Fault Tolerant Mechanism for Space Distributed Systems

Internship Report

Armin Purle-Kopacz ***(s3052907)***

MSc Embedded Systems
04.11.2024 to 21.02.2025

Deutsche Zentrum für Luft- und Raumfahrt (DLR)
Lilienthalplatz 7
38108 Braunschweig
Germany

Supervisor
Dr. -Ing. Zain Alabedin, Haj Hammadeh
Researcher
zain.hajhammadeh@dlr.de

University of Twente
Drienerlolaan 5
7522 NB Enschede
Netherlands

Academic Supervisor
Dr. Ir. Marco Ottavi
m.ottavi@utwente.nl

Version: 1.2
February 26, 2025

PREFACE & ACKNOWLEDGEMENT

From November 2024 until February 2025, I did an Internship at Deutsches zentrum für Luft und Raumfahrt. DLR is the German aerospace research and technology center (or so-called German Space Agency). In its core areas, DLR develops technologies for aeronautics and space, energy and transport, as well as security and defense research. The location of DLR's institutions are presented in figure 1. This internship is part of my master program conducted at University of Twente, Netherlands.

Figure 1: DLR Institutions

The primary objective of this internship project was to evaluate the feasibility and develop a prototype for a fault-tolerant mechanism designed for deployment in a distributed system intended for use in Space.

This topic aligned perfectly with my background, as it required both advanced software development skills and a strong understanding of embedded systems. Throughout the internship, I had the opportunity to apply and refine my existing skills while also gaining valuable experience and knowledge about the unique challenges of the Space environment.

Beyond enhancing my technical expertise, this experience allowed me to develop professionally in a real-world work setting. Equally important was the growth of my communication skills, which improved through interactions with colleagues and supervisors, as well as through delivering presentations, conducting demos, and engaging in discussions and debates.

Therefore, I would like to extend my deepest gratitude to Dr. -Ing. Zain Alabedin Haj Hammadeh, my supervisor at DLR, for his invaluable support throughout my journey. His guidance was instrumental not only during my internship but also in helping me navigate the many chal-

lenges of settling into a new country. From assisting with paperwork and life logistics in Germany to providing insightful mentorship in the workplace, he made my transition seamless and enriching. Under his supervision, I had the opportunity to grow both technically and professionally, gaining a deeper understanding of the company, its projects, and its structure. He taught me how to present my ideas effectively, conduct stakeholder demos, and defend my work with confidence. More than just a mentor in the workplace, Zain's encouragement and advice have had a lasting impact on my aspirations as a person and my vision for the future. For all of this, I am truly grateful. I would also like to express my sincere appreciation to Dr. ir. Marco Ottavi for his permission to be my academic supervisor and, more importantly, for his enthusiastic encouragement and invaluable guidance throughout my internship. His support extended beyond technical aspects, providing me with essential advice and assistance that greatly contributed to my growth. His insights and mentorship have been instrumental in shaping both my academic and professional development, and I am deeply thankful for his support.

Additionally, I would like to thank my internship coordinator, as well as everyone from DLR and the University of Twente, for their support in making this opportunity possible. Their efforts and dedication have enabled me to embark on this invaluable experience, which has profoundly influenced both my personal and professional journey. This experience has been transformative and will continue to have a lasting impact on my life and career.

I would like to also acknowledge that while the content of the report is entirely original, some phrases have been rephrased with the use of LLMs.

SUMMARY

Distributed systems have proven highly valuable in terrestrial environments, offering scalability and flexibility. Their application in Space, however, remains relatively recent due to the unique challenges posed by cosmic radiation, communication latencies, and severe resource constraints. Over the course of this discussion, a Fault-Tolerant Mechanism was proposed to address the difficulties inherent in maintaining reliable distributed systems in orbit. By combining established strategies from ground-based distributed computing with specific adaptations for Space, the FTM seeks to handle node and application failures effectively, ensuring that the system avoids endless reconfiguration cycles caused by faulty software components.

A key motivation for introducing distributed architectures in Space lies in the need for higher computational power, which space-graded hardware alone may not reliably provide due to its slow certification process and limited performance capabilities. Employing a mix of Coordinator Nodes and Worker Nodes can meet emerging demands for intensive onboard data processing. Nonetheless, this arrangement requires effective fault management, especially since non-space-graded nodes are more susceptible to radiation-induced malfunctions, single event upsets, and thermal stress. The FTM design addresses these risks by recognizing the nature of failures—whether they stem from hardware fragility, environmental factors, or application bugs—and reacting accordingly.

To understand the necessity of such a mechanism, a problem scenario was outlined in which repeated crashes of a faulty application trigger never-ending reconfiguration cycles, making essential services like telemetry and telecommand unavailable at critical moments. In a traditional environment, these issues might be resolved via manual interventions or simpler failover mechanisms, but in Space, communication delays and hardware constraints complicate timely responses. The FTM seeks to provide an autonomous solution that eliminates faulty applications quickly and reshapes the system configuration to preserve overall functionality. Its design leverages well-known techniques such as crash tables and issue weight tables. Crash tables count how often each application causes node failures, while issue weight tables classify and weight different failure types according to their impact. A confidence index is then computed to identify if an application is genuinely faulty or if a hardware error is more likely to blame. Tasks within an application are also considered individually so that only the truly problematic portions are halted, reducing disruption to other critical services.

To demonstrate feasibility, the FTM was implemented and integrated with an existing distributed system framework designed for Space. The system relies on specialized services: a System Monitor tracks node health, a TMR Service uses triple modular redundancy to counter soft errors, a Periodic Checkpoint Service stores snapshots for rollback, the System Reconfiguration service helps with changing the system's configuration, an Application Error Alert Service report any known errors to the system, a System Update Service that handles updates to any part of the system and a Telemetry/Telecommand Service allows ground operators to manage the spacecraft remotely. The FTM interacts with these services to coordinate reconfiguration requests, gather relevant logs, and maintain data essential for diagnosing issues. A dynamic configuration module reorganizes the cluster based on the coordinator node's instructions, while a data collection module periodically retrieves logs from each node so that valuable evidence is not lost in the event of a crash.

Validation efforts involved extensive unit and integration tests. Unit tests confirmed the internal logic of the FTM: crash tables, the confidence index, log analysis, and dynamic configuration updates. Integration tests simultaneously launched multiple processes to emulate the spacecraft's nodes, verifying that the FTM could handle scenarios where only specific tasks of an application fail, multiple tasks fail, or every application fails. These tests indicated that the FTM's simplified approach is sufficient to detect and isolate faulty applications under controlled conditions. However, the solution's performance in real missions—both in terms of resource consumption and timing—remains unquantified. Understanding how quickly the system recovers from failures, or how its resource usage scales with larger deployments, requires deeper analysis and real mission data.

Discussions highlighted that, while Space introduces additional constraints such as limited power and infrequent maintenance, the fundamental ideas of replication, checkpointing, consensus protocols, and thorough failure detection are readily adaptable from terrestrial distributed systems. What changes is the need to respond autonomously and efficiently under extreme conditions and with minimal human intervention. The internship objectives—gaining an understanding of fault detection and recovery in Space, familiarizing oneself with onboard software architectures, building and integrating an FTM, and validating its correctness—were all met. The feasibility of the concept is established, but future steps would involve collaborating with mission stakeholders to define performance requirements more precisely, carrying out detailed resource and timing analyses, and refining the mechanisms for even better efficiency. Additional improvements might include optimizing table look-ups or merging the Data Collection component with existing system logging to reduce redundancy.

In conclusion, the FTM shows promise in preventing system-wide breakdowns caused by recurring application faults in a space-deployed distributed system. Although the mechanism's effectiveness was demonstrated through unit and integration tests, the next phase of research should delve into performance measurements, mission-specific constraints, and stakeholder requirements. By addressing these areas, the proposed FTM can be refined into a robust, mission-ready solution capable of handling the rigors of orbital and deep-Space operations.

CONTENTS

Preface & Acknowledgement **1**

Summary **3**

1 INTRODUCTION **9**
1.1 Background . . . 9
1.1.1 What is a Distributed System? . . . 9
1.1.2 Challenges of Space . . . 10
1.2 FTM in Literature . . . 11
1.2.1 General Approaches to Fault-Tolerant Mechanisms . . . 11
1.2.2 Distributed Systems in the Context of Space . . . 11
1.3 Objectives and Motivation . . . 11
1.3.1 Business Requirements . . . 12
1.3.2 Software Requirements . . . 12
1.3.3 Safety/Security/Legal Requirements . . . 13

2 Problem Analysis **14**
2.1 Study's Distributed System . . . 14
2.1.1 System Monitor Service . . . 15
2.1.2 TMR Service . . . 15
2.1.3 System Reconfiguration Service . . . 15
2.1.4 Application Error Alert Service . . . 15
2.1.5 Periodic Checkpoint Service . . . 16
2.1.6 System Update Service . . . 16
2.1.7 Telemetry and Telecommand Service . . . 16
2.2 The Problem . . . 16

3 Approach and Method **20**
3.1 Assumptions . . . 20
3.2 Constraints . . . 20
3.3 FTM Design . . . 21
3.3.1 Service Dependencies . . . 21
3.3.2 FTM Modules . . . 21
3.3.3 FTM settings . . . 24
3.4 Measurement Setup . . . 24

4 Results **26**
4.1 General Approach . . . 26
4.2 Unit Tests . . . 26
4.3 Integration Tests . . . 27

5 Discussion **28**

6 Conclusions and Recommendations **29**

References **30**

List of Abbreviations and Symbols **31**

List of Figures

1 DLR Institutions 1

1.1 Traditional Cluster Computing System (adapted from [1] and sourced from [2]) . 10

2.1 Nominal Configuration 17
2.2 Configuration 1 17
2.3 Configuration 2 18

3.1 System Services 21

List of Tables

1.1 Business Requirements 12
1.2 Non-functional software requirements 13
1.3 Functional software requirements 13
1.4 Safety, Security and Legal Requirements 13

3.1 Assumptions 20
3.2 Crash Table 22
3.3 Issue Weight Table 23
3.4 FTM settings Variables 24

1 INTRODUCTION

To fully comprehend the necessity and application of FTMs, particularly within the context of Space systems, it is imperative to first establish a fundamental understanding of distributed systems and the environments in which they operate. Therefore, this chapter provides an overview of the foundational concepts, objectives, and a concise summary of existing FTMs documented in the literature.

The subsequent chapters systematically explore the problem that the FTM seeks to address (§2). The next two chapters present the proposed solution (§3) and analyze the corresponding results (§4). This is followed by a discussion (§5) that juxtaposes our approach against existing methodologies in the literature while revisiting the primary objectives of this work. Finally, the report culminates in a concluding chapter that summarizes key insights and provides recommendations for future advancements (§6).

1.1 Background

In essence, a distributed system comprises a collection of independent computing entities that communicate over a network to achieve a shared objective. Despite their physical separation, these entities function in a coordinated manner, presenting themselves to users as a unified system.

In our specific case, the distributed system under study is intended for deployment aboard spacecraft. Consequently, it must be designed to withstand the unique challenges posed by the Space environment.

1.1.1 What is a Distributed System?

To better contextualize our study, this report refers to the classification provided in the textbook used for the Distributed Systems course [2]. Our distributed system falls within the category of cluster computing systems. A conventional cluster computing system is illustrated in Figure 1.1. The primary distinctions between a traditional cluster and the system in our study lie in the fact that our system is hardware-agnostic and its nodes are not restricted to executing a single application.

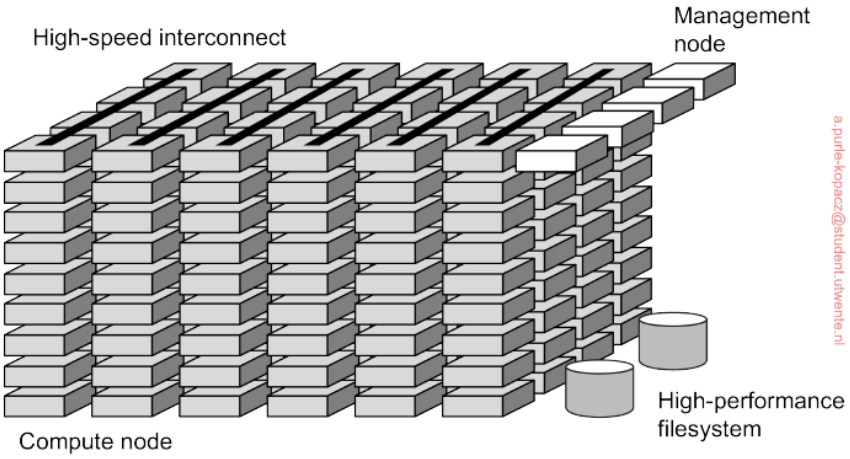

Figure 1.1: Traditional Cluster Computing System (adapted from [1] and sourced from [2])

Several key characteristics define distributed systems. Given that multiple components must operate concurrently, it is essential to implement mechanisms that effectively manage parallel operations. Additionally, due to the absence of a global clock, nodes rely on message passing for synchronization [3]. Furthermore, a distributed system must be designed to gracefully handle partial failures, ensuring that localized disruptions (such as individual node failures) do not compromise overall system functionality [4].

The principal advantages of distributed systems include their high reliability—an indispensable attribute for Space applications, as elaborated in Section §1.1.2. Additionally, distributed systems inherently offer excellent scalability by enabling the integration of additional nodes [5]. However, these benefits come at a cost. Notable challenges include the increased complexity associated with coordinating independent components, maintaining data consistency across nodes, and addressing security concerns stemming from the critical communication dependencies between system elements [3, 4, 5].

1.1.2 Challenges of Space

Space presents a unique set of challenges for distributed systems due to extreme environmental conditions, limited available resources, and the vast distances separating deployed systems from Ground Station Control.

From an environmental perspective, spacecraft are continuously exposed to high levels of cosmic and solar radiation. On Earth, computers and electronic equipment benefit from the protective shielding of the atmosphere and magnetic field, but in Space, these protective layers are absent. Consequently, radiation-induced disruptions such as SEUs [6] can lead to system failures or even cause permanent damage to electronic components [7]. Another significant challenge is the extreme temperature variations that occur due to the lack of an atmosphere. A spacecraft can simultaneously experience extreme heat on one side while facing intense cold on the other. Maintaining optimal operating conditions for computing equipment under such conditions presents a major engineering challenge.

Resource limitations in Space further complicate system operations, with power availability being one of the most critical constraints. Currently, spacecraft rely on solar panels for energy generation. However, for satellites in Earth orbit, there are periods when they pass through Earth's shadow, rendering solar energy temporarily unavailable. Furthermore, due to cost, weight, and Space constraints imposed by both launch providers and spacecraft designers, increasing battery capacity or adding larger solar panels is not always feasible. This makes energy efficiency a critical factor in system design [8].

Additionally, the immense distances between spacecraft and Ground Station Control result in

high communication latency and intermittent connectivity. As a result, spacecraft must be capable of operating with a significant degree of autonomy, handling faults independently without relying on immediate human intervention.

1.2 FTM in Literature

FTMs are not a novel concept, particularly in the domain of distributed systems. A robust FTM is essential for ensuring the stability and reliability of a DS, allowing it to maintain continuous operation despite failures.

1.2.1 General Approaches to Fault-Tolerant Mechanisms

Various strategies are employed in fault-tolerant mechanisms for general distributed systems. The most significant among them include:

- **Replication:** By maintaining multiple copies of data or services across different nodes, the system can continue to function even if one node fails. Replication is implemented using two primary approaches: ”active” replication, in which identical services run on multiple nodes and process the same requests simultaneously, and ”passive” replication, where backup nodes are periodically updated [9].
- **Checkpointing and Rollback Recovery:** The system periodically saves its operational state so that in the event of a failure, it can revert to the last saved state, thereby minimizing data loss and system downtime [9].
- **Consensus Protocols:** These protocols are crucial for maintaining data consistency across distributed nodes. They ensure that all nodes reach agreement on system-wide decisions despite failures [10].
- **Failure Detection:** A fault-tolerant system cannot function effectively without reliable fault detection mechanisms. It is essential for the system to identify node or process failures using techniques such as heartbeat monitoring and timeout-based failure detection [7].

1.2.2 Distributed Systems in the Context of Space

Due to the relative scarcity of distributed systems deployed in Space, there is limited literature on their design and implementation. However, in addition to the general fault-tolerant mechanisms discussed in Section §1.2.1, designing an FTM for space-based distributed systems necessitates special considerations. These include mitigating the effects of radiation-induced faults, addressing the challenges of limited communication and maintenance procedures [7], and optimizing for resource constraints [11].

1.3 Objectives and Motivation

With a clear understanding of the context, the assessment of the necessity regarding the FTM designed and implemented in this study can be done. A more comprehensive analysis of the problem is presented in Chapter §2.

The extreme conditions of Space introduce multiple potential failure points for computational nodes. Hardware malfunctions may arise due to prolonged exposure to high radiation levels, surpassing the TID threshold, or extreme thermal variations. Additionally, SEUs can disrupt node functionality, often necessitating a system reboot. Apart from these hardware-related failures, software anomalies and application crashes further exacerbate system instability.

Given these challenges, the primary objective of the FTM is to effectively distinguish between failures caused by environmental conditions and those originating from faulty applications. Once an application is identified as malfunctioning, the mechanism must ensure its timely termination to prevent cascading system failures and maintain overall stability.

To achieve this objective, the following requirements (§1.3.1, §1.3.2, §1.3.3) have been formulated. Since this study does not focus on the broader domain of systems engineering, only high-level implementation aspects have been considered to facilitate an efficient workflow. While the outlined requirements address key concerns, additional specifications could be introduced depending on the desired level of granularity. Furthermore, fundamental concepts such as services, reconfiguration mechanisms, and other critical system components, which have yet to be discussed, will be explored in greater detail in Chapter §2.

The general objectives of this internship include:

- Gaining an understanding of Fault Detection, Isolation, and Recovery in Space applications.
- Learning about the architecture of distributed onboard software.
- Developing and implementing the mitigation mechanism using C++.
- Integrating the mechanism with the existing reconfiguration framework.
- Testing the implementation under various failure scenarios.

1.3.1 Business Requirements

The business requirements outlined in Table 1.1 define the fundamental needs of the FTM within the study case's distributed system.

Business Requirements	
ID	**Description**
BR_REQ-FTM001	The service shall estimate the APP that is crashing nodes
BR_REQ-FTM002	The service must detect the reason of the nodes' crash
BR_REQ-FTM003	The service shall prevent the system from chain or repeated reconfiguration
BR_REQ-FTM004	The service shall collect data about the APP's fault
BR_REQ-FTM005	The service must send the collected data to the Telemetry Service

Table 1.1: Business Requirements

1.3.2 Software Requirements

The more technical requirements, outlined in Tables 1.3 and 1.2, are detailed in this section.

Non-Functional Requirements

Software Requiremetns	
ID	**Description**
SW_NFKT_REQ-FTM001	Collected data shall not exceed the maximum allocated storage space for Telemetry
SW_NFKT_REQ-FTM002	FTM shall be implemented as a separate service alongside the DS' existing services
SW_NFKT_REQ-FTM003	FTM settings shall be stored in a file delivered with the system configuration
SW_NFKT_REQ-FTM004	FTM settings shall take into consideration the estimated SEU rate of the mission orbit

Table 1.2: Non-functional software requirements

Functional Requirements

Software Requiremetns	
ID	**Description**
SW_FKT_REQ-FTM001	The service shall be able to be enabled / disabled by GS operators
SW_FKT_REQ-FTM002	APP developers shall be able to edit the FTM settings
SW_FKT_REQ-FTM003	FTM shall analyze the behaviour of APPs
SW_FKT_REQ-FTM004	The FTM shall be able to access the APP logs and the system logs

Table 1.3: Functional software requirements

1.3.3 Safety/Security/Legal Requirements

Given the level of control that the FTM has over the system, there are critical safety, security, and legal requirements (Table 1.4) that must be considered.

Safety / Security / Legal Requirements	
ID	**Description**
SSL_REQ-FTM001	The FTM shall contain a self-disable mechanism
SSL_REQ-FTM002	When an APP displays security related faults, the FTM shall consider it a faulty APP
SSL_REQ-FTM003	All legal requirements that apply to the study case's distributed system shall also be applied to the FTM

Table 1.4: Safety, Security and Legal Requirements

2 PROBLEM ANALYSIS

While the context of distributed systems and space environments has already been introduced, it is crucial to delve deeper into the necessity of both. space-graded hardware is exceptionally costly, and somewhat counterintuitively, when it comes to computing hardware, it often exhibits lower performance compared to its terrestrial counterparts. Although new hardware continues to be space-graded, the process is highly intricate and time-consuming due to the extensive testing required to validate its reliability under extreme conditions.

To bridge the gap between the need for high-performance computing in Space and the slow progress of space-grade hardware certification, distributed systems present a viable solution. The distributed system DS in this study case employs a coordinator node as the primary coordinator for the cluster. The CN consists of space-graded computing hardware, ensuring its resilience in the harsh Space environment. The remaining nodes in the system, referred to as worker nodes, are not space-graded but are designed to withstand the extreme temperatures encountered in Space. These WNs may not have undergone the rigorous Space qualification tests, or such tests may not have been conducted at all. Despite this, their significantly higher computational performance and substantially lower costs make them an attractive choice.

The system is inherently scalable, allowing for the incorporation of additional CNs to enhance redundancy and multiple WNs to boost computational capabilities. Spacecraft are typically engineered to ensure that all onboard hardware can endure Space conditions for the duration of the mission. However, in this scenario, the primary vulnerability lies within the WNs. Nevertheless, as this is a distributed system, the failure of an individual node does not compromise overall functionality. The system can continue operating and delivering essential services, albeit with one fewer node.

As previously stated, the Space environment presents extreme challenges. Conventional distributed system solutions, while highly effective in terrestrial applications, are insufficient to guarantee the same level of reliability in space-based deployments. The unique adversities of Space necessitate enhanced fault tolerance and resilience strategies tailored specifically for these harsh conditions.

Once a spacecraft is deployed, maintenance opportunities are virtually nonexistent. While in-orbit servicing has been demonstrated in select missions, it remains exceedingly rare, prohibitively expensive, and fraught with risks. Therefore, when applied to unmanned spacecraft and deep-space missions, this distributed system must be engineered to function autonomously, ensuring continuous operation and self-recovery capabilities. Given the impossibility of routine maintenance, the system must achieve an exceptionally high level of reliability, enabling it to withstand the rigors of Space and sustain mission-critical operations without external intervention.

2.1 Study's Distributed System

To fully comprehend the role of the Fault-Tolerant Mechanism within this Distributed System, it is important to highlight some of the key functionalities that the DS already implements.

2.1.1 System Monitor Service

This service employs a heartbeat mechanism to continuously verify the operational status of nodes. In practice, the CN periodically sends heartbeat requests to the WNs. If an WN fails to respond after three consecutive requests, a heartbeat failure is recorded, triggering a system-wide alert.

To prevent chaotic or unpredictable system behavior—the system running without coordination—the DS designates two WNs as observer nodes (Observer 1 and Observer 2). These observer nodes are responsible for monitoring the health of the CN. If both the CN and at least one observer node fail simultaneously, the system enters a predefined ”Safe Mode.” In this mode, all non-essential operations cease, and the system awaits further intervention from Ground Station operators. Operators can interact with the Telemetry Service (§2.1.7) to diagnose and manually resolve the issue.

2.1.2 TMR Service

The TMR Service employs a well-established fault-tolerance technique known as Triple Modular Redundancy. This technique is widely used to mitigate the effects of soft errors in computational tasks. Essentially, a task is executed three times—either concurrently or sequentially—on the same hardware. The system then compares the results and selects the majority outcome as the correct one, ensuring both data integrity and consistency.

If a majority consensus cannot be reached, the system generates an alert, signaling a potential anomaly. This mechanism is particularly useful in detecting SEUs, as unexplained discrepancies in computation results may indicate radiation-induced bit flips. Additionally, the TMR Service enhances overall system reliability by validating task outputs and maintaining data consistency across nodes.

2.1.3 System Reconfiguration Service

The System Reconfiguration Service ensures system adaptability in the event of node failure by dynamically adjusting the system’s operational configuration. For instance, if the DS is initially configured as a three-node cluster and one node becomes unresponsive, the system seamlessly transitions to a two-node cluster. In this process, all tasks previously assigned to the failed node are redistributed among the remaining nodes, allowing uninterrupted operation with minimal downtime.

By implementing this service, the DS maintains a high level of resilience, ensuring that mission-critical processes continue even in the presence of hardware failures. This capability is particularly vital for autonomous Space missions, where in-orbit maintenance and intervention opportunities are extremely limited.

2.1.4 Application Error Alert Service

The distributed system relies on this service to systematically register and report any errors that occur within its various applications. These errors may originate from the TMR Service, which is designed to enhance fault tolerance, or they may stem from internal application issues that are either predefined or already recognized by the system. In such cases, the system is equipped with a dedicated error-handling routine to efficiently manage and mitigate these faults, ensuring continued stability and reliability in operation.

2.1.5 Periodic Checkpoint Service

This service is designed to periodically capture and record the current states of individual nodes and applications to safeguard against potential failures. These recorded states, known as checkpoints, serve as critical recovery points that enable the system to restore a node to its last known state in the event of an unexpected failure or system crash. By maintaining these checkpoints at regular intervals, the service ensures minimal data loss and facilitates recovery, thereby enhancing the overall reliability, resilience, and fault tolerance of the distributed system.

2.1.6 System Update Service

The SUS plays a crucial role in facilitating modifications and enhancements to the distributed system by managing and deploying updates efficiently. These updates can encompass a wide range of system components, including configuration files, operational settings, and even executable binaries essential for system functionality.

All updates within the system are strictly controlled and can only be issued by the Ground Station, ensuring that changes are authorized and properly managed. Before an update is executed, it must first be processed through the TM/TC Service, which acts as a communication gateway to verify, transmit, and start the update process.

Once an update is applied, the system automatically generates a detailed report indicating the outcome of the update process, whether it was successfully implemented or encountered an issue. This report is then transmitted back to the TM/TC Service to be forwarded to the GS.

In the event of an update failure or system instability following the update, the service initiates an automatic rollback mechanism. This mechanism ensures that any unintended or detrimental changes are reverted, restoring the system to its last known stable and functional state. By implementing this structured update process, the SUS enhances the reliability, security, and maintainability of the distributed system while minimizing disruptions and potential risks associated with system modifications.

2.1.7 Telemetry and Telecommand Service

The Telemetry and Telecommand Service serves as the primary communication link between the spacecraft and the Ground Station, enabling seamless data exchange and system oversight.

Telemetry refers to the transmission of diagnostic and operational data from the spacecraft to the Ground Station, providing operators with real-time insights into system status and health metrics. Telecommand, on the other hand, involves commands issued by the Ground Station to the spacecraft, facilitating remote system control and adjustments.

This service plays a pivotal role in fault reporting, allowing the DS to relay information about application malfunctions or anomalies to Ground Station operators. Additionally, it enables remote system recovery procedures, granting operators the ability to restore functionality by executing corrective commands from Earth. As such, the Telemetry and Telecommand Service is an indispensable component in maintaining mission stability and operational integrity.

2.2 The Problem

The issue addressed in this study arises under the following scenario, as illustrated in Figures 2.1, 2.2, and 2.3.

The system consists of three nodes: one Coordinator Node and two Worker Nodes. The WNs are responsible for executing three applications: APP_A, which runs two tasks on a single node; APP_B, with two tasks distributed across both WNs; and APP_C, which consists of a single task. In its nominal configuration, the system operates as depicted in Figure 2.1.

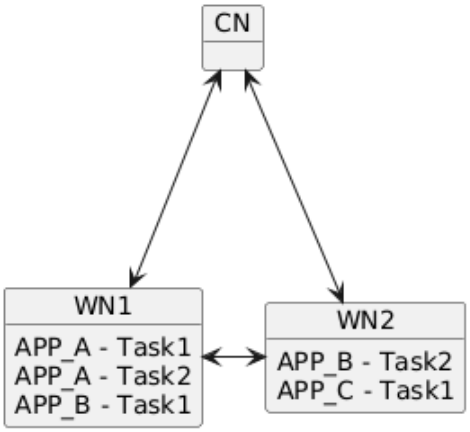

Figure 2.1: Nominal Configuration

When WN2 fails, the system transitions to Configuration 1, in which all applications are migrated to WN1 (Figure 2.2).

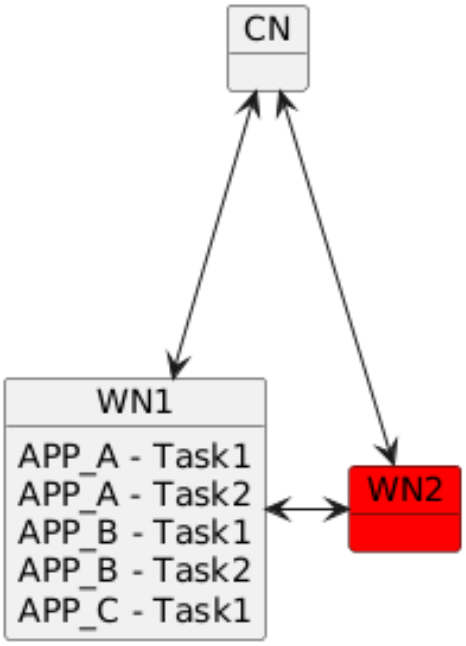

Figure 2.2: Configuration 1

Conversely, if WN1 fails, the system switches to Configuration 2, moving all applications to WN2 (Figure 2.3).

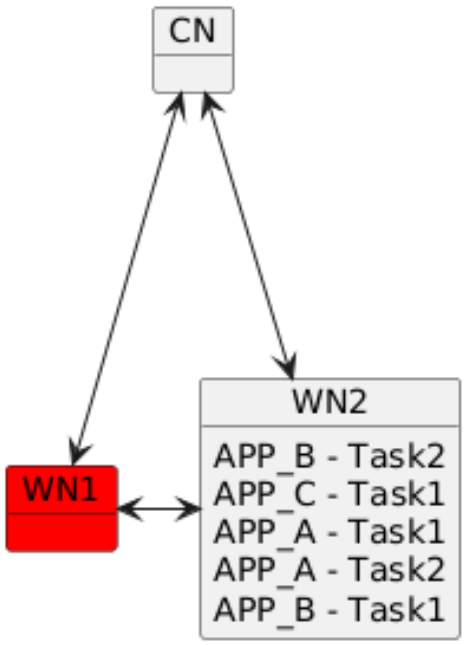

Figure 2.3: Configuration 2

The core problem arises when a faulty application (for instance, APP_C) triggers a node failure. This leads to a continuous reconfiguration cycle between Configuration 1 (Figure 2.2) and Configuration 2 (Figure 2.3).

When WN2 crashes due to APP_C, the application is migrated to WN1. The system then power-cycles the failed node, bringing it back online. However, since APP_C exhibits the same behavior irrespective of the node it is running on, WN1 is also likely to crash when executing it. This results in two problematic scenarios:

If WN1 crashes before WN2 has fully restarted, then as soon as WN2 is available, all applications are transferred back, restarting the failure loop. The system enters an indefinite reconfiguration cycle between Configuration 1 and Configuration 2.

If WN1 remains operational until WN2 comes back online, the system briefly restores the nominal configuration (Figure 2.1). However, once APP_C is executed on WN2, it fails again, triggering a reconfiguration cycle between the nominal configuration and Configuration 1.

In both cases, the system remains stuck in an unstable reconfiguration loop, which severely impacts mission-critical services such as telemetry and telecommands. This is particularly dangerous during contact windows with the Ground Station, where communication outages can lead to mission failure.

On Earth, where hardware failures are less frequent and maintenance or analysis can be conducted with ease, preventing such failure cycles is straightforward, with well-documented methodologies available. However, in the Space environment, where failures are less predictable, repairs are infeasible, and literature on fault mitigation remains scarce, the problem becomes significantly more complex.

The system must therefore be capable of maintaining its core services autonomously and distinguishing between node failures and application-induced crashes. Furthermore, given the constrained resources in Space, any solution must be lightweight to ensure minimal interference with nominal operations.

Identified Error Types

Through analysis, four primary categories of errors have been identified:

1. **Internal Application Error:** Defined by the application developer and recognized by the system. This means that the cause, severity, and impact of the error on the application's tasks and the system as a whole are known and pre-documented.
2. **Unidentified Error:** Occurs when a node suddenly crashes without a predefined error signature. These errors are detected and classified by the Health Service as unknown failures requiring further investigation.
3. **Startup Error:** Arises when a node undergoes reconfiguration, but its last known state was faulty. Consequently, as soon as the node restarts, it crashes again, leading to an immediate failure cycle.
4. **Bad Update Error:** A distinct category of failure that occurs immediately after an update. This classification is crucial as it allows the system to preemptively disable an application suspected of causing failures post-update.

3 APPROACH AND METHOD

The overarching principle guiding this approach is the well-known ”Keep It Simple, Stupid” philosophy. This methodology prioritizes simplicity to enhance code maintainability, facilitate ease of understanding, and streamline the debugging process. These considerations are particularly critical because this FTM serves as a proof of concept rather than a production-ready solution. As such, the primary focus is on demonstrating the viability of the approach rather than implementing fully optimized, industrial-grade software.

3.1 Assumptions

To ensure that the design of the FTM remains both straightforward and practical, several key assumptions have been established, as outlined in Table 3.1. These assumptions help define the constraints and expectations of the system, ensuring that its implementation remains effective within the intended scope.

APP Developers	are tasked with configuring the FTM using the distributed system's configuration file
	agree that the FTM is able to read the log files of the APP
APPS	are using the distributed system's logging system
	have the at least the same quality as expected from published ground applications. For example, the ones published on phones' AppStores
	are composed of one or more Tasks running on one or more Nodes
	do not need FPGA reconfigurations

Table 3.1: Assumptions

Since some boards deployed in space—particularly Worker Nodes—feature onboard FPGAs, it is important to clarify that FPGA reconfigurations are not supported by the . The reason for this exclusion is that detecting and responding to application failures related to FTM misconfigurations would introduce an additional layer of complexity that falls beyond the scope of this internship.

3.2 Constraints

Two primary constraints have significantly influenced the design of the FTM.

The first constraint pertains to log file storage. Each application's log file is stored locally on the node where it is executed. Consequently, if a node crashes, the Coordinator Node is unable to access those files. To address this limitation, a specialized data collection mechanism, described in Section §3.3.2, has been implemented.

The second constraint arises from the lack of centralized data access for the nodes. As a result, when the system transitions to a dynamic configuration (see Section §3.3.2), each node must independently reconstruct its state, as the configuration cannot be transmitted from a central source.

3.3 FTM Design

To ensure minimal system overhead and efficient resource utilization, the FTM employs lightweight yet effective mechanisms. This service is executed on the coordinator node, as it has overarching control over the entire cluster and is responsible for system-wide coordination.

The FTM operates primarily in an event-driven manner, with the exception of the Data Collection module (Section 3.3.2), which is periodically executed.

3.3.1 Service Dependencies

As discussed in Chapter §1, a distributed system comprises various mechanisms that facilitate continuous operation and minimize data loss. Figure 3.1 illustrates the interdependencies between the FTM service and the other system services.

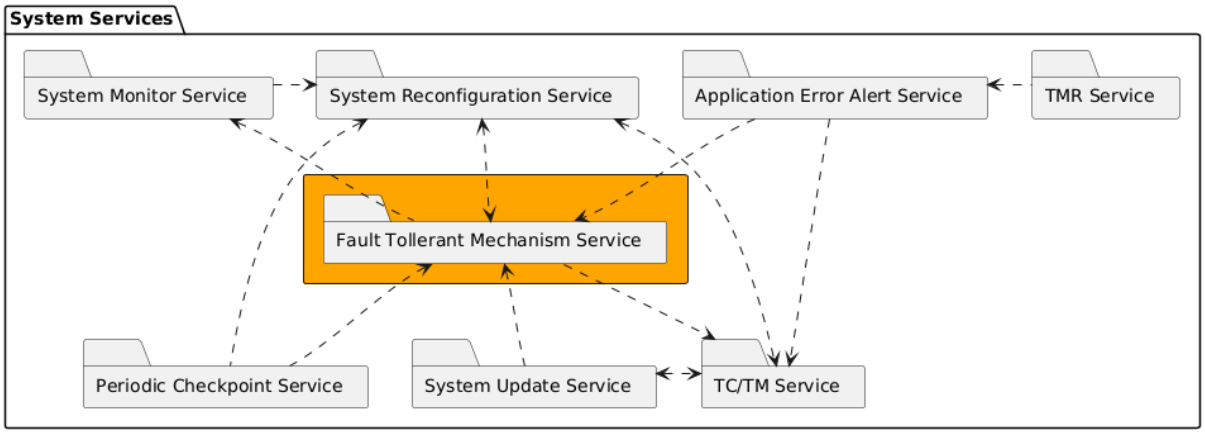

Figure 3.1: System Services

The FTM Service communicates with the Reconfiguration Service to request dynamic configuration changes and interacts with the Telemetry/Telecommand Service for status reporting.

The System Monitor Service reports node health check failures, providing critical input for FTM decision-making. Additionally, the System Alert Service notifies the system of known application failures or issues detected by the Voter Service. The Checkpointing Service, which implements a fault recovery mechanism, transmits its recorded data to the FTM for further analysis. Lastly, the Update Service informs the FTM about recent software updates and modified files, ensuring that the system is aware of changes that may impact its stability and performance.

3.3.2 FTM Modules

The Fault-Tolerant Mechanism is primarily divided into two key modules: the FTM Analysis and Decision Module, responsible for detecting faults and determining the appropriate response, and the Activity and Data Collection Module, which executes actions based on the logic established by the first module.

FTM Analysis and Decision Module

The primary function of this module is to detect application failures. At its core, failure detection relies on the confidence index, which is computed based on two key inputs:

1. Crash Table: This table (3.2) tracks the frequency with which each application crashes a specific node. The crash table is populated with data provided by the System Monitor Service.

2. Issue Weight Table: This table (3.3) categorizes system issues and assigns a severity weight to each type. Based on this table and additional data from the System Alert Service, the confidence index is dynamically adjusted.

Crash Table. This component is relatively straightforward. The coordinator monitors which applications crash on which nodes. Since applications operate in identical environments across different nodes, tracking both the frequency and location of crashes is crucial. This distinction is necessary because if an application crashes only on a specific node, it may indicate a hardware-related issue rather than a faulty application.

When an application exceeds the predefined crash threshold (configured in the system settings), the FTM automatically halts the faulty application to prevent further disruptions.

Since Single Event Upsets are an expected phenomenon, there is a risk of erroneously increasing crash table counters without actual application failures. To mitigate this, the crash table is periodically reset based on the anticipated SEU rate. This rate is determined using known data sources and is dynamically updated according to the spacecraft's current orbital position.

However, relying solely on crash table data is insufficient to precisely identify the faulty application. Consider the scenario illustrated in Figure 2.1, where APP_B is running on both WN1 and WN2. If APP_C is faulty and responsible for system crashes, then both APP_B and APP_C will be halted, as they share the same node. In such cases, there is no direct way to pinpoint the actual faulty application.

To address this challenge, the system refines fault identification by analyzing crash patterns. If an application on a crashed node has also caused failures on other nodes, the system may narrow the investigation from the application level to the task level. For example, in the same scenario, if APP_B did not cause crashes on WN1, the fault is likely between APP_B/Task2 and APP_C. This increases the probability that APP_C is the root cause rather than an isolated issue within APP_B. The system can then halt one application and monitor for further crashes. If another crash occurs, it can revert the decision and apply alternative detection methods.

Another method to refine fault identification involves log analysis. This includes both application logs and system logs. By cross-referencing these logs—mimicking the approach of a ground operator—the system can identify which application executed the last operation before the crash. This same technique is also used to populate the Issue Weight Table, which categorizes the severity and impact of detected failures.

Node / APP	APP_A	APP_B	APP_C	...
WN1	0	3	2	
WN2	N/A	0	2	
...				

Table 3.2: Crash Table

Issue Weight Table. This table categorizes different types of system issues and assigns a weight to each, reflecting its severity and overall impact on system stability. The sum of all assigned weights must equal 1, ensuring a balanced and systematic approach to fault assessment. The primary function of the Issue Weight Table is to fine-tune the confidence index, allowing the system to halt an application either sooner or later than the predefined crash limit, depending on the nature and severity of the detected errors.

For example, configuration errors are generally more critical than user-defined errors. This is because configuration issues cannot be resolved without direct intervention from the Ground Station, meaning that regardless of how many times a node or application is restarted, the fault will persist. In contrast, user-defined errors may be transient or recoverable without requiring external assistance. By incorporating this weighting system, the FTM ensures a more intelligent and adaptive response to different failure scenarios, prioritizing intervention based on the criticality of the detected issue.

Issues	Weight	Rationale
Runtime Errors	TBD	Can compromise the distributed system's operation
I/O Errors	TBD	Can cause conflicts between APPs regarding resource usage.
Configuration Errors	TBD	Configurations are critical for the correct operation of the system.
User-defined Errors	custom	In case the developers are aware of critical errors that may cayse problems.
...	...	...

Table 3.3: Issue Weight Table

Confidence Index. As previously mentioned, the confidence index serves as a key determinant in assessing whether an application (APP) is faulty. This index is calculated using the formula 3.3, incorporating relevant system metrics to provide an objective evaluation of an application's stability and reliability.

The confidence index is computed by summing the Crash Table score and the Weight Table score. The Crash Table has a maximum value of 1 (when the number of crashes hits the limit of maximum allowed crashes) 3.2. The Issue Weight Table score is the sum of all the issue scores 3.1. An issue score is the multiplication between its weight and how many times did it occur.

The confidence index threshold is explicitly defined within the Fault-Tolerant Mechanism configuration. By adjusting this threshold, the system can fine-tune its sensitivity to potential failures, ensuring that erroneous applications are identified and addressed promptly while minimizing false positives. This adaptability allows the FTM to maintain optimal system performance, balancing fault detection with operational continuity.

$$W_t = \sum_{i=1}^{IssueNo} OccNo * W \tag{3.1}$$

$$C_t = \frac{CrashNo.}{MaxCrashNo.} \tag{3.2}$$

$$conf.index = C_t + W_t \tag{3.3}$$

where:

W_t is the Issue Weight Table Score,
$IssueNo$ is the number of issues in the Issue Weight Table,
$OccNo$ is the number of times i-th issue occurred for the APP/Task under analysis ,
W the weight described in the Issue Weight table,
C_t is the Crash Table Score,
$MaxCrashNo.$ is the crash threshold defined in FTM settings,
$conf.index$ is the confidence index,

Dependency Analysis. Building on the issue outlined in the Issue Weight Table section, there remains a possibility that only a specific task within an application is faulty rather than the entire APP. Consider an Earth Observation APP composed of two distinct tasks: Image Acquisition and Image Compression. To ensure mission-critical data is not lost, the Image Acquisition task is of paramount importance, whereas the Image Compression task is not essential

for immediate system functionality. By analyzing the dependencies between these tasks, the system can intelligently disable only the faulty Image Compression task while keeping the Image Acquisition task operational, thereby preserving mission integrity.

Error Timing. The FTM can be further enhanced to proactively mitigate failures by analyzing the timing of error occurrences. For instance, if a configuration error arises immediately following a software update, it may indicate a faulty update. By incorporating temporal analysis into fault detection, the system can identify patterns and act preemptively to prevent further failures.

APP Isolation. When an APP reaches the threshold confidence index and is suspected to be faulty, but may be a false positive, the system can take an additional step to validate its behavior. If sufficient hardware resources are available, the APP can be isolated on a dedicated node for monitoring. If the APP does not crash the node as it previously did, it can be classified as a false positive. The system can then proceed to evaluate the next suspected APP candidate, refining fault detection accuracy.

FTM Dynamic Configuration and Data Collection Module

Beyond fault detection and analysis, the FTM performs two additional critical functions: Dynamic Configuration, which governs the creation and dissemination of updated system configurations, and Data Collection, which periodically retrieves essential system data from active nodes.

Dynamic Configuration. When an APP is deemed faulty, the system must exclude it from its execution schedule. However, in the context of the study case's Distributed System, exclusion is handled through adaptive configuration adjustments. Once the coordinator node generates a revised configuration, it transmits this update to all available nodes. Each node then independently applies the new configuration. Once all nodes confirm successful implementation, the coordinator initiates the transition to the updated configuration, ensuring seamless operation.

Data Collection. As previously mentioned, once a node crashes, the system loses access to its logs and checkpoints. To mitigate this issue, the FTM periodically collects data from all nodes and securely stores it on the coordinator node. This proactive data collection enables post-crash analysis and potential recovery actions. To optimize storage utilization, a predefined allocation is reserved for this purpose, with the storage limit configurable within the FTM settings.

3.3.3 FTM settings

Variable	Type	Rationale
ftmEnabled	Boolean	Enables / Disables the FTM System Service
maxCrash	uint_8	Defines the maximum number of allowed APP crashes
confIndexThr	uint_8	Specifies the confidence index'es threshold
maxDynConfig	uint_8	Helps the FTM to shut itself off in case too many different dynamic configurations are made. If this limit is reached, help from Ground is requested.

Table 3.4: FTM settings Variables

3.4 Measurement Setup

The primary metric for evaluating the effectiveness of this project is its ability to prevent nodes from entering a reconfiguration cycle. Therefore, the validation process involves a comprehensive set of unit and integration tests to ensure that both the system and the FTM function as intended. A detailed breakdown of these tests is provided in the subsequent section.

For unit testing, Google Test (gtest) was employed. Google Test is a widely-used C++ testing framework designed for writing and executing unit tests efficiently. It offers a comprehensive set of assertions, test fixtures, and test runners to validate code behavior. Additionally, gtest supports automatic test discovery, parameterized testing, and seamless integration with Google Mock for creating and managing mock objects, which helps simulate complex dependencies within the system.

The integration testing process posed additional challenges due to the nature of inter-service communication. The system's services interact using socket-based communication, making conventional gtest-based approaches less suitable, as proper testing requires launching and managing multiple processes simultaneously. When processes are executed independently, direct access to internal variables and objects is lost, making validation more complex.

To address this limitation, a hybrid approach was adopted. The gtest framework was utilized to execute tests in a manner consistent with the unit testing methodology. However, instead of relying on direct object access, the validation of inter-service communication was performed using a log-based analysis system. A dedicated text parser was implemented to analyze the logs generated during execution, enabling verification of expected behaviors and ensuring that service interactions function correctly under real-world operational conditions.

This approach ensures that the system's fault-tolerant capabilities are thoroughly tested, providing confidence that the implemented mechanisms effectively prevent reconfiguration cycles and maintain system stability.

4 RESULTS

As previously stated, the primary metric for evaluating the effectiveness of the FTM Service is its ability to prevent the scenario described in Section §2.2. To ensure thorough validation, an extensive suite of tests has been designed to analyze multiple scenarios in detail.

4.1 General Approach

The FTM operates through three core activities:

1. **Detection:** When a fault event occurs, the system must quickly and accurately determine whether an application is faulty.
2. **Action:** Once an application is confirmed to be faulty, a dynamic configuration is created to exclude the APP from execution. Additionally, logs and checkpoints from each APP and node are periodically collected.
3. **Reporting:** After the system transitions into a dynamic configuration, all relevant logs and checkpoints are forwarded to the Telemetry Service for transmission to the Ground Station for further analysis.

During the detection phase, four possible outcomes must be tested to ensure the robustness of the confidence index and the accuracy of the system's ability to distinguish between faults. The test scenarios aim to challenge the detection mechanism and identify any weaknesses in handling SEUs or misclassifications.

- **True Positive:** The APP is genuinely faulty and correctly identified as such.
- **False Positive:** The APP is not faulty, yet the confidence index is close to the threshold. The test verifies that the FTM does not incorrectly classify the APP as faulty.
- **False Negative:** The APP is faulty, but its confidence index remains near the threshold. The test ensures that the FTM does not incorrectly classify the APP as non-faulty.
- **True Negative:** The APP is correctly identified as functional, confirming that the FTM does not unnecessarily exclude it.

Testing is divided into two categories: unit tests, which evaluate individual features of the FTM, and integration tests, which assess its interactions with other system services. All tests utilize the same system configurations and application topology as the scenario outlined in Section §2.2.

4.2 Unit Tests

Using the setup described in Section §3.4, the unit tests validate the following functionalities and scenarios:

1. Verification that the crash table correctly identifies and limits faults, ensuring only the faulty APP (e.g., APP_C) is flagged.
2. Validation that all issues listed in the Gravity Table are correctly identified through controlled faults in APP_C.
3. Ensuring that the confidence index only triggers when reaching the predefined threshold in the FTM settings.
4. Direct testing of the confidence index computation by inputting raw values.
5. Validation of dependency identification, ensuring that APP_B Task 1 remains operational even when APP_B Task 2 is disabled.
6. Confirmation that the FTM correctly detects and differentiates two errors occurring in APP_C at the same time of day.
7. Verification that the FTM correctly isolates an APP when two applications have closely matching confidence indices exceeding the threshold.
8. Testing the correctness of dynamic configurations when APP_A, APP_B, and APP_C are faulty individually, in pairs, and simultaneously.
9. Ensuring that data is correctly collected from nodes at designated intervals.

4.3 Integration Tests

As described in Section 3.4, integration tests run three parallel processes representing the three nodes in the system. Their configurations match those in Section 2.2. These tests validate that system components function correctly under the following scenarios:

1. APP_A is faulty. Only Task 1 and Task 2 of APP_A are stopped, as they are interdependent.
2. APP_B Task 1 is faulty. APP_A remains operational, but APP_B is entirely stopped due to Task 2's dependency on Task 1.
3. APP_B Task 2 is faulty. APP_A remains unaffected, and only Task 2 of APP_B is stopped since Task 1 can operate independently.
4. APP_C is faulty. Only APP_C is stopped, leaving APP_A and APP_B running.
5. APP_A and APP_C are faulty, leaving only APP_B operational.
6. APP_A and APP_B are faulty, leaving only APP_C operational.
7. APP_B and APP_C are faulty, leaving only APP_A operational.
8. All three applications (APP_A, APP_B, and APP_C) are faulty, resulting in no active applications on the nodes.

These scenarios cover both expected operational cases and extreme edge cases. Notably, each test is conducted for multiple fault types, including unknown issues, known Gravity Table issues, startup errors, and update-related errors.

The FTM's effectiveness is assessed based on its performance in these extensive test cases. Additionally, all pre-existing system tests for the study case's distributed system must pass. Upon successful completion of all tests, the FTM is deemed suitable for deployment within the system.

5 DISCUSSION

The implementation of distributed systems in Space is a relatively new and evolving field, meaning that there is limited literature on Fault-Tolerant Mechanisms specifically tailored for Space-based applications. However, FTMs have been extensively researched and implemented in terrestrial distributed systems, particularly in the Ground Segment, where fault tolerance is a critical requirement for maintaining reliable operations.

The proposed solution and the study case's distributed system leverage many of the well-established fault-tolerance strategies documented in literature. These strategies have been rigorously tested and validated in ground-based distributed systems, ensuring their reliability. The primary distinction in this study is the adaptation of these methodologies to accommodate the unique challenges of the Space environment. These include considerations for radiation-induced errors, extreme thermal fluctuations, and communication latency due to the vast distances between spacecraft and Ground Station Control. By integrating these additional factors into the existing framework, the system ensures robust fault mitigation tailored specifically for Space operations.

The development of the FTM service required an in-depth understanding of Fault Detection, Isolation, and Recovery techniques within Space applications. To successfully implement this solution, I first gained a strong foundational knowledge of distributed system architectures and onboard software frameworks. This knowledge enabled me to design and develop the FTM service, integrating it seamlessly into the study case's existing FTM infrastructure.

Following its implementation, the FTM service underwent extensive testing, including unit and integration tests, to validate its effectiveness and ensure compatibility with the study case's distributed system. These tests confirmed the service's ability to prevent reconfiguration cycles, maintain system stability, and accurately identify and mitigate faults within the system. The rigorous validation process reinforced the reliability of the proposed solution and demonstrated its capability to handle fault scenarios effectively.

Through this comprehensive process, all the objectives of this internship were successfully achieved. The development and testing of the FTM service not only provided valuable insights into fault tolerance in distributed Space systems but also contributed to a growing body of knowledge in this emerging field. This experience has strengthened my technical expertise in distributed computing, Space systems engineering, and fault-tolerant software design, further reinforcing the viability of distributed architectures for future Space missions.

6 CONCLUSIONS AND RECOMMENDATIONS

The findings of this study demonstrate that with minimal adaptations to existing fault-tolerance mechanisms for terrestrial distributed systems, reliable fault management solutions can be developed for Space-based distributed systems. However, this research primarily focused on assessing feasibility rather than conducting an in-depth performance evaluation.

Although the FTM Service was successfully implemented and tested on the hardware used for the Flight Model, several aspects remain unverified in real-world operational scenarios with actual applications. One of the most significant unknowns is resource utilization under mission conditions. The actual computational load and memory footprint of the service when deployed with real applications need to be assessed to determine its viability for long-term Space missions. Additionally, the service lacks a defined time guarantee, meaning that the detection time for faulty applications and the duration of system downtime during dynamic configuration propagation are currently unknown. Understanding these factors is crucial for determining the overall effectiveness of the FTM in mission-critical environments.

While theoretically, these performance analyses are relatively straightforward, their practical execution is highly dependent on specific mission requirements and stakeholder expectations. Therefore, before proceeding with further optimizations or enhancements, it is strongly recommended to engage with mission stakeholders to establish clear performance criteria, fault-detection time constraints, and system availability requirements. Once these parameters are well-defined, a set of structured requirements and constraints can be formulated. This will allow for a more comprehensive performance analysis and guide the necessary improvements to the system.

If further optimizations are required, several enhancements could be considered. For example, more efficient search algorithms could be implemented, particularly for table look-ups, to improve processing speed. Additionally, integrating the existing system logging and checkpointing services with the Data Collection component could significantly reduce resource consumption by streamlining data retrieval processes. This adjustment would shift the Data Collection functionality away from the FTM Service, allowing it to focus solely on fault detection and dynamic reconfiguration.

In conclusion, the results of this study confirm the feasibility of the proposed FTM Service. However, without a thorough performance assessment, the system's limitations and its compatibility with real applications remain uncertain. Therefore, the next logical steps involve defining clear requirements and constraints in collaboration with mission stakeholders, followed by a detailed performance and timing analysis. These steps will provide the necessary insights to refine and optimize the service, ensuring its reliability and efficiency in operational Space environments.

Keywords— Distributed Systems, Fault-Tolerant Mechanism, Single Event Upsets, Space Environment, Confidence Index, Dynamic Configuration

REFERENCES

[1] Balazs Gerofi, Yutaka Ishikawa, Rolf Riesen, and Robert Wisniewski. *Operating Systems for Supercomputers and High Performance Computing*. 01 2019.

[2] Martinus Richardus van Steen and Andrew S. Tanenbaum. *Distributed Systems*. 3rd edition, February 2017. Self-published, open publication.

[3] Nitin Naik. Comprehending concurrency and consistency in distributed systems. In *2021 IEEE International Symposium on Systems Engineering (ISSE)*, pages 1–6. IEEE, 2021.

[4] Nitin Naik. Demystifying properties of distributed systems. In *2021 IEEE International Symposium on Systems Engineering (ISSE)*, pages 1–8. IEEE, 2021.

[5] Stephen S Yau. Challenges and future trends of distributed computing systems. In *2011 IEEE International Conference on High Performance Computing and Communications*, pages 758–758. IEEE, 2011.

[6] Robert Ecoffet. Overview of in-orbit radiation induced spacecraft anomalies. *IEEE Transactions on Nuclear Science*, 60(3):1791–1815, 2013.

[7] Muhammad Fayyaz and Tanya Vladimirova. Survey and future directions of fault-tolerant distributed computing on board spacecraft. *Advances in Space Research*, 58(11):2352–2375, 2016.

[8] Weisen Liu, Zeqi Lai, Qian Wu, Hewu Li, Qi Zhang, Zonglun Li, Yuanjie Li, and Jun Liu. In-orbit processing or not? sunlight-aware task scheduling for energy-efficient space edge computing networks, 2024.

[9] Arif Sari and Murat Akkaya. Fault tolerance mechanisms in distributed systems. *International Journal of Communications, Network and System Sciences*, 8(12):471–482, 2015.

[10] Michel Raynal. *Fault-Tolerant Message-Passing Distributed Systems: An Algorithmic Approach*. 01 2018.

[11] Irina V. Asharina. *The Concept of Failure- and Fault-Tolerance on Base of the Dynamic Redundancy for Distributed Control Systems of Spacecraft Groups*, pages 181–191. Springer International Publishing, Cham, 2020.

LIST OF ABBREVIATIONS AND SYMBOLS

- **APP** Application composed of one or more Tasks that run on different threads
- **CN**: Coordinator Node
- **DS**: Distributed System
- **FDIR**: Fault Detection, Isolation and Recovery
- **FN**: False Negative
- **FP**: False Positive
- **FTM**: Fault-Tolerant Mechanism
- **SEU**: Single Event Upset
- **TID**: Total Ionizig Dose
- **TN**: True Negative
- **TP**: True Positive
- **TMR**: Triple Modular Redundancy
- **WN**: Worker Node