Mälardalen University Press Licentiate Theses
No. 104

NEW STRATEGIES FOR ENSURING TIME AND VALUE CORRECTNESS IN DEPENDABLE REAL-TIME SYSTEMS

Hüseyin Aysan

2009

MÄLARDALEN UNIVERSITY
SWEDEN

School of Innovation, Design and Engineering

ISSN 1651-9256
ISBN 978-91-86135-28-7
Printed by Mälardalen University, Västerås, Sweden

Abstract

Dependable real-time embedded systems are typically composed of tasks with multiple criticality levels allocated to a number of heterogeneous computing nodes connected by heterogeneous networks. The heterogeneous nature of the hardware, results in a varying level of vulnerability to different types of hardware failures. For example, a computing node with effective shielding shows higher resistance to failures caused by transient faults, such as radiation or temperature changes, than an unshielded node. Similarly, resistance to failures caused by permanent faults can vary depending on the manufacturing procedures used. Task vulnerability to different types of errors, potentially leading to a system failure, varies from task to task, and depends on several factors, such as the hardware on which the task runs and communicates, the software architecture and the implementation quality of the software. This variance, the different criticality levels of tasks, and the real-time requirements, necessitate novel fault-tolerance approaches to be developed and used, in order to meet the stringent dependability requirements of resource-constrained real-time systems.

In this thesis, we provide four major contributions in the area of dependable real-time systems. Firstly, we describe an error classification for real-time embedded systems and address error propagation aspects. The goal of this work is to perform the analysis on a given system, in order to find bottlenecks towards satisfying dependability requirements, and to provide guidelines on the usage of appropriate error detection and fault tolerance mechanisms.

Secondly, we present a time-redundancy approach to provide a-priori guarantees in fixed-priority scheduling (FPS) such that the system will be able to tolerate a single value error per every critical task instance, while keeping the potential costs minimized.

Our third contribution is a novel approach, Voting on Time and Value (VTV), which extends the N-modular redundancy approach by explicitly con-

sidering both value and timing errors, such that a correct value is produced at a correct time, under specified assumptions. We illustrate our voting approach by instantiating it in the context of the well-known triple modular redundancy (TMR) approach. Further, we present a generalized voting algorithm targeting NMR, that enables a high degree of customization from the user perspective.

Finally, we propose a novel cascading redundancy approach within a generic fault tolerant scheduling framework. The proposed approach is i) capable of tolerating errors with a wider coverage (with respect to error frequency and error types) than our proposed time and space redundancy approaches in isolation, ii) handles tasks with mixed criticality levels, iii) is independent of the scheduling technique, and above all, iv) ensures that every critical task instance can be feasibly replicated in both time and/or space.

The fault-tolerance techniques presented in this thesis address various error scenarios that can be observed in real-time embedded systems with respect to the types of errors and frequency of occurrence, and can be used to achieve the high levels of dependability, required in many critical systems.

Acknowledgments

My trip to Sweden was intended to be a short one. Definitely not to get another degree or do research, but rather, to experience a new culture, live in a different climate and learn a new language (!). It would indeed have been a short one, as intended, if I didn't meet Lars Asplund who introduced me with all the fun things he works with, at the very first hours of my arrival in Västerås. Of course, he kept on introducing them with a constant dose, which eventually made me get a degree, and even continue to work as a research engineer at this department. Thank you Lars for making a lot of things entertaining and keeping me here!

Many thanks go to Sasikumar Punnekkat and Radu Dobrin, for teaching me a lot of new stuff, for guidance and support, for all the fruitful discussions, and for the company during the conference trips. Also, I am grateful to Hans Hansson, Mikael Sjödin, Ivica Crnkovic and Malin Rosqvist for their reviews and feedback, as well as for the humour which they bring to the PROGRESS research centre.

I would like to thank my officemates, Moris, Aida and Mikael for the good times we have had, but especially Moris for reminding me all the deadlines of any kind and Aida for keeping my plants alive. I would like to thank many more people at this department, Farhang, Jörgen, Fredrik, Andreas, Bob, Séverine, Peter, Hongyu, Antonio, Batu, Marcelo, Yue, The Balkanian Gang (Aneta, Luka, Juraj, Ana, Iva, Leo, Adnan and Damir), Pasqualina, Dag, Nolte, Cristina, Tiberiu, Kathrin, Jagadish, Nikola, Johan Fredriksson, Johan Kraft, Markus Lindgren, Jan Carlson, and others for all the fun during coffee breaks, lunches, parties, conferences and PROGRESS trips!

I would also like to thank Jörgen one more time for the daily Swedish words, Fredrik for the daily strips, Antonio and Conny for their training partnerships, Maria Lindén for motivating me to learn ice-skating and for her excellent guidance, Christer Norström for the skiing lesson, Harriet and Monica for making the life at the department easier, Baran and Ceren for their support and friendships.

Finally, many thanks to my parents, my brother Ilhan, Lena, Ümran, Murat, Engin, Erhan, Hazim and Ege who all contributed a lot (probably the most) to my life during this period in many ways!

This work has been supported by the Swedish Foundation for Strategic Research (SSF), via the strategic research centre PROGRESS.

Hüseyin Aysan
Västerås, June, 2009

List of Publications

Papers included in the thesis [1]

Paper A *Towards an Error Modeling Framework for Dependable Component-based Systems*, Hüseyin Aysan, Radu Dobrin, Sasikumar Punnekkat, In Proceedings of the DATE Workshop on Dependable Software Systems, Munich, Germany, March, 2008.

Paper B *Maximizing the Fault Tolerance Capability of Fixed Priority Schedules*, Radu Dobrin, Hüseyin Aysan, and Sasikumar Punnekkat, In Proceedings of the 14^{th} IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'08), KaoHsiung, Taiwan, August, 2008.

Paper C *A Voting Strategy for Real-Time Systems*, Hüseyin Aysan, Sasikumar Punnekkat, and Radu Dobrin, In Proceedings of the 14^{th} IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'08), Taipei, Taiwan, December, 2008.

Paper D *A Cascading Redundancy Approach for Dependable Real-Time Systems*, Hüseyin Aysan, Sasikumar Punnekkat, and Radu Dobrin, In Proceedings of the 15^{th} IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'09), Beijing, China, August, 2009 *(to appear)*.

[1] The included articles are reformatted to comply with the licentiate thesis specifications

Other relevant publications

Conferences and Workshops

- *Error Modeling in Dependable Component-based Systems*, Hüseyin Aysan, Sasikumar Punnekkat, Radu Dobrin, IEEE International Workshop on Component-Based Design of Resource-Constrained Systems (CORCS'08), Turku, Finland, July, 2008

- *Adding the Time Dimension to Majority Voting Strategies*, Hüseyin Aysan, Sasikumar Punnekkat, Radu Dobrin, Proceedings of the Work-In-Progress (WIP) session of the 14th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'08), St. Louis, MO, United States, April, 2008

- *A Generalized Task Allocation Framework for Dependable Real-Time Systems*, Hüseyin Aysan, Radu Dobrin, Sasikumar Punnekkat, Proceedings of the Work-In-Progress (WIP) session of the 19th Euromicro Conference on Real-Time Systems (ECRTS'07), Pisa, Italy, July, 2007

Technical Report

- *FT-Feasibility in Fixed Priority Real-Time Scheduling*, Hüseyin Aysan, Radu Dobrin, Sasikumar Punnekkat, MRTC report ISSN 1404-3041 ISRN MDH-MRTC-210/2007-1-SE, Mlardalen Real-Time Research Centre, Mlardalen University, March, 2007

Contents

I

Thesis

Chapter 1

Introduction

Most real-time systems typically have to satisfy high dependability requirements due to their interactions with and possible impacts on the environment. The real-time nature of such systems requires that the delivered services must be both value-wise correct and timely, i.e. not too late or too early. Satisfying systems' dependability requirements typically requires using both fault prevention and fault tolerance (FT) approaches. However, implementation of such approaches can be very costly, requiring a significant amount of extra resources, and the designers have to judiciously select efficient and cost-effective approaches specific to the system context.

A major step in designing dependable real-time systems is modeling of the error scenarios, by explicitly stating several characteristics of each error that forms a threat to satisfy the specified dependability requirements, and disregarding the errors that are too unlikely to occur. Once having such a model, the next step towards efficiently and effectively satisfying the dependability requirements is to introduce FT approaches, in the form of error masking, error detection and error recovery, that are capable of addressing only the errors stated in the error model.

In this thesis, we propose an error modeling approach considering various aspects, such as domain, consistency, impact, criticality and persistence of errors. Although several works exist on error modeling and error classification, our modeling approach can be seen as an all-encompassing view, elaborating upon many of these works. Based on the impact, criticality and persistence of errors, we propose the usage of appropriate redundancy approaches to provide FT. For instance, for tolerating critical transient errors, we use a time redundancy approach in the form of task re-executions, or executions of alternate tasks. On the other hand, for tolerating critical permanent errors, we use adaptations of N-modular redundancy approaches [1]. We address the domain property of errors by explicitly considering time and value correctness in the voting procedure. Furthermore, we use both approaches in combination to achieve a more comprehensive error coverage.

1.1 Thesis Outline

This thesis consists of two main parts. The first part comprises seven chapters. Chapter 1 describes the motivation for the research and provides an introduction to the research domain. Chapter 2 introduces the basic concepts related to real-time systems and dependability. Chapter 3, 4 and 5 provide overviews of our time-redundancy, space redundancy and cascading redundancy strategies, respectively. Chapter 6 gives a technical overview of the papers, and finally, Chapter 7 concludes and summarizes the thesis, and gives directions to possible future works. The second part of the thesis is a collection of peer-reviewed conference and workshop papers, which are briefly described below:

1.1.1 Paper A

Towards an Error Modeling Framework for Dependable Component-based Systems, Hüseyin Aysan, Radu Dobrin, Sasikumar Punnekkat, In Proceedings of the DATE Workshop on Dependable Software Systems, Munich, Germany, March, 2008.

Summary In this paper, we propose an approach to model errors that can occur in the system components based on a synthesized view of several works [2, 3, 4, 5, 6]. It presents various aspects of errors in two categories based on their influence on the error handling mechanisms. These categories essentially determine 'which mechanisms' and 'how much' are needed for adequate error handling. The various aspects considered are domain, consistency, impact, criticality and persistence of errors. The domain and consistency determine what kind of error handling mechanisms are appropriate while the rest determine the amount of error handling needed. The former is more relevant for design of the system and for providing qualitative guarantees, whereas the latter is important for quantitative predictions.

My contribution I was the main author of this paper and contributed with the survey on existing error models and the proposed error modeling approach. The co-authors contributed with valuable advice on the validity of the classification of errors as well as on the formulation the research challenges.

1.1.2 Paper B

Maximizing the Fault Tolerance Capability of Fixed Priority Schedules, Radu Dobrin, Hüseyin Aysan, and Sasikumar Punnekkat, In Proceedings of the 14^{th} IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'08), KaoHsiung, Taiwan, August, 2008.

Summary This paper focuses on incorporating time redundancy as a part of the scheduling strategy used in a real-time node of a system in order to tolerate transient errors occurring in critical task instances. The proposed methodology is intended for a task set with fixed priorities resulting in a fault-tolerant task set that provides timing guarantees for the failed critical tasks to re-execute or run an alternate version under certain assumptions on the processor utilization of critical and non-critical tasks. The resulting task set will have new task attributes to provide such guarantees.

My contribution I was the second author of this paper, contributed with the literature survey, took part in the development of the methodology and performed the evaluations on the validity of the approach. The basic idea for the methodology has its roots in Radu Dobrin's Ph.D thesis which we further adapted to fault-tolerant scheduling.

1.1.3 Paper C

A Voting Strategy for Real-Time Systems, Hüseyin Aysan, Sasikumar Punnekkat, and Radu Dobrin, In Proceedings of the 14^{th} IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'08), Taipei, Taiwan, December, 2008.

Summary A widely used approach to ensure fault tolerance in dependable systems is the N-modular redundancy (NMR) which typically uses a majority voting mechanism. However, NMR primarily focuses on producing the correct value, without taking into account the time dimension. In this paper, we propose a new approach, Voting on Time and Value (VTV), applicable to real-time systems, which extends the modular redundancy approach by explicitly considering both value and timing errors, so that correct value is produced at correct time, under specified assumptions. We present an algorithm directly applicable for triple modular redundancy (TMR) as well as a generalized version targeting NMR. Both intermittent and permanent errors are tolerated by this approach.

My contribution I was the main author of this paper and contributed with the idea, literature survey, development of the methodology and the algorithms. The co-authors contributed with discussions during the development of the methodology and provided continuous feedbacks.

1.1.4 Paper D

A Cascading Redundancy Approach for Dependable Real-Time Systems, Hüseyin Aysan, Sasikumar Punnekkat, and Radu Dobrin, In Proceedings of the 15^{th} IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'09), Beijing, China, August, 2009

Summary In this paper, we combine our time redundancy approach proposed as a scheduling strategy with our space redundancy approach to cover errors in both time and value domains together with all degrees of persistence from transient to permanent errors. As a result, our framework provides a comprehensive methodology to enable synergistic usage of these redundancy techniques.

My contribution I was the main author of this paper and contributed with the idea for integration, development of the algorithm, and part of the methodology. The co-authors provided technical contributions to the time redundancy part as well as useful feedbacks for the overall paper.

Chapter 2

Dependability in Real-Time Systems

2.1 Real-Time Systems

Real-time systems are computing systems whose correctness depends not only on the correctness of the outputs produced, but also on the timeliness of these outputs [7]. Failing to meet the timeliness requirement may result in catastrophic consequences, such as loss of human life, in *hard real-time systems*, while decrease in Quality of Service (QoS), or degraded service can be the results of missing deadlines in *soft real-time systems*.

Fast computing or performance optimizations are not direct solutions for satisfying the timeliness requirement, since increasing the speed of computations does not mean that meeting the deadlines will be guaranteed [8]. Real-time research strives for assuring that the systems will behave predictably with respect to time, e.g., execute their tasks before their predefined deadlines, while enabling efficient usage of the limited resources such as processor and memory.

Real-time systems are typically composed of a set of *tasks*, where each task performs a certain function satisfying certain *timing constraints*. The timing constraints are specified by special attributes, such as *offsets* which specify the earliest time points at which the tasks can start executing, and *deadlines* which specify the latest time points at which the tasks should complete their executions. Tasks may have *periodic*, *aperiodic* or *sporadic* activations which are controlled by a *scheduler* based on a *scheduling policy*. Each periodic task consists of an infinite sequence of activations, which are called *instances*. The scheduling policy can either be *off-line* or *on-line*. In the off-line scheduling policies, the time points for each activation of task instances are decided at design-time, whereas in on-line scheduling, these decisions are made during run-time based on, e.g., task *priorities*. On-line scheduling policies can further be decomposed into *fixed-priority scheduling* (FPS), and *dynamic-priority scheduling* policies depending on whether the task priorities are decided during design-time or run-time [9].

Real-time systems consist of some sort of hardware, often relatively complex real-time software and a dynamic environment that the systems interact with. Despite the advances in the production techniques of computer hardware, there remains a possibility that the hardware may fail. Similarly, despite the advances in software engineering, bug free software development is considered as infeasible due to the costs, if at all practically possible. Furthermore, due to the non-deterministic nature of the environments in which the real-time systems operate, there is always a possibility of external interferences that may adversely affect the correctness or timeliness of their functioning. Therefore, special attention has to be paid in order to have the confidence in the real-time systems at acceptable levels. This is the basic reason for the close coupling between real-time systems and dependability concerns.

2.2 Dependability

Dependability of a system is a property which indicates the degree to which extent its services can be trusted by its users. A systematic decomposition of the dependability concept can be done as shown in Figure 2.1 where the main components are the *threats* to dependability, *attributes* of dependability and the *means* to achieve dependability [2, 10]:

2.2.1 Failures, Errors and Faults

A system *failure* is the deviation of its delivered service from the specified service, therefore threatening the confidence degree of the system to deliver a service that can be trusted. A system can fail in different ways based on the domain of application. Typical failure modes, presented in ([2, 11]), are as follows:

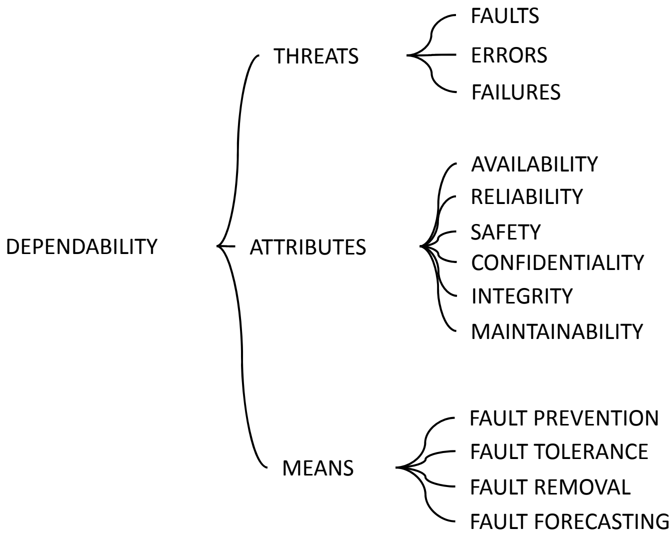


Figure 2.1: The dependability tree [2]

Fail-soft: In the fail-soft failure mode, the system continues functioning but provides a degraded service until the system is restored.

Fail-safe: In the fail-safe failure mode the failure does not result in severe consequences.

Fail-silent: In the fail-silent failure mode, the system does not produce any services that are erroneous, but may still be functioning and could deliver correct services.

Fail-stop: In the fail-stop failure mode, the system does not produce any outputs and continues to stay in this mode until restarted. Furthermore it is assumed that the error is detected and signalled by an error message.

Crash failure: In the crash failure mode, the system does not produce any outputs and continues to stay in this mode until restarted. Furthermore it is assumed that the error is not signalled [12].

Babbling idiot failure: In this mode, the system produces untimely outputs, often loads of junk outputs, possibly threatening the availability of resources.

Deceptive failure: In Deceptive failure mode, the system pretends some other identity, without authority such as sending and receiving messages in another identity.

Byzantine failure: This is the mode where the system can fail in any possible arbitrary way.

Each failure mode corresponds to different degrees of *severity* and *controllability*. One of the biggest challenges in system design is to introduce a certain degree of restrictions, i.e., to set one, or a set of allowed failure modes, on how the target system can fail.

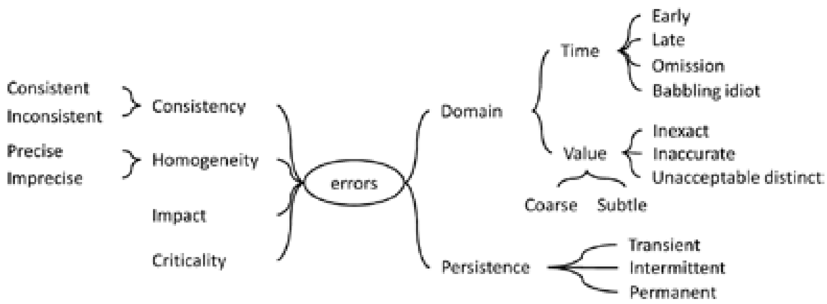


Figure 2.2: Error Classification

An *error* is a system state that may lead to a system failure through propagations, i.e., valid state transformations. Errors can be classified into several categories, such as, domain, persistence, consistency, homogeneity, impact and criticality [4, 3, 13, 14] as shown in Figure 2.2. The domain and consistency properties determine the types of error handling mechanisms to be used. The other properties describe with what frequency the errors may occur, and once they occur, the probability of causing a system failure as well as the severity of the consequences of such failures. Hence, they determine where these mechanisms should be located and how much resources should be reserved for adequate handling of the expected errors. Modeling error scenarios, error transformations and error propagations, is a crucial step in the dependable real-time systems design, in order to effectively and efficiently introduce mechanisms to prevent system failures.

A *fault* is the adjudged cause of an error. The relationship between faults, errors and failures is shown in Figure 2.3 where each arrow represents a relation of cause and effect [2].

$$\text{Fault} \longrightarrow \text{Error} \longrightarrow \text{Failure} \dashrightarrow \text{Fault}$$

Figure 2.3: The chain of dependability threats [2]

2.2.2 Reliability and Availability

Reliability is the ability to continue delivering correct service, i.e., perform failure-free operation, for a specified period of time. Availability is the probability of being operational and deliver correct service at a given time. These two concepts are often mixed with each other, however, they do not mean the same thing. A system that fails very frequently has a low reliability, but can still have very high availability provided that the recoveries are performed very quickly. Similarly, if a system breaks down very rarely, but the repair action takes a long time, its reliability is high, while its availability is low.

Though being different concepts, they are closely connected in the sense that, if system reliability is improved, then its availability is improved as well (although the opposite case is not always true). In this thesis, we propose strategies for improving reliability of real-time systems which also improves the availability for the stated reason.

2.2.3 Fault Tolerance

Fault tolerance is the set of measures and techniques that are used to enable continuity of correct service delivered by a system even in case of errors. It is one of several complementary techniques to attain dependability along with *fault prevention*, *fault removal* and *fault forecasting* [2]. Two essential steps for providing fault tolerance are the *error detection* and the *recovery* from errors. An optional third step is the *fault diagnosis* and *fault isolation* which causes the error, in order to prevent them from causing more errors.

There exist various types of error detection strategies targeting different types of errors. Examples are *timing checks* for timing errors, *reasonableness* *checks* for coarse value errors and *replica comparison* for subtle value errors. Each detection approach has a different resource requirement and error coverage, where resource requirement generally grows as the coverage increases. Apparently, this may not be a linear relation since the error coverage may consist of various error types which cannot be compared with each other.

Error recovery is the action to transform the system state into an error-free state. There are three main approaches to perform error recovery:

1. Backward error recovery is the technique to take the system back to a correct state that was saved before the error has been detected. The saved state is called a *checkpoint*.

2. Forward error recovery is the technique to transform the system state to a state known to be error-free.

3. Compensation through redundancy is the third error recovery approach which uses the error-free replicas to compensate the error state. The redundancy can be achieved in space by replicating the computing nodes, or in temporal domain, by execution of recovery blocks [15], re-execution of the same actions or execution of alternate actions.

This thesis focuses on the third type of error recovery approach, viz., compensation through redundancy. In the following chapters, we present new time and space redundancy techniques.

Chapter 3

Time Redundancy in Real-Time Systems

In this chapter, we discuss the time redundancy approach used in real-time systems, (also known as *dynamic redundancy*,) give pointers to the existing approaches, and present an overview of our time redundancy approach targeting FPS which is a fairly matured scheduling technique commonly used in complex industrial real-time systems.

Time redundancy is a widely used fault tolerance technique to recover from transient and intermittent errors, which involves repeating the execution of a failed action in the system. It is most often used in computer communications in the form of message re-transmissions. In hard real-time systems, repetitions of the executions should be performed before deadlines, therefore adequate measures are needed to be taken to reserve sufficient spare times in schedules, in order to assure that the deadlines will be met.

Large number of publications have addressed incorporating time redundancy into various real-time scheduling paradigms. Liestman and Campbell [16] investigated a fault tolerant scheduling problem where they tried to schedule primary and alternate versions of a task in the same schedule to attain software redundancy. The alternate version of a task can either be a simplified version of the primary task, that gives an approximate result in a shorter time, or the same as the primary task, which then becomes a basic time redundancy solution. Krishna and Shin [17] used a dynamic programming algorithm to embed backup schedules into the primary schedule, so that hard deadlines of critical tasks will be met in the event of temporary processor failures up to a specified number. Pandya and Malek [18] showed that single faults with a minimum inter-arrival time equal to the largest period in the task set can be recovered by re-executing the failed task, if the processor utilization is less than 0.5 under the Rate Monotonic scheduling policy. Ramos-Thuel and Strosnider [19] used the Transient Server approach to handle transient errors which arrive as aperiodic recovery requests. Ghosh et al. [20] presented a method for guaranteeing that the real-time tasks will meet the deadlines under transient faults, by resorting to reserving sufficient slack in queue-based schedules. Burns et al. [21] provided exact schedulability tests for fault-tolerant task sets under specified failure hypotheses where the fault tolerance is employed in the form of recovery blocks, re-execution of the affected task, checkpointing schemes, or forward recovery methods, like exception handlers. Han et al. [22] scheduled primary and alternate versions of each task, using the imprecise computation model, and aimed to guarantee either the primary or alternate version of each task to be executed before deadlines, such that if the primaries are not successfully executed before certain times, then the corresponding alternates are executed.

Each of the above works has advanced the field of fault tolerant scheduling within the contexts mentioned above. However, some of the disadvantages are restrictive task and fault models, non-consideration of task sets with mixed criticality, non-consideration of scenarios where multiple types of faults cause failures, high computational requirements of complex online mechanisms, and scheduler modifications which may be unacceptable from an industrial perspective. We try to address these limitations in our time-redundancy approach which is outlined below.

3.1 System Model

We assume a single node real-time system consisting of a set of periodic tasks whose deadlines are equal to their periods. The task set consists of critical and non-critical tasks where the criticality of a task could be seen as a measure of the impact of the correctness of the output it delivers on the overall system correctness. Each critical task has an alternate task with a worst case execution time less than or equal to that of its primary and a deadline equal to the deadline of the primary. This alternate task can typically be the same as the primary task, a recovery block, an exception handler or a program with imprecise computations to perform the desired task.

The maximum utilization of the original critical tasks together with their alternates can be up to 100%. This will imply that, during error recovery, execution of non-critical tasks cannot be permitted as it may result in overload conditions. We assume that the scheduler has adequate support for flagging non-critical tasks as unschedulable during such scenarios, along with appropriate error detection mechanisms in the operating system.

We assume that the transient and intermittent errors can effectively be tolerated by a simple re-execution of the affected task whilst the effects of software design faults could be tolerated by executing an alternate action such as recovery blocks or exception handlers. Both of these situations could be considered as execution of another task (either the primary itself or an alternate) with a specified computation time requirement.

We assume that an error can adversely affect only one task at a time, and is detected before the termination of the current execution of the affected task instance. This can be achieved by commonly applied techniques such as acceptance checks at the end of task executions or watchdog timers that interrupt the execution of the task once the worst case execution time has been exhausted.

Our proposed approach enables masking of up to *one error per each task instance* which is a more demanding scenario compared to earlier assumptions such as one error per longest task period, or an explicit minimum inter-arrival time between consecutive error occurrences.

3.2 Time Redundancy in Fixed Priority Scheduling

The goal of our approach is to derive feasibility windows for each task in the task set, which guarantees the fulfillment of the dependability requirements by providing recovery for the errors that are specified in the error model, and to assign FPS attributes (new attributes in case of legacy systems) that ensure task executions within these feasibility windows. An overview of the proposed methodology is shown in Figure 3.1.

While executing non-critical tasks in the background can be a safe and straight forward solution, in our approach we aim to provide non-critical tasks a better service than background scheduling. Hence, depending on the criticality of the original tasks, the feasibility windows we are looking for differ as following:

1. *Fault Tolerant* (FT) feasibility windows for critical task instances
2. *Fault Aware* (FA) feasibility windows for non-critical ones

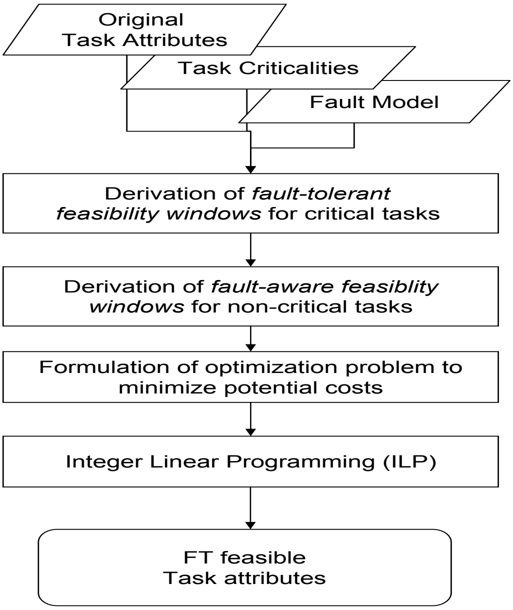


Figure 3.1: Methodology overview

While critical task instances must execute within their FT feasibility windows to be able to re-execute feasibly upon an error, the derivation of FA feasibility windows has two purposes: to prevent non-critical task instances from interfering with critical ones, and to enable their execution at high priority levels, i.e., not only executing in the background. Since the size of the FA feasibility windows depend on the size of the FT feasibility windows, in our approach we first derive FT-feasibility windows and then FA feasibility windows. Then, we assign fixed priorities to ensure the task executions within their newly derived feasibility windows. In some cases, however, it is not possible to achieve the required fault tolerance for the error assumptions with the same priorities for all instances directly. In certain cases reaching the desired degree of fault tolerance may require that instances of a given set of tasks need to be executed in different order on different occasions which is obviously not directly possible in FPS. Our approach detects such situations, and circumvents the problem by splitting certain tasks into their instances. Then, the algorithm assigns different priorities to the newly generated ”artifact” tasks, which are the former instances. Key issues in resolving the priority conflicts are the number of artifact tasks created, and the number of priority levels. Depending on how the priority conflict is resolved, the number of resulting tasks may vary, i.e., based on the size of the periods of the split tasks. Our algorithm minimizes the number of artifact tasks by using Integer Linear Programming (ILP) for solving the priority relations.

Chapter 4

Space Redundancy in Real-Time Systems

Space redundancy is a *static* redundancy technique for achieving fault tolerance, and it is implemented in a large number of critical applications [23, 24]. It is most often used in the form of triple-modular redundancy (TMR) where a critical node is triplicated, and the outputs of the *replicas* are delivered to a voting mechanism which outputs the majority of the delivered values [1] (see Figure 4.1). The key attraction of this technique lies in its low overhead and

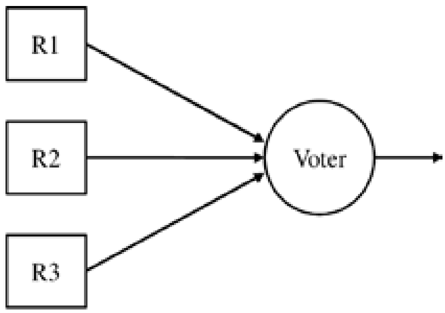


Figure 4.1: Triple modular redundancy

instant detection and masking abilities of errors occurring in a single replica, without requiring backward recovery [10]. Furthermore, with this technique it is possible to detect and mask errors that are not easily detectable due to the lack of prior knowledge on the error behaviour of the critical components. Hence, it provides a wide coverage of different types of errors. The disadvantages include, e.g., cost of redundancy in terms of additional resources, and single point of failure mode. Traditionally, voters are constructed as simple electronic circuits, so that a very high reliability can be achieved. Replication of voters has also been employed to take care of the single-point failure mode in case of highly critical systems [25, 26] as shown in Figure 4.2.

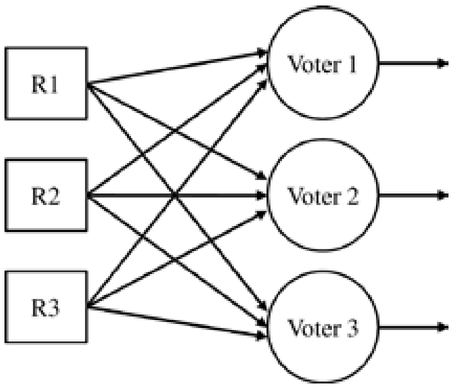


Figure 4.2: Triple modular redundancy with triplicated voters

With the additional cost of increased computation time and software complexity of the voter, more enhanced voting strategies are proposed, such as *plurality*, *median* and *average* voters [27, 28, 29]. *Plurality* voters (or *m-out-of-n* voters) require m corresponding outputs out of n, where m is less than the majority, to reach a consensus [30, 31]. *Median* voters output the middle and *average* voters output the average value of the replica output values.

The primary goal of space redundancy approaches has been the assurance of correctness in the value domain. This implies that tight synchronization is assumed in order to use this approach in time-critical applications. However, providing tight synchronization to compensate for the variations in replicas' output delivery times due to several factors, such as clock drifts, node failures, processing and scheduling variations at node level, as well as communication delays, can be either too costly or simply infeasible. Therefore, a solution based on loose synchronization may be an attractive alternative due to low overheads, requiring, however, specifically designed asynchronous voting algorithms to compensate for the timing variations [32].

A simple approach towards tolerating both value and timing errors in replicas using the NMR approach could be adding time stamps to the replica outputs. Then, voting on time stamp values could detect possible timing anomalies of the replicas, under the assumption that even erroneous nodes issue correct time stamps and no node ever halts. Moreover, this approach is unable to mask late timing errors since the voter has to wait for all the values to be delivered from the replicas.

4.1 Quorum Majority Voting and Compare Majority Voting

Two adaptations of majority voting techniques to time-critical systems, viz., *Quorum Majority Voting (QMV)* and *Compare Majority Voting (CMV)* have been proposed by Shin et al., [33], where voting is performed among a quorum or a majority of responses received, rather than waiting for all.

In QMV, majority voting is performed among the received values as soon as $2n+1$ out of $3n+1$ replicas deliver their outputs to the voter. This $2n+1$ values form the quorum. This approach guarantees detection of a majority consisting of matching values in a bounded time, under the assumption that maximum n replicas can be erroneous in the value domain, be late or both.

In CMV, the voter keeps track of the received values, and outputs the majority value as soon as it is formed without waiting for the possibly late arriving other replica outputs. This approach can mask up to n replica errors out of $2n+1$ replicas as in basic majority voting.

Both QMV and CMV provide outputs within a bounded time interval, as long as the assumptions regarding the maximum number of errors hold. This implies that the late timing errors of replicas will be masked. However, QMV and CMV are unable to mask any early timing errors. The coverage of timing errors for different techniques are shown in Figure 4.3.

In the rest of this chapter, we present an overview of our new approach which tolerates value errors, as well as both early and late timing errors, targeting loosely synchronized time-critical systems.

	Timestamps	QMV	CMV	VTV
Early timing errors	✓	✗	✗	✓
Late timing errors	✗	✓	✓	✓

Figure 4.3: Error coverage comparison for different techniques

4.2 System Model

We assume a distributed real-time system where each critical node is replicated and the outputs are voted to ensure correctness in both value and time. The replicas receive identical inputs and requests for computation. The delivery times of any two replica outputs to the voter are assumed to differ at most by δ. This bound can be achieved by relatively inexpensive software clock synchronization algorithms (compared to hardware-based tight clock synchronization implementations) and using reliable communication techniques that have bounded message transmission times. The graph in Figure 4.4 shows the typical characteristics of a replica output.

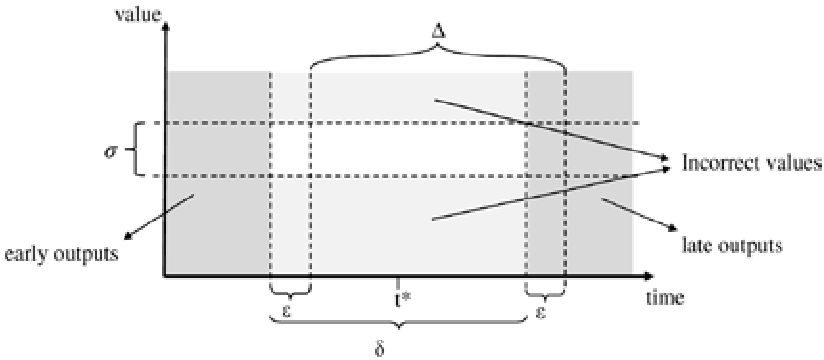


Figure 4.4: Replica output characteristics

Here, t^* is correct time point (seen by a perfect observer) of the output delivery, ϵ is the computation time of the voting algorithm, σ is the maximum admissible deviation in value domain and Δ is the minimum value of a maximum admissible time window for the voter output as per the real-time system specifications, i.e., what the rest of the system can tolerate. It will be still acceptable if Δ is greater than this value shown on the graph, although the opposite scenario can result in timing failures. In such cases, additional efforts should be spent for tightening the range δ by, for instance better clock synchronization or faster message transmissions.

Basic assumptions:

1. After each computation block, the non-faulty nodes produce values within
   - a specified admissible value range,
   - a specified admissible time interval.
2. Incorrect replica outputs do not form a consensus.
3. The voting mechanism does not fail.

4.3 Voting on Time and Value (VTV)

VTV is a voting strategy which tolerates value and timing errors with any persistence, and enables the use of space redundancy in real-time applications with high dependability requirements, while eliminating the tight synchronization requirement for the replica nodes. The key point in this approach is to incorporate the time domain into the voting procedure by performing majority (or plurality) voting on both the delivery times and the values of the replica outputs.

Early outputs can result in taking wrong decisions, even if they are value-vise correct at the time they are produced, depending on the application. Let us assume that R_1 to R_5 in Figure 4.5 represent altitude sensor replicas which form 5-modular redundancy on a descending airplane. Let us also assume that all replicas produce correct altitude values at the time points they deliver the outputs. At time t_5, if we consider a majority of all the received values, the plane (the overall system) will assume that the correct altitude is c. However, if the majority voting is performed only among the freshly produced, i.e., timely, data, then the correct altitude (a) can be obtained. In this example early values should be considered *invalid* for voting.

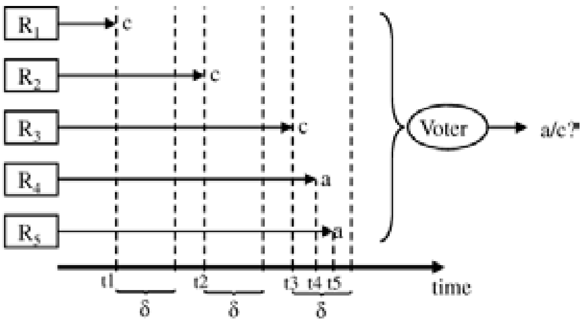


Figure 4.5: Voting dilemma

On the other hand, let us assume that R_1 to R_5 in Figure 4.5 represent acceleration sensor replicas in an airbag system on a car. These sensors are triggered as soon as a crash has detected to measure the intensity of the crash. The sensors output value c if the detected acceleration during the crash is greater than a threshold value to indicate that airbags should be inflated, and value a otherwise. Since the values in this example are not dynamically changing as in the altitude measurement example, the voting should be performed in all received values rather than the timely values alone.

The correct functioning of our methodology relies on two conditions regarding the permissible number of errors for various types:

Condition 1 The number of timely replica outputs should be greater than or equal to the minimum number of replica outputs required for consensus in the time domain (M_t):

$$N - F_t \geq M_t$$

where N is the total number of replica tasks, F_t is the number of replica outputs with incorrect timing, and M_t is either majority $(M_t \geq \lceil \frac{N+1}{2} \rceil)$ or plurality $(M_t \leq N/2)$ in time domain.

Condition 2 Depending on the validity of early replica outputs we have two different scenarios:

- **Early values are invalid:** In order to achieve a consensus in the value domain, the number of replica outputs with correct values should be greater than or equal to the number of minimum number of replicas required to achieve consensus in value domain (M_v). However, since timeliness is a precondition for value correctness, i.e., since we want to compare only relatively timely replica outputs and we cannot wait for late outputs, the replica outputs forming the consensus in value domain needs to be free from any type of errors:

$$N - F \geq M_v$$

where F is the number of replica outputs with incorrect timing and/or value, and M_v is either majority $(M_v \geq \lceil \frac{N+1}{2} \rceil)$ or plurality $(M_v \leq N/2)$ in value domain.

- **Early values are valid:** The number of error-free replica outputs, except the ones with early timing errors, must be greater than or equal to the minimum number of replica outputs required to achieve consensus in the value domain (M_v):

$$N - (F - F_t^e) \geq M_v$$

where F is the number of replica outputs with incorrect timing and/or value and F_t^e is the number of replica outputs with early timing error.

In this thesis, we have provided an algorithm to be run on the voter which performs the voting in both value and time domains. The output of our voter algorithm is the correct value delivered within the feasible time window provided that the conditions on the maximum number of errors that can occur hold. In case conditions are violated, the algorithm enables the voter to provide information about such violations to the rest of the system.

Chapter 5

Cascading Redundancy in Real-Time Systems

In this chapter, we present an overview of our cascading redundancy framework capable of tolerating errors with a wider coverage (when considering both error frequency and error types) than our time and space redundancy techniques in isolation. The framework handles tasks with various criticality levels for more flexibility, and, above all, ensures that every critical task instance can be feasibly replicated in both time and space, independent of the real-time scheduling policy.

5.1 System Model

In our cascading redundancy approach, we assume a distributed real-time architecture consisting of processing nodes, sensors and communication media. Similar to the system model in our space redundancy approach, each node has its own clock allowed to drift from the correct time (i.e., seen by a perfect observer) by a maximum permissible deviation δ which is bounded by relatively inexpensive clock synchronization algorithms implemented in software (as opposed to expensive tight clock synchronization implementations) and reliable communication techniques that have bounded message transmission times.

On each processing node, a periodic task set is allocated, where each task represents a real-time thread of execution with a specified criticality. We assume that tasks' deadlines are equal to the end of their periods. The criticality of a task is a combined measure which indicates the severity of the consequences caused by its failure together with its vulnerability to different types of errors, as well as different error intensities. Without loss of generality, in this framework, we specified four different criticality levels, addressing different error scenarios:

1. **Non-critical tasks:** The failure of these tasks does not have any major impacts on the system performance.

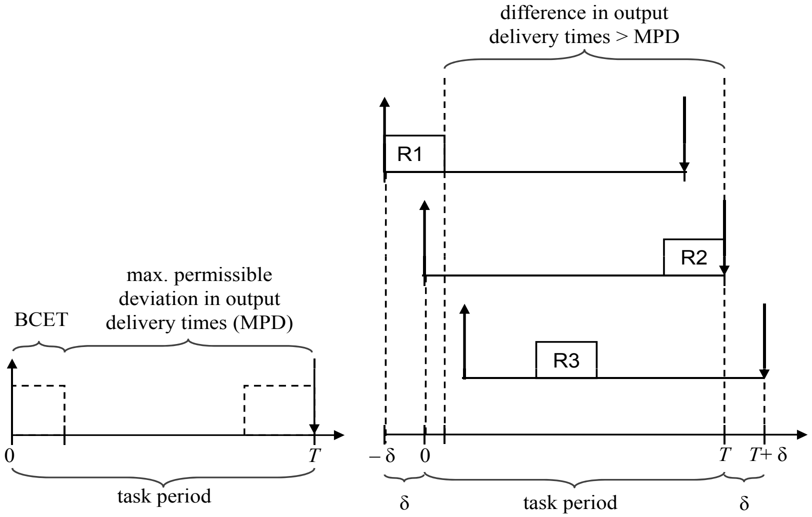
2. **Critical tasks:** These tasks are vulnerable to value errors of transient or intermittent nature.

3. **Highly-critical tasks:** These tasks are vulnerable to both value and time errors of transient, intermittent or even permanent nature.

4. **Ultra-critical tasks:** These tasks are vulnerable to similar types of errors as *highly-critical tasks*. However, the intensity of errors and/or the severity of its failure consequences are higher.

Compared to the task model in our space redundancy approach, we relax the assumption of starting the replicas at the same time and performing the requested operations simultaneously, by allowing them to execute and complete anywhere during their periods. In a system with no redundancy, or in a redundant system with tight synchronization, the deviation in output delivery times of a task is less than or equal to the maximum permissible deviation (MPD) that is equal to its feasibility window, i.e., the time interval between its release time and deadline, minus its best case execution time ($BCET$) (Figure 5.1 (a)). However, in loosely synchronized redundant systems, local clocks on the processing nodes are allowed to drift by a specified value (δ), which can, in certain scenarios, result in delivery of replica outputs that are farther apart from each other than they are designed for with respect to time (Figure 5.1 (b)), and cause system failures.

We assume that the total utilization of the critical, highly-critical, and ultra-critical tasks together with all the alternate tasks (or the primary re-executions) on a processing node is less than or equal to 100%. This will imply that, during error recovery, the execution of non-critical tasks cannot be permitted as it may result in overload conditions. We assume that the scheduler has adequate support for flagging non-critical tasks as unschedulable during such scenarios, along with appropriate error detection mechanisms for the errors that can be tolerated by time redundancy.



(a) Single processor (b) Loosely synchronised multiple processors

Figure 5.1: The timing issue in redundant real-time systems

We assume that the value errors caused by a large variety of transient and intermittent hardware faults can effectively be tolerated by a simple re-execution of the affected task, whilst the value errors caused by software design faults could be tolerated by executing an alternate action such as recovery blocks or exception handlers. Both situations could be considered as executions of another task (either the primary itself or an alternate) with a specified computation time requirement. On the other hand, in addition to all types of errors that can be tolerated by a time redundancy mechanism, value errors caused by permanent hardware faults, and timing errors can be tolerated by a space redundancy mechanism. If ultimate dependability is desired, using both approaches together will provide recovery from a wide range of errors, as well as from an increased number of error occurrences, with the obvious additional cost. The error type coverage achieved by each technique is shown in Figure 5.2.

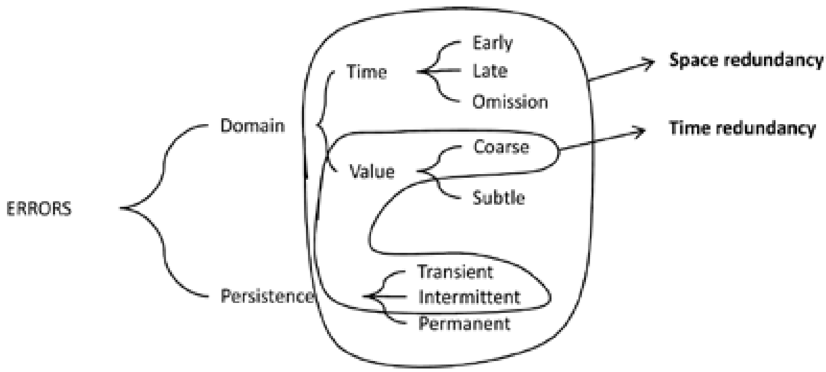


Figure 5.2: Coverage of error types achieved by time and space redundancy

5.2 Methodology

The cascading redundancy approach addresses the different error and cost models of tasks with various criticality by allowing the following configuration levels for each predetermined criticality:

1. no redundancy for *non-critical tasks*
2. only time redundancy for *critical tasks*
3. only space redundancy for *highly-critical tasks*
4. both time and space redundancy for *ultra-critical tasks*

Highly-critical and ultra-critical tasks are replicated, and the replica outputs are voted by the voting mechanism implemented on a a stand-alone node, to ensure correctness in both value and time. Upon receiving identical requests or inputs, replicas start their executions on separate processors whose clocks are allowed to drift from each other by a maximum deviation. When the highly-critical task replicas complete their executions, the outputs are sent to the stand-alone voter. For ultra-critical tasks, before sending the outputs to the voter, error detection for transient coarse value errors and re-executions of primaries or executions of alternates are performed upon error occurrences. Upon receiving the required replica values, the voter starts executing the voting algorithm and sends the agreed value back to the processing nodes, or signals the non-existence of a correct output before the latest deadline among the tasks whose outputs are being subjected to voting. In some cases, however, the voter output may be delivered after one, or several task deadlines. This scenario can occur in cases when the last required replica output is provided close to its deadline, while the first replicas are executing on nodes drifting ahead of the clock, thus, with deadlines before the latest admissible voter output. In these cases, the schedules on nodes which are drifting ahead of time are rolled back to the end of the replica task period, with adequate checkpointing mechanisms. This action will ensure that the subsequent tasks will process the consensus value delivered by the voter and, in the worst case, force the nodes executing such tasks to drift from the correct time by maximum δ, which is still admissible with respect to the real-time specifications.

As the replicas of a task need to agree on an output value, the only feasible order for performing cascading redundancy is first time redundancy followed by space redundancy. Whenever a consensus is achieved for a replicated critical task, any uncompleted replica instances, or their alternate actions becomes obsolete for the purpose of voting, thus, can be shed to improve the service to the non-critical tasks. In Figure 5.3, neither the ultra-critical task B on Node 3 nor its alternate is executed since the voter reports that majority has been formed before B starts executing on that node.

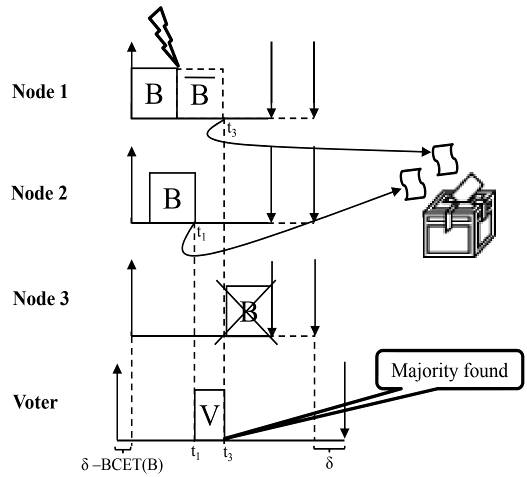


Figure 5.3: Cascading redundancy

Chapter 6

Conclusions and Future Work

In this thesis, we have presented methods that improve the state of the art for development of dependable real-time embedded systems. In particular, taking Laprie's definition of dependability as a reference [10, 2], we have mainly focused on the reliability attribute of dependability, contributed with a comprehensive error modeling approach and new fault tolerance techniques applicable to dependable real-time embedded systems (Figure 6.1).

The main goal of our proposed fault tolerance techniques is to ensure that the delivered services are both correct value-wise and timely, i.e., not too late or too early. Furthermore, as our domain of interest is embedded systems which are often constrained with respect to available computational resources, we pay special attention to the usage of such resources in our proposed techniques. We provide several degrees of fault tolerance corresponding to different levels of error coverage, each of which has different resource requirements.

6.1 Contributions

The following are the main contributions of this thesis:

Error modeling We present our findings from our survey of various error classifications and failure modes in the literature with the aim of identifying

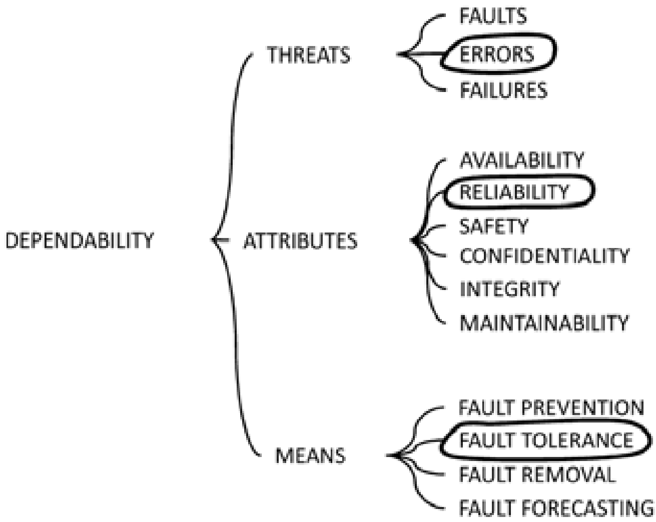


Figure 6.1: Thesis research focus, highlighted on the Dependability Tree [2]

their relations/contrasts as well as in arriving at an 'all-encompassing compilation of classifications'. We also discuss the effects of various error handling mechanisms on error propagation.

Time redundancy technique in fixed-priority scheduling We present a fault tolerant scheduling framework which allows the system designer to schedule a set of real-time tasks with mixed criticalities and FT requirements. Our main contribution is a methodology which guarantees every critical task instance to be re-executed upon an error before its deadline, provided the combined utilization of primaries and alternates is less than or equal to 100%. Additionally our methodology allows non-critical tasks to execute at priority levels higher than the critical ones, in an error-aware manner. Hence, the non-critical tasks are provided a better service than using background scheduling.

Space redundancy technique with voting on time and value We present a new voting strategy, Voting on Time and Value (VTV), for redundant real-time systems, to explicitly consider both value and timing failures for achieving fault tolerance in real-time applications. Our method produces the correct output value, as well as identifies the correct window of time in which the output has to be delivered, provided the conditions on the maximum number of permissible failures are not violated. Moreover, our method is capable of providing information about violation of such conditions, if any. We present algorithms for the particular case where the nodes are triplicated, as well as a generalized version that enables a high degree of customization with respect to the number of replicas as well as the validity of early values for the purpose of voting.

Cascading redundancy technique We present a framework that enables the use of both time and space redundancy in dependable real-time systems. We propose a cascading redundancy approach that is capable of tolerating a wider range of errors (with respect to error frequency and error types) than either time or space redundancy alone. The approach is independent from the underlying scheduler, can cope with tasks of mixed criticality levels, and guarantees the feasible time and space replication of every critical task instance while fully utilizing the resources.

6.2 Future work

This thesis work brings possibilities to conduct further research in certain areas that are not thoroughly addressed. Some of these possibilities include:

- Adding formalizations to the error modeling approach including modeling of the error propagations and the effects of error handling mechanisms.

- Development of error handling mechanisms for consistency and homogeneity related errors.

- Incorporating more complex error models to our time redundancy approach by enabling different assumptions on the maximum frequency of errors.

- Handling babbling idiot errors in our space redundancy approach.

- Defining an end-to-end error model for component-based embedded software systems and adapting this work to the Progress Component Model [34].

- Investigating the task distribution and migration aspects together with on-line adaptations in our cascading redundancy approach, in order to optimize the schedulability of the lower critical tasks, e.g., non critical, in error free scenarios.

Bibliography

[1] J. Von Neuman. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata Studies*, 34:43–98, 1956.

[2] A. Avizienis, J. Laprie, and B. Randell. Fundamental concepts of dependability. *UCLA CSD Report no. 010028, LAAS Report no. 01-145, Newcastle University Report no. CS-TR-739*, 2001.

[3] D. Powell. Failure mode assumptions and assumption coverage. In *Proceedings of the 22nd International Symposium on Fault-Tolerant Computing*, pages 386–395, 1992.

[4] A. Bondavalli and L. Simoncini. Failure classification with respect to detection. In *Proceedings of the 2nd IEEE Workshop on Future Trends in Distributed Computing*, pages 47–53, 1990.

[5] M. Hiller, A. Jhumka, and N. Suri. EPIC : Profiling the propagation and effect of data errors in software. *IEEE Transactions on Computers*, 53(5):512–530, 2004.

[6] K. Echtle and A. Masum. A fundamental failure model for fault-tolerant protocols. In *Proceedings of the Computer Performance and Dependability Symposium*, pages 69–78, 2000.

[7] J.A. Stankovic and K. Ramamritham, editors. *Hard Real-Time Systems Tutorial*. IEEE-Computer Society Press, 1988.

[8] J.A. Stankovic. Misconceptions about real-time computing. *Computer*, pages 10–19, October 1988.

[9] L. Sha, T.F. Abdelzaher, K-E. Årzén, A. Cervin, T.P. Baker, A. Burns, G.C. Buttazzo, M. Caccamo, J.P. Lehoczky, and A.K. Mok. Real time

scheduling theory: A historical perspective. *Real-Time Systems*, 28(2-3):101–155.

[10] J-C. Laprie. Dependable computing and fault-tolerance: Concepts and terminology. In *Proceedings of the 25th International Symposium on Fault-Tolerant Computing, ' Highlights from Twenty-Five Years'.*, 1995.

[11] N.G. Leveson. Software safety: Why, what and how. *ACM Computing Surveys*, 18(2):125–164, 1986.

[12] R. Obermaisser and P. Peti. A fault hypothesis for integrated architectures. In *Proceedings of the 4th Workshop on Intelligent Solutions in Embedded Systems*, June 2006.

[13] C.J. Walter and N. Suri. The customizable fault/error model for dependable distributed systems. *Theoretical Computer Science*, 290(2):1223–1251, 2003.

[14] H. Kopetz. On the fault hypothesis for a safety-critical real-time system. In *Proceedings of the Workshop on Future Generation Software Architectures in the Automotive Domain*, January 2004.

[15] J.J. Horning, H.C. Lauer, P.M. Melliar-Smith, and B. Randell. A program structure for error detection and recovery. *Lecture Notes in Computer Science*, 16:177–193, 1974.

[16] A.L. Liestman and R.H. Campbell. A Fault-Tolerant Scheduling Problem. *IEEE Transactions on Software Engineering*, 12(11):1089–95, 1986.

[17] C.M. Krishna and K.G. Shin. On scheduling tasks with a quick recovery from failure. *IEEE Transactions on Computers*, 35(5):448–455, 1986.

[18] M. Pandya and M. Malek. Minimum achievable utilization for fault-tolerant processing of periodic tasks. *IEEE Transactions on Computers*, 47(10), 1998.

[19] S. Ramos-Thuel and J.K. Strosnider. The transient server approach to scheduling time-critical recovery operations. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 286–295, 1991.

[20] S. Ghosh, R. Melhem, and D. Mosse. Enhancing real-time schedules to tolerate transient faults. In *Proceedings of the IEEE Real-Time Systems Symposium*, 1995.

[21] A. Burns, R.I. Davis, and S. Punnekkat. Feasibility analysis of fault-tolerant real-time task sets. In *Proceedings of the IEEE Euromicro Real-Time Systems Workshop*, pages 29–33, 1996.

[22] C-C. Han, K.G. Shin, and J. Wu. A fault-tolerant scheduling algorithm for real-time periodic tasks with possible software faults. *IEEE Transactions on Computers*, 52(3):362–372, 2003.

[23] A.L. Hopkins, T.B. Smith, and J.H. Lala. FTMP: A highly reliable fault-tolerant multiprocessor for aircraft. *Proceedings of the IEEE*, 66(10):1221–1239, 1978.

[24] J.H. Wensley, L. Lamport, J. Goldberg, M.W. Green, K.N. Levitt, P.M. Milliar-Smith, R.E. Shostak, and C.B. Weinstock. SIFT: Design and analysis of a fault-tolerant computer for aircraft control. *Proceedings of the IEEE*, 66(10):1240–1255, 1978.

[25] V. De Florio, G. Deconinck, and R. Lauwereins. The EFTOS voting farm: A software tool for fault masking in message passing parallel environments. In *Proceedings of the 24th IEEE Euromicro Conference*, pages 379–386, 1998.

[26] R.E. Lyons and W. Vanderkulk. The use of triple-modular redundancy to improve computer reliability. *Journal of Research and Development*, 6:200–209, 1962.

[27] D.M. Blough and G.F. Sullivan. A comparison of voting strategies for fault-tolerant distributed systems. In *Proceedings of the 9th Symposium on Reliable Distributed Systems*, pages 136–145, 1990.

[28] F. Di Giandomenico and L. Strigini. Adjudicators for diverse-redundant components. In *Proceedings of the 9th Symposium on Reliable Distributed Systems*, pages 114–123, 1990.

[29] G. Latif-Shabgah, J.M. Bass, and S. Bennett. A taxonomy for software voting algorithms used in safety-critical systems. *IEEE Transactions on Reliability*, 53(3):319–328, 2004.

[30] P.R. Lorczak, A.K. Caglayan, and D.E. Eckhardt. A theoretical investigation of generalized voters for redundant systems. In *Proceedings of the 19th International Symposium on Fault-Tolerant Computing*, pages 444–451, 1989.

[31] A. Grnarov, J. Arlat, and A. Avizienis. Modeling of software fault-tolerance strategies. In *Proceedings of the 11th Annual Pittsburgh Modeling and Simulation Conference*, pages 571–578, 1980.

[32] J. Lala, L. Alger, S. Friend, G. Greeley, S. Sacco, and S. Adams. An analysis of redundancy management algorithms for asynchronous fault tolerant control systems. *Research Report NASA-TM-100007, NASA*, 1987.

[33] K.G. Shin and J.W. Dolter. Alternative majority-voting methods for real-time computing systems. *IEEE Transactions on Reliability*, 38(1):58–64, 1989.

[34] T. Bures, J. Carlson, I. Crnkovic, S. Sentilles, and A. Vulgarakis. Procom - the progress component model reference manual, version 1.0. *Mälardalen University, Technical Report ISSN 1404-3041 ISRN MDH-MRTC-230/2008-1-SE*, 2008.