FAULT TOLERANCE IN A TWO-STATE REGULARITY-BASED CHECKPOINTING SYSTEM

A Senior Honors Thesis

Submitted to

the Department of Computer Science,

College of Natural Sciences and Mathematics

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science

in Computer Science

By

Elena Torre

May 2021

FAULT TOLERANCE IN A TWO-STATE REGULARITY-BASED CHECKPOINTING SYSTEM

Elena Torre

APPROVED:

Dr. Albert M. K. Cheng, Thesis Director

Dr. Yuhua Chen

Dr. Stuart Long

Dr. Dan Wells, Dean

College of Natural Sciences and Mathematics

ACKNOWLEDGEMENTS

Firstly, I would like to thank my thesis advisor Dr. Albert M. K. Cheng for his guidance. He never failed to be approachable and encouraging, and I can say without a doubt that I would not have pursued the project had I not had him as a mentor. My time as a member of his Real-Time Systems research group bolstered my interest in the field and helped me build the skills to pursue this research project.

Secondly, I would like to thank Dr. Guangli Dai for all his help and support over the past two years. He spent countless hours helping me acclimate to the research process, explaining difficult concepts and offering invaluable advice, all while working on his own PhD dissertation. I cannot overstate how thankful I am for his time and how important his assistance was to the completion of this project.

Finally, I would like to thank my family for supporting me throughout my undergraduate studies at the University of Houston. I would not be where I am without them and I am beyond grateful for the opportunities they have afforded me.

FAULT TOLERANCE IN A TWO-STATE REGULARITY-BASED CHECKPOINTING SYSTEM

An Abstract of a Senior Honors Thesis

Submitted to

the Department of Computer Science,

College of Natural Sciences and Mathematics

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science

in Computer Science

By

Elena Torre

May 2021

ABSTRACT

Embedded real-time virtualized systems serve a wide range of functions in many industries. They can encompass multiple independent applications that must share limited computational resources. The tasks running within these applications may vary in criticality and have different timing requirements. Many models have been introduced to ensure reliability and efficiency when scheduling tasks in these systems. Models in the Hierarchical Real-time Scheduling (HiRTS) framework can enable the virtualization and sharing of resources. The Regularity-based Resource Partition model (RRP) can be used to achieve transparent scheduling for such models. Many use resource-level checkpointing with rollback recovery as a method to resolve transient faults without modifying application code. However, checkpoint insertions are known to incur high time and energy overheads.

This thesis project proposes the Two-state Regularity-based Checkpointing model. This HiRTS model will ensure fault tolerance when scheduling independent, mixed-criticality real-time task sets on limited resources. By reducing checkpoint insertions before the first fault, the system will achieve a lower time overhead while still ensuring fault tolerance. Simulation-based experiments were performed using a simple implementation of the proposed scheduling model. Results indicate that the model allows independent mixed-criticality task sets to maintain real-time performance guarantees for their high-priority tasks, even under a high fault rate. In addition, results show that unaffected task sets will still not suffer delays, even if other sets are experiencing an elevated number of faults.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iii
ABSTRACT v
TABLE OF CONTENTS vi
LIST OF TABLES vii
LIST OF FIGURES viii
CHAPTER 1: Introduction 1
CHAPTER 2: Background Works 7
2.1 Embedded Virtualized Systems and Real-time Scheduling 7
2.2 Hierarchical Scheduling 7
2.3 The Regularity-based Resource Partition Model 8
2.4 Scheduling Mixed-criticality Task Sets 11
2.5 Fault-tolerant Scheduling 12
2.6 The Two-state Checkpointing Model 16
CHAPTER 3: The Two-state Regularity-based Checkpointing Model 18
3.1 An Overview 18
3.2 Applicable Task Sets 18
3.3 Algorithm 20
3.4 Space Complexity 22
3.5 A Sample Case 23
3.6 Time Complexity 26
3.7 Potential Benefits 26
CHAPTER 4: Experimental Evaluation 28
4.1 Motivation 28
4.2 Experiment Setting 28
4.3 Results for Group A 35
4.4 Results for Group B 40
CHAPTER 5: Conclusions and Future Work 43
BIBLIOGRAPHY 45

LIST OF TABLES

Table 1: Fault distribution for simulations in Group A. The partition set experienced between 4 and 46 faults across all partitions. 33

Table 2: Fault distribution for simulations in Group B. The partition set experienced between 0 and 7 faults in $P0$. 33

Table 3: Ending time for each partition during the simulations in Group A. 35

Table 4: Ending time for each partition during the simulations in Group B. 40

LIST OF FIGURES

Figure 1: A 2-layer HiRTS model [21]. 2

Figure 2: A sample RRP schedule [7]. 9

Figure 3: Supply function and instant regularity of a regular partition [24]. 10

Figure 4: Time slices to be re-executed after a fault is spotted [7]. 13

Figure 5: An example of checkpoint placement in a schedule generated by the Fault-tolerant RRP model [7]. 14

Figure 6: Schedules for $S1$ and $S2$ generated by the proposed model. 23

Figure 7: Sub-schedules generated by the proposed model. 24

Figure 8: An example illustrating how the proposed model would detect and resolve a fault in $P3$ during execution. 25

Figure 9: CPU usage on the test bed used for experimental simulations. 29

Figure 10: Major schedules used for experimental simulation. 30

Figure 11: Plans used for experimental simulation. Each one represents a sub-schedule for the partition set. 31

Figure 12: Results for Case 3 in Group A. The partition set experienced 7 faults which affected $P1$, $P2$, $P3$, $P4$ and $P5$. 36

Figure 13: Results for Case 6 in Group A. The partition set experienced 42 faults which affected all partitions. 38

Figure 14: Results for Case 7 in Group A. The partition set experienced 46 faults which affected all partitions. 39

Figure 15: Results for all cases in Group B. The partition set experienced between 0 and 7 faults in $P0$. 41

CHAPTER 1: Introduction

Embedded real-time virtualized systems have become an invaluable asset to many industries. These systems are defined as any combination of hardware and software designed to perform time-bound, dedicated functions within a larger system or product [2]. They may encompass multiple independent applications that depend on a limited number of computational resources. A surveillance system, for example, may encompass functions like robot control and image processing, which must adhere to strict time constraints while relying on a limited computational resource supply [26]. In addition, tasks performed within these applications may control functions with widely different criticalities and timing constraints [4].

As such, several scheduling models have been developed to address common issues of embedded real-time virtualized systems. Efficient resource distribution, fault tolerance, prioritization of critical tasks and adherence to deadlines are just a few examples [35]. Hierarchical Real-time Scheduling (HiRTS) [19] is a useful framework for such cases, as it separates the virtualization and distribution of computational resources from the scheduling of tasks.

Fig. 1 depicts a two-layer HiRTS model that manages a group of independent applications. On the resource layer, a virtual representation of the computational resource is known as a partition, each one representing access to the resource for a period of time. On the task layer, the scheduler assigns each partition to a task set, corresponding to a specific application. Models within this framework have been known to provide reliability

when scheduling real-time tasks [7]. The separate layers increase portability, as schedulers can manage applications without task-level information.

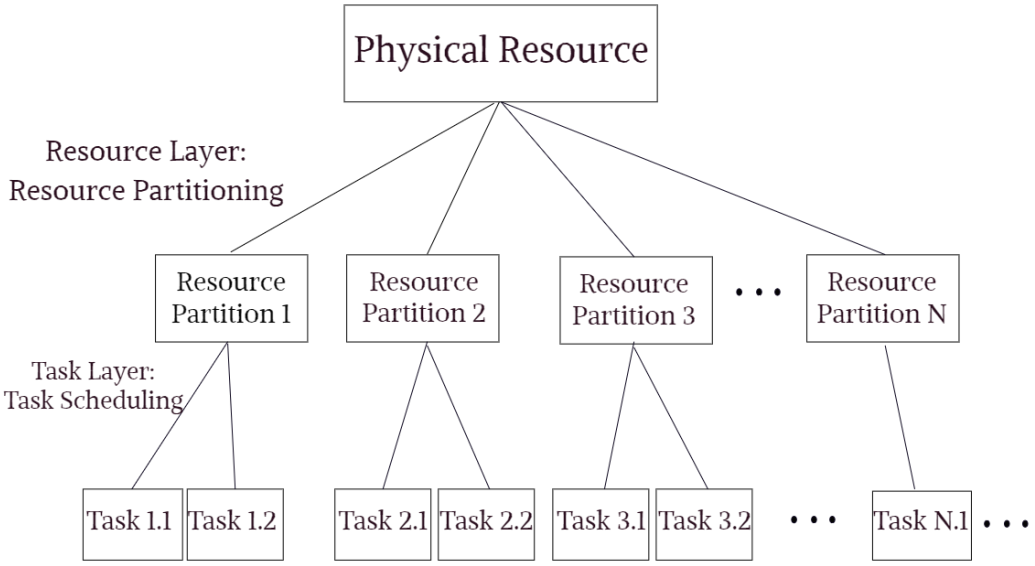

Figure 1: A 2-layer HiRTS model [21].

HiRTS models require a partitioning model to manage the division of resources among tasks [7]. The Regularity-based Resource Partition (RRP) [20] model is considered particularly useful, as it allows for resource partitions that are as evenly distributed as possible. The model bounds the deviation between the ideal and actual distribution of resources among partitions. It is known for achieving a high level of transparency for task-level scheduling.

Fault resolution is another important aspect of an effective HiRTS model. Transient faults must be detected and resolved without missing any critical deadlines [21]. Checkpointing with rollback recovery is a well-established method to deal with transient

faults and increase reliability [28]. Many models within the HiRTS framework ensure fault tolerance and transparency by inserting checkpoints at the resource layer. These models initiate a checkpointing partition for fault detection and resolution and schedule it as another application in the system [7]. Every checkpoint has an adjacent redundancy area, used to re-execute the faulty sections of tasks after detection. This checkpointing method has been shown to guarantee a high level of fault tolerance, even in systems with high loads and frequent fault occurrence [7]. However, checkpoint insertions can result in high time and energy overheads. A single checkpoint insertion can encompass up to 24% of an application's total execution time and up to 17% of its total energy consumption, according to experiments performed on an embedded processor using emerging non-volatile memory technology [28]. While other works have not indicated such results, efficiency is always a concern when designing embedded systems.

The simplest method to address these issues is to reduce the number of checkpoint insertions. Previous works [24, 28] have achieved this through the Two-state Checkpointing model. After the first occurrence of a fault, the system enters a "faulty" state during which faults are assumed to happen at a higher rate. Until that first fault, intervals between checkpoints are kept significantly larger, thereby reducing checkpoint insertions [28].

Previous implementations of the Two-state Checkpointing model have shown promising results but have been limited to a single application executing hard real-time tasks [26]. If this strategy were to be used in conjunction with hierarchical scheduling and the RRP model, it would provide increased efficiency and reliability for independent task

sets in embedded systems. Implementing a two-state checkpointing scheme within the resource layer would keep the addition and removal of applications easy, enabling resource division and task scheduling without changes to the applications' code.

Mixed-criticality applications would particularly benefit from this strategy. When the system enters the "faulty" state, resource utilization for the partitions becomes lower. Each partition receives less access to the computational resource, as more time is spent in checkpointing and re-execution [28]. This may lead to missed deadlines for hard real-time applications, as access to the resource will be granted in longer intervals. Mixed-criticality applications, on the other hand, only have strict deadline requirements for high-priority tasks [32]. Lower priority or "soft" real-time tasks have more flexible deadlines or often no deadlines at all. They can be temporarily disregarded without major consequences, safely transitioning the system into the "faulty" state.

Mixed-criticality scheduling has been a popular topic within the field of real-time systems for some time, as there are many challenges when scheduling tasks of varying priority. Recent works in the field have introduced a variety of techniques and algorithms meant to address specific issues within these systems. For example, a recent model introduced by Zhang [34] uses dynamic processor speed scaling, to ensure deadline adherence and low energy overhead. The model is meant to manage energy aware sporadic tasks and prioritizes low energy consumption. Likewise, Agrawal et al. [1] presented an algorithm to synthesize communication schedules for shared media networks employed by hard real-time tasks. All tasks have strict deadlines, so their level of priority is defined by the number of faults they can tolerate. Neither model addresses resource distribution and

both require that all tasks have a pre-determined static priority. In a recent paper Choi et al. [8] presented a new scheduling technique to improve the accuracy of Worst-case Response Time (WCRT) analysis for mixed-criticality task sets on multi-core embedded systems. Their analysis is based on a scheduling model that allows for the partial dropping of low-priority tasks to guarantee the schedulability of safety-critical ones. This model emphasizes fault tolerance for high priority tasks but does not utilize resource virtualization to manage access to the processors.

As previously mentioned, hierarchical scheduling with resource virtualization can be a very useful tool for more complex mixed-criticality scheduling models and is present in many current works in the field. For instance, Kadeed et al. [18] presented a HiRTS model that can re-configure resource utilization boundaries for mixed-criticality task sets during run-time. Similarly, Yang and Dong [33] presented a model that could assign multiple resource utilization budgets for depending on the specifically assigned criticality of a task within a set. While both models show promising results, all require detailed task-level information and focused on task-level scheduling.

The goal of my undergraduate research, as outlined in this thesis, is to introduce a scheduling model for real-time mixed-criticality embedded virtualized systems that will integrate HiRTS, the RRP model and aspects of the Two-state Checkpointing model. It will handle the scheduling of real-time mixed-criticality tasks while ensuring a reasonable tolerance of transient faults. Unlike other mixed-criticality hierarchical scheduling models [18, 33], the Two-state Regularity-based Checkpointing model will minimize the use of

task-level information and implements regularity-based resource distribution. This will ensure transparent scheduling, portability and real-time guarantees for high-priority tasks.

This thesis is organized as follows:

Chapter 2 summarizes the development of HiRTS, the RRP model, two-state checkpointing and other important works in the field. Chapter 3 introduces the Two-state Regularity-based Checkpointing model and outlines its potential advantages. Chapter 4 discusses experimental simulations performed using an implementation of the proposed model. The settings, results and implications of these simulations are outlined in this section. Lastly, Chapter 5 details the conclusions and plans for further research.

CHAPTER 2: Background Works

2.1 Embedded Virtualized Systems and Real-time Scheduling

Due to rapid advances in computer hardware and software, the use of embedded devices has become increasingly widespread. Many real-time applications make use of these devices, such as environmental monitoring, intelligent transportation and avionics [35]. Embedded real-time virtualized systems often encompass many independent applications that share limited computational resources [7]. In addition, each application may be performing tasks of varying criticality for the system. For example, an aircraft may have to manage high priority tasks, like a real-time cabin pressure monitoring system, and low priority tasks, like air conditioning for the passengers. As such, many scheduling models have been developed to address these concerns.

2.2 Hierarchical Scheduling

HiRTS is a useful framework to schedule real-time applications in embedded virtualized systems. Models within this framework separate resource virtualization and partitioning from task scheduling. Deng and Liu [11] were the first to present a two-layer hierarchical scheduling model for real-time tasks. The model was intended to manage independent applications in an open system, which is defined as a system in which some timing attributes of real-time applications are not known [11]. As such, it could schedule both real-time and non-real-time applications on a single processor. It was later expanded to include non-predictable tasks as well as local and global resource distribution [10]. Shigero et al. [30] later implemented a time slot-based algorithm to handle resource distribution in

hierarchical real-time systems. Both models showed improvements in reliability and efficiency when compared to contemporary scheduling algorithms, like Earliest Deadline First (EDF) and Rate-monotonic [10, 11, 30].

2.3 The Regularity-based Resource Partition Model

Resource partitioning is an important aspect of HiRTS, as the method of distribution chosen for the resource layer can have a tremendous impact on the efficiency of a model. These policies are known as resource models [20]. Several have been implemented in two-layer HiRTS, including the Bounded-delay model [22], the Periodic model [29] and the Explicit Deadline Periodic model [12]. However, none can provide transparency for task scheduling except the Regularity-based Resource Partition (RRP) model [7]. Mok and Feng [24] first introduced the RRP model as a resource virtualization and division strategy for open real-time systems sharing a single resource. Li and Cheng [20, 21] recently expanded the model to encompass a multi-resource platform, as embedded systems may manage access to multiple processors, input devices or power sources [23].

The RRP model virtualizes access to a resource and splits it into regularly-sized time slices. A time slice is defined as a unit of time that cannot be partitioned any further [7]. The length of this unit can vary and is decided before generating any static schedules. These slices are allocated to different resource partitions, each of which is mapped to a task set in the system. The Least Common Multiple (LCM) of the periods of these tasks is referred to as a hyper-period, as shown in Fig. 2. Each resource partition *P* is considered to

have a certain capacity, indicated by two values: its *availability factor* and its *supply regularity*. They are represented by the tuple $(\alpha, R_s(P))$.

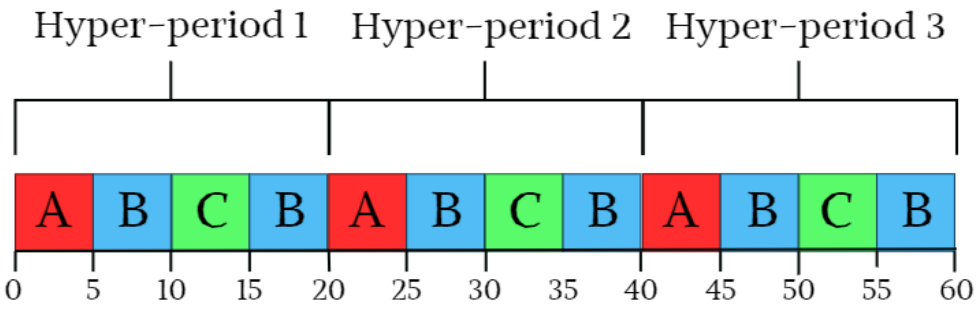

Figure 2: A sample RRP schedule [7].

The availability factor (α) is the ratio between the time slices allocated to a partition during its period and said period's length. In order to understand the supply regularity, we must first understand the concepts of *supply function* and *instant regularity*.

The supply function $S(t)$ of a partition *P* is the total number of time-slices that are allocated to *P* from time 0 to *t*. For example, some resource partition *A* given time slices 0, 2, 3 and 5 during a hyper-period with a length of 7 has an availability factor of $\frac{4}{7}$. Fig. 3 depicts the ideal distribution for a partition with this availability factor through the grey line. Unfortunately, this distribution is rarely achievable [7]. The red line represents the value of the supply function, which is a more realistic representation of the resource distribution.

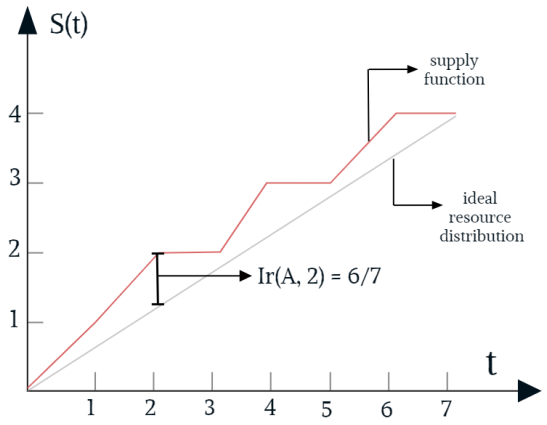

Figure 3: Supply function and instant regularity of a regular partition [24].

The instant regularity $Ir(t)$ at time t of a partition P is given by $S(t) - t \times \alpha(P)$. This value represents the difference between the ideal and realistic allocation of time slices for some partition. For the partition depicted on Fig. 3, the instant regularity at $t = 2$ would be $\frac{6}{7}$. In order to measure the stability of a partition, the RRP model calculates the largest difference between the instant regularity of any two time spots during a hyper-period. This difference known as the supply regularity.

Let i, j and k be non-negative integers. The supply regularity $R_s(P)$ of a resource partition P equals to the minimum integer k such that $\forall_{i,j}, |I(P,i) - I(P,j)| < k$. where $i < j$.

Partitions with supply regularity values no larger than 1 are referred to as *regular partitions*. As shown by Mok and Feng [24], regular partitions preserve the utilization bounds for both fixed and dynamic priority scheduling. That is, each task set in the system

will perform as if it were executing on a dedicated resource with the same availability, maintaining task set independence and preventing fault propagation.

2.4 Scheduling Mixed-criticality Task Sets

Ensuring deadline adherence and efficient resource distribution are not the only challenges for HiRTS models. As previously mentioned, many embedded real-time systems must schedule tasks of varying priority. Zhang [34] recently introduced a scheduling model for mixed-criticality real-time tasks that dynamically updated the frequency of the processor to conserve energy and ensure performance for high-priority tasks. While effective, these techniques can be highly expensive and has only been implemented on a single mixed-criticality task set. While most HiRTS models have been developed to handle hard real-time tasks, mixed-criticality task sets are becoming a popular topic within embedded systems [13]. However, most models [3, 4, 13, 18] developed since Vestal [32] first introduced multi-criticality analysis to real-time scheduling have focused on task-level scheduling and require task-specific information.

High priority tasks within embedded real-time systems will often manage safety-critical functions with strict fault tolerance requirements [7]. While much research has been done on fault tolerance within real-time systems [22], research into fault-tolerant HiRTS is still in its early stages. Hyun and Kim [14] first presented a HiRTS model in which fault detection and resolution were treated as an independent task within the system. Jin [15] later introduced a similar model, where each partition was assigned an adjacent backup partition for fault resolution. Similarly, Tchamgoue et al. [31] developed a model which

considered the transient fault requirements of each task set in the system to design more efficient backup partitions. Hyun et al. [15] later expanded upon their previous model, introducing temporal partitioning, and adding task-level and resource-level schedulers.

2.5 Fault-tolerant Scheduling

While the models listed above were shown to perform well [14, 15, 17, 30], all were relatively inefficient when it came to resource partitioning. They used the Periodic model for resource distribution, which lacks the stability and portability that the RRP model provides. In addition, the backup partition strategy employed resolved faults at the task level, making implementation for cross-platform applications more difficult. In order to resolve these issues, Cheng et al. [7] introduced a fault-tolerant hierarchical scheduling model that kept transient fault resolution within the resource layer. As such, the Fault-tolerant RRP model can ensure fault tolerance for independent applications without making any changes to their code, increasing portability [7]. The proven transparency property of the RRP model ensures that, once resource division is complete, task scheduling can be done using single-resource techniques [21]. Thus, the model can ensure task-level and resource-level fault tolerance as well as cross-layer transparency.

The Fault-tolerant RRP model was developed under several assumptions [7]. First, the system is assumed to be fault-free at start time. Furthermore, faults are assumed to be independent from one another, but can overlap once they occur, creating faulty intervals. As shown in Fig. 4, if a fault occurs during time slice 1 and is detected at time slice 4, all time slices belonging to partition B in that interval must be re-executed. It is assumed that

faults always result in errors, like abnormal bit flips or energy corruption, that could cause the calculations after to be invalid. Finally, the fault occurrence rate in the system is assumed to follow the Poisson Distribution. This last assumption can be adjusted to account for changes in hardware and dynamic environments, which can affect the fault rate.

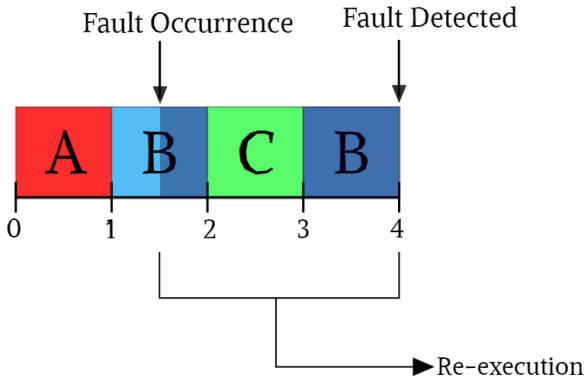

Figure 4: Time slices to be re-executed after a fault is spotted [7].

The model is given a set of partitions $P = \{P_i : 1 \leq i \leq n\}$, the number of checkpoints N, the length of the hyper-period hp and size in time slices of the redundancy area l_R as input.

The value of N represents the expected number of faults during a hyper-period. While there is always a possibility of experiencing countless faults, but the Poisson distribution makes this situation is very unlikely [7]. As such, the model defines the variable β as the probability of experiencing at most N faults during a hyper-period. It then sets a reasonably high value for β and finds a corresponding value for N using the following inequality:

$$\beta \geq \sum_{i=0}^{N} \frac{\gamma^i}{i!} e^{-\gamma} \quad (1)$$

These *N* checkpoints are then inserted at equidistant intervals. Fig. 5 depicts a sample schedule generated by the Fault-tolerant RRP model. Checkpoints are assumed to have a size of one time slice [7]. Each one has an adjacent redundancy area, used to re-execute faulty time slices. The length of these *N* redundancy areas is based upon the expected fault rate and an approximation algorithm based on the Adjusted Availability Factor (AAF). Many HiRTS models that implement RRP, including those presented in [14, 15, 17], utilize this algorithm to map the availability factor of each of their resource partitions to a higher value. This allows the scheduling of multiple partitions within polynomial time and with a minimum overhead in resource utilization.

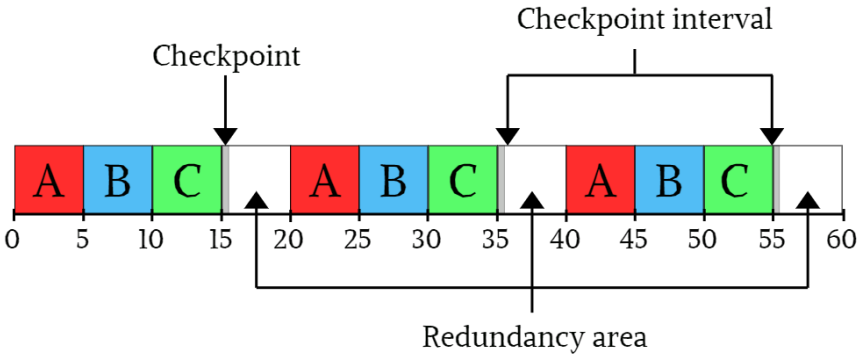

Figure 5: An example of checkpoint placement in a schedule generated by the Fault-tolerant RRP model [7].

The size of this redundancy area l_R is based upon the following equation:

$$l_R = \left\lceil 1.1 \times \sum_{i=1}^{n} AAF((\alpha, R_s(P_i))) \times \frac{N_p}{N} \times (1 - e^{-\gamma}) \right\rceil \quad (2)$$

The sum $\sum_{i=1}^{n} AAF((\alpha, R_s(P_i)))$ represents the value of the adjusted availability of every partition in the set. It is multiplied by the probability of at least one fault under the Poisson distribution $(1 - e^{-\gamma}) \times N_p$, where N_p is the number of time slices given to each regular partition. This value is then divided between the *N* redundancy areas during the hyper-period. Finally, the model increases the redundancy area by 10% and rounds the value up to an integer. This is done to compensate for potential overhead from context-switching [7].

In addition to the partitions in *P*, the model initializes a checkpoint redundancy regular partition with an availability factor of $\frac{(N+N\times \mathrm{l_R})}{hp}$. As previously mentioned, the model distributes access to the computational resource among these partitions, such as access to a processor. The Magic7 algorithm [18] has been shown to efficiently divide a single resource. The MulZ algorithm [19] was later introduced for multi-resource scheduling and was shown to outperform other resource partitioning algorithm through the use of multiple approximation sequences. Both algorithms use AAF to approximate the availability of partitions.

The model initializes an array Q and a FIFO queue G. The length of Q will be hp and each element will represent a time slice in the schedule. This array maps each time slice to a specific resource partition. G is used to store faulty time slices that need re-execution. Time slices are evenly divided among partitions using Magic7 or MulZ. The distribution of these slices is recorded in Q. Once the schedule has been determined, the

algorithm simply transverses Q time slice by time slice, granting access to the resource and re-executing faulty partitions as needed.

The resource-level checkpointing scheme used by the Fault-tolerant RRP model has been shown to ensure schedulability and deadline adherence, even in systems with high utilization and fault occurrence [7]. As previously mentioned, checkpointing with rollback recovery is one of the most commonly used methods to ensure fault recovery, despite the relatively high overhead caused by checkpoint insertions [15]. Due to the nature of distributed systems, eventual fault occurrence is considered inevitable, and fault tolerance models designed for them tends to be pessimistic in nature [25]. However, efficiency is always a concern when designing embedded systems, and many scheduling models have been developed to compensate for the cost of checkpointing.

2.6 The Two-state Checkpointing Model

Salehi et al. [28] presented a Two-state model for energy efficiency in hard real-time systems that use checkpointing for fault resolution. The model achieves low time and energy overheads while ensuring tolerance of k faults in hard real-time systems [26]. It considers real-time applications, each having a corresponding triple (t, D, k). These values represent the application's execution time, deadline and number of tolerable faults respectively. For periodic tasks, D represents the period.

This model takes advantage of both online and offline schemes. The offline section is performed at design time and determines the optimized checkpoint intervals for both

faulty and non-faulty states. The online section is performed at runtime and it simply selects the pre-calculated checkpoint intervals based on fault occurrence.

Non-uniform checkpoint intervals are calculated considering the worst-case fault scenario while still seeking to postpone checkpoint insertions as much as possible. The system must reserve enough slack time to recover if a fault occurs and to tolerate an additional $k - 1$ faults. Each non-uniform interval has a corresponding uniform interval that becomes effective if a fault is detected by a non-uniform interval's checkpoint. Muhammad et al. [25] later demonstrated that this checkpoint policy considerably reduces checkpoint overheads when compared to periodic checkpoints. While these results are promising, these models only encompass task-level fault tolerance for a single application and have not been implemented in tandem with hierarchical scheduling.

CHAPTER 3: The Two-state Regularity-based Checkpointing Model

3.1 An Overview

The Two-state Regularity-based Checkpointing model will schedule mixed-criticality real-time task sets on one or more limited computational resources. This two-layer HiRTS model will implement regularity-based resource partitioning to schedule both its assigned task sets and transient fault resolution. This will be done on the resource layer, as shown in [7]. On the task layer, each mixed-criticality task set can schedule its own periodic tasks with using single-resource techniques. Task-level information remains encapsulated on this layer, with the resource-level scheduler receiving only the tuple $(\alpha, R_s(P))$ as a measurement of the needs of each task set.

3.2 Applicable Task Sets

The proposed model is designed for independent task sets and does not consider situations where a task set may be waiting for the output of another [7]. As mentioned in the previous section, resource partitions mapped to each task set must be regular, ensuring that balanced resource distribution is possible. Due to the schedulability test used by the RRP model, periodic tasks within these sets must be aligned. The key parameters of an aligned task must be multiples of the selected time slice length [27]. These parameters include arrival time, Worst-case Execution Time (WCET), deadline and period length. The Two-state Regularity-based Checkpointing model seeks to minimize the use of task-level

information, and as such alignment will be determined using only period length and execution time. In addition, the laxity of each task set must be greater than the cost of checkpointing. That is, each task must have enough time to re-execute faulty partitions before its next instance. Tasks with a laxity lower than the cost of checkpointing must be assigned to alternate resources that use more expensive techniques to achieve reliability, like voltage scaling [7].

As mentioned in the previous chapter, the model was developed for mixed-criticality task sets. These sets encompass periodic tasks with different levels of priority, based on their deadline, period, designated function, and many other factors [32]. In order to ensure deadline adherence for critical tasks, scheduling system designers must be conservative when determining the WCET of a high-priority task in the set. This often makes is difficult to determine the schedulability of a task set, as the WCET value used for analysis is often an upper bound estimate that far exceeds the real execution time. However, due to the special schedulability test employed by the RRP model [7], hard real-time tasks with utilization ratios smaller than their availability factor are still expected to be schedulable. The utilization ratio is the ratio between the actual number of time slices used by a task set and the length of the hyper-period, which can be much smaller than the availability provided by a regular resource partition [7]. As a result, the model can assign a range of valid availability factors for regular partitions, with a lower bound that must be satisfied and an optimistic upper bound that can be achieved if time allows. Soft real-time tasks of lower priority can be executed using the remaining portion of the time slices, but do not have real-time performance guarantees.

3.3 Algorithm

Much like the Two-state Checkpointing model presented in [29], the proposed model will have an offline portion and an online portion. During the online portion, the system will be in one of two states: S_1, the fault-free state, and S_2, the faulty state. The schedules for both states will be determined during the offline portion. During S_1, the schedule will have significantly less checkpoint insertions, as previous works [25, 29] demonstrate that fault occurrence is more likely after the first fault. As such, the system is given a different expected fault rate for each state. The fault rate after the first fault occurrence γ_2 is expected to be two times larger than γ_1. This assumption is based failure trace data collected from various distributed systems by Javadi et al. [16] but can be adjusted.

During the offline portion, the model receives the following input:

- The partition set $P = \{P_i : 1 \leq i \leq n\}$.
- The size of the redundancy area l_R.
- Number of checkpoints during S_1, N_1.
- The number of checkpoints during S_2, N_2.
- Length of hyper-period during each state, hp_1 and hp_2.

The value of N_1 and N_2 are determined by the following equations:

$$\beta \geq \sum_{i=0}^{N_1} \frac{\gamma_1^i}{i!} e^{-\gamma_1} \quad (3)$$

$$\beta \geq \sum_{i=0}^{N_2} \frac{\gamma_2^i}{i!} e^{-\gamma_2} \quad (4)$$

During S_1 the schedule will only contain *critical checkpoints*. These N_1 checkpoints do not have an adjacent redundancy area, as fault resolution is done exclusively in S_2. Critical checkpoints will also be present in the S_2 schedule, alongside (N_2 - N_1) *additional checkpoints*. All checkpoints will have an adjacent redundancy area during this state. As such, the model will determine the value of l_R by inserting N_2 and γ_2 into Equation 2. In addition, the model will use the lower bound availability factor, as the schedule during this state seeks to maximize reliability while ensuring that partitions are given enough resources to execute high-priority tasks.

Unlike the previous Two-state Checkpointing model [28], checkpoint intervals will be uniform in both states, as the proposed model must be able to generate a schedule without task-level information. Furthermore, K.M. Chandy [5, 6] has proven that the most effective checkpoint placement strategy is to insert them at equidistant intervals.

The proposed model will use this input to generate two *major schedules*, which are stored in arrays Q_1 and Q_2 respectively. In addition to the partition set *P*, two additional checkpoint redundancy partitions are initialized. P_{CR1} represents the critical checkpoints during S_1 and is given an availability factor of $\frac{(N_1)}{hp_1}$. P_{CR2} represents the N_2 critical and additional checkpoint during S_2 and their adjacent redundancy areas. It is given an availability factor of $\frac{(N_2+N_2\times \mathrm{l_R})}{hp_2}$. In addition to checkpointing and redundancy, these partitions will contain a special module called the *schedule switcher*, used to swap schedules when appropriate. The model then uses Magic7 or MulZ to divide the time slices

available among the partition set and both checkpoint redundancy partitions, generating the major schedules for both states.

3.4 Space Complexity

Each of the two schedules will be split into *sub-schedules*, each one corresponding to a checkpoint interval. This will allow the model to transition from S_1 to S_2 before the end of the hyper-period, as critical checkpoints hold the same relative position in both major schedules. This will result in a total of $(N_2 + N_1 + 2)$ schedules. In addition, the model initializes n re-execution sub-schedules, one for each regular partition. These sub-schedules are meant to re-execute a portion of one of the partitions in the set and then immediately switch back to the main schedule.

This results in a space complexity of $(N_2 + N_1 + 2 + n)$. The implementation used for experimental simulations in Chapter 4, for example, is given a set of 6 regular partitions. The schedules for S_1 and S_2 are given 2 critical checkpoints and 2 additional checkpoints respectively. This results in an additional 2 sub-schedules for S_1 and 4 sub-schedules for S_2. The model also uses one re-execution schedule for each partition. In total, the model uses 14 schedules. The number of schedules is expected to grow linearly as the number of partitions and checkpoints increase. As such, the algorithm will not experience the state explosion problem.

3.5 A Sample Case

Given a set of regular partitions $P = \{P_i : 0 \leq i \leq 3\}$, the model generates Q_1 and Q_2 as shown in Fig. 6. Critical checkpoints are shown in green while additional checkpoints are shown in dark blue. Checkpoints are used to detect faulty time slices and record their position on a queue *G*, as done in [7]. Redundancy areas are shown in light blue. Faulty partitions are re-executed within these reserved time slices.

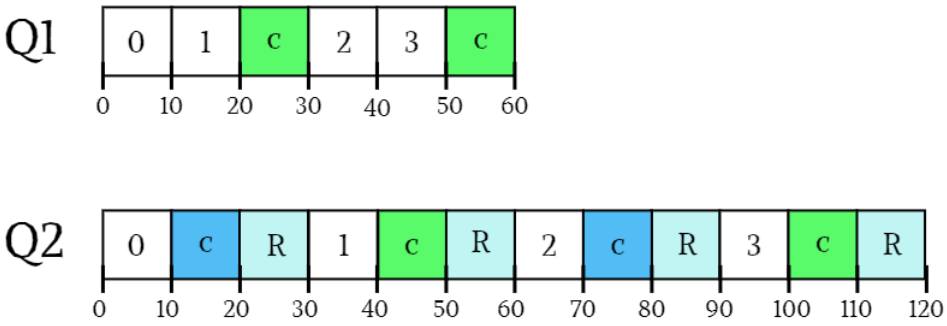

Figure 6: Schedules for S_1 and S_2 generated by the proposed model.

As previously explained, the schedules are then split into sub-schedules. In addition, the model generates four schedules for re-execution. These sub-schedules are to be executed within the redundancy area and they contain reserved time slices for partition re-execution and a small checkpoint.

This checkpoint is only used to switch back to the regular execution schedule, which can be done in negligible time. It does not check for faults, as the model operates under the assumption that transient faults will be resolved after a single re-execution. All sub-schedules are shown on Fig. 7.

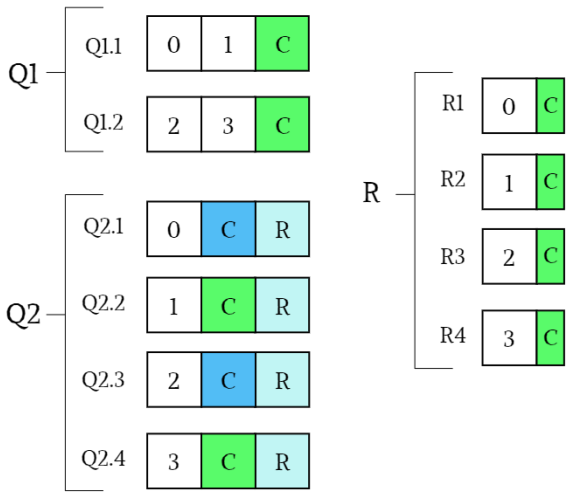

Figure 7: Sub-schedules generated by the proposed model.

Once the model enters the online portion, it will begin execution. Since the system is assumed to be fault-free at start time [7], the model will begin iterating through $Q_{1.1}$. If a time slice is not part of the checkpoint partition, the model grants its assigned task access to the resource. If it is, the algorithm checks if any faults occurred during the checkpoint interval. If there are no faults, the algorithm switches to the next sub-schedule for S_1, in this case $Q_{1.2}$. If there were faults, the algorithm will switch to the corresponding sub-schedule for S_2. The algorithm will attempt re-execute all affected time slices within its adjacent redundancy area by switching to the pertinent re-execution schedule. Any time slices that cannot be re-executed in the redundancy area will be added to G.

If the following checkpoint does not detect any faults, the algorithm will use its adjacent redundancy area to re-execute any remaining time slices in G. If there are no time slices awaiting re-execution, the partition in Q with the highest priority will be given access

to the resource and the idle time slices will be allocated to the next redundancy area in the schedule.

Fig. 8 depicts how the model would handle a fault on the first instance of P_3. The model remains in S_1 until the fault is detected on the last critical checkpoint. It then switches to the corresponding sub-schedule in S_2 and immediately resolves the fault by re-executing P_3 in its adjacent redundancy area. The next hyper-period follows the S_2 schedule, with additional checkpoints. Since no more faults are detected, the model skips the redundancy areas after each checkpoint and immediately grants access to the next partition on the schedule.

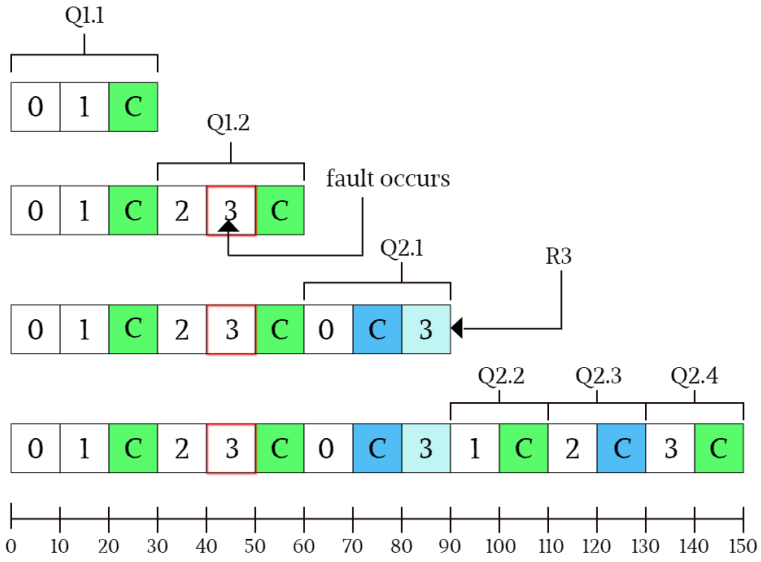

Figure 8: An example illustrating how the proposed model would detect and resolve a fault in P_3 during execution.

3.6 Time Complexity

Time complexity for the algorithm was determined based on an implementation of the Two-state Regularity-based Checkpointing model using the XtratuM hypervisor [8]. Due to the hypercalls available through this hypervisor, the model is able to perform many operations in negligible time.

The model generates major schedules for S_1 and S_2 using the Magic7 algorithm, which can be costly [7]. Fortunately, the model only needs to run the algorithm twice before execution. During execution, the model simply switches between sub-schedules to resolve faults, with no need to modify them or generate new ones. Prior implementations of fault-tolerant RRP [7] indicate that the cost of rescheduling when the redundancy area is not used during S_2 is $O(\frac{hp_2}{N_2})$. However, the XtratuM implementation further explained in Chapter 4 can reschedule in $O(1)$, as it does not modify sub-schedules to append unused time slices. Instead, the model simply switches to the next sub-schedule immediately, resulting in a shorter execution time for partitions in the set. In addition, XtratuM allows the model to perform schedule switching, check for faults and access the *G* queue in $O(1)$.

3.7 Potential Benefits

The Two-state Regularity-based Checkpointing model seeks to ensure time efficiency for a HiRTS system that handles transient faults the resource level. It will offer an adjustable trade-off between the degree of fault tolerance and resource utilization efficiency. The model is designed to schedule mixed-criticality task sets on limited computational

resources with minimal task-level information and employs a two-state checkpointing scheme for efficient fault resolution.

The RRP model provides transparent task scheduling, so applications can be added and removed from the system without modifying their code, simplifying implementation. As such, the model ensures that high-priority tasks within mixed-criticality sets maintain their real-time performance guarantees, even under a high fault rate. Benefits of this model include resource-level and task-level fault tolerance, low time overhead, high portability, and transparent task scheduling.

CHAPTER 4: Experimental Evaluation

4.1 Motivation

The purpose of our experimental simulations was to provide evidence of the model's validity. Firstly, we must prove that it is an implementable scheduling model for embedded real-time systems. Secondly, we must demonstrate that the model can ensure fault tolerance for mixed-criticality task sets even under a high fault rate. As previously mentioned, every task set is given an upper bound availability factor, an optimistic limit to be achieved if possible, and a lower bound availability factor, which is the minimum utilization required to satisfy the timing constraints of high-priority tasks. During execution, the model must maintain an average availability factor above the lower bound for all partitions, regardless of faults.

Lastly, we must show that the model can handle both a high overall fault rate for the partition set and a high fault rate for a single partition. That is, if one partition is "unlucky" and experiences a disproportionately high number of faults, it may lose its real-time performance guarantees. However, the model must ensure that other partitions in the set are not delayed as a result.

4.2 Experiment Setting

Case study simulations were performed using a Zybo Z7 development board. The board features a dual-core processor, but only one core was used. Fig. 9 depicts the usage of both cores on the test bed. All simulations were done using XtratuM [9], an open-source

hypervisor designed for embedded systems. XtratuM provides spatial and temporal isolation for independent task sets, as it ensures that partitions do not share memory and uses a fixed cyclic scheduler to manage their execution.

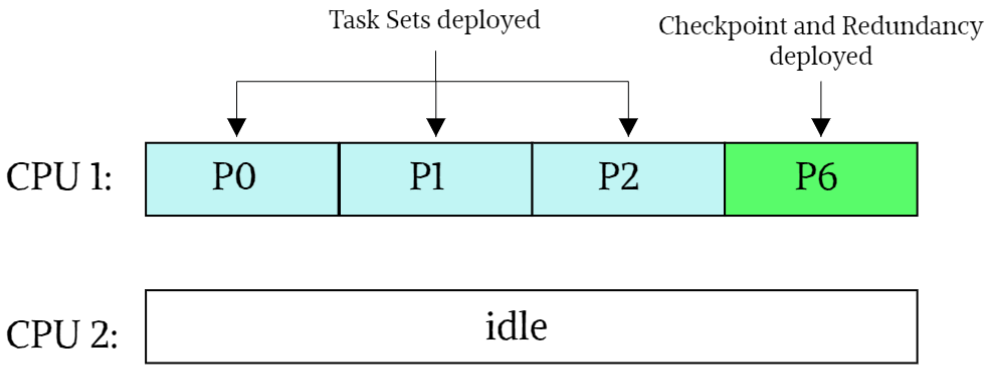

Figure 9: CPU usage on the test bed used for experimental simulations.

XtratuM initializes a set of regular partitions $P = \{P_i : 0 \leq i \leq 5\}$. Each partition represents an independent real-time mixed-criticality task set that requires 100ms of CPU time to complete execution. These partitions do not execute any actual tasks. Instead, they each contain a counter that keeps track of the execution time for the partition set, which they increase by one with every passing millisecond. When a partition has been given enough CPU time to finish execution and has resolved any faults through re-execution, it outputs the value of its counter.

In addition, XtratuM initializes a checkpoint redundancy regular partition P_6. This partition simulates a checkpoint by generating and displaying transient fault warnings in a predetermined pattern. It is also used for schedule switching. As explained in the

previous section, P_6 will be given more time slices during S_2 to account for time spent on re-execution.

The simulations performed do not encompass the offline portion of the Two-state Regularity-based Checkpointing model. As such, XtratuM receives all the sub-schedules required for execution prior to start time. Fig. 10 depicts the major schedules for the partition set in S_1 and S_2. Each time slice is given a length of 10 ms. Q_1 has a length of 80 ms, giving hp_1 a value of 8 time slices. Similarly, Q_2 has a length of 140 ms and hp_2 has a value of 14 time slices.

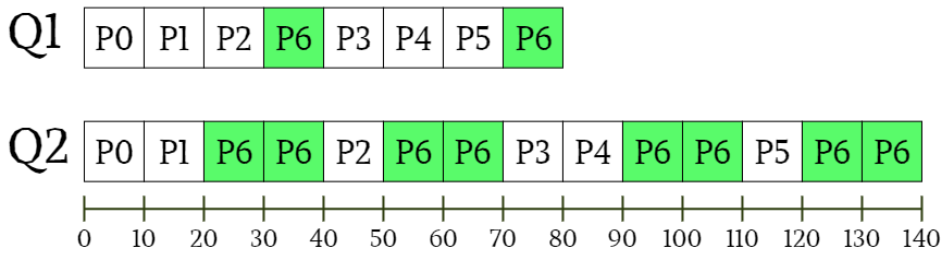

Figure 10: Major schedules used for experimental simulation.

The corresponding sub-schedules are shown on Fig. 11. They are referred to as scheduling plans. XtratuM can easily switch between them using a hypercall. Plans 1 and 2 represent the hyper-period during S_1 and contain fewer checkpoints with no adjacent redundancy areas. Plans 3, 4, 5 and 6 represent the hyper-period during S_2, while plans 7, 8, 9, 10, 11 and 12 are used to simulate re-execution. Re-execution plans have a length of 10ms, as P_6 is only used to switch back to the regular execution schedule, which can be done in around 0.1 milliseconds with XtratuM [9].

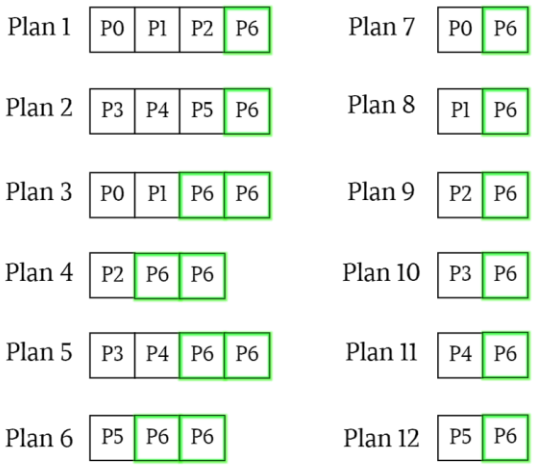

Figure 11: Plans used for experimental simulation. Each one represents a sub-schedule for the partition set.

As previously mentioned, each regular partition is mapped to a mixed-criticality task set. High-priority tasks within each set require a minimum availability factor of $\frac{1}{14}$ in order to maintain real-time performance guarantees. Likewise, with minimal checkpoint insertions and under the assumption that no faults will occur, each partition can expect an availability factor of $\frac{1}{6}$. This is because XtratuM can simulate checking for transient faults in very little time and if no faults are detected, the model immediately switches to the next sub-schedule. As a result, most of the time reserved for P_6 is not used. In addition, when no faults occur the model will only iterate between plans 1 and 2, minimizing the time spent on schedule switching. Therefore, when no faults occur XtratuM only spends a cumulative 10ms on P_6 during execution. If the actual observed availability of a partition remains within the range $[\frac{1}{14}, \frac{1}{6}]$, it can be said that it maintains real-time performance guarantees for its critical tasks.

Another important variable is the *critical number* of faults. This value represents the maximum number of time slices that can be re-executed during the 10 hyper-periods needed complete execution for the partition set. Since the maximum number of redundancy areas per hyper-period is 4, which corresponds to the Q_2 schedule, the critical number for this partition set is 40. If the number of faults exceeds this value, the model will have to carry faulty partitions from one hyper-period to the next through the queue *G*, which may result in delays.

The simulations performed are divided into two groups: A and B.

Group A encompasses 7 different cases in which the system experiences between 4 and 46 faults distributed among tasks. Table 1 describes each case, including the total number of faults and which partitions they affected. In addition, each row indicates which time slices were affected for each partition. The row corresponding to Case 1, for example, shows P_3, P_4 and P_5 experienced faults during the fourth hyper-period. P_4 experienced an addition fault during the fifth hyper-period.

This group of simulations determine whether the model can ensure fault tolerance and real-time performance guarantees for high-priority tasks when the number of faults exceeds the critical number. In addition, they demonstrate how the time spent on checkpointing and schedule switching increases as faults grow more numerous.

Table 2: Fault distribution for simulations in Group A. The partition set experienced between 4 and 46 faults across all partitions.

Case	Faulty Time Slices						
	P_0	P_1	P_2	P_3	P_4	P_5	Total Faults
1				[40]	[40, 50]	[40]	4
2		[30]	[30]	[30]	[40, 50]	[40]	6
3		[20]	[30]	[40]	[40, … , 60]	[40]	7
4	[20]	[30]	[30]	[30]	[40, … , 60]	[40]	8
5	[20]	[30]	[30]	[30]	[40, … , 80]	[40]	10
6	[10, … 70]	[10, … , 70]	[10, … ,90]	[10, … ,90]	[10, … ,90]	[40]	42
7	[10, … ,90]	[10, … ,90]	[10, … ,90]	[10, … ,90]	[10, … ,90]	[40]	46

Group B encompasses 8 different cases. The system experiences up to 7 faults during these simulations, but they all happen exclusively to P_0. As shown on Table 2, P_0 will begin experiencing faults on its first time slice, immediately switching to S_2.

Table 1: Fault distribution for simulations in Group B. The partition set experienced between 0 and 7 faults in P_0.

Case	Faulty Time Slices	
	P_0	Total Faults
1		0
2	[10]	1
3	[10, 20]	2
4	[10, … , 30]	3
5	[10, … , 40]	4
6	[10, … 50]	5
7	[10, … ,60]	6
8	[10, … , 70]	7

The purpose of this group of simulations is to determine if the system can resolve faults for an “unlucky” partition without losing real-time performance guarantees for its high-priority tasks or causing delays to other partitions. In addition, the results will allow us to compare the performance of the model when it experiences faults across several partitions against the performance when the model experiences the same number of faults within a single partition.

It must be noted that the settings of these simulations will affect the quality of the results. The simulations were designed under the assumption that faulty time slices will fully recover after being re-executed. In addition, both schedule switching and checkpointing can be done at very little cost with XtratuM, which may not apply to implementations in different platforms. The lower and upper bound for the availability factor for the partition set were chosen arbitrarily, as partitions are not executing any real tasks. Furthermore, a more realistic simulation would generate faults randomly, rather than in a pre-determined pattern.

4.3 Results for Group A

The results for Group A met all performance expectations for the model. As shown on Table 3, the maximum expected execution time was never exceeded, even when faults exceeded the critical number.

Table 3: Ending time for each partition during the simulations in Group A.

Case	Ending Time (*ms*)					
	P_0	P_1	P_2	P_3	P_4	P_5
1	645.653	645.693	645.828	645.811	645.88	645.996
2	666.463	666.534	666.659	666.64	666.696	666.816
3	676.957	676.998	677.133	677.116	677.191	677.285
4	687.157	687.208	687.323	687.322	687.398	687.482
5	707.479	707.525	707.65	707.641	707.743	707.803
6	961.921	961.969	992.67	1063.446	1073.593	1083.789
7	1063.268	1063.309	1073.593	1083.756	1083.773	992.658

Exceeding the critical number of faults resulted in delays for heavily affected partitions, but less affected partitions still terminated within a reasonable time frame. As expected, the model performed very efficiently during the first 5 cases.

During Case 3, for example, the model experienced 7 total faults. Fig. 12 shows the ending time for each partition, as well as the upper and lower bounds for execution time. The upper bound corresponds to a total partition set execution time of 1400ms, which would give each partition an availability factor of $\frac{1}{14}$. Similarly, the lower bound corresponds to a total execution time of 600ms and an availability factor of $\frac{1}{6}$.

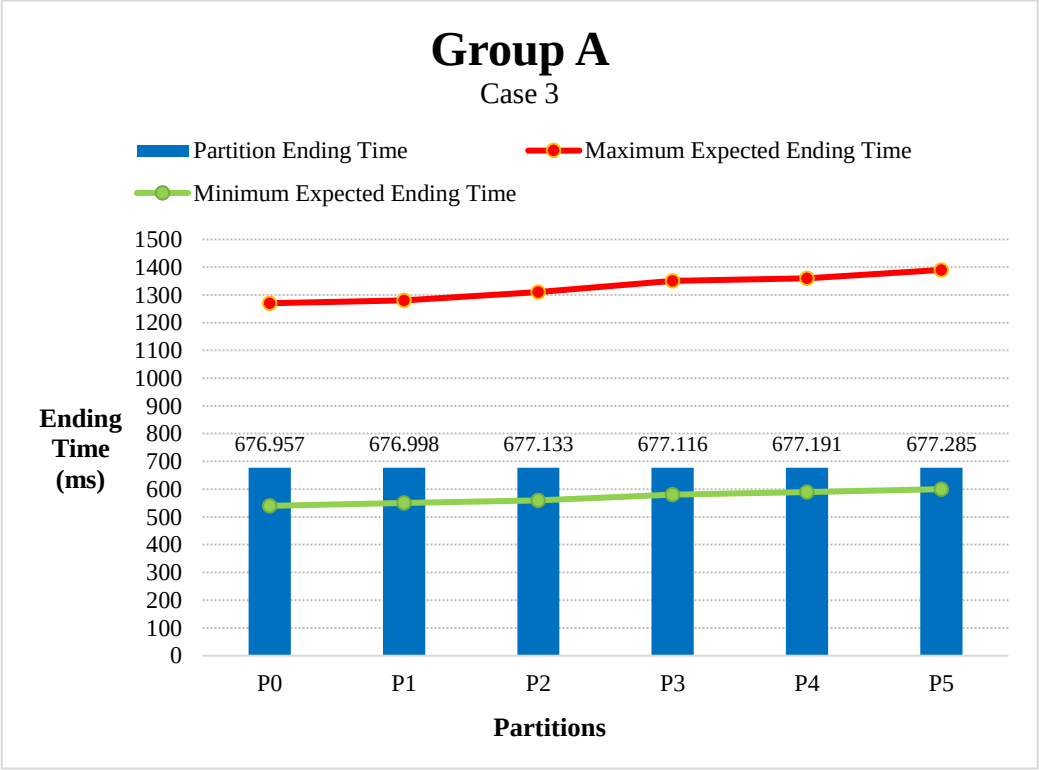

Figure 12: Results for Case 3 in Group A. The partition set experienced 7 faults which affected P_1, P_2, P_3, P_4 and P_5.

As shown, all partitions in the set terminated in the same order they began execution and had an observed availability factor of $\frac{100}{677} \approx \frac{1}{7}$. Given that the model spends 600ms executing the partition set and 70ms re-executing faulty time slices, the cost of checkpointing and schedule switching is only 7.3ms or around 10% of the time spent on re-execution. This ratio remained true in all cases where the number of faults remained below the critical number, as shown in Table 3. While the low cost of checkpointing in this particular implementation makes it difficult to measure what the realistic overhead savings would be, the model is easily able to handle faults after switching to S_2. In addition, P_4 suffered no significant delays, despite being affected by the most faults.

During Case 6, however, the fault occurrence began impacting the performance of the model. The number of faults exceeded the critical number, which means that the model had to carry over faulty time slices from one hyper-period to the next to re-execute them. Five different partitions experienced faults in the first hyper-period and continued to experience faults up to the seventh during this case, as shown in Table 1. Despite this, all partitions remained within the acceptable bounds for execution time, with an observed availability factor of $\frac{100}{1083} \approx \frac{1}{11}$. Furthermore, all partitions terminated in the right order. However, there was an increase in overhead from checkpointing. In addition to the 600ms used to execute partitions, the model spent a total of 420ms re-executing faulty time slices and 63ms on checkpointing and schedule switching. This indicates a decrease in performance as prior simulations did not spend more than 10% of the time spent in re-execution on checkpointing and switching schedules.

As shown on Fig. 13, while the partitions did not experience any delays, execution time increased significantly. The ending times for partitions P_2, P_3 and P_4 were significantly more spread out than in prior simulations. Since the number of faulty time slices per hyper-period exceeded the number of available redundancy areas, partitions had to wait significantly longer to re-execute. Still, the model was able to ensure fault tolerance and real-time performance guarantees for high-priority tasks in all partitions.

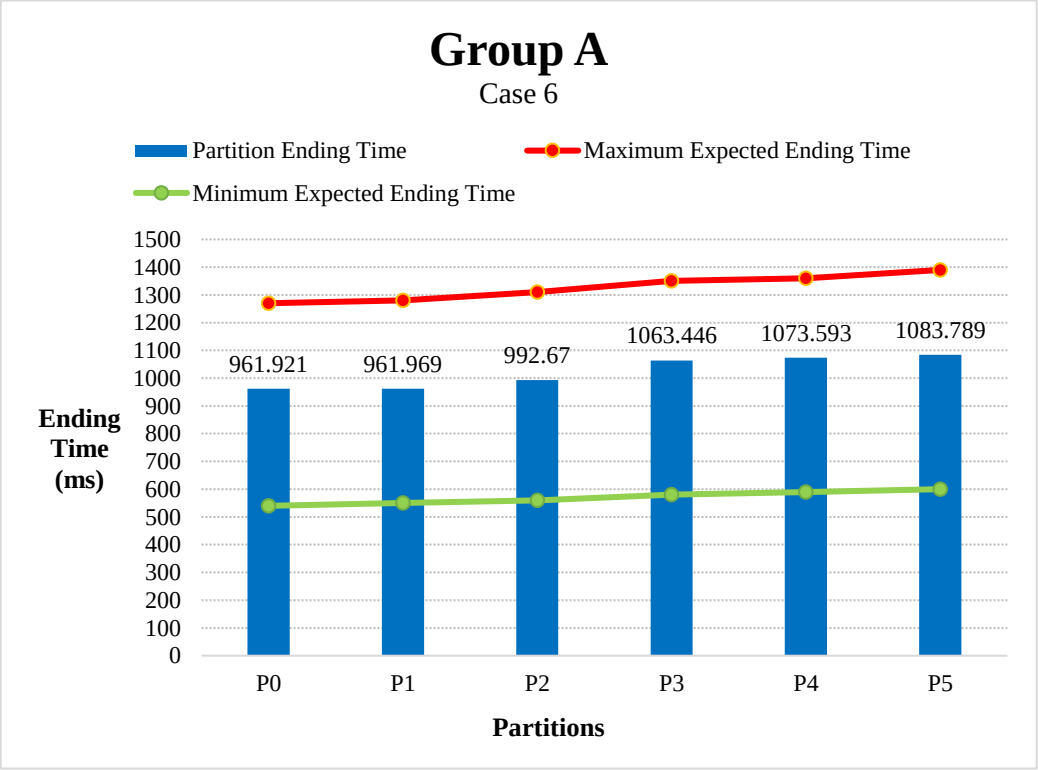

Figure 13: Results for Case 6 in Group A. The partition set experienced 42 faults which affected all partitions.

Despite only experiencing 4 additional faults, partitions begin experiencing serious delays in Case 7. While execution time remains within the established bounds, partitions begin to terminate in the wrong order and the cost of checkpointing and schedule switching increases significantly.

As shown on Fig. 14, while the partition set maintained the same total execution time and observed availability factor as in Case 6, P_5 finished executing first despite being deployed last. Because P_5 is only affected by one fault, it does not need to wait for access to the redundancy area to terminate as the other partitions do. This demonstrates that even in situations where the number of faults exceeds the critical number, only partitions

affected by faults will experience delays and loss of their real-time performance guarantees. As such, it can be said that the model can provide high fault tolerance and independent scheduling for mixed-criticality task sets.

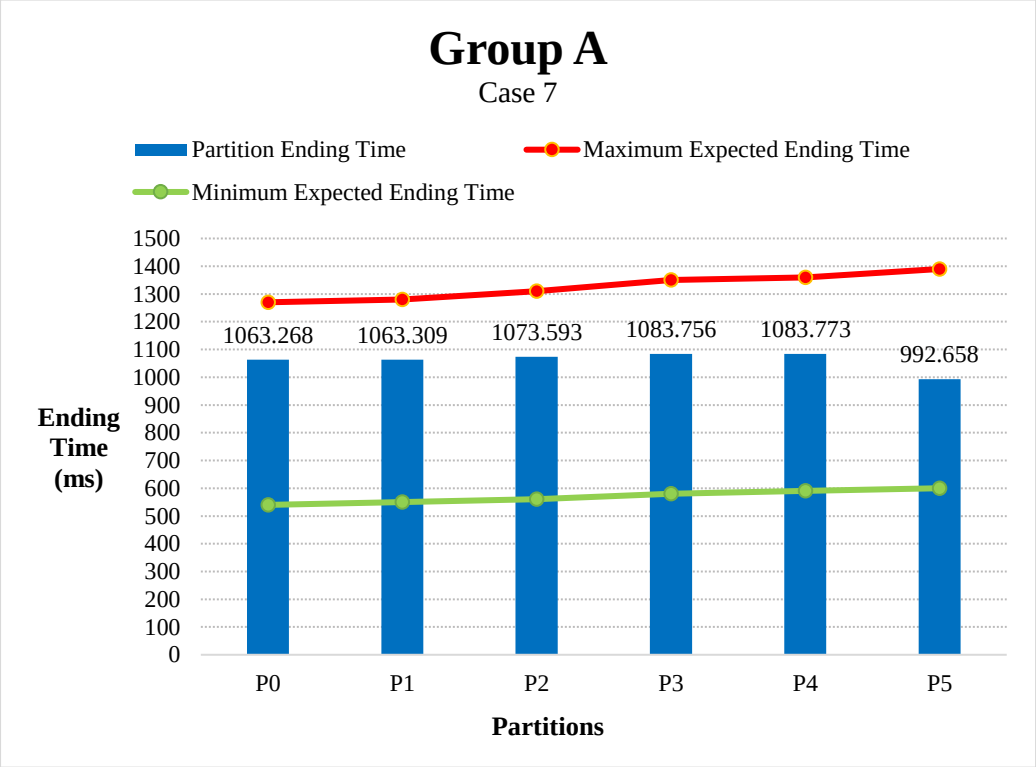

Figure 14: Results for Case 7 in Group A. The partition set experienced 46 faults which affected all partitions.

4.4 Results for Group B

The results for Group B were within expectations as well. As shown on Table 4, partitions stayed within the acceptable bounds for execution time and terminated in the right order in every case. The model was able to handle faults within one partition just as well as faults happening among several partitions, as the redundancy area is efficiently utilized.

Table 4: Ending time for each partition during the simulations in Group B.

Case	Ending Time (*ms*)					
	P_0	P_1	P_2	P_3	P_4	P_5
1	603.366	603.402	603.42	603.367	603.399	603.411
2	614.52	614.556	614.707	614.675	614.731	614.859
3	625.322	625.367	625.506	625.479	625.555	625.661
4	635.481	635.527	635.658	635.635	635.726	635.82
5	645.967	646.013	646.143	646.127	646.229	646.303
6	656.449	656.492	656.621	656.6	656.725	656.78
7	666.936	666.998	667.109	667.101	667.228	667.263
8	677.1	677.148	677.258	677.252	677.392	677.424

As shown on Table 1 and Table 2, the partition set experienced 4 total faults in Case 1 of Group A and Case 5 of Group B. The model only spent an additional 1ms execution in Group B. This is likely due to the system entering S_2 on the first hyper-period in Group B, while the first fault in Group A did not occur until the fourth. Similarly, the partition set experienced the same number of faults in cases 2 and 3 of Group A and cases 7 and 8 of Group B. The 1ms difference remained consistent for these cases as well. In all cases the model maintained a similar overhead from checkpointing and schedule switching, around 10% of time spent on re-execution. In addition, all partitions terminated in the right order and fairly close to each other.

When the system experienced no faults, partitions came very close to the lower bound for execution time, with an observed availability factor of $\frac{100}{603} \approx \frac{1}{6}$. In addition, the model only spent 3.4ms checkpointing. This is because the system never entered S_2. As such, the model only had to switch between plan 1 and plan 2, making the overhead from schedule switching minimal. Overall, it is reasonable to expect that the model can handle a partition with faults on every time slice, as the S_2 schedule provides enough redundancy to re-execute the one time slice given to each partition per hyper-period. As shown in Fig. 15, partitions terminated near-simultaneously in every case and overall, the availability factor of the partition set never fell bellow $\frac{1}{7}$.

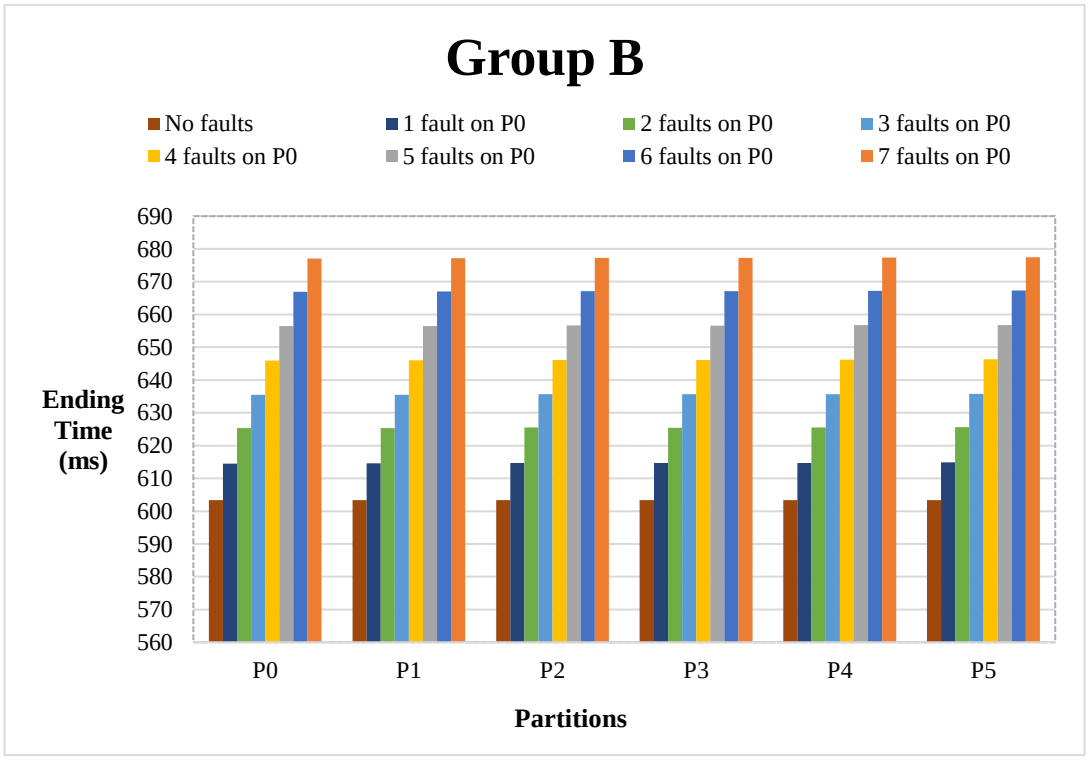

Figure 15: Results for all cases in Group B. The partition set experienced between 0 and 7 faults in P_0.

It must be noted that the schedule used to test the model had the caveat of only having one instance of each partition per hyper-period. Results with a partition set containing a regular partition with much higher resource needs than the others could result in delays under situations with a high fault rate. Simulations in future work may be introduced to address the model's behavior and performance during such situations.

CHAPTER 5: Conclusions and Future Work

The main goal of this thesis is to introduce a fault-tolerant, low-overhead solution for mixed-criticality task sets on embedded virtualized real-time systems. The Two-state Regularity-based Checkpointing model enables resource virtualization and distribution among task sets through hierarchical scheduling. The RRP model ensures transparent scheduling and balanced resource distribution, making the addition and removal of applications from the system simple. In addition, the model implements a two-state resource-level checkpointing scheme to ensure transient fault resolution. Prior to the first occurrence of a fault, the system will perform less checkpoint insertions and will not reserve time slices for redundancy. After the first fault, the model will follow a new schedule with additional checkpoints, each one with an adjacent redundancy area. By keeping checkpoints on the resource level, the model can provide fault tolerance with minimal task-level information and without modifying the applications' code.

Experimental simulations performed on a Zybo Z7 development board with the XtratuM hypervisor demonstrated that the model can ensure efficient scheduling and fault tolerance in a variety of situations. The simulations performed in Group A showed that as long as the number of faults experienced does not exceed the critical number, the partitions experience no delays and are able to achieve a high level of utilization. In addition, when the system exceeded the critical number of faults, less affected or unaffected partitions did not experience delays and no partition lost real-time performance guarantees for its critical tasks. The simulations performed in Group B proved that the model is easily able to handle a single unlucky partition, as P_0 did not experience delays even when seven of the ten time

slices assigned to it experienced faults. Thus, the model was able to ensure fault tolerance even with less checkpoint insertions.

As such, the model has been proven to be implementable. It will provide fault-tolerance and maintain real-time performance guarantees for high-priority tasks in mixed-criticality applications. Even under high fault rates, the model can maintain an observed availability factor within the necessary bounds for each partition. Future work will focus on measuring the actual time and energy overhead reductions provided by the model, specifically when compared to fault-tolerant RRP models that do not implement two-state checkpointing. This will be calculated using a more realistic cost for checkpointing, as the implementation on XtratuM was able to perform checkpoints in very little time. In addition, future work will measure model performance when fault occurrence is generated randomly or according to distributions in specific application domains rather than pre-determined. Finally, the implementability of the model will be confirmed by measuring the time complexity of its algorithm when the offline portion is included in execution.

BIBLIOGRAPHY

[1] K. Agrawal, S. Baruah and A. Burns, "Fault-tolerant Transmission of Messages of Differing Criticalities Across a Shared Communication Medium," *27th International Conference on Real-Time Networks and Systems (RTNS)*, Toulouse, France, pp. 41-49, 2019.

[2] M. Barr, "Embedded Systems Glossary," *Barr Group*, 11-Apr-2020. [Online]. Available: https://barrgroup.com/embedded-systems/glossary. [Accessed: 27-Mar-2021].

[3] S. Baruah, "Certification-cognizant Scheduling of Tasks with Pessimistic Frequency Specification," *7th IEEE International Symposium on Industrial Embedded Systems* (SIES), Karlsruhe, pp. 31-38, 2012.

[4] S. K. Baruah, A. Burns and R. I. Davis, "Response-time Analysis for Mixed Criticality Systems," *32nd IEEE Real-Time Systems Symposium*, Vienna, pp. 34-43, 2011.

[5] K. M. Chandy, "A Survey of Analytic Models of Rollback and Recovery Strategies," *Computer*, vol. 8, no. 5, pp. 40–47, 1975.

[6] K. M. Chandy, J. C Browne, C. W. Dissly, and W. R. Uhrig. "Analytic Models for Rollback and Recovery Strategies in Data Base Systems," *IEEE Transactions on Software Engineering*, vol. SE-1, no. 1, pp. 100-110, March 1975.

[7] A. M. K. Cheng, G. Dai, P. K. Paluri, M. Ansari, D. Knape and Y. Li, "Fault-tolerant Regularity-based Real-time Virtual Resources," *25th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications* (RTCSA), Hangzhou, China, pp. 1-12, 2019.

[8] J. Choi, H. Yang, and S. Ha, "Optimization of Fault-tolerant Mixed-criticality Multi-core Systems with Enhanced WCRT Analysis," *ACM Transactions on Design Automation of Electronic Systems*, vol. 24, no. 1, pp. 1-26, January 2019.

[9] A. Crespo, I. Ripoll and M. Masmano, "Partitioned Embedded Architecture Based on Hypervisor: The XtratuM Approach," *European Dependable Computing Conference*, Valencia, Spain, pp. 67-72, 2010.

[10] Z. Deng, J. W. Liu and J. Sun, "A Scheme for Scheduling Hard Real-time Applications in Open System Environment," *Proceedings Ninth Euromicro Workshop on Real Time Systems*, Toledo, Spain, pp. 191-199, 1997.

[11] Z. Deng and J. W. Liu, "Scheduling Real-time Applications in an Open Environment," *Proceedings Real-Time Systems Symposium*, San Francisco, CA, USA, pp. 308-319, 1997.

[12] A. Easwaran, M. Anand and I. Lee, "Compositional Analysis Framework Using EDP Resource Models," *28th IEEE International Real-Time Systems Symposium (RTSS)*, Tucson, AZ, pp. 129-138, 2007.

[13] P. Ekberg and W. Yi, "Bounding and Shaping the Demand of Generalized Mixed-criticality Sporadic Task Systems," *24th Euromicro Conference on Real-Time Systems* (ECRTS), Pisa, Italy, pp. 48–86, 2014.

[14] J. Hyun and K. H. Kim, "Fault-tolerant Scheduling in Hierarchical Real-time Scheduling Framework," *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, Seoul, pp. 431-436, 2012.

[15] J. Hyun, S. Lim, Y. Park, K. S. Yoon, J. H. Park, B. M. Hwang, and K. H. Kim. "A Fault-tolerant Temporal Partitioning Scheme for Safety-critical Mission Computers," *31st IEEE/AIAA Digital Avionics Systems Conference* (DASC), Williamsburg, VA, pp. 1-24, 2012.

[16] B. Javadi, D. Kondo, A. Iosup and D. Epema, "The Failure Trace Archive: Enabling the Comparison of Failure Measurements and Models of Distributed Systems," *Journal of Parallel and Distributed Computing*, vol. 73, no. 8, pp. 1208-1223, 2013.

[17] H. W. Jin, "Fault-tolerant Hierarchical Real-time Scheduling with Backup Partitions on Single Processor," *ACM SIGBED Review*, vol. 10, no. 4, pp. 25-28, December 2013.

[18] T. Kadeed, B. Nikolic and R. Ernst, "Safe Online Reconfiguration of Mixed-criticality Real-time Systems," *IEEE 25th Pacific Rim International Symposium on Dependable Computing (PRDC)*, pp. 140-149, 2020.

[19] Y. Li and A. M. K. Cheng, "Static Approximation Algorithms for Regularity-based Resource Partitioning," *33rd IEEE Real-Time Systems Symposium*, San Juan, pp. 137-148, 2012.

[20] Y. Li and A. M. K. Cheng. "Toward a Practical Regularity-based Model: The Impact of Evenly Distributed Temporal Resource Partitions," *ACM Transactions on Embedded Computing Systems* (TECS), vol. 16, no. 4, pp. 111, 2017.

[21] Y. Li and A. M. K. Cheng, "Transparent Real-time Task Scheduling on Temporal Resource Partitions," *IEEE Transactions on Computers*, vol. 65, no. 5, pp. 1646-1655, May 2016.

[22] J. Lin, A. M. K. Cheng, D. Steel, M. Y. C. Wu and N. Sun, "Scheduling Mixed-criticality Real-time Tasks in a Fault-tolerant System," *International Journal of Embedded and Real-Time Communication Systems* (IJERTCS), vol. 6, no. 2, 2016.

[23] A. K. Mok, X. Feng and D. Chen, "Resource Partition for Real-time Systems," *Proceedings Seventh IEEE Real-Time Technology and Applications Symposium*, Taipei, Taiwan, pp. 75-84, 2001.

[24] A. K. Mok and X. Feng, "Towards Compositionality in Real-time Resource Partitioning Based on Regularity Bounds," *22nd IEEE Real-Time Systems Symposium* (RTSS 2001), London, UK, pp. 129-138, 2001.

[25] S. Muhammad Abrar Akber, H. Chen, Y. Wang and H. Jin, "Minimizing Overheads of Checkpoints in Distributed Stream Processing Systems," *7th IEEE International Conference on Cloud Networking* (CloudNet), Tokyo, pp. 1-4, 2018.

[26] J. Niu, C. Liu, Y. Gao and M. Qiu, "Energy Efficient Task Assignment with Guaranteed Probability Satisfying Timing Constraints for Embedded Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 8, pp. 2043-2052, August 2014.

[27] P. K. Paluri, G. Dai, A. M. K. Cheng, "ARINC 653-Inspired Regularity-based Resource Partitioning on Xen," *22nd ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems (LCTES)*, June 2021.

[28] M. Salehi, M. Khavari Tavana, S. Rehman, M. Shafique, A. Ejlali and J. Henkel, "Two-state Checkpointing for Energy-efficient Fault Tolerance in Hard Real-time Systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 7, pp. 2426-2437, July 2016.

[29] I. Shin and I. Lee, "Periodic Resource Model for Compositional Real-time Guarantees," *24th IEEE Real-Time Systems Symposium*, Cancun, Mexico, pp. 2-13, 2003.

[30] S. Shirero, M. Takashi and H. Kei, "On the Schedulability Conditions on Partial Time Slots," *Proceedings Sixth International Conference on Real-Time Computing Systems and Applications* (RTCSA), Hong Kong, China, pp. 166-173, 1999.

[31] G. M. Tchamgoue, J. Seo, J. Hyun, K. H. Kim and Y. Jun, "Supporting Fault-tolerance in a Compositional Real-time Scheduling Framework," *ACM SIGBED Review*, vol. 12, no. 2, pp. 7-15, 2015.

[32] S. Vestal, "Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance," *28th IEEE International Real-Time Systems Symposium* (RTSS), Tucson, AZ, pp. 239-243, 2007.

[33] K. Yang and Z. Dong, "Mixed-criticality Scheduling in Compositional Real-time Systems with Multiple Budget Estimates," *IEEE Real-Time Systems Symposium* (RTSS), pp. 25-37, 2020.

[34] Y. W. Zhang, "Energy-Aware Mixed-criticality Sporadic Task Scheduling Algorithm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 1, pp. 78-86, January 2021.

[35] X. Zhao, Y. Wei and W. Li, "The Improved Earliest Deadline First with Virtual Deadlines Mixed-criticality Scheduling Algorithm," *IEEE International Symposium on Parallel and Distributed Processing with Applications*, Guangzhou, China, pp. 444-448, 2017.

[36] P. Zhu, D. Luo and X. Chen, "Fault-tolerant and Power-aware Scheduling in Embedded Real-time Systems," *International Conference on Computer, Information and Telecommunication Systems* (CITS), Hangzhou, China, pp. 1-5, 2020.