2023 Design, Automation & Test in Europe Conference (DATE 2023)
Special Initiative "Autonomous Systems Design"

Formal Analysis of Timing Diversity for Autonomous Systems

Anika Christmann, Robin Hapka, Rolf Ernst
{christmann, hapka, ernst}@ida.ing.tu-bs.de
Institute of Computer and Network Engineering
Technische Universität Braunschweig
Germany

***Abstract*—The design of autonomous systems, such as for automated driving and avionics, is challenging due to high performance requirements combined with high criticality. Complex applications demand the full performance of commercial high performance multi-core systems of-the-shelf (COTS), with or without accelerators. While these systems are optimized for performance, hard real-time requirements and deterministic timing behavior are major constraints for safety-critical systems. Unfortunately, infrequent timing outliers caused by interleaved hardware-software effects of COTS systems complicate traditional worst-case design. This conflict often prohibits deploying COTS hardware and consequently prevents sophisticated applications, too. Recently, an approach called Timing Diversity was introduced, which proposes to exploit existing dual modular redundant hardware platforms to mask deadline violations. This paper puts Timing Diversity on a theoretical foundation and provides specification for different implementations. It demonstrates that Timing Diversity needs fast recovery to be effective, proposes a recovery strategy and provides a mathematical model for the reliability of the resulting system. Using experimental data in a Linux based system, it shows that fast recovery is useful, making Timing Diversity a realistic option for compute demanding hard real-time applications.**

I. Introduction

In autonomous systems, the complexity and number of system applications running on a hardware platform are constantly increasing. Especially vision-based applications, e.g., visual landing guidance in avionics or lane departure and lane keeping assist, traffic light and traffic sign detection in automated driving, require large processing capabilities. Often, only modern COTS multi-/many-core processors are able to process camera frames sufficiently fast to achieve reasonable frame rates. Fast frame rates and low latencies are required by control systems and improve the quality of motion control, therefore timing constraints are imposed. Violation of expected timing can cause failures in motion control, resulting in fatal damage to passengers and goods. Apparently, real-time constraints are crucial for such autonomous systems.

A drawback is that the timing behavior of modern COTS hardware platforms is not reliable [1]. Due to shared resources inherent to multi-core processors in combination with multiple applications or a single application featuring multiple threads, resource contention occurs. This can prolong the execution time, thus violating deadlines and affecting the system's real-time capability, consequently jeopardizing commercial off-the-shelf (COTS) hardware deployment in safety-critical systems.

The real-time community suggests predicting or restricting the worst-case execution time (WCET). But, both have been major challenges for WCET analysis [2] [3]. The first proposes often overly pessimistic timings and the latter accepts huge performance penalties for deterministic timing. In consequence, neither predicting nor restricting is simple or convenient.

Recently, an alternative approach based on modular redundancy has been presented, called *Timing Diversity* [4], [5]. It executes an application on two hardware platforms simultaneously and assumes diverse worst-case timing behavior among the platforms. Hence, a deadline violation of one platform can be corrected, if the other platform finishes the calculation in time. Although, proof of diverse worst-case timing behavior has been presented [4], a theoretical foundation and a sound system model for its impact on reliability are still missing.

In this paper we present and analyze design principles of a system exploiting Timing Diversity and give insight into the compatibility with existing dual modular redundancy (DMR) systems. We identify open issues and propose a solution including application to an example.

In Sec. II, the paper starts with an overview of related work and state-of-the-art approaches for fault-tolerance and deploying multi- and many-core COTS hardware in safety-critical systems. The system model is introduced in Sec. III and in Sec. IV the compliance with existing modular-redundant implementation is shown. Followed by Sec. V, which discusses the impact of different failures and analytically shows whether recovery is possible, which is completed by an example in Sec VI. At last, a conclusion is drawn and outlook presented in Sec. VII.

II. Related Work and Background

In avionics, as a representative of safety-critical industry, strict guidance for design and implementation exists, which is issued by certification authorities, e.g., the European Union Aviation Safety Agency (EASA). In their document named *AMC 20-193* they deal with the usage of multi-core processors. Among other aspects, they address challenges arising from complex timing by proposing to determine WCET analytically [6, p. 669]. For determining WCETs, nowadays, measurement-based approaches are state-of-the art, because static timing analysis suffers from the high complexity and low availability of sufficiently precise processor models [2]. As a drawback, the

978-3-9819263-7-8/DATE23/© 2023 EDAA

number of measurements required for high evidence is significant. Although, the effort can be reduced using extreme value theory (EVT) [3], critics claim the method to be immature, still [7], [8].

Determining a WCET under final configuration with all software components running, is complicated and needs to be repeated for every update. Regardless of whether the system is a homogeneous multi-core hardware architecture or a hardware architecture with accelerators, all face similar challenges by determining WCETs as discussed in [4]. Therefore by guidance the need of mitigating interference arises. Often memory access is considered the most important cause of interference, hence memory bandwidth allocation is a favored mitigation technique [9]–[13]. Alternatively, deterministic and predictable execution models are proposed [14]–[16]. Unfortunately, both approaches suffer from substantial performance penalties.

Motivated by the traditional fault-tolerant system design, the Timing Diversity approach was introduced in [4] [5], which does not mitigate interference on a single platform, but preclude deadline violations at a compound system based on dual modular redundancy. In [4] the statistical independence of timing overshoots has been investigated, presented experimental results underpin that even identical systems do not exhibit simultaneous deadline violations. Moreover, in combination with hardware and software diversity common cause failures will become even more unlikely. Consequently, modular redundancy could be deployed to mask timing outliers. As a result, the WCET can be lowered, the probability of keeping a given deadline increased or the number of measurements reduced for an identical confidence level. But, the presented work [4] lacks a design specification, how to exploit Timing Diversity concretely to achieve the proposed benefits. Therefore, we introduce our system model and investigate the feasibility of Timing Diversity for safety-critical autonomous systems.

III. System Model

Consider a system model running a single application A featuring a set of tasks τ defined as $\tau = \{\tau_1, \tau_2, \cdots, \tau_i\}$. Each task is activated according to its period T_i. A task activation is called job and the jth activation of task τ_i is denoted as job $\tau_{i,j}$. Each task is constrained by its relative task deadline D_i which is given by the system specifications. Unlike in classical system models, we define $\overline{C_i}$ as the WCET of a task, C_i is the average execution time and $C_{i,j}$ corresponds to the execution time of the jth job of task τ_i. The worst-case response time (WCRT) of task τ_i is referred to as $\overline{RT}_i$. Furthermore, we view each task and job as a tuple of its properties: $\tau_i = (\overline{C}_i, T_i, D_i)$ and $\tau_{i,j} = (C_{i,j})$.

Since we are considering a DMR system, i.e., a system with two hardware platforms, we account for differences in these by writing a channel index Ch as superscript. Here, $Ch \in \{A, B\}$, where channel A, for short, Ch A and Ch B are denoted as identifiers for the hardware platforms. As a result, we formulate $\tau_i^{Ch} = (\overline{C}_i^{Ch}, T_i, D_i)$ and job $\tau_{i,j}^{Ch} = (C_{i,j}^{Ch})$ for $Ch \in \{A, B\}$. Each hardware platform, i.e. channel, may deploy a different hardware or exhibit a different software configuration. Therefore the WCRT, respectively the WCET, as well as job's execution time depend heavily on it. If both channels are identical, it causes the WCRT per channel to equal each other: $\overline{RT}_i^A = \overline{RT}_i^B = \overline{RT}_i$. Same holds true for the WCET with $\overline{C}_i^A = \overline{C}_i^B = \overline{C}_i$. But still, the job's execution times for different channels equal each other only by pure chance, if at all, thus $C_{i,j}^A \neq C_{i,j}^B$ [4]. In contrast, timing constraints such as task period T_i and task deadline D_i are predefined by system specification and therefore do not depend on the channel. Moreover, we omitted defining a task set τ^{Ch} for each channel, because we are considering only those tasks running on both channels.

At last, we define the reliability for each channel at time t as $R^{Ch}(t)$, which is the probability that the given channel operates without a failure in time interval $[0, t]$. Conversely, $P^A(x)$ is the probability of an error x in Ch A and $P^B(x)$ for Ch B, where $R^{Ch}(t) = 1 - P^{Ch}(x)$.

We assume the existence of an application- or system-specific *safety layer*. Such is inherent to safety-critical systems and often implemented in software. It performs emergency means, if the system exhibits an unexpected behavior. Furthermore, we consider the loss of jobs as catastrophic and require all jobs to be processed in order. Therefore, jobs exceeding their task period T_i will cause job queuing. For simplicity, we further assume implicit deadlines, i.e., $T_i = D_i$.

IV. Compliance to Dual Modular Redundancy

Active dual modular redundancy is already used to detect and correct transient hardware and software errors. Given a hardware and software diverse redundant system implementation, then it can be assumed that the channels are mutually independent and the error probability of a dual channel system can be calculated with $P^{A|B}(x) = P^A(x) \cdot P^B(x)$. Due to the low error probability and the assumption of mutually independent channels, the single-error model can be applied. Within such a model, it is unlikely that an error will occur on both redundant channels simultaneously.

Dynamic effects of the software architecture and the platform control cause rare but substantial worst-case timing outliers as has been shown in [4]. Based on Fig. 1, note that the occurrence of an error in a double system is also approximately statistically independent (compare red line and black-dotted line). Therefore, it is possible to exploit the properties of modular redundancy to decrease the timing error probability. The concept is not exclusive to DMR implementations only and can be applied to N-modular redundancy as well. But in this paper, we focus on DMR and introduce two different types of deadlines to be compatible with an existing redundant system implementation (cf. Fig 2).

Definition 1 (Recoverable Deadline): The relative and recoverable deadline $D_{R,i}$ depends on the specification of task τ_i and corresponds to the task deadline D_i. Therefore, it is applied to Ch A and Ch B. At $D_{R,i}$ at least one channel must provide the result to continue normal operation.

Definition 2 (Non-Recoverable Deadline): The relative, non-recoverable deadline $D_{NR,i}^{Ch}$ is set at a time after the recoverable

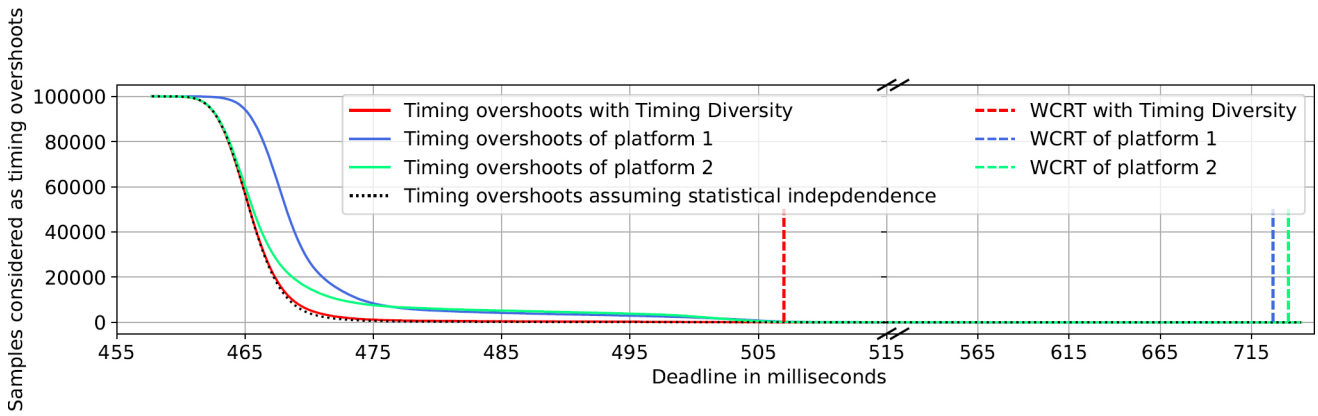

Fig. 1: Deadline-distribution taken from [4]. It shows the number of timing overshoots by setting an arbitrary deadline. The dotted, black line indicates the number of timing overshoots per deadline under the assumption of statistical independence. Since the red line and the black line are very similar, you can assume timing errors to be approximately independent.

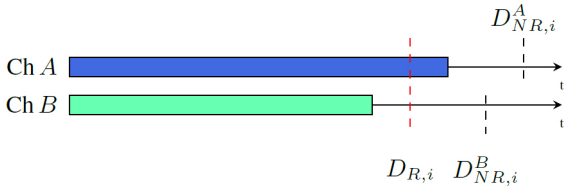

Fig. 2: DMR system model with different deadlines

$D_{R,i}$ and may be different for Ch A and Ch B channel. If one channel exceeds $D_{R,i}$ and $D^{Ch}_{NR,i}$, the system is immediately switched to the safety layer.

Accordingly, we adapt our software model: The task's deadline D_i corresponds to the recoverable deadline $D_{R,i}$ and the non-recoverable deadline per channel $D^{Ch}_{NR,i}$ is added. Thus, τ_i is characterized by $\tau_i^{Ch} = \{\overline{C}_i^{Ch}, T_i, D_{R,i}, D^{Ch}_{NR,i}\}$. Furthermore, redundant implementations distinguish between recoverable and non-recoverable errors, defined as follows:

Definition 3 (Recoverable error): A recoverable error compromises the system but subsides after a certain time. Due to redundancy, the error was never effective from system's perspective. It can be masked and after a certain time the channel works reliable again.

Definition 4 (Non-recoverable error): A non-recoverable error endangers the safety of the system. It can not be corrected or masked, thus the system must enter the safety layer.

According to traditional fault tolerant system design, we adopt the error types: hardware error x_H and software error x_S. In fault-tolerant systems, $\frac{N-1}{2}$ hardware and software errors can be tolerated and corrected, where N is the number of active channels. In a DMR system ($N = 2$) one x_H and x_S can only be detected, but not corrected. Thus, the error is non-recoverable and the system has to enter the safety layer.

We additionally introduce the error types: recoverable deadline miss $x_{R,i}$ and non-recoverable deadline miss $x_{NR,i}$. In contrast to the detection of hardware and software errors, for which at least two channels are required, only one channel is needed to detect deadline violations. In order to detect a deadline miss it is sufficient to check if a value is available at

TABLE I: Types of errors that can occur in one channel

Hardware Error	$x_H \wedge x_S \wedge x_R \wedge x_{NR}$
	$x_H \wedge \bar{x}_S \wedge x_R \wedge x_{NR}$
	$x_H \wedge \bar{x}_S \wedge x_R \wedge \bar{x}_{NR}$
	$x_H \wedge x_S \wedge \bar{x}_R \wedge \bar{x}_{NR}$
	$x_H \wedge \bar{x}_S \wedge \bar{x}_R \wedge \bar{x}_{NR}$
Software Error	$\bar{x}_H \wedge x_S \wedge x_R \wedge x_{NR}$
	$\bar{x}_H \wedge x_S \wedge x_R \wedge \bar{x}_{NR}$
	$\bar{x}_H \wedge x_S \wedge \bar{x}_R \wedge \bar{x}_{NR}$
Deadline Miss	$\bar{x}_H \wedge \bar{x}_S \wedge x_R \wedge x_{NR}$
	$\bar{x}_H \wedge \bar{x}_S \wedge x_R \wedge \bar{x}_{NR}$

the deadline. Accordingly, in a DMR setup, a deadline violation of one channel can be masked since the redundant channel provides the correct result in time (recoverable error). If no result is available at $D_{NR,i}$, so a non-recoverable deadline miss $x_{NR,i}$ occurred, it cannot be ensured that the channel works reliable and thus the system must switch to the safety layer (non-recoverable error). This is caused by the fact that an error in one channel can cause other errors in the same channel. For example, a hardware error x_H in Ch A can cause a software error x_S, which in turn causes a recoverable deadline miss $x^A_{R,i}$. Tab. I lists further error dependencies that can occur in one channel and for which the operational mode of the system is endanger. To determine whether a hardware or software error and thus a non-recoverable error is present, the result must be available at time $D^{Ch}_{NR,i}$ to enable the comparison.

For traditional DMR the function for entering the safety layer is given by

$$f_D(x_H^{Ch}, x_S^{Ch}) = \sum_{Ch=1}^{2} x_H^{Ch} + \sum_{Ch=1}^{2} x_S^{Ch}. \quad (1)$$

With Tab. I and (1) we extend the function for entering the safety layer to

$$f_D(x) = \sum_{Ch=1}^{2} x_H^{Ch} + \sum_{Ch=1}^{2} x_S^{Ch} + \prod_{Ch=1}^{2} x_{R,i}^{Ch} + \sum_{Ch=1}^{2} x_{NR,i}^{Ch} \quad (2)$$

for $x = \{x_H^{Ch}, x_S^{Ch}, x_{R,i}^{Ch}, x_{NR,i}^{Ch}\}$. Note, if Ch A and Ch B miss $D_{R,i}$, so the deadline miss cannot be masked by the redundant channel, the system must switch to the safety layer, which also corresponds to a non-recoverable error.

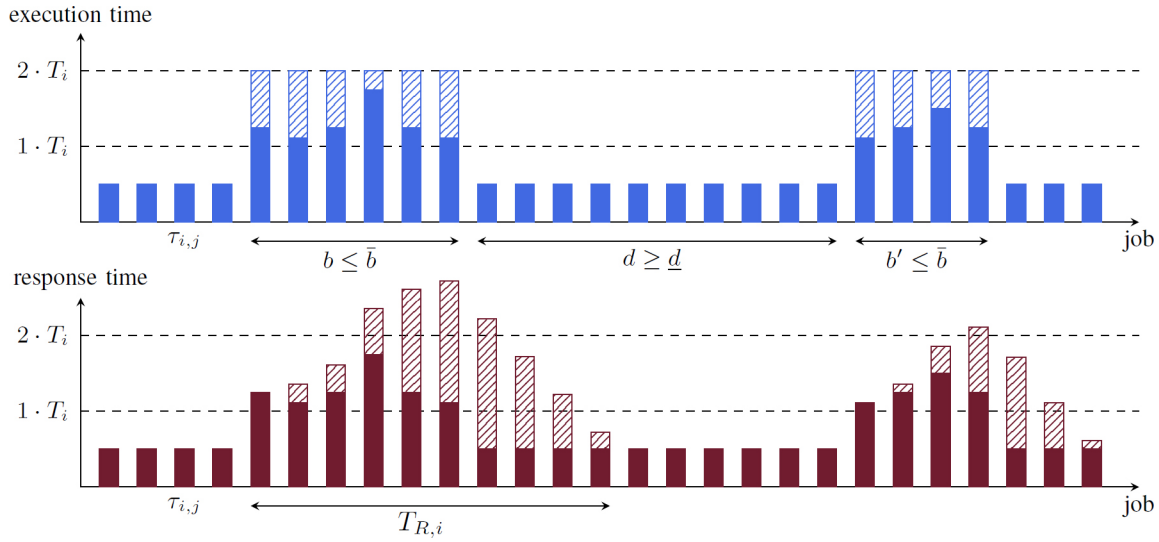

Fig. 3: Conceptual trace of a task τ_i. Above: The execution times normalized to the period T_i of a job are plotted on the y-axis and the length of the bursts b and b' as well as the distance d between two bursts are shown. The shaded area (pessimistic approximation) is used to analyze whether the system is recoverable. Below: The response times normalized to the period T_i of task τ_i are plotted on the y-axis. Further the recovery time $T_{R,i}$ of a burst is illustrated. The shaded area corresponds to the accumulated time of the previous jobs.

The error probability of the system can be derived from (2) with

$$P_D(x) = \sum_{Ch=1}^{2} P^{Ch}(x_H) + \sum_{Ch=1}^{2} P^{Ch}(x_S) + \prod_{Ch=1}^{2} P^{Ch}(x_{R,i}) + \sum_{Ch=1}^{2} P^{Ch}(x_{NR,i}). \quad (3)$$

But, if one channel misses $D_{R,i}$, then only one channel works reliable with

$$f_S(x) = x_H^{Ch} + x_S^{Ch} + x_{R,i}^{Ch} + x_{NR,i}^{Ch} \quad (4)$$

and therefore, the error probability is given by

$$P_S(x) = P^{Ch}(x_H) + P^{Ch}(x_S) + P^{Ch}(x_{R,i}) + P^{Ch}(x_{NR,i}). \quad (5)$$

The probability of a recoverable deadline miss $x_{R,i}$ is higher then the probability of x_H, x_S and $x_{NR,i}$. Thus, (5) is dominated by $P^{Ch}(x_{R,i})$ and the reliability of the system is drastically decreased. So, if one channel misses $D_{R,i}$, the system would have to switch to the safety layer immediately. In other words, Timing Diversity would only be effective to the first deadline miss of a single channel, which drastically limits its applicability.

V. Recoverable System and Recovery Interval

To increase the permanent reliability of the system and thus make timing diversity applicable, the only possibility is to recover the failed channel. However, this approach is not trivial, as the channel must be recovered within a short time period to ensure high reliability of the system over time. If too much time is spent on recovery, then the probability of the second channel failing increases and the reliability decreases drastically. In other words, an error on the second channel during the recovery time leads to a completely failed system. This concept is based on Fig. 4 and we can conclude that the reliability is

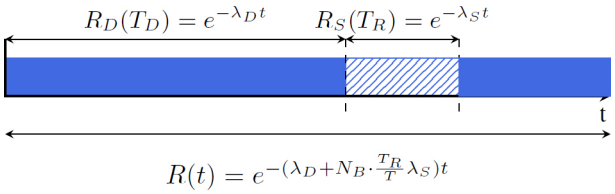

Fig. 4: Reliability of the system assuming recovery

$$R(t) = e^{-(\lambda_D + N_B \cdot \frac{T_R}{T} \lambda_S)t} \quad (6)$$

for N_B is the number of times in which only one channel works reliable. Previously, we impose a condition that the single system is recoverable.

We are assuming a deadline violation, on one platform, therefore we omit the superscripted channel index Ch. It often happens that not only a single but multiple jobs exhibit worst-case timing behavior, consecutively. So, the system suffers from a burst of prolonged execution times. To model such a behavior, we define the maximum burst length and the minimum distance between bursts, as follows:

Definition 5 (Maximum burst length $\bar{b}$): A burst corresponds to multiple jobs with execution times $C_{i,j} > D_{R,i} (= T_i)$. The maximum burst length $\bar{b}$ describes the maximum number of successive jobs exceeding their recoverable Deadline $D_{R,i}$.

Definition 6 (Minimum burst distance $\underline{d}$): The minimum burst distance $\underline{d}$ defines the minimum number of jobs between two consecutive bursts for which applies $C_{i,j} < D_{R,i} (= T_i)$.

Since we do not accept the loss of jobs, a burst will postpone every successive job, leading to a backlog of jobs (cf. lower

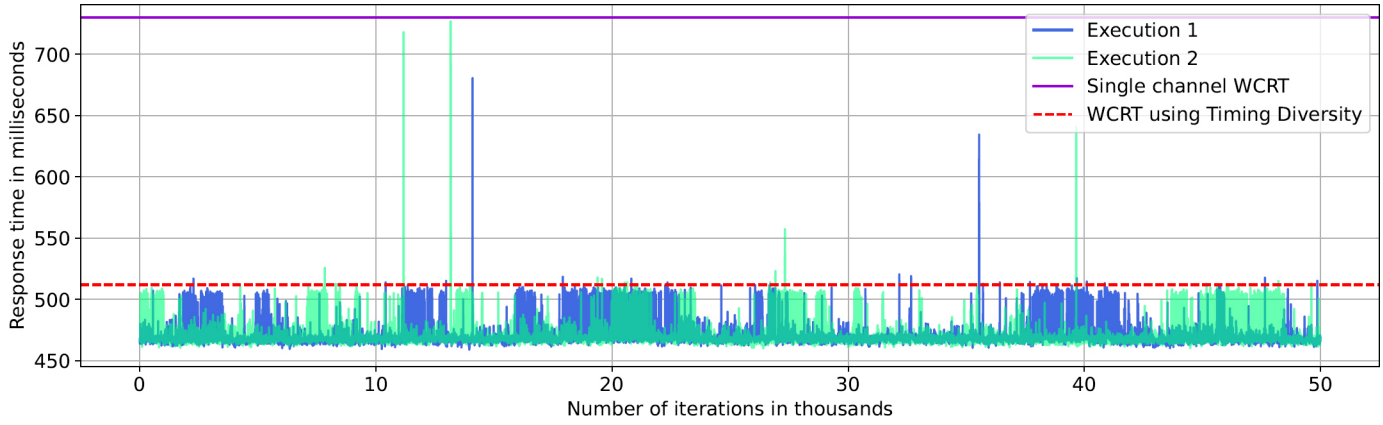

Fig. 5: Data trace taken from [4][Fig. 1]

half of Fig. 3). Still after the burst has subsided, the queued jobs need to be processed, to achieve the normal timing behavior. Consequently, during this time only one channel works reliably and requires the other to work without any failure. The time interval, from a burst's beginning until the backlog is processed, is called the recovery interval $T_{R,i}$. We now state a formula to check if a system is recoverable and proof by contradiction.

Theorem 1: Given the minimum distance $\underline{d}$ and the maximum burst length $\bar{b}$, a system is recoverable if

$$\sum_{j=d_0}^{\underline{d}} \underbrace{(T_i - C_{i,j})}_{T_D} \geq \underbrace{\sum_{j=b_0}^{\bar{b}} T_i \cdot \left(\left\lceil \frac{C_{i,j}}{T_i} \right\rceil - 1 \right)}_{T_B} \tag{7}$$

where b_0 is the first job that exceeds its deadline, and d_0 is the first job that works reliably again after the burst. Furthermore, T_B is the processing time required by the burst and T_D is the remaining processing time per job inherent to normal operation.

Proof 1: Proven by contradiction. Let a system be recoverable, then

$$\sum_{j=d_0}^{\underline{d}} (T_i - C_{i,j}) < \sum_{j=b_0}^{\bar{b}} T_i \cdot \left(\left\lceil \frac{C_{i,j}}{T_i} \right\rceil - 1 \right) \tag{8}$$

$$\sum_{j=d_0}^{\underline{d}} (T_i - C_{i,j}) - \sum_{j=b_0}^{\bar{b}} T_i \cdot \left(\left\lceil \frac{C_{i,j}}{T_i} \right\rceil - 1 \right) < 0. \tag{9}$$

Equation (9) is only valid if the the sum over $\underline{d}$ is smaller than the sum over $\bar{b}$, and therefore $\underline{d} < \bar{b}$. This means that the maximum burst length is greater than the distance between two bursts. The queued job can only be processed at the minimum distance, since according to Def. 6 $T_i > C_{i,j}$ and thus there is a slack that can be used for processing. Since $\underline{d} < \bar{b}$, there is no sufficient slack available to process the backlogged job. As a result, an overload situation is caused in the channel and all subsequent jobs of τ_i will exceed $D_{R,i}$. Therefore, the system is not recoverable and thus (7) applies.

By taking (7), we can calculate the maximum recovery interval for each channel as

$$\overline{T_{R,i}^{Ch}} = T_i \cdot \frac{T_B}{T_D}. \tag{10}$$

To show the impact of our formulas, we present an example.

VI. Example

We recently conducted a case study, the results were consistent across all cases. Thus, we have chosen to include the results from one representative use case illustrated in Fig. 5 [4] as an example. Based on this setup and data trace, we demonstrate the impact of our recovery mechanism (see Fig. 5). The system consists of two identical platforms and a single task τ_1, so we omit the task index for legibility. Based on Fig. 5 we define $\tau = \{\overline{C} = 730\,\text{ms}, D = T = 550\,\text{ms}\}$. But instead of the proposed deadline of [4], we assume a slightly higher deadline to make deadline violations clearly identifiable. Additionally, we use an average execution time of $C = 480\,\text{ms}$, since we cannot know the job's execution times a priori. A series of $5 \cdot 10^4$ measurements were conducted for each platform in the DMR setup. For Ch A 2 deadline violations were observed and 4 for Ch B, respectively, and an (observed) WCET of $670\,\text{ms}$ for Ch A $730\,\text{ms}$ for Ch B, respectively. Even tough both channel were configured identically, the measurements resulted in different WCETs for each channel. This implies that the number of tests were not sufficiently large to discover the more infrequent WCET on Ch B, so we assume the more pessimistic WCET of $730\,\text{ms}$ for Ch B as well. As a result, we define $\tau_1^A = \tau_1^B = \tau$. The maximum burst length on both platforms are given by $\bar{b} = 1$ and we estimate the minimum burst distance on $\underline{d}^A = 20,000$ and $\underline{d}^B = 2,000$, respectively.

Given 2 and 4 deadline violations for $5 \cdot 10^4$ measurements, respectively, we approximate the probability of a timing error as $P^A(x_R) = 4 \cdot 10^{-5}$ and $P^B(x_R) = 8 \cdot 10^{-5}$. Under the assumption of mutually independent channels, it follows $P^{A|B}(x_R) = 3.2 \cdot 10^{-9}$. Consequently, a timing error statistically occurs for Ch B every $12,500$ jobs, i.e., $6875\,\text{s}$ with a period of $T = 550\,\text{ms}$. Now, we can calculate the error rate of the DMR system to $\lambda_D = \frac{1}{47\,743.056\,\text{h}}$ and for the single system to $\lambda_S = \frac{1}{1.910\,\text{h}}$. Assuming no recovery mechanism, the DMR system would break-down after the second deadline violation, so statistically, it would fail in less than four hours. Therefore, despite redundancy it is necessary to deploy recovery.

To calculate the time to failure and the overall reliability with recovery, we need to calculate the recovery interval T_R, first.

Using (10), it follows $T_B = 550\,\text{ms}$ and $T_D = 70\,\text{ms}$. So, the maximum recovery interval for Ch A and Ch B is

$$\overline{T}_R = T \cdot \frac{550\,\text{ms}}{70\,\text{ms}} \approx 4.321\,\text{s} \tag{11}$$

Considering a constant average execution time, the recovery interval depends primarily on the period T. This correlation is illustrated in Fig. 6. Note that at $T = 730$ the recovery time jumps at $0\,\text{ms}$, because $T > \bar{C} = 730$.

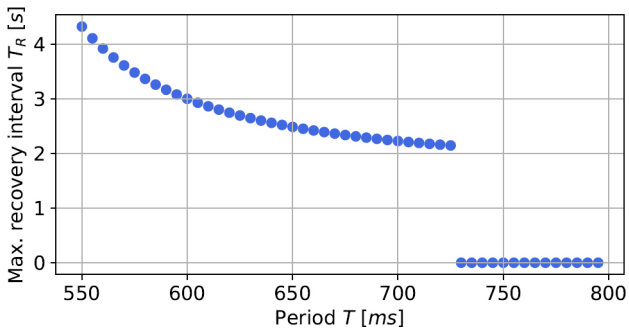

Fig. 6: Recovery interval T_R for periods T

With $T_R = 4.321\,\text{s}$, $N_B = 6$ deadline violations, and (6), integration over reliability yields the mean time to failure as

$$\int_{t=0}^{\infty} e^{-(\frac{1}{47\,743.056\,\text{h}} + \frac{6 \cdot 4.321\,\text{s}}{0.55\,\text{s} \cdot 50000 \cdot 3600} \cdot \frac{1}{1.910\,\text{h}})t} dt \approx 47\,432\,\text{h} \tag{12}$$

Due the short maximum recovery interval $\bar{T}_R$, we achieve a time to failure almost as good as under ideal conditions. Given a instantaneous recovery, the time to failure would be statistically $47\,743\,\text{h}$. Our recovery mechanism does decrease this time only by marginal 0.65%.

VII. Conclusion

We presented a theoretical foundation of the Timing Diversity concept proposed by [4], [5] and showed compliance with existing DMR systems w.r.t. hardware and software errors. From the analysis, we saw that Timing Diversity alone would still be insufficient to achieve a level of reliability that is required by safety-critical systems. The drop in reliability following a channel deadline violation would enforce entering the safety layer already after a mean time of few hours of operation, as a comparison with experimental data showed. We presented a fast recovery method and derived a formal model to calculate the resulting combined system reliability. We used that model to investigate and quantify the relation between task period, recovery time and reliability. Using experimental data we saw that rare timing outliers could be compensated by fast recovery and formulated conditions that guarantee outlier compensation for continuous service with sufficiently high reliability, even for higher levels of criticality.

VIII. Acknowledgment

This work was funded by the German Federal Ministry of Economic Affairs and Climate Action (BMWK) within the Many-core Avionics Design, Architecture, Modeling and Simulation (MC-ADAMS) project, funding number 20E1920B.

We would like to thank the other members of the MC-ADAMS project for their support in our research.

References

[1] J. Bin, D. Girbal, Sylvai nand Gracia Pérez, A. G. Grasset, and A. Mérigot, "Studying co-running avionic real-time applications on multi-core COTS architectures," in *ERTS 2014*, 2014.

[2] C. Maiza, H. Rihani, J. M. Rivas, J. Goossens, S. Altmeyer, and R. I. Davis, "A survey of timing verification techniques for multi-core real-time systems," *ACM Comput. Surv.*, vol. 52, no. 3, jun 2019. [Online]. Available: https://doi.org/10.1145/3323212

[3] R. I. Davis and L. Cucu-Grosjean, "A survey of probabilistic timing analysis techniques for real-time systems: 03:1-03:60 pages / leibniz transactions on embedded systems, vol 6, no 1 (2019)," *LITES: Leibniz Transactions on Embedded Systems*, pp. 1–60, 2019.

[4] R. Hapka, A. Christmann, and R. Ernst, "Controlling high-performance platform uncertainties with timing diversity," in *2022 IEEE 28th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2022, pp. 212–219.

[5] A. Christmann, A. Kostrzewa, R. Ernst, M. Rockschies, M. Halle, F. Thielecke, A. Peuker, A. Kuzolap, M. Steen, P. Hecker, K.-F. Nessitt, and S. Saidi, "Integrating multi-/many-cores in avionics: Open issues and future concepts," in *2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*. IEEE, 2021, pp. 1–8.

[6] "Amc-20 amendment 23 - amc 20-136a," Jan 2022. [Online]. Available: https://www.easa.europa.eu/en/document-library/certification-specifications/amc-20-amendment-23

[7] F. Reghenzani, G. Massari, and W. Fornaciari, "chronovise: Measurement-based probabilistic timing analysis framework," *Journal of Open Source Software*, vol. 3, no. 28, p. 711, 2018. [Online]. Available: https://doi.org/10.21105/joss.00711

[8] F. Reghenzani, G. Massari, W. Fornaciari, and A. Galimberti, "Probabilistic-wcet reliability: On the experimental validation of evt hypotheses," in *Proceedings of the International Conference on Omni-Layer Intelligent Systems*, ser. COINS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 229–234. [Online]. Available: https://doi.org/10.1145/3312614.3312660

[9] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms," in *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2013.

[10] H. Kim, D. de Niz, B. Andersson, M. Klein, O. Mutlu, and R. Rajkumar, "Bounding and reducing memory interference in COTS-based multi-core systems," *Real-Time Systems*, vol. 52, no. 3, pp. 356–395, Feb. 2016. [Online]. Available: https://doi.org/10.1007/s11241-016-9248-1

[11] A. Agrawal, G. Fohler, J. Freitag, J. Nowotsch, S. Uhrig, and M. Paulitsch, "Contention-aware dynamic memory bandwidth isolation with predictability in cots multicores: An avionics case study," in *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*, 2017.

[12] M. Hassan and R. Pellizzoni, "Bounding dram interference in cots heterogeneous mpsocs for mixed criticality systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2323–2336, 2018.

[13] ——, "Analysis of Memory-Contention in Heterogeneous COTS MPSoCs," in *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), M. Völp, Ed., vol. 165. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, pp. 23:1–23:24. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2020/12386

[14] A. Schranzhofer, J.-J. Chen, and L. Thiele, "Timing predictability on multi-processor systems with shared resources," in *Embedded Systems Week-Workshop on Reconciling Performance with Predictability*, 2009, p. 87.

[15] F. Boniol, H. Cassé, E. Noulard, and C. Pagetti, "Deterministic execution model on cots hardware," in *Proceedings of the 25th International Conference on Architecture 700 of Computing Systems*. Springer, Berlin, Heidelberg, 2012, pp. 98–110.

[16] R. Pellizzoni, E. Betti, S. Bak, G. Yao, J. Criswell, M. Caccamo, and R. Kegley, "A predictable execution model for cots-based embedded systems," in *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 2011.