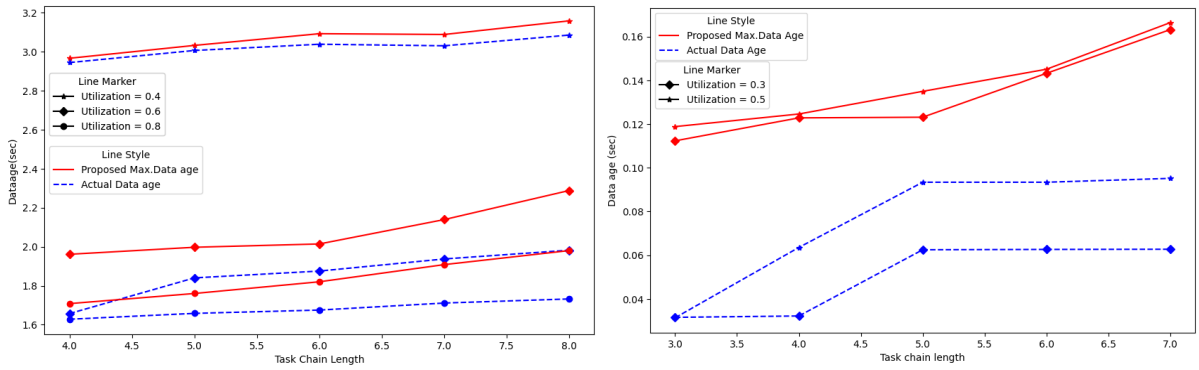
Figure 3.6 Variation of maximum data age with checkpointing period under varying fault rates for single-core processor

Figure 3.6(a) illustrates the fluctuation in data age concerning different checkpoint frequencies for a synthetic task set. Notably, in this task set, periods are non-harmonic. Conversely, Figures 3.6(b), 3.6(c), and 3.6(d) showcase the maximum data age for an automotive benchmark task

set characterized by semi-harmonic periods. It is observed that, even with higher fault rates, the data age in benchmark taskset remains lower than synthetic tasksets.

We've observed that higher fault rates result in increased data age. This underscores the need for dataage calculation techniques in understanding the adverse impact of faults on data age in real-time systems.

Experiment 2 (Varying task chain length): In this experimental study, we investigate the influence of varying task chain length on data age in a real-time system, considering different checkpointing frequencies and system utilization. The task chain lengths are explored in the range from 3 to 8 while maintaining a fault rate of 5 faults per second for a synthetic taskset and 50 faults per second for a benchmark taskset. As we can see in Fig. 3.7, dataage of system increases with an increase in task chain length.



(a) Synthetic taskset with 5 faults/second (b) AUTOSAR taskset with 50 faults/second

Figure 3.7 Variation of maximum data age with task chain length under varying utilization for single core processor

3.6.2 Multi core processor

Experiment 1 (Checkpoint period variation): For this experiment, we focus on the impact of checkpointing periods on data age on a dual-core platform for a fault rate of 50 faults per second. Our system comprises 14 tasks, with task chain lengths spanning from 3 to 8. In Fig. 3.8, for benchmark tasksets we observe a dip in data age between 0.012 second and 0.014 second checkpoint this signifies a local optimal checkpoint period in the experimented range.

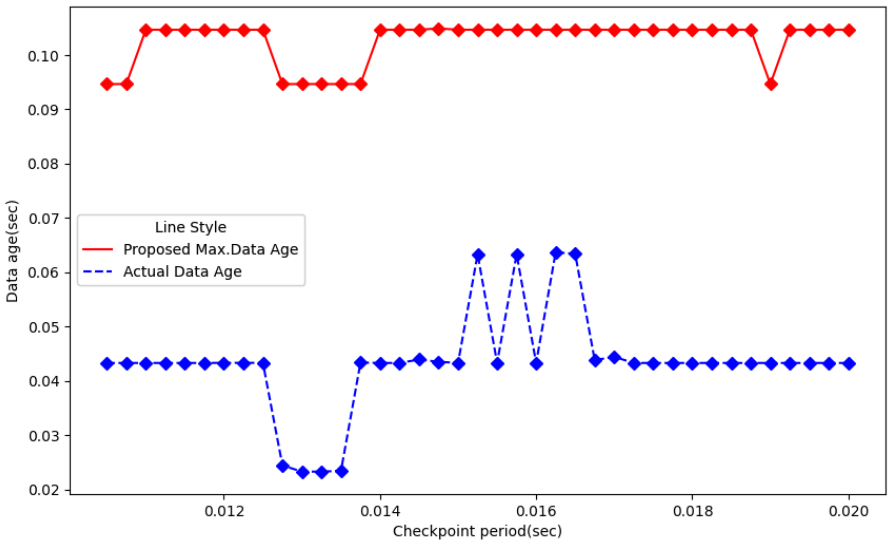


Figure 3.8 Variation of maximum data age with increasing checkpointing period with 0.8 utilization and 50 faults per second

Experiment 2 (Varying task chain length): In this experiment, conducted on a dual-core platform, we observed a trend, as the task chain length and utilization increase, the data age exhibits a corresponding upward trajectory. Fig. 3.9 suggests a direct correlation between the distribution of tasks of taskchain and memory access overhead incurred.

3.6.3 Computation time analysis

The objective of this experiment was to assess the CPU runtime necessary for computing data age under the specified parameters. Fig. 3.10 displays the time required to determine the maximum data age of the system using simulations when only one of the potential cause-effect chain scenarios occurred. Meanwhile, in Fig. 3.11, the total simulation time is presented, considering a substantial portion of the potential paths. It's worth noting that there might be additional paths beyond what is depicted. This calculation of the number of data propagation paths is adapted from [7].

The computational time for estimating the maximum data age, considering all potential chain instances, utilizing our proposed method approximates $1e2$ seconds. Nonetheless, when we venture into the simulation of the possible number of instances[7], aimed at calculating the maximum data age in the presence of faults, we encounter a substantial increase in computational

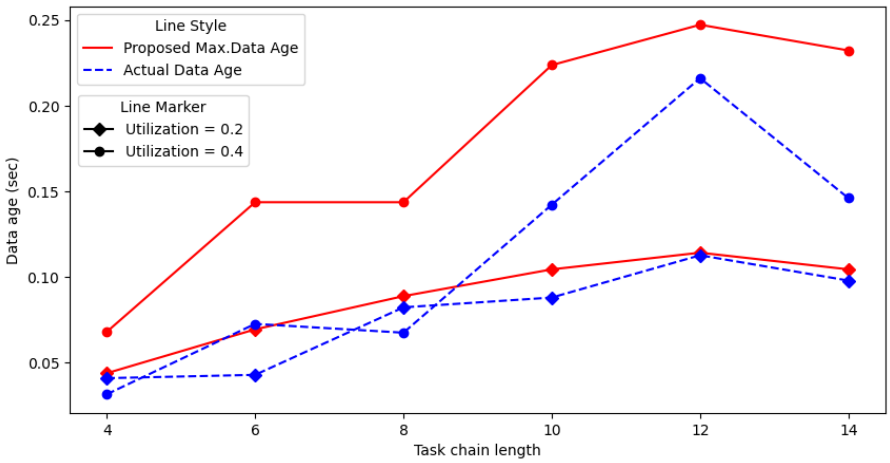


Figure 3.9 Variation of maximum data age with task chain length under varying utilization for multicore processor. Fault rate is 50 per second

time, scaling to approximately $1e6$ seconds. This notable escalation in processing duration can be attributed to the exponential surge in potential scenarios, particularly when dealing with higher number of node task chains or heightened system utilization, as shown in Fig. 3.11. Note that computation time depends on the taskset chosen and the observation window. Such scenarios demand an extensive amount of computational resources to explore comprehensively, hence contributing to the observed increase in computational time.

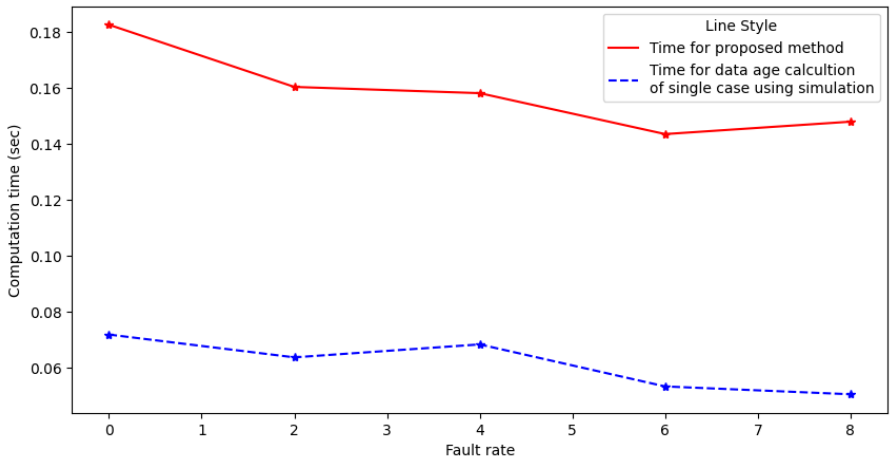


Figure 3.10 Time needed to calculate maximum data age of the system with 4 faults per second. The proposed method finds the maximum data age for all possible chain instances in a cause-effect chain while the simulation is for just one of the possible scenarios

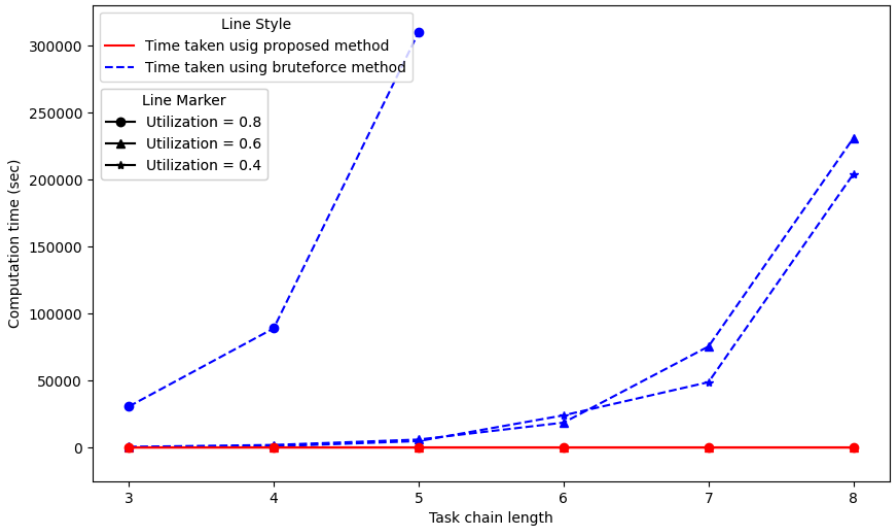


Figure 3.11 Time needed to calculate maximum data age of the system with 5 faults per second. Both, the proposed method and simulation, find the maximum data age of the cause-effect chain in all possible scenarios

Chapter 4

Thermal-Aware Data age Analysis of Task Chains

4.1 Problem Formulation

4.1.1 Task Model

The application is modeled as a set of n data-dependent periodic tasks, denoted as $\Gamma = \{\tau_1, \tau_2, \tau_3, ..., \tau_n\}$. A task τ_j is represented as a tuple $(P_j, e_j^{max}, e_j^{min}, D_j)$, where P_j denotes the period, e_j^{min} indicates the best-case execution time (BCET), e_j^{max} signifies the worst-case execution time (WCET), and D_j denotes the relative deadline of τ_j. The i^{th} job of task τ_j is denoted as $\tau_{j,i}$, and the execution time of $\tau_{j,i}$ is denoted as e_j^i. Implicit deadlines are assumed for this study, where $D_j = P_j$, although the proposed method can be readily extended to cases where $D_j \neq P_j$. A task is classified as a *producer* when it generates data for another task to process and consume. The receiving task is termed the *consumer*. The hyperperiod (HP) is defined as the least common multiple of all task periods in the task set, i.e., $HP = LCM(P_i,\ \forall\ \tau_i\ \epsilon\ \Gamma)$. Consequently, τ_i can execute multiple jobs within one HP.

Task Chains: Within this set of tasks, a subset of tasks can be organized into specific sequences called task chains. A task chain represents a series of data-dependent tasks where the output data generated by a task serves as the input data for the subsequent task in the sequence. There can be multiple task chains within an application, and a single task might participate in multiple chains depending on the data flow.

Context Switch Time: The additional delay incurred due to context switching, within the range denoted as $[ct^{min}, ct^{max}]$ can be considered as part of the execution time. Thus, the total execution time range for task τ_j becomes $[e_j^{min} + ct^{min}, e_j^{max} + ct^{max}]$. Consequently, the entire

proof discussed in subsequent sections remains valid even if a context switch or any similar delay is taken into account, as it can be accommodated within the execution time.

4.1.2 System Model

This work focuses on a single-core processing platform for executing task chains [32]. Each task communicates via a globally shared register, a common feature in task chains. Tasks retrieve necessary inputs upon initiation and store local copies in their buffer memory. During execution, tasks access these local copies, and upon completion, outputs are transferred to global shared memory, becoming available for dependent tasks. Notably, input and output operations occur at deterministic points in time, ensuring the deterministic behavior of the system.

4.1.3 Thermal Model

If the processor's average power is $P_{proc}(t)$ and ambient temperature is $T_{amb}(t)$, then the processor's instantaneous temperature is modeled as follows-

$$T(t) = T(0) \cdot e^{-t/RC} + (T_{\text{amb}}(t) + P_{\text{proc}}(t) \cdot R) \cdot (1 - e^{-t/RC}) \tag{4.1}$$

where t is the time instant, R and C are the thermal resistance and capacitance, respectively, and $T(0)$ is the initial temperature of the processor[39]. The peak temperature that a processor should not exceed is denoted by T_{max}.

The thermal management mechanism used in this work is based on the insertion of idle times. The idle times can be inserted as one chunk before the task (as shown in the top half of Fig. 4.1) or the task can be split into multiple chunks and an idle time can be inserted before each chunk (as shown in the bottom half of Fig. 4.1). The idle time required to cool the processor is derived based on the following two cases -

- The temperature at the start time of another task is T_{max}
- The temperature at the start time of the task is less than T_{max} but the execution of the fraction of the task leads to the increase in the temperature to T_{max}, which requires an idle time for the processor to cool off.

In this work, we follow the task splitting mechanism[39], which is described next.

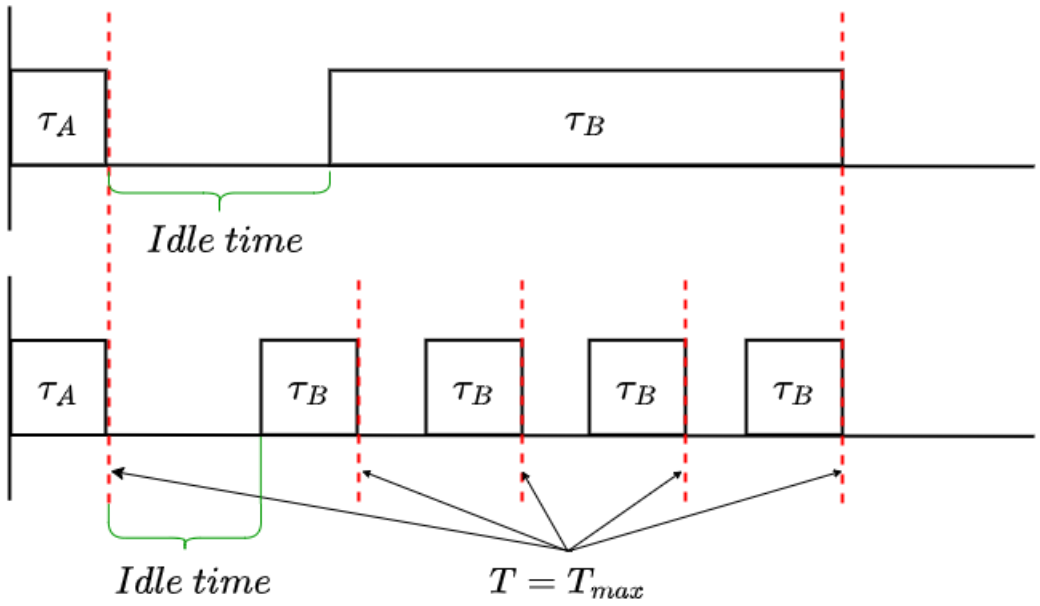


Figure 4.1 Non Splitting vs Splitting Mechanism - The first case (top figure) denotes the non-splitting and the second case (bottom figure) denotes the splitting mechanism. Red lines denote all the cases of temperature, $T = T_{max}$ while considering splitting

Task Splitting: Consider a situation where the temperature at the start of a task is T_{max}, thereby having the need to insert idle time to reduce the processor's temperature. As illustrated in the Fig. 4.1, there are two ways of insertion of idle times -

- Insert the idle time to lower the temperature such that the whole task τ_A can run without reaching T_{max} in between.
- Split task τ_A into m parts such that after the execution of every part, the temperature rises to T_{max}. "m" is chosen optimally such that the preemption overhead cost never outweighs the benefit of splitting.

Theorem 4.1.1. *The total idle time required when the task splitting technique is followed is always lesser than the original non-splitting mechanism.*

Our work builds upon Theorem 4.1.1 proven by Pratyush et al. in [36]. However, Lee et al. in [39] demonstrated that the splitting mechanism can be less effective when preemption

overhead outweighs its benefits. Despite this, we focus on the splitting mechanism for this work.

Let's consider a job execution where idle time is inserted at time t. Denote the system temperature at this time as $T(t)$. We define the execution time of the job's split fraction as $e_j^{i,f}$. Here, the superscript i denotes the job instance and f **represents the fraction of the job that is remaining after the split**. i.e., $e_j^{i,1}$ means no portion of job is finished, while $e_j^{i,0}$ means job execution is complete.

Based on these parameters, we define the idle time function $T_{idle}(T(t), e_j^{i,f})$. This function expresses the duration of idle time inserted strategically within a task chain. The rationale is to manage thermal conditions for the given job by considering:

(a) The temperature, $T(t)$, at the time the idle period is inserted.

(b) The remaining execution time, $e_j^{i,f}$. Where, The fraction, f, denotes execution that's left to be completed

Furthermore, we denote the fraction of the job τ_j that has already been executed by $\tau_{j,i}^f$. Here, i again refers to the job instance and f indicates the remaining portion of the job.

4.2 Two Task Result

In analyzing the dynamics of the two-task scenario, a comprehensive understanding of the intricate interactions between task jobs is crucial for gaining insights into the broader system behavior. Within this system, we have two key players: τ_A, the producer, and τ_B, the consumer. Our objective is to explore all potential job combinations and formulate worst-case data age scenarios for each. Specifically, we focus on observing the behavior over two consecutive periods of τ_A, during which $\tau_{B,1}$ receives input from $\tau_{A,1}$. Additionally, we assume τ_B holds higher priority, preempting $\tau_{A,2}$ and necessitating the retrieval of output from $\tau_{A,1}$.

In this two-task system, the initial temperature significantly shapes its dynamics. **Notably, when task splitting occurs, it indicates that the system tends to reach its peak temperature during job execution, i.e.** $T_{proc} \rightarrow T_{max}$. In this section, to ensure the maximum data age, we systematically explore different job execution sequences, considering whether the initial temperature prompts job splitting or not-

1. $\tau_{A,1}$ Splitting Scenarios:

   (a) Scenario 1: $\tau_{A,2}$ Executes Before $\tau_{B,1}$:

   - Exploring sub-scenarios where $\tau_{A,2}$ may or may not further split.
     - $\tau_{B,1}$ may also split.

   (b) Scenario 2: $\tau_{B,1}$ Executes Before $\tau_{A,2}$:

   - Here, we investigate distinct outcomes based on $\tau_{B,1}$'s splitting behavior.

2. $\tau_{A,1}$ Non-Splitting Scenario:

   - With derived information, we provide insights into the system dynamics when $\tau_{A,2}$ executes before $\tau_{B,1}$, and vice versa, while $\tau_{A,1}$ remains unsplit.

3. Designing parameters for τ_A

   - Formulating period
   - Formulating initial temperature

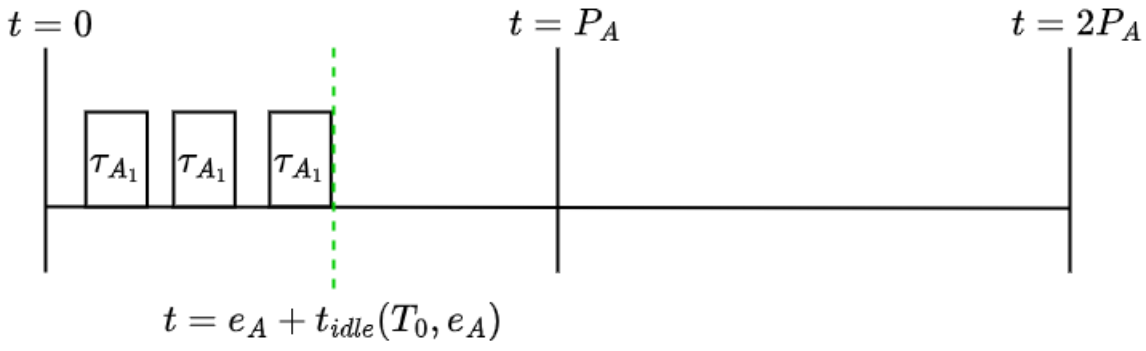


Figure 4.2 Generic scenario when $\tau_{A,1}$ splits

Let's delve into each of these scenarios individually in upcoming subsections.

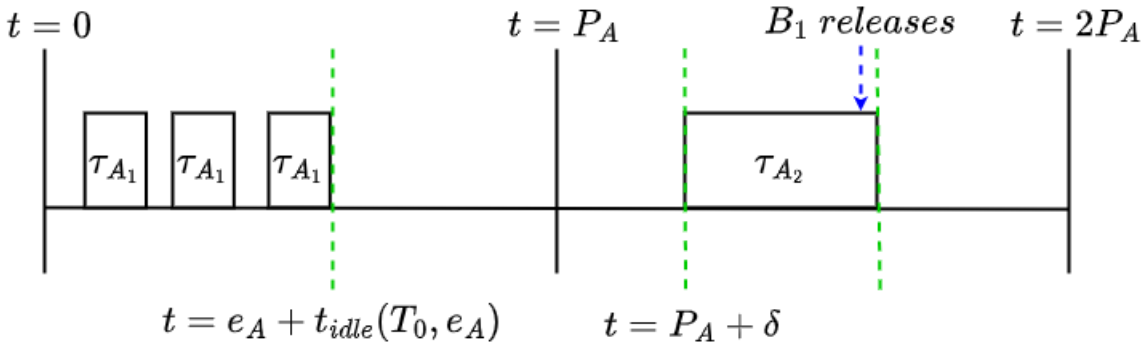


Figure 4.3 Scenario for $\tau_{B,1}$ executes after start time of $\tau_{A,2}$ when $\tau_{A,2}$ didn't split

4.2.1 $\tau_{A,1}$ Splitting Scenarios -

The idle time insertion for $\tau_{A,1}$ is illustrated in Fig.4.2. In this case, $e_A^1 + T_{idle}(T_0, e_A) \leq P_A$ must follow, as $\tau_{A,1}$ has to meet its deadline for maintaining the schedulability. Now there are 2 possibilities either $\tau_{A,2}$ executes first or $\tau_{B,1}$.

4.2.1.1 Scenario 1 ($\tau_{A,2}$ executes before $\tau_{B,1}$)

In this scenario, let's assume the processor gets some time $t = \delta$ before $\tau_{A,2}$ executes. Even in such a situation $\tau_{A,2}$ may reach peak temperature during execution and has task splitting.

4.2.1.1.1 $\tau_{A,2}$ does not split

Figure 4.3 illustrates how inserting idle time can influence processor temperature. As you can see, after executing task $\tau_{A,1}$, the processor has some idle time equal to $t = P_A + \delta - (e_A^1 + T_{idle}(T_0, e_A))$. This allows the processor to cool down before starting task A_2, meaning the temperature at the beginning of $\tau_{A,2}$ is not necessarily T_{max} (the maximum safe temperature).

It's important to remember that while the overall theory is independent of the task schedule before the job begins, it does depend on the initial temperature at the job start. Here, we'll explore two scenarios:

$\tau_{B,1}$ **splits:** In this case, take $T(t = P_A + \delta + e_A^{2,1-f}) = T_{max} - \alpha$, such that $\alpha > 0$. Here, f is the fraction of the job that's remaining to be executed, so $1 - f$ is the fraction that has been executed. $\tau_{B,1}$ execution happens at time instant defined as ϵ before the finish time of $\tau_{A,2}$ where $\epsilon \rightarrow 0$, this also means that $f \rightarrow 0$.

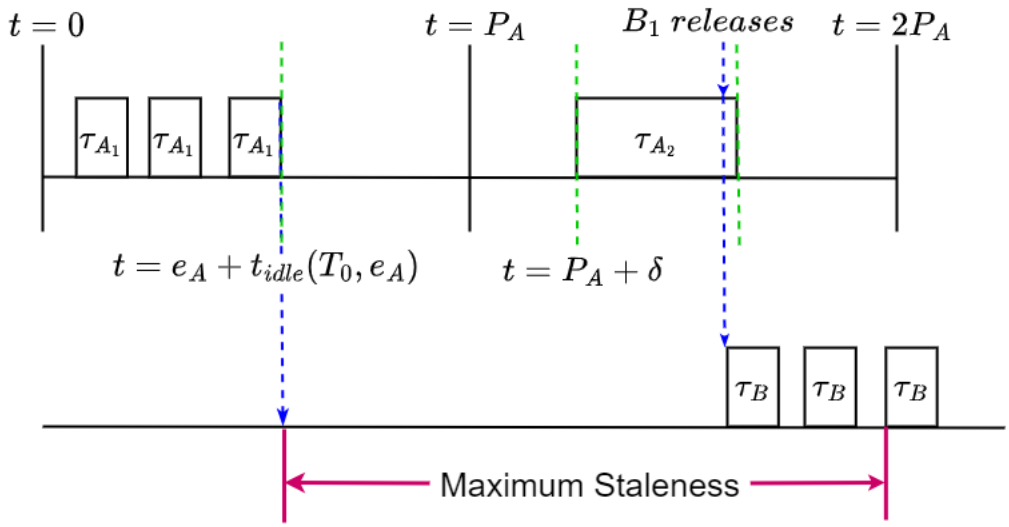


Figure 4.4 Maximum data age scenario when $\tau_{A,2}$ never splits

Theorem 4.2.1. *When temperature $T(t = P_A + \delta + e_A^{2,f}) < T_{max}$, the maximum data age for $\tau_{B,2}$ can be given as-*

$$DA_{max} = 2P_A - (e_A^1 + T_{idle}(T_0, e_A)) - (e_B^{1,f}/m) \tag{4.2}$$

Proof. The execution of $\tau_{B,1}$ may rise to T_{max} in between. As illustrated in Fig.4.4, if $\tau_{B,1}$ splits, then the data generated by $\tau_{A,1}$ should be consumable till the last part of $\tau_{B,1}$ as no task can generate its local copy, thereby concluding that every part of the task $\tau_{B,1}$ should be able to consume the data generated by $\tau_{A,1}$. There is no need to include the last part's execution in the data age calculation, as no further data is being consumed by $\tau_{B,1}$.

Assuming, from the splitting policy, m is the optimal number of parts of $\tau_{B,1}$ is split into after it first tends to reach T_{max}-

$$\begin{aligned} DA =& P_A + \delta - (e_A^1 + T_{idle}(T_0, e_A)) + (e_A^2 - \epsilon) - (e_B^{1,f}/m) \\ &+ (e_B^1 + T_{idle}(T_{max} - \alpha, e_B^{1,1-f})) \end{aligned} \tag{4.3}$$

where f fraction of $\tau_{B,1}$ is that is left and $e_B^{1,f}$ is the execution time left for $\tau_{B,1}$ to execute, reaching T_{max} in between.
Maintaining the strict thermal constraint and obtaining the maximum data age, also requires the schedulability to be met by the task set. So, imposing the condition, we get-

$$P_A + \delta + e_A^2 + (e_B^1 + T_{idle}(T_{max} - \alpha, e_B^{1,f})) \leq 2P_A \tag{4.4}$$

To find DA_{max} we can state that the Eq.4.4 can be used with the equality operation otherwise it won't be the worst case.

Using the Eq.4.3 and Eq.4.4 i.e. substituting, the variable δ, we obtain

$$DA_{max} = 2P_A - (e_A^1 + T_{idle}(T_0, e_A)) - (e_B^{1,f}/m) \tag{4.5}$$

assuming $\epsilon \rightarrow 0$ and m chosen optimally for obtaining the minimum total idle time. □

$\tau_{B,1}$ **does not split:** We now see the impact on data age when $\tau_{A,2}$ does not cause $\tau_{B,1}$ to reach T_{max}

Lemma 4.2.2. *DA_{max} remains the same even when $\tau_{B,1}$ doesn't split, when $\tau_{B,1}$ executes after the start time of $\tau_{A,2}$.*

Proof. Unlike depicted in Fig. 4.4, the maximum data age in the case where the $\tau_{B,1}$ doesn't split, will be considered only till the start of $\tau_{B,1}$. This would result in a difference of $e_B - e_B/m$ (Assuming m is the number of parts of $\tau_{B,1}$ derived from the splitting policy) in the data age, thereby reducing the data age, hence making no difference in DA_{max}. □

4.2.1.1.2 $\tau_{A,2}$ splits

In Fig.4.5 we can see the case where $\tau_{A,2}$ reaches peak temperature while execution and $\tau_{B,1}$ executes after $\tau_{A,2}$.

Theorem 4.2.3. *When temperature at $t = P_A + \delta + e_A^2$ is T_{max}, maximum data age for $\tau_{B,2}$ can be given as-*

$$DA_{max} = 2P_A - (e_A^1 + T_{idle}(T_0, e_A)) - (e_B/m) \tag{4.6}$$

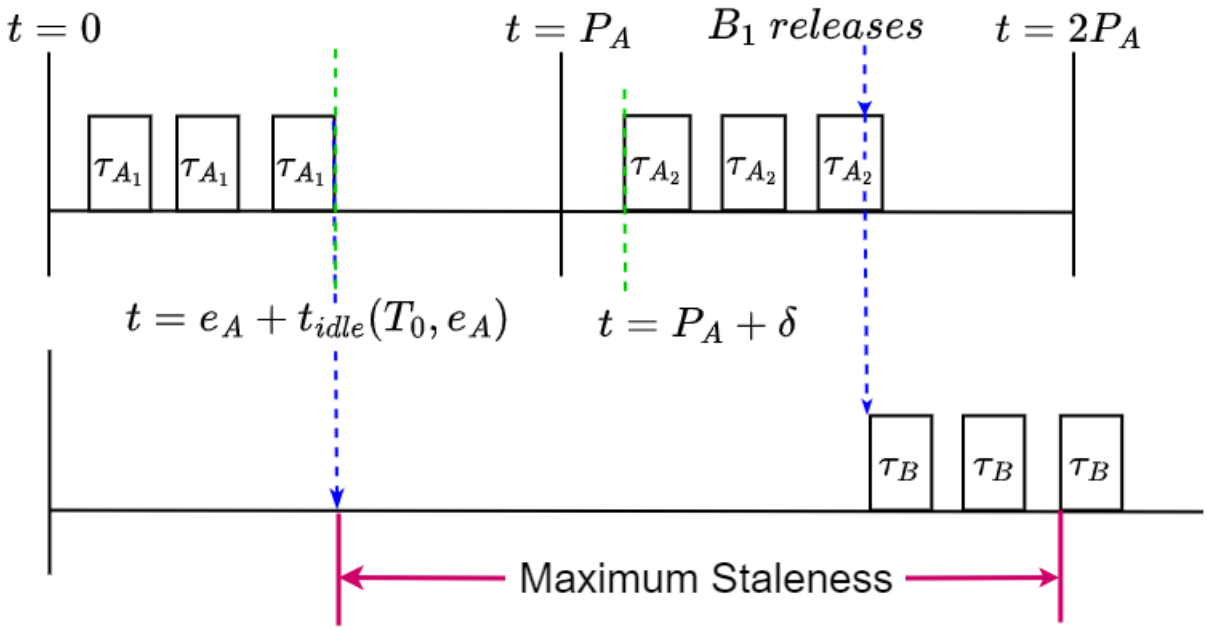


Figure 4.5 Maximum data age scenario for $\tau_{B,1}$ to consume data when $\tau_{A,2}$ splits.

Proof. Consider, the temperature at $t = P_A + \delta$ as T_{start}. Also, the temperature at the beginning of τ_B's execution is considered $\approx T_{max}$ as $\epsilon \rightarrow 0$.

$$DA = (P_A - e_A^1 - T_{\text{idle}}(T_0, e_A)) + \delta + (e_A^2 + T_{\text{idle}}(T_{\text{start}}, e_A^{2,f}) - \epsilon) + (e_B^1 + T_{\text{idle}}(T_{\text{max}}, e_B)) - (e_B^1/m) \tag{4.7}$$

where $e_A^{2,f}$ is the execution time left for $\tau_{A,2}$ after reaching T_{max} in between. Here, the schedulability is defined as-

$$P_A + \delta + e_A^2 + T_{\text{idle}}(T_{\text{start}}, e_A^{2,f}) + e_B^1 + T_{\text{idle}}(T_{\text{max}}, e_B) \leq 2P_A \tag{4.8}$$

To find DA_{max} we can state that the Eq. 4.8 can be used with the equality operation making it the worst case. Using the Eq. 4.7 and Eq. 4.8, i.e. substituting δ, we get the final equation as-

$$DA_{max} = 2P_A - (e_A^1 + T_{idle}(T_0, e_A)) - (e_B/m) \tag{4.9}$$

assuming $\epsilon \rightarrow 0$ and m is chosen optimally for obtaining the minimum total idle time. □

Notice that in Eq. 4.5 and Eq. 4.9, m are two different values. By the theory of approximation, if we assume α to be very small quantity then we can say that $e_B^{1,f}/m_2 \approx e_B/m_1$. This approximation helps us maintain a stricter thermal constraint.

4.2.1.2 Scenario 2 ($\tau_{B,1}$ executes before $\tau_{A,2}$)

Consider Fig. 4.2 as the starting point for this scenario too. Assume that δ time units post $e_A^1 + T_{idle}(T_0, e_A)$, B_1 starts its execution, such that $\delta \geq 0$. Here also, we can see two sub-cases - $\tau_{B,1}$ executes and never reaches T_{max} and another one where it reaches T_{max} in between.

4.2.1.2.1 $\tau_{B,1}$ splits

In Fig.4.6 we can see the case where $\tau_{B,1}$ reaches peak temperature during execution. As $\tau_{A,2}$ executes later, it does not have any effect on the data age of $\tau_{B,1}$

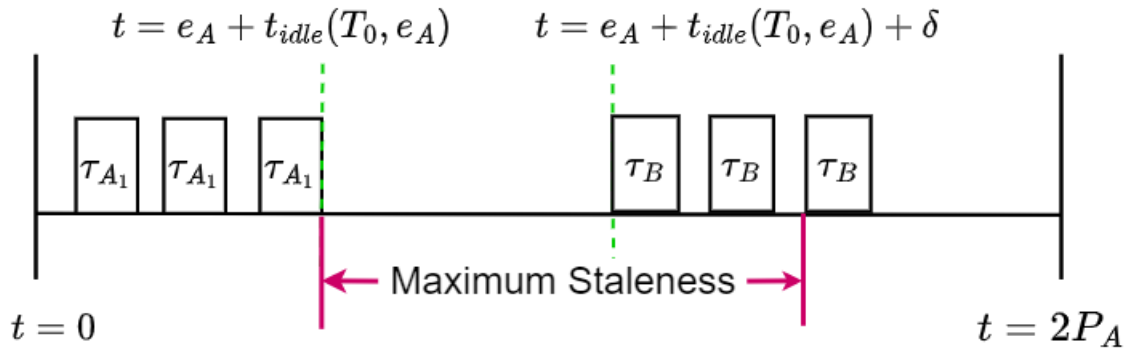


Figure 4.6 Maximum data age scenario when $\tau_{B,1}$ splits

Theorem 4.2.4. *The maximum data age when $\tau_{B,1}$ reaches $T(t = e_A^1 + T_{idle}(T_0, e_A) + \delta + e_B^{1,f}) = T_{max}$ in between its execution can be calculated as-*

$$DA_{max} = 2P_A - (e_A + T_{idle}(T_\Delta, e_A) + \Delta) - (e_B^{1,f}/m) - (e_A + T_{idle}(T_0, e_A)) \tag{4.10}$$

where, Δ represents the time difference between the finish of $\tau_{B,1}$ and the start of $\tau_{A,2}$, and the temperature at the start of $\tau_{A,2}$ is T_Δ

Proof. Here, take a general case where the temperature at the start time of $\tau_{B,1}$ is T_{start} and during execution temperature $T(t)$ reaches T_{max}. As shown in Fig. 4.6,

$$DA = \delta + e_B^1 + T_{idle}(T_{start}, e_B^{1,f}) - (e_B^{1,f}/m) \tag{4.11}$$

where $e_B^{1,f}$ is the execution time of $\tau_{B,1}$ left after it reaches T_{max} in between and $e_B^{1,f}/m$ is the execution time of last fraction of $\tau_{B,1}$. Here, the schedulability condition is stated as follows-

$$e_A^1 + T_{idle}(T_0, e_A) + \delta + e_B^1 + T_{idle}(T_{start}, e_B^{1,f}) + \Delta + e_A^2 + T_{idle}(T_\Delta, e_A) \leq 2P_A \tag{4.12}$$

For data age to be maximum, δ needs to be maximized and it is possible when the Eq.4.12 follows the equality as when it becomes equal, δ reaches its peak. Using Eq.4.11 and Eq.4.12, substituting δ, we get-

$$DA_{max} = 2P_A - (e_A + T_{idle}(T_0, e_A)) - \delta - (e_A + T_{idle}(T_\delta, e_A)) - (e_B^{1,f}/m) \tag{4.13}$$

□

4.2.1.2.2 $\tau_{B,1}$ does not split

In this case, temperature at t, $T(t = e_A^1 + T_{idle}(T_{max}, e_A) + \delta + e_B) < T_{max}$ As shown in Fig. 4.6 and Fig. 4.7, we can say that the data age, in this case, will be less, implying that the maximum data age for the system remains unaffected.

4.2.2 Designing parameters for τ_A

4.2.2.1 Formulating the period (P_A)

Let $DA_{A\to B}$ denote the upper bound on data age for data produced by task τ_A and consumed by task τ_B.

Theorem 4.2.5. *If the first instance of τ_A splits, the period of task τ_A would be-*

$$P_A \leq \frac{1}{2}[DA_{A\to B} + e_A + T_{idle}(T_0, e_A)) + (e_B/m)] \tag{4.14}$$

Proof. From Eq.4.5, Eq.4.9 and Eq.4.13, we can deduce that the maximum data age is,

$$DA_{max} = 2P_A - (e_A^1 + T_{idle}(T_0, e_A)) - (e_B/m) \tag{4.15}$$

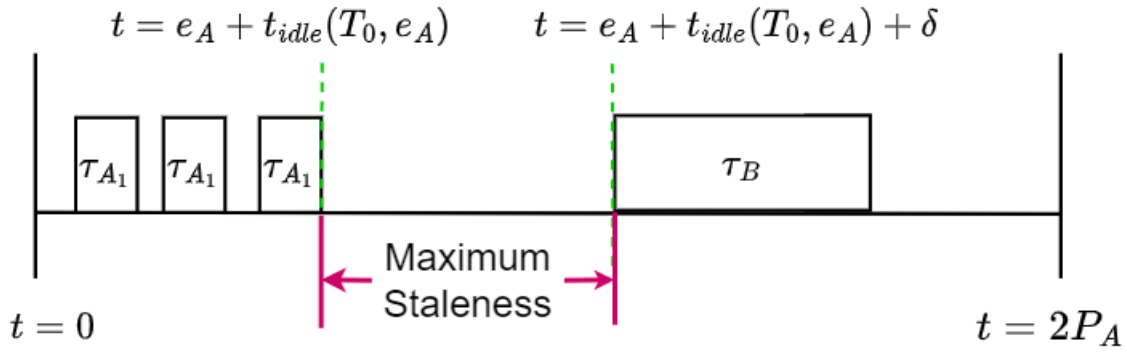


Figure 4.7 Maximum data age Scenario when the first instance of B never splits

Therefore,

$$DA_{A\to B} \geq 2P_A - (e_A + T_{idle}(T_0, e_A)) - (e_B/m) \tag{4.16}$$

Therefore, the period of task τ_A-

$$P_A \leq \frac{1}{2}[DA_{A\to B} + e_A + T_{idle}(T_0, e_A)) + (e_B/m)] \tag{4.17}$$

□

Our result aims to obtain the minimum data age under strict thermal bounds, satisfying the objective of minimizing the maximum CPU utilization.

$$U(t) = \frac{e_A^{max}}{P_A} \tag{4.18}$$

Notice that we assumed the minimum execution times of the task set for maximum data age in the formulation.

To minimize utilization, we need to ensure the period is at its maximum. Using period obtained from Eq.4.17, we get utilization in a two-task scenario as-

$$U(t) = \frac{e_A^{max}}{\frac{1}{2}[DA_{A\to B} + e_A^{min} + T_{idle}(T_0, e_A^{min})) + (e_B^{min}/m)]} \tag{4.19}$$

Here we have considered e_A^{max} because our solution will ensure data validity even with best-case execution of the first task, while minimizing the maximum utilization the system could experience.

4.2.2.2 Formulating the initial temperature

We formulate the initial temperature of the first instance of τ_A such that it doen't split. We will follow the same exact procedure as done in the case when $\tau_{A,1}$ splits, but before that let's look at the result we have obtained in Eq.4.15.

Lemma 4.2.6. *Data age is maximum when $\tau_{B,1}$ executes after the start time of $\tau_{A,2}$.*

Lemma 4.2.7. *Under thermal constraints, maximum data age depends on two factors, the initial temperature $T(0)$ and the start temperature of $\tau_{B,1}$.*

From the above two lemmas, we can say that there are only two cases we need to consider-

4.2.2.2.1 B_1 splits From Eq.4.15, we can say that if $\tau_{A,1}$ does not split, then $T_{idle}(T_0, e_A)) = 0$. The data age becomes,

$$DA_{max} = 2P_A - e_A^1 - (e_B/m) \tag{4.20}$$

4.2.2.2.2 $\tau_{B,1}$ does not split From Eq.4.20, we can say that if $\tau_{B,1}$ does not split, then the data age is measured only till the start of $\tau_{B,1}$, implying that e_B/m need not be considered as τ_B executes without any split. The data age becomes-

$$DA_{max} = 2P_A - e_A^1 \tag{4.21}$$

Amongst these two cases, the maximum data age is given by Eq.4.21, therefore the utilization becomes:

$$U(t) = \frac{e_A^{max}}{\frac{1}{2}[DA_{A\to B} + e_A^{min}]} \tag{4.22}$$

4.3 N Task Result

In this section, we extend our analysis from the two-task scenario to consider a general case with an N-task chain. Here, the task chain is denoted as $\tau_{C,1}$ to $\tau_{C,n}$, where n represents the

chain length and each $\tau_{C,i}$ signifies a task within the chain. Each task acts as a producer for the subsequent task, $\tau_{C,i+1}$, in the chain and a consumer for the preceding task, $\tau_{C,i-1}$.

We aim to leverage the insights from the two-task formulation (Eq. 4.15) for the N-task scenario. The key takeaways from Eq. 4.15 include:

- Maximum Data Age: The data age reaches its maximum value when task $\tau_{A,2}$ starts executing before the beginning of $\tau_{B,1}$ execution.
- Intermediate Task Execution: Unlike the two-task case where the data age bound didn't consider any intermediate task execution between the producer and consumer tasks, the N-task scenario allows for such occurrences. However, upon closer inspection, the derived data age bound remains independent of any activities happening within these intermediate tasks.

Explanation: The data age bound is independent of the idle time associated with $\tau_{A,2}$. This implies that the initial temperature at the start of $\tau_{A,2}$ is not a factor in the equation. Also, even if any job execution in between affects the temperature, the formulation only considers the temperature at the beginning of the job execution. Hence, we can conclude that the data age bound remains unchanged regardless of any fraction of tasks running between the producer and consumer tasks within the chain.

Theorem 4.3.1. *In N-task, where $\tau_{C,i}$ denotes the task in a task chain where $1 \leq i \leq n$, the end to end data age bound is given as-*

$$DA_{C_1 \to C_{n+1}} = \sum_{i=1}^{i=n} DA_{C_i \to C_{i+1}} + \sum_{i=2}^{i=n} \gamma_i * (e_{C_i}/m_i) \tag{4.23}$$

where $\gamma_i \ \epsilon \ \{0, 1\}$

Proof. Proving using the Principle of Mathematical Induction,

Base Case :

As shown in the Fig. 4.8, we can say that if C_1, C_2 and C_3 all split, then

$$DA_{C_1 \to C_3} = DA_{C_1 \to C_2} + e_{C_2}/m + DA_{C_2 \to C_3} \tag{4.24}$$

In case $\tau_{C,1}$ or $\tau_{C,3}$ does not split, it does not make any difference to the Eq.4.24 but if $\tau_{C,2}$ does not split then the equation transforms as,

$$DA_{C_1 \to C_3} = DA_{C_1 \to C_2} + DA_{C_2 \to C_3} \tag{4.25}$$

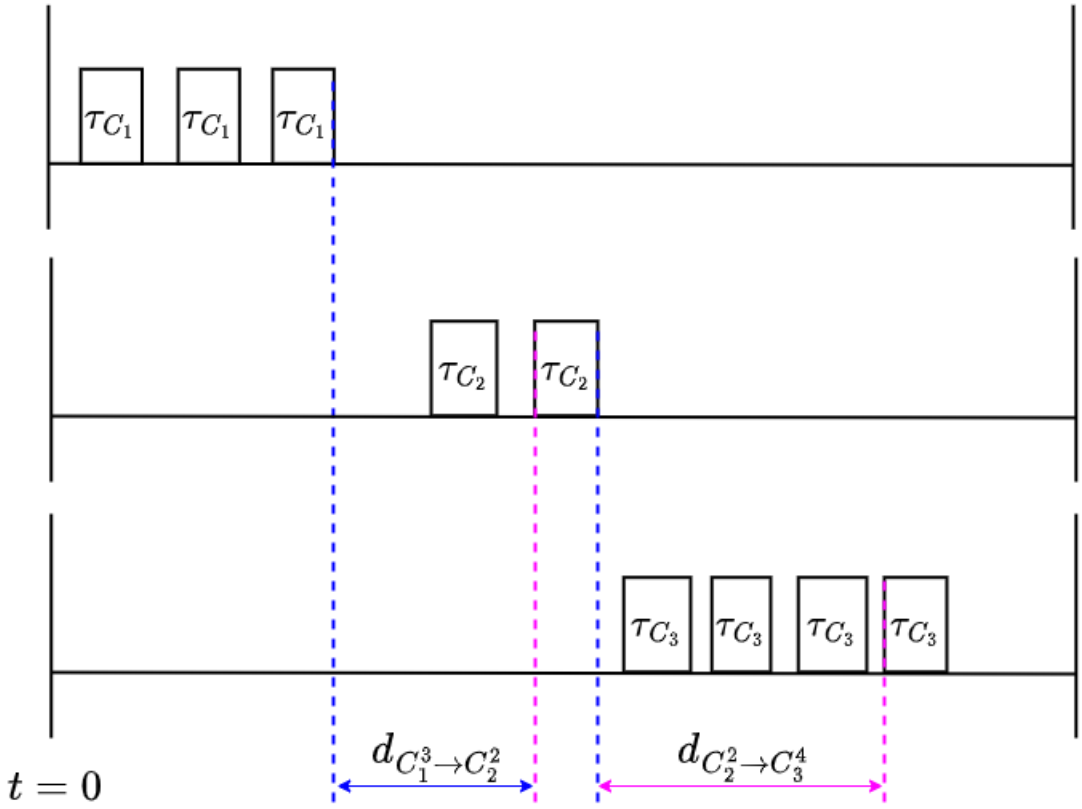


Figure 4.8 Data age scenario For Three-Task set

$\tau_{C,1}$ and $\tau_{C,3}$ split influences the data age value.

So we can conclude that if $\gamma \, \epsilon \, \{0, 1\}$ -

$$DA_{C_1 \to C_3} = DA_{C_1 \to C_2} + \gamma * (e_{C_2}/m) + DA_{C_2 \to C_3} \tag{4.26}$$

Induction Step :

Assuming that the result is true for the n task result, we will try to prove that it exists for the $(n+1)th$ task result.

As shown in the Fig.4.9, for n task set, we get-

$$DA_{C_1 \to C_n} = \sum_{i=1}^{i=n-1} d_{C_i \to C_{i+1}} + \sum_{i=2}^{i=n-1} \gamma_i * (e_{C_i}/m_i) \tag{4.27}$$

. where $\gamma_i \, \epsilon \, \{0, 1\}$

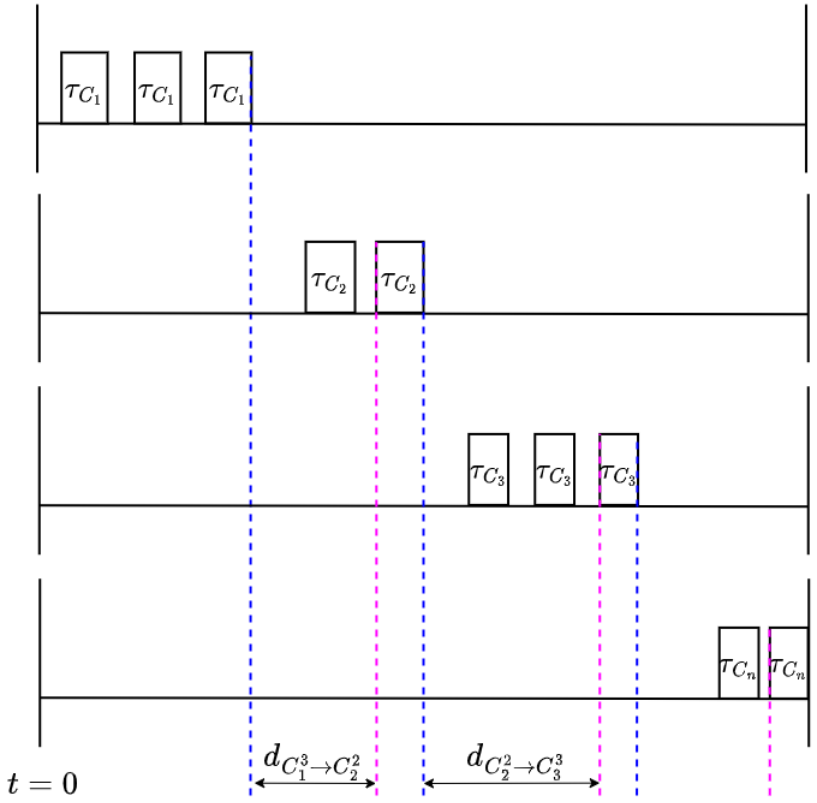


Figure 4.9 Data age scenario For N-Task set

So, we have to obtain the result for $(n+1)th$ task which should be -

$$DA_{C_1 \to C_{n+1}} = \sum_{i=1}^{i=n} DA_{C_i \to C_{i+1}} + \sum_{i=2}^{i=n} \gamma_i * (e_{C_i}/m_i) \tag{4.28}$$

where, $\gamma_i \ \epsilon \ \{0, 1\}$

One more addition of a task leads to the equation-

$$DA_{C_1 \to C_n} = \sum_{i=1}^{i=n-1} DA_{C_i \to C_{i+1}} + DA_{C_n \to C_{n+1}} + \sum_{i=2}^{i=n-1} \gamma_i * (e_{C_i}/m_i) + \gamma_n * (e_{C_n}/m_n) \tag{4.29}$$

where, γ_n represents the possibility of the split of n^{th} task. If we look carefully at Eq.4.29, then we can see that it is-

$$DA_{C_1 \to C_{n+1}} = \sum_{i=1}^{i=n} DA_{C_i \to C_{i+1}} + \sum_{i=2}^{i=n} \gamma_i * (e_{C_i}/m_i) \tag{4.30}$$

. where, $\gamma_i \epsilon \{0, 1\}$

Hence proved □

In the above derivation, the consecutive two-task stitching mechanism is considered. The above case can be extended to m task stitching depending on which tuple represents the minimum data age among all.

So, we get our optimization formulation as,

Given : Let $E = \{1, 2, 3, \ldots, n\}, e_i^{max}, e_i^{min}$ for each $i \in E$

Find : $DA_{C_i \to C_j}$ for each $(i, j) \in E$

Objective : Minimize $U = \sum_{i=1}^{i=n-1} \frac{e_{C_i}^{max}}{P_{C_i}}$ which is,

$$\sum_{(i,*) \in E} \frac{2 \cdot e_{C_i}^{max}}{\min(DA_{C_i \to C_*}) + e_{C_i}^{min} + \gamma_i \cdot (T_{idle}(T_{0,i}, e_{C_i}^{min})) + \beta_i \cdot (e_{C_*}^{min}/m_{C_*})}$$

Here, γ_i and $\beta_i \in \{0, 1\}$ and $T_{0,i}$ denotes the temperature at the start of the execution of task $\tau_{C,i}$. γ_i and β_i are 0 or 1 depending on whether in between tasks split or not and the initial temperature of every part splits their first task or not respectively.

Subject To : $\forall k, \sum_{i:\{i,j\} \in E_k} DA_{C_i \to C_j} + \sum_{i:\{i,*\} \in E_k} (e_{C_i}/m_i) \leq DA_{C_1 \to C_n}$

and, $i : \{i, j\} \in E, DA_{C_i \to C_j} \geq 0$

In this formulation, stitching of the tasks has been done such that it yields the minimum data age.

4.4 Evaluation

4.4.1 N-Task Theory Evaluation

We implemented and evaluated our proposed theory using task set values obtained from the MiBench automotive benchmark suite [24]. The evaluation platform consisted of an i.MX6 processor with an ARM A9 core. We focused on tasks within the automotive section of the benchmark. Our evaluation aimed to achieve three key objectives:

1. Verification: Verify if the proposed theory guarantees adherence to both thermal and real-time scheduling constraints while maintaining acceptable data age for the tasks.

2. Impact Analysis: Analyze how schedulability and taskset utilization change as we vary the desired data age and temperature ranges.

3. Scalability Analysis: Examine how schedulability changes with varying task chain lengths under different temperature constraints.

Experimental Setup: The task graphs extracted from the benchmark suite contained between 2 and 12 tasks each. We collected a total of 16 two-task scenarios and 20 three-task scenarios from the benchmark data. To simulate the EDF scheduling algorithm, we employed a Linux kernel-based simulation environment.

For enriching the dataset for special cases, our evaluation captured as many relevant special cases as possible, we combined several originally serial tasks within the benchmark into larger composite tasks. However, we maintained the original temperature ranges associated with these tasks in the real-world use cases to ensure the resulting data remained meaningful. And, for task execution time parameters, we utilized WCETs from the first hardware configuration provided in the benchmark for a subset of the given period value range. BCETs were set to half the corresponding WCET values. Additionally, a context switch time equivalent to 1% of the WCET was incorporated into the execution time for evaluation purposes. **Experiments:** Optimization code used for derivation of periods of the tasks in a chain and the EDF Simulation along with the thermal model used to schedule the task chain with given freshness bound and temperature constraints are shown in Fig.4.10.

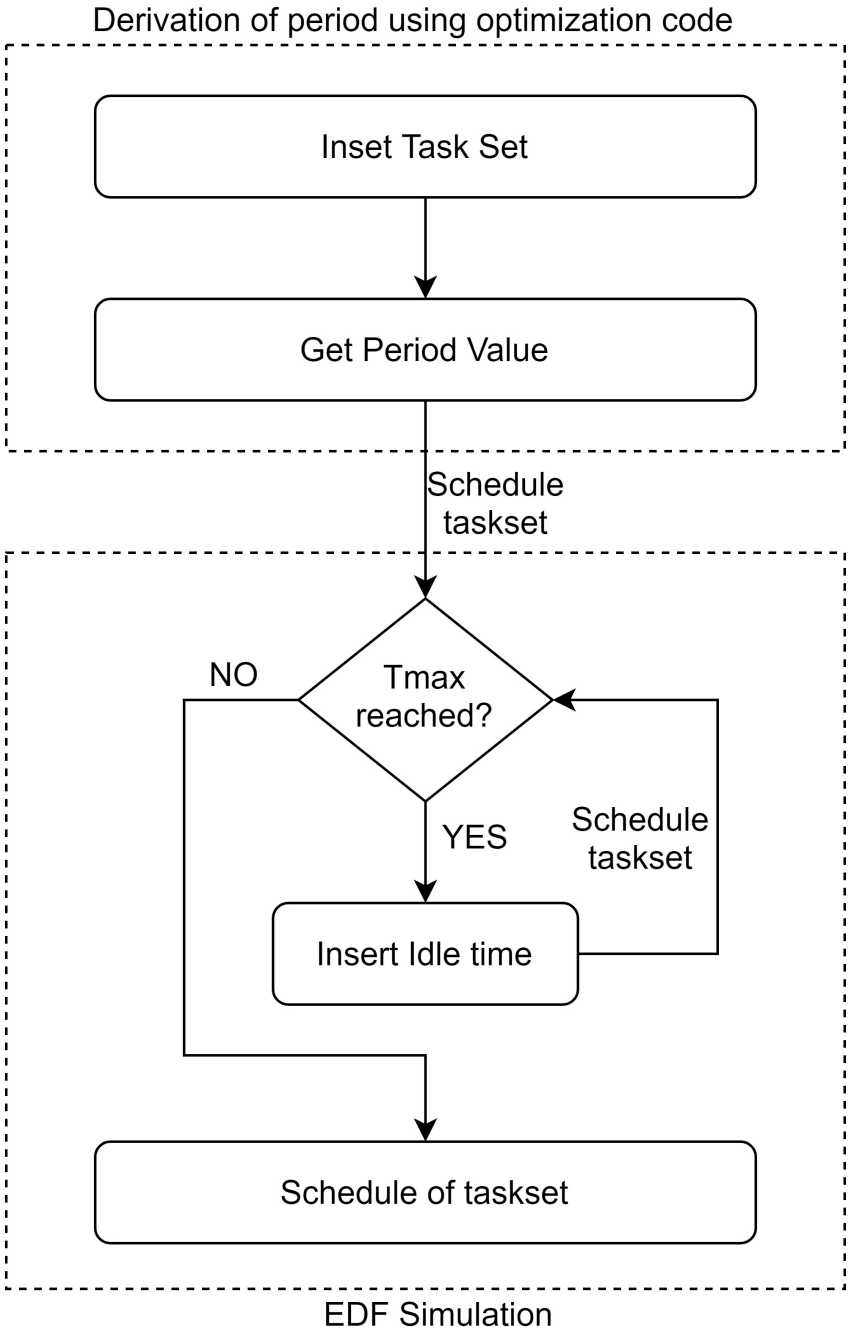


Figure 4.10 Evaluation Flow Diagram

4.4.1.0.1 Schedulability variation with Utilization factor

For the first experiment, we ran task chains of different lengths for various range of data age bounds. We set up the tasks to run for 99% of the specified WCET to account for scheduler overhead and context switching. The final values we got are the schedulability percentage for uniformly spread utilization values over a set of temperature ranges defined for the processor.

Our simulation algorithm determined the parameters of the thermal model, including T_{idle} and m, for all task sets across the considered temperature ranges. This computation followed a comprehensive scheduling and data age experiment across all task sets. Notably, all tasks were executed on the same core, without any external loads.

To explore the impact of varying data age bounds relative to the end-to-end data age bound of the task set, we depicted the results in Fig. 4.15. Notably, the graph demonstrates how schedulability varies across different temperature ranges and various data age bounds, including 12%, 30%, 50%, and 80%.

Examining Fig. 4.15(a), where we fixed the data age bound at 80%, we observed a significant decrease in average schedulability as the thermal constraints became more stringent. This observation holds true for a constant utilization scenario. Conversely, considering a fixed schedulability percentage, we noted a substantial reduction in processor utilization as thermal constraints tightened. This implies that longer task chains are less likely to be scheduled under strict thermal limits.

Shifting our focus to the variation in data age bound from 80% (Fig. 4.15(a)) to 12% (Fig. 4.15(d)), we observed a transition from a concave downward curve to a convex upward curve across different temperature differences. This shift is attributed to the decrease in task periods as the data age bound diminishes, thereby making schedulability more challenging due to tighter deadlines.

Moreover, across different data age bounds, a significant decline in schedulability is evident when both strict thermal constraints ($T_{max} - T(0) = 0$) and stringent data age bounds (12%) are imposed. Relaxing either constraint leads to a decrease in schedulability, albeit less severe than when both constraints are stringent. Ultimately, when both constraints are relaxed, schedulability reaches its optimal level.

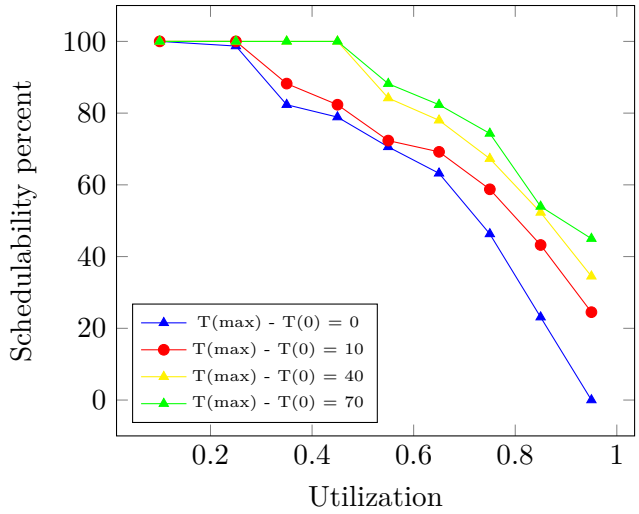


Figure 4.11 Data age bound = 80%

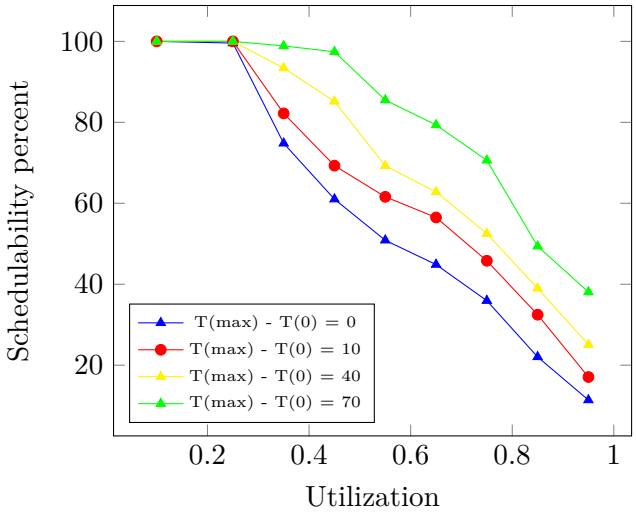


Figure 4.12 Data age bound = 55%

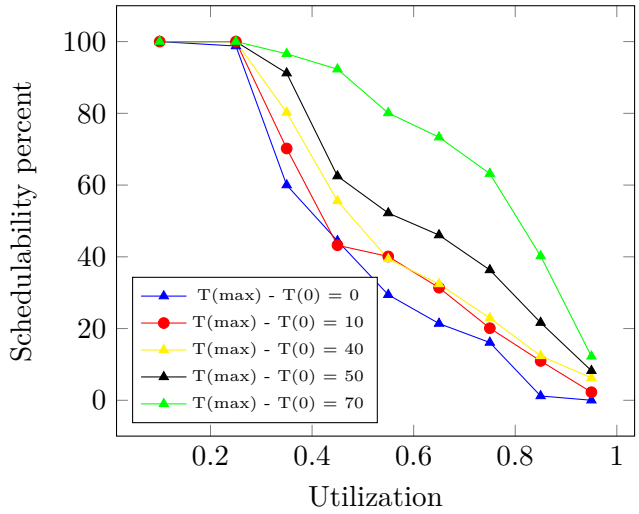


Figure 4.13 Data age bound = 30%

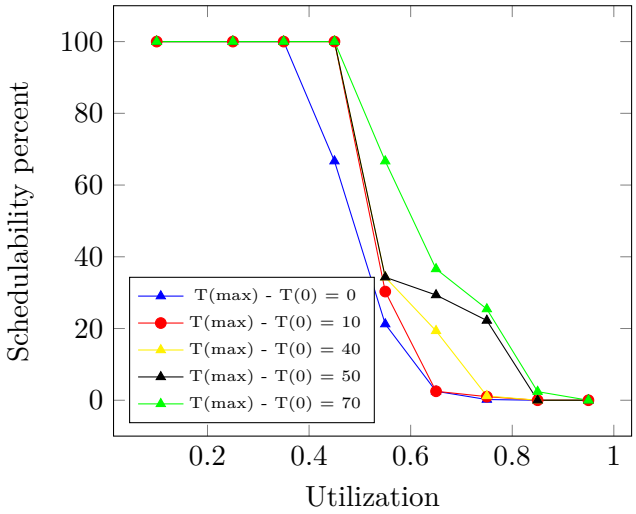


Figure 4.14 Data age bound = 12%

Figure 4.15 Schedulability Percentage and Utilization at various temperature ranges for different data age bounds. (a)80%, (b)50%, (c)30%, and (d)12%

4.4.1.0.2 Schedulability variation with data age bound In our second experiment, we delved into the intricacies of each task set, determining the idle time and the 'm' value across different temperature ranges. Leveraging these parameters, we calculated the maximum data age, DA_{max}, for each task set. Subsequently, we identified the minimum data age value, denoted as DA_{min}, required for the task set to remain schedulable. In essence, this value represents the threshold beyond which further reduction in data age would render the task set unschedulable. Armed with this range of data age values (from DA_{min} to DA_{max}), we scheduled all possible periods across various thermal constraints, spanning temperature differences from 0 to 80.

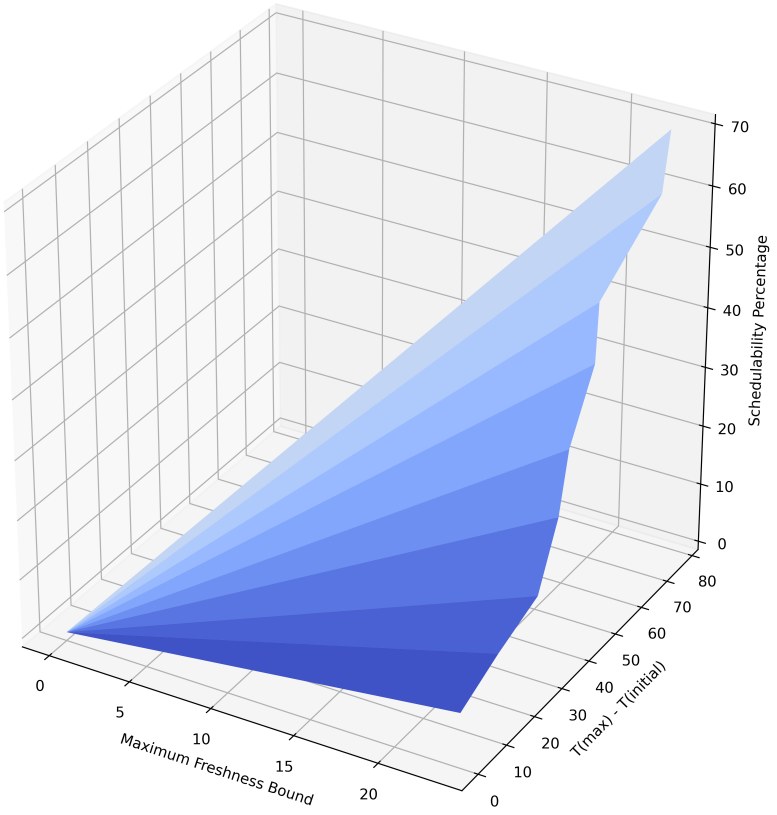


Figure 4.16 Schedulablilty with different data age bounds under various temperature constraints

With the implementation of these thermal constraints, we assessed the percentage of task chains that remained schedulable and plotted these values in Fig. 4.16. Analyzing the graph, we observe several key trends. Firstly, for a constant data age, as the temperature difference increases, the schedulability of tasks also rises. For instance, within a data age range of 21.79s

to 23.2s, when the temperature difference varies from 0 to 80, the average schedulability is 46.75%, with a minimum of 12.32% and a maximum of 76.8%. Secondly, for a fixed temperature difference, the schedulability increases in a concave upward manner as the data age bound expands. Lastly, when maintaining a constant schedulability percentage, an increase in the thermal bound leads to a corresponding increase in the maximum data age, aligning with the theoretical formulation as described in Eq. 4.15.

4.4.1.0.3 Schedulability with varying chain length In our experiment, we standardized the data age for every task chain to be 30% of the data age upper bound. Subsequently, we utilized the idle time and 'm' values corresponding to each task across various temperature ranges as input parameters. It's noteworthy that we maintained the BCET of every task exactly half of their WCET.

The experiment's output consisted of the task periods for each task chain, which we executed for chain lengths ranging from 2 to 12 under various temperature constraints. The resulting schedulability percentages, both with and without thermal constraints, were plotted in Fig. 4.17, with detailed values provided in Table 4.1. Examining the graph, we discern several notable

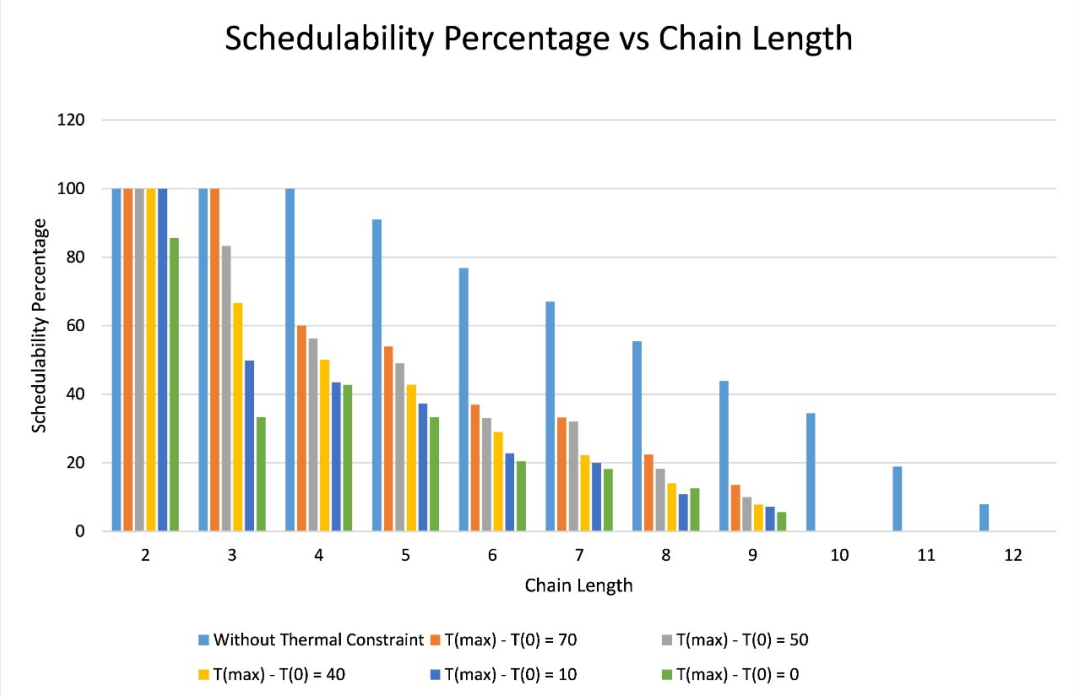


Figure 4.17 Schedulablilty with Different Chain Lengths

trends. Firstly, for a given chain length, schedulability decreases as the temperature difference decreases, owing to the rapid increase in required idle time. Notably, task sets with chain lengths exceeding 9 exhibit unschedulable behavior, as depicted in Fig. 4.17. Furthermore, achieving a specific schedulability percentage necessitates a higher temperature difference for longer chain lengths, while even a stringent thermal bound can suffice for shorter chain lengths.

The sharp decline in average schedulability percentages with thermal constraints, compared to those without, underscores the challenge faced by schedulers in accommodating strict data freshness bounds and thermal constraints, especially for longer task chains. Additionally, Table 4.1 reveals that the average drop in schedulability percentage increases with decreasing temperature differences across all chain lengths considered.

Temperature bound	70	50	40	10	0	NIL
Average schedulability	52.52	47.73	41.56	36.44	31.49	79.27
Percent drop in schedulability	33.75	39.79	47.58	54.04	60.18	-

Table 4.1 Average schedulability at different temperature bounds

Percent drop in schedulability is a metric which defines the drop in the schedulability under temperature constraints w.r.t to without any temperature constraint for all the chain lengths. Average in this case, has been considered for all chain lengths unlike in Fig 13, where we consider average for all the cases of a particular length.

It's important to acknowledge that the optimistic schedulability values obtained without thermal constraints contrast with real-world scenarios where thermal bounds must be considered for processor safety and durability. Nonetheless, Fig. 4.17 highlights the benefits of incorporating thermal constraints, particularly for task sets with longer chain lengths. Remarkably, the schedulability percentage for chain lengths exceeding 3 remains relatively stable across different temperature differences, suggesting resilience to unexpected temperature rises in the processor. However, it's worth noting that the observed variations for chain length 3 underscore the significance of considering task set characteristics in real-life scenarios, where task sets often exhibit considerable lengths.

Chapter 5

Conclusion

In concluding this comprehensive exploration into the world of real-time systems, particularly focusing on data age management under thermal and fault tolerance constraints, this thesis has highlighted several critical areas that impact system performance and reliability. By rigorously investigating idle time insertion as solution to thermal thresholds, checkpointing as fault tolerance mechanisms, and the integration of scheduling techniques, this thesis has paved the way for a deeper understanding of how data validity and timeliness can be maintained in safety-critical applications.

The critical factors influencing data age in real-time systems have been identified and analyzed, leading to the development of a novel data-age analysis framework. This framework is specifically tailored to address the unique challenges posed by thermal constraints and fault recovery mechanisms, offering a more holistic and adaptable approach to managing data age.

One of the key takeaways from this study is the profound impact of thermal management and fault tolerance techniques on data age. The research has shown that even task splitting and effective checkpointing that significantly mitigate the risks associated with high temperatures and system faults, thereby enhancing overall system stability and data reliability, can significantly degrade data age parameter of the task set.

Moreover, the research emphasizes the significance of a tailored approach to maximum data age bound, suggesting that a one-size-fits-all solution is often not feasible due to the diverse nature of real-time systems and their operational environments. The developed framework tries to addresses current challenges while providing a scalable foundation for future research and development in this field.

As a concluding remark, the journey through this thesis reaffirms the essential role of continuous innovation and adaptation in the field of real-time systems. The insights gained and the methodologies developed here contribute substantially to the ongoing discourse in data age management, offering actionable strategies and a comprehensive analytical tool set for enhancing system performance and reliability.

5.1 Summarising the work done

Summarizing what has happened in the thesis, the research began with a detailed literature review to set the groundwork and identify initial factors impacting data age. From there, specific methodologies such as thermal modeling, fault tolerance mechanisms analysis, and the application of task splitting and checkpointing were explored and developed into a cohesive framework. Rigorous testing through simulations provided empirical support for the framework's effectiveness, confirming its potential in real-world applications.

5.2 Summarising the final set of factors

The final set of factors that were identified as most influential in data age management include thermal thresholds, task scheduling flexibility, fault tolerance strategies, and the interaction between these elements. The initial set of factors were identified from the literature study that was done. These were then taken through a process of experimental validation and refinement, resulting in a robust set of strategies and practices that can be applied to optimize data age across various real-time systems. This evolution from theory to practice not only highlights the complexity of managing data age but also showcases the potential for significant advancements in system design and operation.

5.3 Future works

The research presented in this thesis lays a solid foundation for further exploration and development in the field of real-time systems, particularly concerning data age management under varying operational conditions. While the outcomes of this study provide valuable insights, there are several areas where future work can extend the current findings:

The following are some of the ways in which this study can be expanded:

5.3.1 Expanding Thermal Models

Future research could explore more comprehensive thermal models that encompass multi-core and heterogeneous processor architectures. This expansion would enhance the framework's applicability to a broader range of real-time systems, including those in high-performance computing environments.

5.3.2 Incorporating Diverse Fault Tolerance Techniques

The current framework primarily focuses on checkpointing and task replication as fault tolerance mechanisms. Investigating additional strategies, such as predictive fault management and more advanced error recovery algorithms, could provide deeper insights into optimizing data age and system reliability.

5.3.3 Advanced Scheduling Algorithms

Further development of scheduling algorithms that can dynamically adapt to changes in thermal conditions and fault occurrence is crucial. Future studies could develop and test algorithms that prioritize tasks based not only on their criticality and deadlines but also on their thermal and fault tolerance characteristics.

5.3.4 Real-World Implementation and Testing

Applying the developed framework in real-world scenarios across different industries could validate its effectiveness on a larger scale and under diverse operational stresses. This practical implementation would also help identify unforeseen challenges and areas for improvement.

5.3.5 Machine Learning Techniques

Employing machine learning algorithms to predict system behaviors and optimize scheduling and fault tolerance decisions in real-time could significantly enhance the adaptability and efficiency of real-time systems. Future work could focus on integrating AI-driven analytics to predict and mitigate potential data age and integrity issues before they affect system performance.

5.3.6 Cross-Domain Applications

Exploring the application of the developed data age management framework in other domains, such as IoT networks and autonomous vehicles, where real-time data processing is critical, could help in tailoring the framework to different technological needs and operational environments.

5.3.7 Addressing the limitations of this study

The research conducted in this thesis, while thorough, is subject to several limitations that are important to acknowledge. Firstly, the study is primarily focused on single-core and multi-core architectures for fault-tolerant systems and predominantly single-core for thermal-aware systems, potentially limiting its applicability to more complex or heterogeneous computing environments. Additionally, the framework may require further enhancement to fully integrate and leverage a broader spectrum of scheduling techniques, which are crucial for optimizing performance and reliability in dynamic real-time systems. In terms of thermal management, the research covers basic strategies but does not delve into more advanced thermal-aware resource management techniques that might significantly improve system performance under diverse operational conditions. Furthermore, the scope of fault tolerance mechanisms considered is somewhat narrow, potentially overlooking newer or more comprehensive fault management approaches. Lastly, the findings and the developed framework may face challenges in generalization and scalability when applied to different configurations of real-time systems or when scaled up for larger, more intricate applications. These limitations highlight the focused scope of the thesis and suggest areas where future research could expand and refine the initial findings to enhance both the robustness and applicability of the research.

Related Publications

Thesis Publications

Accepted Publication

- S. Mallareddy, P. K. Kondooru, D. Gangadharan. "Checkpointing-Aware End-to-End Data Age Analysis of Task Chains under Transient Faults". In *27th IEEE International Symposium on Real-time Distributed Computing (ISORC)*, 2024

Publication in Preparation

- S. Mallareddy, S. P. Singhal, D. Gangadharan. "Thermal-Aware Data age: Formulation and Evaluation".

Bibliography

[1] R. Ahmed, P. Ramanathan, and K. K. Saluja. Necessary and sufficient conditions for thermal schedulability of periodic real-time tasks. In *2014 26th Euromicro Conference on Real-Time Systems*, pages 243–252, 2014.

[2] R. Ahmed, P. Ramanathan, and K. K. Saluja. Temperature minimization using power redistribution in embedded systems. In *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, pages 264–269, 2014.

[3] AUTOSAR. Specification of timing extensions, r21-11, 2021.

[4] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. Power-aware scheduling for periodic real-time tasks. *IEEE Transactions on Computers*, 53(5):584–600, 2004.

[5] P. Bailis, V. J. Reddi, S. Gandhi, D. Brooks, and M. Seltzer. Dimetrodon: processor-level preventive thermal management via idle cycle injection. In *Proceedings of the 48th Design Automation Conference*, DAC '11, page 89–94, New York, NY, USA, 2011. Association for Computing Machinery.

[6] R. Baumann. Radiation-induced soft errors in advanced semiconductor technologies. *IEEE Transactions on Device and Materials Reliability*, 5(3):305–316, Sept. 2005.

[7] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte. Synthesizing job-level dependencies for automotive multi-rate effect chains. In *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 159–169, 2016.

[8] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte. End-to-end timing analysis of cause-effect chains in automotive embedded systems. *Journal of Systems Architecture*, 80:104–113, 2017.

[9] A. M. Bedewy, Y. Sun, and N. B. Shroff. Optimizing data freshness, throughput, and delay in multi-server information-update systems. *arXiv preprint arXiv:1603.06185*, 2016.

[10] E. Bini and G. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Syst*, 30:129–154, 2005.

[11] M. Bouzeghoub and V. Peralta. A framework for analysis of data freshness. In *Proceedings of the International Workshop on Information Quality in Information Systems*, pages 59–67, 2004.

[12] T. Chantem, X. S. Hu, and R. P. Dick. Temperature-aware scheduling and assignment for hard real-time applications on mpsocs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(10):1884–1897, 2011.

[13] J.-J. Chen, C.-M. Hung, and T.-W. Kuo. On the minimization fo the instantaneous temperature for periodic real-time tasks. In *13th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'07)*, pages 236–248, 2007.

[14] J.-J. Chen, S. Wang, and L. Thiele. Proactive speed scheduling for real-time tasks under thermal constraints. In *2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 141–150, 2009.

[15] A. Davare, Q. Zhu, M. Di Natale, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli. Period optimization for hard real-time distributed automotive systems. In *2007 44th ACM/IEEE Design Automation Conference*, pages 278–283, 2007.

[16] M. Dürr, G. Von Der Brüggen, K.-H. Chen, and J.-J. Chen. End-to-end timing analysis of sporadic cause-effect chains in distributed systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 2019.

[17] N. Feiertag, K. Richter, J. Nordlander, and J. Jönsson. A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics. In *IEEE Real-Time Systems Symposium (RTSS)*, 2008.

[18] N. Fisher, J.-J. Chen, S. Wang, and L. Thiele. Thermal-aware global real-time scheduling on multicore systems. In *2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 131–140, 2009.

[19] Y. Fu, N. Kottenstette, Y. Chen, C. Lu, X. D. Koutsoukos, and H. Wang. Feedback thermal control for real-time systems. In *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 111–120, 2010.

[20] Y. Fu, N. Kottenstette, C. Lu, and X. D. Koutsoukos. Feedback thermal control of real-time systems on multicore processors. In *Proceedings of the Tenth ACM International Conference on Embedded Software*, EMSOFT '12, page 113–122, New York, NY, USA, 2012. Association for Computing Machinery.

[21] Y. Gao, S. Gupta, and M. Breuer. Using explicit output comparisons for fault tolerant scheduling (fts) on modern high-performance processors. In *Proceedings of the Conference on Design Automation and Test in Europe DATE '13*, pages 927–932, 2013.

[22] P. Gohari, M. Nasri, and J. Voeten. Data-age analysis for multi-rate task chains under timing uncertainty. In *Proceedings of the 30th International Conference on Real-Time Networks and Systems (RTNS 2022)*, page 24–35. ACM, 2022.

[23] M. Günzel, K. Chen, N. Ueter, G. Von Der Brüggen, M. Dürr, and J.-J. Chen. Timing analysis of asynchronized distributed cause-effect chains. In *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 40–52, 2021.

[24] 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 *Fourth Annual IEEE International Workshop on Workload Characterization*, pages 3–14, 2001.

[25] M. Haque, H. Aydin, et al. Real-time scheduling under fault bursts with multiple recovery strategy. In *RTAS 2014*, pages 259–268, 2013.

[26] H. Huang, G. Quan, J. Fan, and M. Qiu. Throughput maximization for periodic real-time systems under the maximal temperature constraint. DAC '11, page 363–368, New York, NY, USA, 2011. Association for Computing Machinery.

[27] J. Huang, A. Raabe, K. Huang, C. Buckl, and A. Knoll. A framework for reliability-aware design exploration on mpsoc based systems. *Design Automation for Embedded Systems*, 16(4):189–220, 2012.

[28] R. Jayaseelan and T. Mitra. Temperature aware task sequencing and voltage scaling. In *2008 IEEE/ACM International Conference on Computer-Aided Design*, pages 618–623, 2008.

[29] K. Kang, S. H. Son, and J. A. Stankovic. Managing deadline miss ratio and sensor data freshness in real-time databases. *IEEE Transactions on Knowledge and Data Engineering*, 16(10):1200–1216, 2004.

[30] J. Kim, G. Bhatia, R. Rajkumar, and M. Jochim. Safer: System-level architecture for failure evasion in real-time applications. In *Real-Time Systems Symposium (RTSS) 2012 IEEE 33rd*, pages 227–236, 2012.

[31] T. Kloda, A. Bertout, and Y. Sorel. Latency analysis for data chains of real-time periodic tasks. In *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 360–367, 2018.

[32] T. Kloda, A. Bertout, and Y. Sorel. Latency analysis for data chains of real-time periodic tasks. In *23rd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 360–367, 2018.

[33] J. Kong, S. W. Chung, and K. Skadron. Recent thermal management techniques for microprocessors. *ACM Comput. Surv.*, jun 2012.

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

[35] P. Kumar and L. Thiele. Cool shapers: shaping real-time tasks for improved thermal guarantees. In *Proceedings of the 48th Design Automation Conference*, DAC '11, page 468–473, New York, NY, USA, 2011. Association for Computing Machinery.

[36] P. Kumar and L. Thiele. Cool shapers: shaping real-time tasks for improved thermal guarantees. DAC '11, page 468–473, New York, NY, USA, 2011. Association for Computing Machinery.

[37] P. Kumar and L. Thiele. System-level power and timing variability characterization to compute thermal guarantees. In *2011 Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 179–188, 2011.

[38] A. Labrinidis and N. Roussopoulos. Exploring the tradeoff between performance and data freshness in database-driven web servers. *The VLDB Journal*, 13(3):240–255, 2004.

[39] Y. Lee, H. S. Chwa, K. G. Shin, and S. Wang. Thermal-aware resource management for embedded real-time systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2857–2868, 2018.

[40] G. d. A. Lima and A. Burns. An optimal fixed-priority assignment algorithm for supporting fault-tolerant hard real-time systems. *Computers IEEE Transactions on*, 52(10):1332–1346, 2003.

[41] J. Liu, M. Wei, W. Hu, X. Xu, and A. Ouyang. Task scheduling with fault-tolerance in real-time heterogeneous systems. *Journal of Systems Architecture*, 90:23–33, 2018.

[42] V. Peralta. Data freshness and data accuracy: a state of the art. Technical report, Reportes Técnicos 06-13, UR. FI–INCO, 2006.

[43] C. Pinello, L. Carloni, and A. Sangiovanni-Vincentelli. Fault-tolerant distributed deployment of embedded control software. *Computer-Aided Design of Integrated Circuits and Systems IEEE Transactions on*, 27(5):906–919, 2008.

[44] P. Pop, V. Izosimov, P. Eles, and Z. Peng. Design optimization of time-and cost-constrained fault-tolerant embedded systems with checkpointing and replication. *Very Large Scale Integration (VLSI) Systems IEEE Transactions on*, 17(3):389–402, 2009.

[45] S. Punnekkat, A. Burns, and R. Davis. Analysis of checkpointing for real-time systems. *Real-Time Syst. J.*, 20(1):83–102, 2001.

[46] L. Schor, I. Bacivarov, H. Yang, and L. Thiele. Worst-case temperature guarantees for real-time applications on multi-core systems. In *2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium*, pages 87–96, 2012.

[47] S. Wang and R. Bettati. Delay analysis in temperature-constrained hard real-time systems with general task arrivals. In *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*, pages 323–334, 2006.

[48] C. Weaver, J. Emer, S. Mukherjee, and S. Reinhardt. Techniques to reduce the soft error rate of a high-performance microprocessor. In *ACM SIGARCH Computer Architecture News*, volume 32, page 264, 2004.

[49] Y. Wei, S. H. Son, and J. A. Stankovic. Maintaining data freshness in distributed real-time databases. In *Proceedings of 16th Euromicro Conference on Real-Time Systems*, pages 251–260, 2004.

[50] M. Xiong, S. Han, and K.-Y. Lam. A deferrable scheduling algorithm for real-time transactions maintaining data freshness. In *Proceedings of 26th IEEE Real-Time Systems Symposium*, pages 11–pp, 2005.

[51] S. Zhang and K. S. Chatha. Thermal aware task sequencing on embedded processors. DAC '10, New York, NY, USA, 2010. Association for Computing Machinery.

[52] Y. Zhu and F. Mueller. Feedback edf scheduling exploiting dynamic voltage scaling. In *Proceedings. RTAS 2004. 10th IEEE Real-Time and Embedded Technology and Applications Symposium, 2004.*, pages 84–93, 2004.