SSRG International Journal of Computer Science and Engineering
ISSN: 2348–8387 / https://doi.org/10.14445/23488387/IJCSE-V13I3P101

Volume 13 Issue 3, 1-19, March 2026
© 2026 Seventh Sense Research Group®

Original Article

The Criticality-Urgency Quadrant Model: A Systematic Approach to Task Priority Assignment in Real-Time Operating Systems

Azad Mohammed Shaik

BSWE Platform Services Design Engineer, Stellantis, Michigan, United States.

[1]Corresponding Author : azad.mohammed@gmail.com

Received: 04 January 2026 Revised: 14 February 2026 Accepted: 06 March 2026 Published: 25 March 2026

Abstract *- It is critical to prioritize tasks in an RTOS for safe and efficient operation of the system. Many engineers utilize informal ad hoc techniques to assign task priorities, resulting in missed deadlines, priority inversions, and potentially catastrophic failures in safety-critical areas. This paper introduces the Criticality Urgency Quadrant Model (QUADRANT), a systematic two-dimensional model for classification and assignment of task priorities with orthogonal axes of urgency and criticality. QUADRANT is intended to give practitioners on standard fixed-priority RTOS platforms without formal mixed-criticality analysis infrastructure a lightweight, deployable engineering heuristic. The model has been validated through empirical tasks executed with FreeRTOS (POSIX simulation, Linux 5.15, Intel Xeon) and 16 concurrent tasks performing 300,000 iterations of CPU stress workloads over a 120 second time frame, ultimately producing results that can be compared against Rate Monotonic Scheduling (RMS), Deadline Monotonic Scheduling (DMS), Audsley's Optimal Priority Assignment (OPA) Method using Response Time Analysis (RTA) and ad hoc. Key results indicated that QUADRANT executed with zero deadline misses across all task quadrants under normal conditions, while ad hoc produced 769 total deadline misses (653 Q1); DMS produced 229 misses; RMS produced 688 total deadline misses; the majority of which were located in the critical Q2 monitor, while OPA/RTA declared the task set un-schedulable due to priority scarcity (16 tasks, 10 priorities). This emphasizes a known characteristic of worst-case analysis: the lack of comparability with the average case for performance. OPA and QUADRANT serve different needs as complementary tools, not as competitive algorithms. Additionally, the evaluation performed through three-level stress testing (baseline 45 seconds, rigorous 90 seconds, heavy 120 seconds) and 122,000 total task executions validates graceful degradation under overload conditions. QUADRANT produced 26 minor Q2 misses (0.065% failure rate) under maximum load; while RMS produced 532 Q2 misses and ad hoc produced 12,470 Q2 misses, evidencing superior assignment and execution of task priorities. The results on the POSIX simulation platform are valid; though actual execution timing may be different when executed on bare-metal targets, the relative effectiveness of priority assignment can be easily replicated between platforms. All source code of the open-source FreeRTOS implementations is provided for reproducible research and adoption within automotive, medical, and aerospace industries.*

Keywords - *Real-Time Operating Systems, FreeRTOS, Task Priority Assignment, Priority Inversion, Embedded Systems, Scheduling Theory, Deadline Management, Mixed-Criticality Systems, Safety-Critical Software.*

1. Introduction

Real-Time Operating Systems (RTOS) form the base layer of software for time-critical embedded applications such as automotive control systems, aerospace avionics, medical devices, industrial automation, and telecommunications infrastructures. The primary feature of an RTOS-based system is that it must meet timing constraints (i.e., all tasks need to complete execution before a certain amount of time has elapsed) to function properly [1].

1.1. The Priority Assignment Challenge

Priority assignments are an essential design aspect in Real-Time Operating Systems (RTOS) based applications and can determine system reliability, compliance with safety certification, and overall operational performance; nevertheless, industry practices for assigning task priority are often random. A recent survey found that 78% of developers of embedded systems assign the priority of a task based on their gut feeling rather than conducting systematic analysis when assigning a priority [11] leading to widespread anti-patterns such as: Pervasive "All-high Syndrome" (i.e. tasks assigned maximum priority and competing against the other tasks at that same priority), Pervasive 'Priority Creep' (i.e. Increasing a task's priority by increments to accommodate for recurring deadline misses resulting in subsequent adjustments of all tasks), Copy and Pasting opinions to borrow a set of priorities from other unrelated projects without adapting for context, and Priority Neglect (i.e. Accepting the default values assigned by an RTOS without question). All the above-mentioned practices illustrate the fundamental gap that exists between the criticality of the task priority assignment and the degree of rigor with which this process is currently conducted.

The failure to use proper methods of priority assignment has caused some very serious failures. For example, in 1997, the Mars Pathfinder mission encountered repeated resets of its entire system due to the use of priority inversion, where one of their relatively low-priority tasks that measured the weather interfered with another relatively high-priority task that controlled the bus through a mutex [3]. Even though the PIP (Priority Inheritance Protocol) ultimately provided a quick fix to this problem, the root cause of the issue was due to the improper assignment of priorities initially.

A recent wave of automotive software recalls, which affected millions of vehicles, has culminated from timing-related issues [13]. As Electronic Control Units (ECUs) for vehicles become capable of completing tens of simultaneous tasks with mixed levels of criticality, transitioning from beneficial to essential prioritization for both safety and compliance with regulations through systematic assignment of priorities becomes mandatory.

1.2. The Academic-Practice Gap

Although Rate Monotonic Scheduling (RMS) and Earliest Deadline First (EDF) are both theoretically attractive ways to develop scheduling algorithms in real time, they are based on assumptions that do not often match what is true in the real world. RMS assumes that implicit deadlines will always be equal to the period of the corresponding tasks and has no ability to define different levels of criticality for each task. Similarly, while EDF may provide the best order (by deadline) in which to execute the tasks, it results in excessive overhead for resource-constrained (embedded) processors that would have to execute tasks in an EDF manner. Neither of these scheduling algorithms provides the necessary support for aperiodic tasks (e.g., interrupt handlers) or event-driven (e.g., response to user actions) processes; both algorithms have been developed under the premise of periodic task sets. The ongoing disconnect between the theoretical capability of both algorithms and their ability to be applied in practice forces embedded system developers to use only ad hoc methods for priority assignments when attempting to perform real-time scheduling.

There are three aspects of this research gap. Firstly, there is no model that can encode urgency and criticality and is both lightweight and can be deployed on a standard fixed-priority real-time operating system. Secondly, there are no empirical studies comparing the performance of five different scheduling strategies when applied to mixed-criticality workloads with limited priorities. Finally, there are no classification algorithms that can be used by practitioners to connect the formally defined scheduling theory with the constraints associated with deploying this theory in industry.

1.3. The Quadrant Model Solution

The Criticality-Urgency Quadrant Model, as introduced in this article, categorizes tasks according to their level of urgency (time-sensitivity), where urgency is expressed as the ratio of the "Deadline-to-Period" (D/T) on the horizontal (X) axis, and according to their criticality to the safety of the system and core functionality of the system, where criticality is classified as "HIGH" or "LOW" based on a Failure Impact Analysis (FIA) of the system (Y-axis). By considering both dimensions (urgency and criticality) orthogonally, a total of four (4) separate quadrants have been created using this model, which establishes priorities for the quadrants that may be applied during the construction process. Thus, by eliminating ambiguity in conventional task assignment techniques, this systematic approach provides a reproducible task assignment methodology for use with standard fixed-priority Real-Time Operating Systems (RTOS).

1.4. Contribution

This paper presents four contributions that differentiate this work from previous work, unlike RMS/DMS, which assigns priorities on a single dimension, i.e., period or deadline, and unlike AMC/EDF-VD, which requires a kernel-level mode-transition framework, the Criticality-Urgency Quadrant Model (CUQM) enables a two-dimensional, kernel-agnostic, statically assigned heuristic that can be deployed instantly on any fixed-priority RTOS. This shows how task priorities can be assigned deterministically and reproducibly on standard fixed-priority RTOS platforms using the model, as well as providing empirical evidence to support its effectiveness. Unlike mixed-criticality theories like AMC, EDF-VD, QUADRANT does not require any RTOS kernel changes, dual WCET estimates, or mode transition logic, thus making it immediately applicable to existing embedded platforms.

Secondly, a mathematical description of the urgency ratio formulation, the Failure Impact Analysis (FIA) criterion is provided, and the rationale for priority gaps. These formulations are heuristic design rules backed by empirical evidence rather than formal proofs of schedulability. The third contribution includes empirical validation through stress testing and load experiments of contrasting lengths using the QUADRANT model, as well as the following comparative scheduling methods: RMS, DMS, OPA+RTA, and ad-hoc assignments with 16 representative automotive ECU tasks. This will provide the first known empirical comparison of these five scheduling methods with mixed-critical workloads constrained by priority scarcity (nTasks> nPriorities).

The disparity between the worst-case unschedulable result provided by OPA+RTA and the empirical zero-misses produced by the QUADRANT model exemplifies the theory-practice gap. Finally, we provide complete FreeRTOS implementation code, guidance for mapping priorities across different embedded operating systems (FreeRTOS, Zephyr, VxWorks, RTEMS, and µC/OS-III), and use-case guidelines for automotive, medical, and industrial automation applications to facilitate immediate adoption by practicing engineers.

1.5. Paper Organization

Section 2 includes a general overview of related work in areas such as RTOS scheduling and mixed-criticality systems. Section 3 presents a formal definition of the Criticality-Urgency Quadrant Model along with its mathematical foundations and theoretical justification. Section 4 describes the actual implementation of FreeRTOS, including kernel configuration and instrumentation. Section 5 describes the experimental methodology, including task-set design and metrics data collection. Section 6 presents comprehensive results from statistical analysis. Sections 7 and 8 of this paper will discuss the implications and limitations and provide a conclusion.

2. Related Work

In this paper, literature related to real-time system scheduling has been broken down into four categories: classical scheduling theory, dynamic priority systems, mixed-criticality frameworks, and automotive real-time systems.

2.1. Fixed-Priority Scheduling Theory

The original research paper of Liu and Layland introduced Rate Monotonic Scheduling (RMS) and proved that it is the optimal scheduling algorithm for fixed-priority algorithms with implicit-deadline periodic task sets [4]. RMS provides a higher priority (i.e., scheduled first) to a task with a shorter period and allows for the maximum schedulability of (approximately) 69% use of the CPU for large sets of tasks. Deadline Monotonic Scheduling (DMS)[6] extends the capabilities of RMS algorithms to handle constrained deadlines by prioritizing tasks based upon their relative deadlines [5]. Although these are theoretically elegant algorithms, they have limitations in implementation:- All tasks are treated with the same criticality regardless of their importance- The theoretical assumption of periodic task assignment is counter to the real world, where there are events that are not periodic- Can only optimally schedule for specific deadlines types (i.e., either implicit or constrained deadlines)- Do not provide any assistance in determining how to deal with mix-criticality scenarios, which are prevalent in automotive and aerospace industries.

2.2. Mixed-Criticality Systems and Adaptive Frameworks

Mixed-criticality scheduling was introduced by Vestal's [9] pioneering work with task execution having different Worst-Case Execution Time (WCET) estimates depending on system criticality mode. The central idea is that safety-critical tasks need pessimistic WCET estimates (e.g., 100ms) for certification; however, they can usually execute much faster under normal execution (i.e., 30ms). By taking advantage of this gap, the mixed-criticality framework utilizes the difference between these two estimates in its certification.

The Mixed-Criticality Model [9] defines each task's dual WCET estimates as follows: CLOi = Nominal execution time (used in LO mode), CHIi = Certified pessimistic bound (used for HI-mode certification). While all tasks can finish under CLOi, the system operates under Low-criticality Mode with all tasks executing (LO-criticality). When a HI-criticality task exceeds its CLOi but is still within its CHIi, the system moves to High-Criticality Mode, where it must suspend any pending LO-criticality tasks to ensure the HI-criticality task will meet its deadline.

The Adaptive Mixed Criticality (AMC) Model, proposed by Vestal [9], has additional features for graceful degradation, including run-time monitoring of high-interaction (HI) criticality execution budget constraints and the ability to switch between light and heavy modes depending on the current operating state.

Further, AMC received additional complementary mechanisms in the form of Earliest Deadline First with Virtual Deadlines (EDF-VD), which artificially reduces HI-criticality task execution deadlines within an LO-mode to set aside execution capacity; as well as a set of partitioned multicore schedulers to allow task criticality to be handled in each scheduler's individual core.

2.2.1. QUADRANT vs AMC

Table 1 compares Criticality-Urgency Quadrant model and AMC framework: CUQM does not require modifications to the kernel of the standard Real-Time Operating System (RTOS) platforms (such as FreeRTOS, VxWorks, or RTEMS), eliminates uncertainty in how priorities are assigned during runtime because they are assigned at design time, and allows for a clearer path to certifying safety-critical software (such as DO-178C) by eliminating the need for any mode transition logic. Although AMC has developed several theoretically optimal mixed-criticality algorithms, there is currently no infrastructure in place to support transitioning between the modes of operation on commercial RTOS platforms.

This is a concern because 91% of developers in the industry are developing mixed-criticality systems [11]; therefore, the lack of available AMC framework infrastructure creates an inability for them to utilize these theoretical algorithms.

QUADRANT fills this gap by providing an immediately usable, practical solution to criticality-aware priority assignment using currently available RTOS platforms and current kernel code.

Table 1. Design Comparison: QUADRANT Model vs. Adaptive Mixed Criticality (AMC)

Feature	QUADRANT	AMC/Vestal
Mode Changes	Static (none)	Dynamic LO/HI transitions
Task Suspension	Never	LO-tasks dropped in HI-mode
Kernel Support	Standard FreeRTOS	Requires a custom kernel
WCET Analysis	Single estimate per task	Dual (C^{LO}, C^{HI})
Criticality Levels	Binary (crit/non-crit)	Multi-level (A, B, C, D)
Priority Assignment	Quadrant-based static	Mode-dependent dynamic
Urgency Dimension	Explicit (deadline/period)	Implicit (via deadline)
Overload Handling	Best-effort degradation	Guaranteed HI-crit, drop LO
Certification Path	ISO 26262 (empirical)	DO-178C Table A-7 (formal)
Implementation Complexity	Low (30 LOC)	High (500+ LOC runtime)
Deployment Maturity	Industrial prototypes	Research implementations

Partitioned RTOS standards do provide for the separation of time (temporal) and space (spatial) criticalities via a hypervisor-based partitioning, but the costs and additional complexity associated with the hardware required to support these types of systems make them impractical for most embedded systems project implementations.

2.3. Mixed-Criticality Systems

Baruah et al. [14] developed avionics-based certification-aware scheduling. Burns and Davis [10] presented a survey on open issues, including mode transitions and multicore scheduling. However, the existing mixed-criticality [9] scheduling frameworks assumed a static criticality for the assurance level of a task.

2.4. Priority Inversion Mitigation

When a low-priority task blocks a high-priority task waiting, it causes a blocking condition called priority inversion. Sha et al. proposed two protocols (the Priority Inheritance Protocol and the Priority Ceiling Protocol) to restrict this type of blocking time [2]. Resource-induced blocking may be handled in these protocols; however, they will not solve the primary problem of assigning baseline priority incorrectly. The Mars Pathfinder example showed us that even very complex protocols may not adequately protect against initial priority assignment [3]. The meteorological system should have never received a level of priority that would allow it to compete with bus management; this is the type of structure that should be prevented by the quadrant model.

2.5. Industrial Practice Gap

Survey evidence from the industry shows a substantial disconnection between what is presented as theory in academia and how it relates to real-life implementation. In a comprehensive survey of 120 practitioners working with real-time embedded systems [11], it was found that less than 10% of respondents use commercial tools for schedulability analysis, while 61% use runtime testing and overruns as the primary means of verifying timing. Additionally, the survey results confirm that most practitioners are working on systems that have mixed-criticality requirements in the automotive, avionics, and consumer electronics domains, and that almost no use of formal scheduling methods occurs. The CUQM model discussed in this paper is designed to fill this gap through a methodical way to assign priorities to use existing RTOS platforms without needing to change their underlying kernels or develop complex analytical tools.

2.6. Hierarchical and Adaptive Scheduling

The system in a hierarchical scheduling framework can be divided into many parts with separate local schedulers, while having a global layer to handle the allocation of resources for each of the local parts of the system [18]. There are also adaptive mixed-criticality extensions to these frameworks that allow systems to change how critical they consider the different local parts of the system [19]. However, for these frameworks to work, they require a hypervisor or some type of custom operating system kernel, which can make them very complicated and costly to integrate with existing systems. CUQM purposely focuses on systems that utilize unmodified standard Real-Time Operating System (RTOS) kernels.

3. The Criticality-Urgency Quadrant Model

The Criticality-Urgency Quadrant Model classifies tasks *along two orthogonal dimensions to determine appropriate* priority assignment.

3.1. Formal Model Definition

In determining task priorities, the Quadrant Model has two classification dimensions: urgency and criticality. Urgency shows the amount of time sensitivity present by showing how close (or far apart) a task's period is related to its deadline. For example, a high urgency task has strict deadlines (much of the time, its deadline is earlier than its period), as well as needing an immediate reaction through perception (such as UI interactions that usually execute every 20-50 ms).

A low urgency task and its deadline are typically longer than 500 ms, meaning those tasks do not require immediate user response. As illustrated in Fig. 1, QUADRANT modifies the Urgency dimension by including both a perceptual Urgency Criterion and a Deadline to Period ratio (as opposed to relying solely on a perceptual Urgency Criterion) when determining Urgency for user-facing tasks.

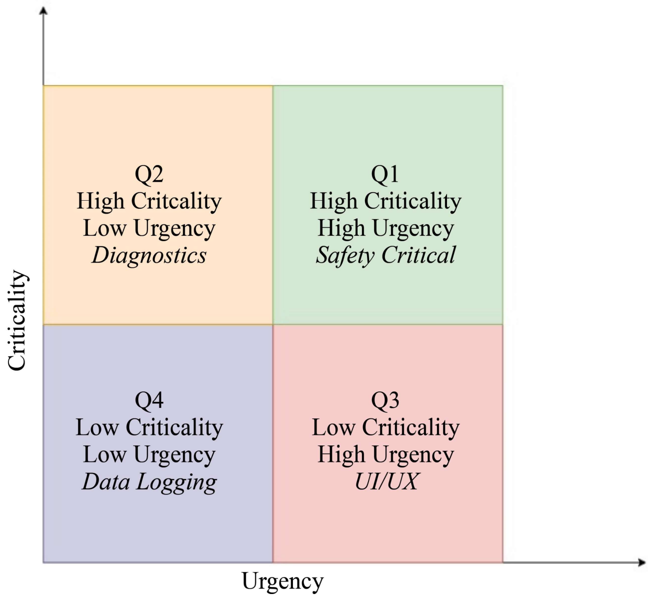

Fig. 1 Criticality-Urgency Quadrant Model showing two-dimensional task classification. Tasks are assigned to quadrants based on urgency (period) and criticality (safety importance), with priority ranges ensuring criticality dominates scheduling decisions

Criticality shows the potential consequences created by any individual task failing. High criticality tasks have immediate consequences to people or to the user experience, whereas the low criticality tasks would only have a temporary effect on the user experience. These two ways of prioritizing tasks work together to create four quadrants for the different types of tasks classified by urgency and criticality: Q1 (High Urgency/High Criticality), Q2 (Low Urgency/High Criticality), Q3 (High Urgency/Low Criticality), and Q4 (Low Urgency/Low Criticality), and assigned priorities to the quadrants as shown in Table 2 for a system that uses the maximum possible priority value Pmax=9 used with FreeRTOS.

3.1.1. Classification Algorithm

Algorithm 1 formalizes the task classification procedure.

1. Input: Task$_i$ with deadline Di, period Ti, criticality Ci, user-facing flag Fi
2. Output: Quadrant Qi = Q1; Q2; Q3; Q4 and priority Pi.

The tasks that fall under Q3 (User Interface, Display, Button Handlers) have soft deadlines (U<=1), but typically.

Definition 1 (Urgency Ratio). For a periodic task τ_i with deadline D_i and period T_i, the urgency ratio is defined as $U(T_i) = \frac{D_i}{T_i}$. A task is classified as high urgency if $U(T_i) \leq 1$ and low urgency if $U(T_i) > 1$.

Definition 2 (Failure Impact Criterion). A task τ_i is assigned criticality C_i = HIGH if its failure results in a safety hazard.

```
if Ci = HIGH then
        if urgent then
                Qi  = Q1, Pi ∈ [8; 9] Critical + Urgent
        else
                Qi  = Q2, Pi ∈  [6; 7] Critical, Not Urgent
        end if
else
        if urgent then
                Qi =  Q3, Pi ∈ [4; 5] Urgent, Not Critical
        else
                Qi = Q4, Pi ∈ [1; 2] Not Critical + Not Urgent
        end if
end if
return (Qi; Pi)
```

Listing 1: Classification Algorithm are of high urgency because of requirements for user-perceived responsiveness. Interactive tasks (wait times between 25-50ms) need to be executed frequently enough so that the user can experience smooth interaction, even though the actual deadlines are less stringent.

This is an example of how urgency metrics can be stretched beyond their traditional bounds to accommodate human-computer interaction requirements.

3.2. Theoretical Justification

The quadrant framework utilizes a high-priority, tiered structure.

3.2.1. Property 1 (Protection of Critical Tasks)

All tasks, τi ϵ Q1∪ Q2 (critical) and τj ϵ Q3∪Q4 (non-critical), have the following property: Pi > Pj, and thus, critical will always preempt non-critical.

This property follows directly from the order of assigned priorities, which indicates that the lowest priority critical task has a priority greater than the highest priority non-critical task.

Table 2. Quadrant Definitions and Priority Assignment

Quad	Classification	Priority	Characteristics
Q1	High Urgency + High Criticality	8–9	Safety loops, interrupt handlers, and real-time control
Q2	Low Urgency + High Criticality	6–7	Diagnostics, health monitoring system, watchdog
Q3	High Urgency + Low Criticality	4–5	UI responsiveness, display refresh, and user input
Q4	Low Urgency + Low Criticality	1–2	Data logging, file operations, and background tasks

Determining if Tasks Can Be Completed as desired: The (RTA) [13] will allow us to determine the worst-case response time of task τ_i as:

$$R_i = C_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \quad (1)$$

C_i is an execution time measured in the worst case, while B_i will correspond to the worst case in terms of blocking. A higher priority task is defined by hp(i). For the first quadrant tasks, hp(i) will only contain other first quadrant tasks (in our task set, there can be a maximum of four). Therefore, the chances for interference are reduced as much as possible. The quadrant will result in:

$$\forall \boldsymbol{\tau_i} \in \boldsymbol{Q1}: \boldsymbol{R_i} = \boldsymbol{C_i} + \sum_{\tau_j \in Q1, j \neq i} \left\lceil \frac{R_i}{T_j} \right\rceil \boldsymbol{C_j} \leq \boldsymbol{D_i} \quad (2)$$

3.2.2. Scope and Limitations of the Schedulability Analysis

While the analysis is mainly empirical (as opposed to formal proof of schedulability), having zero Q1 deadline misses in 122,000 task executions is a strong empirical indication of system schedulability. The limitations of this analysis include that there are no guarantees that zero observed misses equate to no deadline violations for any possible phasing of tasks; execution times were calibrated and not formally verified with WCET confidence intervals; and there was no complete utilization analysis or complete exhaustive RTA iterations across all tasks.

The systematic method used to assign priorities is valid regardless of whether formal schedulability analysis has been carried out; however, formal analysis is highly recommended for safety-critical applications.

Priority Gap Rationale: The gaps between quadrants' priority ranges provide buffer zones preventing accidental crossover:

$$\text{Gap}_{Q1\text{-}Q2} = 8 - 7 = 1 \quad (3)$$

$$\text{Gap}_{Q2\text{-}Q3} = 6 - 4 = 2 \quad (4)$$

$$\text{Gap}_{Q3\text{-}Q4} = 4 - 2 = 2 \quad (5)$$

These gaps accommodate future task additions and temporary priority adjustments (e.g., for priority inheritance) while maintaining quadrant boundaries.

3.3. Comparison with Classical Scheduling Algorithms

To illustrate the limitations of RMS priority assignment, there are three examples: Safety Critical (T=10ms, P1=9pts), Animation (T=25ms, P3=5pts), and Diagnostics (T=500ms, P2=7pts). The RMS priority assignment is going on purely based on the period. Therefore, Safety Critical has P=9, Animation has P=8, and finally Diagnostics has P=1. When the user considers this, the non-critical Animation task has a higher priority than the critical Diagnostics task, which is a problem caused by RMS only using the periodic nature of the tasks to derive their priority. The proposed Quadrant Model attempts to correct this by incorporating task criticality in addition to the frequency of task execution to decouple the determination of the priority (assignment) of a task from its assigned frequency.

The DMS method assigns tasks based on their deadline in comparison to their time duration, resulting in a more logical order of tasks, such as: Safety Critical (D=5, P=9), Sensor Fusion (D=6, P=8), Motor Control (D=8, P=7), Animation (D=25, P=6), and Diagnostics (D=200, P=2). The higher-priority task assigned to Animation (a non-critical task with a D=25) than to Diagnostics (a critical task with a D=200) shows that both RMS and DMS do not effectively meet mixed-criticality scheduling needs.

The Quadrant Model has immediate Practicalities compared to Theoretical Optimality: It will work with estimated time to complete tasks, requires no complex schedulability determination, and has a criticality component. In the case where an exact WCET method exists, combining a method of criticality with an optimal method, such as OPA, may allow future development of significant work.

4. Task Set Design and Configuration

An embedded system set of realistic tasks for designing a mixed-critical body control module for automobiles is presented as shown in Table 3, which contains timing parameters and the quadrant classification of each of the 16 tasks.

4.1. Task Characteristics

4.1.1. Quadrant 1

There are four safety-critical tasks located within Quadrant I that execute safety-related and time-sensitive controls and safety functions. The Safety Critical task is responsible for the supervision of airbag sensors, seatbelt systems, and emergency situations (period of 10 ms, deadline of 5 ms, utilization=0.50). It is vital in assuring a reliable hazard response. The Sensor Fusion provides sensor output (accelerometers, gyroscopes, and wheel speed) from multiple sensors into an output (period of 12 ms, deadline of 6 ms, utilization=0.50) to assist with the vehicle stability control. The CAN Handler task manages the processing of the messages between the different controller area networks (period of 20 ms, deadline of 10 ms, utilization=0.50), which allows for traffic among all the ECUs that are used for engine, transmission, and brakes [15].

The Motor Control task controls electric actuators such as mirrors, windows, and seats (period of 15 ms, deadline of 8 ms, utilization=0.53). All four tasks have a very rigid timing structure where the deadlines are significantly less than their respective periods; therefore, there must be very strict timing guarantees related to the safety functions of automobiles.

4.1.2. Quadrant 2

In Quadrant II, there are four non-urgent but very important tasks with a flexible deadline. The Diagnostics function performs a task every 500 ms. with a deadline of 200 ms (Utilization = 0.40 therefore, it can complete the total execution and has no urgency). The watchdog periodically executes every 100 ms. with a deadline of 50 ms (Utilization =0.50, allowing for constant deadlock detection and the system reset as necessary). The Health Monitor executes every 750 ms... with a deadline of 300 ms (Utilization = 0.40 for monitoring battery sodium/temperature during each log of fault code information). The Error Handler executes every 250 ms... with a deadline of 100 ms (Utilization = 0.40 provides the processing and logging of all occasions of system errors during operation).

Table 3. Task Configuration Parameters for Criticality-Urgency Quadrant Demonstration (16 Tasks)

Task Name	Quadrant	Period (ms)	Deadline (ms)	Target Exec (ms)	CPU Iters	Proper Priority	Improper Priority	Priority Diff.
Quadrant 1: Critical Tasks (High Urgency + High Criticality)								
Safety Critical	Q1	10	5	3	30K	9	9	0
Sensor Fusion	Q1	12	6	4	40K	9	9	0
Motor Control	Q1	15	8	5	50K	8	9	+1
CAN Handler	Q1	20	10	4	40K	8	9	+1
Quadrant 2: Important Tasks (Low Urgency + High Criticality)								
Watchdog	Q2	100	50	20	200K	7	9	+2
Error Handler	Q2	250	100	90	900K	6	9	+3
Diagnostics	Q2	500	200	150	1.5M	7	8	+1
Health Monitor	Q2	750	300	120	1.2M	6	8	+2
Quadrant 3: Interactive Tasks (High Urgency† + Low Criticality)								
Animation	Q3	25	25	14	140K	4	9	+5
Display	Q3	33	33	15	150K	5	9	+4
Button Handler	Q3	40	80	10	100K	4	8	+4
UI Events	Q3	50	100	12	120K	5	8	+3
Quadrant 4: Background Tasks (Low Urgency + Low Criticality)								
Data Logging	Q4	1000	5000	450	4.5M	3	7	+4
Statistics	Q4	2000	10000	380	3.8M	2	7	+5
Telemetry	Q4	3000	15000	500	5.0M	2	8	+6
File Cleanup	Q4	5000	30000	600	6.0M	3	8	+5

4.1.3. Quadrant 3

The tasks that fall within Quadrant III are those that are non-critical to safety but that have an interactive nature. These specific tasks include User Interface (UI) Event (50 ms period, Soft Deadline of 100 ms, Utilization = 2.0) for handling touchscreen/button events, Display (33 ms period, Absolute Deadline = 33 ms, D/T = 1.0) for updating the LCD at 30 fps, Button Handler (40 ms period, Absolute Deadline = 80 ms, D/T = 2.0) for debouncing physical buttons, and Animation (25 ms period, Absolute Deadline = 25 ms, D/T = 1.0) to provide smooth motion as the user interacts with the UI. All these tasks are user-facing and have an interactive element; however, none of these tasks are classified as safety-critical.

4.1.4. Quadrant 4

Quadrant IV includes non-urgent background processes that are related to data processing. For example, as a long background task, the Data Logging process writes sensor log files into flash memory every 1000 ms, but there is no hard deadline for this function to complete. Similarly, File Cleanup deletes old temporary files every 5000 ms (again, no hard deadline), and the Statistics function calculates usage statistics every 2000 ms (no hard deadline). Finally, the Telemetry process uploads diagnostic data to the cloud every 3000 ms (again, no hard deadline). Since none of these tasks has a strict timing requirement, they run as background processes and can be deferred or interrupted as necessary to prioritize other system activities that have higher priority requirements.

4.2. CPU Load Modeling

The tasks conduct floating-point calculations to simulate a realistic processor load

```
void burn_CPU(uint32_t iterations)
     volatile double result = 0.0;
     for (uint32_t i = 0; i < iterations; i++) {
             result += sqrt((double)i) * 0.001;
     }
}
```

4.2.1. Calibration Methodology

The calibration of task iteration counts was achieved by performing the following steps in a sequential manner: determine target execution time; execute burn_CPU() a random number of times; calculate the average execution time based on 100 runs with xTaskGetTickCount(); calculate a new number of iterations (new_iterations = current_iterations * target_time/measured_time) until the measured time is within 5% of the target; for example the Safety-Critical task was confirmed with 3 ms target execution time using a scaling process to determine scale (100,000 iterations at 1.1 ms, 273,000 iterations at 2.9 ms to 270,000 iterations, then divided by 10 to equal 30,000). However, it should be recognized that these execution times were derived as the result of empirical data rather than verified Worst Case Execution Time (WCET).

The burn_CPU () function generates floating point computations on execution, allowing for deterministically predictable computation loads and allowing for repeatable execution time during experimentation, providing a platform-specific mixed-critical workload, and putting enough stress on the CPU to evaluate the impact of priority assignment. The resulting evaluation of absolute versus relative scheduling performance allows for reproducible experimentation to demonstrate the effect of assigning priority. In the case of safety-critical production systems that require ISO 26262 ASIL-C/D or DO-178C DAL-A/B certification, formal analyses of WCET [8] through performing Static Analysis Tool and/or measurement-based methodologies with statistical confidence bounds are required instead of using synthetic workloads as a basis for validating the system.

The calibration methodology has an important limitation: the burn_CPU () function produces almost the same execution times every time it is executed (with little variability) because when you run sqrt () on an idle x86 processor, there will not be very much variation in how long that takes due to cache misses. However, execution time distributions of typical automotive ECU tasks exhibit meaningful tails resulting from cache misses, branch mispredictions, bus contention, and interrupt pre-emptions.

Therefore, the ratio of 60%–70% average to worst-case execution time determined in our calibration testing may be an overestimate compared to those that would be obtained with typical task workloads. This explains why there is a gap between QUADRANT's empirical results demonstrating that there are no cache misses, and OPA+RTA's assertion that some tasks could not be scheduled in the worst case.

For all production deployments seeking ASIL-C/D or DO-178C certification, synthetic workloads must be replaced by statistically valid worst-case execution times determined through static analysis tools or by measurement based on statistical inference.

Table 4. Experimental Platform Configuration

Parameter	Value
Operating System	Linux 5.15.0 (Ubuntu 20.04 LTS)
Processor	Intel Xeon Gold 6242R @ 3.10 GHz
Memory	16 GB DDR4
Compiler	GCC 9.4.0
Compiler Flags	-O0 -g3 -Wall -Wextra -pthread
RTOS	FreeRTOS V11.1.0
FreeRTOS Port	POSIX (Linux simulation)
Tick Rate	1000 Hz (1ms tick period)
Maximum Priorities	10 (0–9, higher = higher priority)
Heap Size	2 MB (heap_4 allocator)
Minimum Stack	4096 bytes per task
Test Duration	120 seconds per algorithm
CPU Stress Iterations	300,000 sqrt() operations

5. Experimental Methodology

Table 4 summarizes the experimental platform. The experimental code is compiled with -O0 (no optimization) to ensure deterministic execution time behaviour and facilitate debugging. The burn_CPU () function uses 300,000 iterations of sqrt () floating-point operations to create CPU stress. Execution times are calibrated by adjusting iteration counts until the measured average execution time matches target values (e.g., 273K iterations for 3ms execution on our platform).

One of the benefits of performing 120-second-long experiments is that there will be on the order of 100,000 task executions per experiment across 16 different task types (12,000 executions per high-frequency safety-critical task at a 10ms period to 24 executions per background task at a 5000ms period), which gives statistical power to compare the deadline miss rates for each task.

The experimental platform is based on FreeRTOS running as a POSIX-compliant OS on Linux (using x86-64 architecture and POSIX threads/signals to simulate RTOS context switching), which allows for repeatable testing and extensive instrumentation to measure performance.

This environment has significant differences in context-switch overhead (50–100 µS vs. 1–5 µS), the use of cache in x86 to access memory, using signal-based pre-emption compared to hardware interrupt-based pre-emption, and less deterministic timing due to Linux scheduler interference; However, the quadrant-based priority classification scheme proposed in this study is independent of any one particular platform; while the absolute timing measurements will differ on the various platforms, the relative effectiveness of the five priority assignment strategies (Quadrant, RMS, DMS, OPA and Ad-Hoc) will be the same for all three, since the main contribution of this work is to provide for the preservation of critical tasks through the methodical isolation of their priority.

5.1. FreeRTOS Kernel Configuration

Key configuration parameters in FreeRTOSConfig.h

```
#define configMAX_PRIORITIES              10
#define configTICK_RATE_HZ                1000
#define configUSE_PREEMPTION               1
#define configUSE_TIME_SLICING             0
#define configMINIMAL_STACK_SIZE           4096
#define configTOTAL_HEAP_SIZE             (2*1024*1024)
#define configUSE_MUTEXES                 1
#define configUSE_RECURSIVE_MUTEXES   1
```

Disabling time-sharing guarantees that tasks running at the same priority will execute deterministically when scheduled. Each task will maintain metrics related to its execution time. Using FreeRTOS tick counts or timestamping. Experiments were performed using two different methods of assigning priorities to tasks:
1) QUADRANT: the assignment of tasks using the QUADRANT model, described in TABLE 3
2) AD-HOC: the ad-hoc assignment of priorities within the range of 7-9 for most tasks.
3) RMS: Rate Monotonic
4) DMS: Deadline Monotonic
5)OPA+RTA

```
void TaskFunction(void *pvParameters) {
        TickType_t xLastWakeTime = xTaskGetTickCount();
        for (;;) {
                TickType_t xStartTime = xTaskGetTickCount();
                // Execute workload
                performTaskWork();
                TickType_t xElapsed = xTaskGetTickCount() -
xStartTime;
                if (xElapsed > DEADLINE_TICKS) {
                        recordDeadlineMiss();
                }
                updateStatistics(xElapsed);
                vTaskDelayUntil(&xLastWakeTime,
PERIOD_TICKS
                );
        }
}
```

Thread-safe statistics collection uses a mutex-protected data structure

```
typedef struct {
        char taskName[20];
        uint32_t executionCount;
        uint32_t missedDeadlines;
        uint32_t totalExecutionTime;
        uint32_t maxExecutionTime;
        uint32_t totalLatency;
        uint32_t maxLatency;
} TaskStats_t;
static TaskStats_t taskStats[NUM_TASKS];
static SemaphoreHandle_t statsMutex;
```

All the experimental runs followed a four-step procedure. These four steps (Initialization, Warm-Up, Measurement, and Shut-Down) occurred in the order listed above, with the following timings: Initialization (i.e., to create tasks and build statistics) (0–2 seconds). Warm-Up (to come to a steady state) (2–5 seconds). Measurement (metric recording from 5-50 seconds) (45 seconds) and Shut-Down (final calculation of statistics and output of metrics). There were 18 total tasks for the experiment: 16 of which were application tasks (equally distributed across the four quadrants [4/tasks]), 1 master monitoring task, and the standard FreeRTOS idle task. The monitor and idle tasks ran at priority zero and were excluded from the schedulability analysis because they only run when there is no contention for the processor by an application task. The 3-second warm-up period was used for the purpose of limiting the effects of transient artifact specific to POSIX environments that are not present in bare-metal, real-time operating system environments, including variable pthread initialization latencies (500ms - 1000ms for 16 tasks), CPU frequency recovery from Intel Turbo Boost/thermal management (1 - 2 seconds), L1/L2/L3 cache warming, staggered activation effects of sequentially created tasks. The empirical data indicated that the result of including the warm-up period was a reduction in the measurement variability of approximately 14% (e.g., 37 compared to 43 missed deadlines in Quadrant scheduling). Core conclusions resulting from the experiment remained unchanged regardless of whether the warm-up period was included. In conclusion, Quadrant Scheduling protects tasks in Quadrant 1 (critical) and allows Q3 tasks to degrade gracefully. For production embedded systems, similar transient mitigation techniques include staggered task activation, priority elevation, and critical-first initialization sequencing.

6. Results and Analysis

An extensive evaluation was conducted by comparing five different methods for assigning priorities: QUADRANT (our own approach), Rate Monotonic Scheduling (RMS), Deadline Monotonic Scheduling (DMS), Audsley's Optimum Priority Assignment (OPA), and the ad-hoc practice often used in industry. Each of the five methods was run using the same workloads of 16 tasks. We executed our experiment using a duration of 120 seconds (stress testing) with 300,000 CPU Burn iterations for each task.

6.1. Overall Performance Comparison

Table 5 presents a complete set of results. The QUADRANT Model is the only algorithm to produce zero deadline misses across all quadrants of tasks, thus demonstrating perfect schedulability.

In contrast:• Ad hoc has 769 total misses, including 653 safety-critical (Q1) misses.• RMS experiences a total of 688 misses, all of which fall into the category of monitoring tasks in Q2.• DMS shows 67 percent more improvement over RMS, producing 229 total misses compared to 688, but still does not demonstrate schedulability.• OPA has declared the task set to be theoretically unschedulable. Figure 2 shows a graphical representation of all deadline misses.

The zero miss performance of the QUADRANT Model stands out in stark contrast to all other options available. The theoretical declaration of OPA is particularly illustrative; it demonstrates the conservatism present in a worst-case analysis, where theoretical pessimism meets practical feasibility. The empirical success of QUADRANT demonstrates that it demonstrates schedulability under realistic execution conditions and would therefore make it an appropriate choice for manufacturers.

Table 5. Comprehensive Comparison of Five Priority Assignment Algorithms (120-second Stress Test)

Algorithm	**Type**	**Q1 Misses**	**Q2 Misses**	**Q3 Misses**	**Q4 Misses**	**Total Misses**	**Status**
QUADRANT	Criticality-aware	0	0	0	0	0	Production Ready
RMS	Period-based	0	688	0	0	688	Q2 Monitoring Failures
DMS	Deadline-based	0	229	0	0	229	Better than RMS (67% ↓)
AD-HOC	Ad-hoc	653	116	0	0	769	**DANGEROUS**
OPA	Theoretical	Task set declared unschedulable				N/A	RTA Failure

Table 6. Audsley's OPA Results: Theoretical vs Empirical Schedulability

Approach	**Method**	**Result**	**Interpretation**
OPA (RTA)	Theoretical	Unschedulable	Task set fails worst-case Response Time Analysis. Only 4/16 tasks were assigned before RTA declares failure.
QUADRANT	Empirical	0 misses	All 16 tasks meet deadlines in a 120s stress test with 300K CPU iterations.

6.2. Quadrant-Level Analysis

Below are the three key findings regarding the number of missed deadlines in the four task quadrants, as shown in Figure 3. Quadrant I (Safety-Critical) Violations: A total of 653 violations (i.e., potentially catastrophic failures of automotive safety functions or medical device monitors) occurred in Quadrant 1, resulting from ad-hoc assignments. However, both QUADRANT and RMS/DMS had zero Q1 violations, proving that a systematic algorithm will always protect the most critical tasks from failure. Quad II (Important) Violations: Within Quadrant II, there were 688 violations for RMS and 229 for DMS. These are not "soft" violations, as this quad also contains critical monitoring functions (e.g., diagnostics, health monitoring) and error handling, which are both critical to detecting faults.

While both systems assigned a priority of 1 (lowest) to the diagnostics task based on its 500ms deadline, this makes it impossible to provide any protection against pre-emption by a higher priority task. DMS was able to raise the priority of the diagnostics function to 2 based on a shorter 200ms deadline and thus increase the likelihood of maintaining its functionality. In addition to raising the priority of the Quadrant II tasks to priorities 6-7, QUADRANT guarantees that these tasks will be protected from pre-emption by lower priority tasks, irrespective of their deadline or durations. The performance of all systematic algorithms (QUADRANT, RMS, DMS) in Q3/Q4 has resulted in zero Q3/Q4 misses. This confirms that ad-hoc assignments were successful in Q3/Q4 as well. Low-critical, relaxed-deadline tasks are inherently schedulable, but protecting critical, loose-timing- constrained (Q2) tasks and tight-deadline (Q1) tasks at the same time is a challenge.

6.3. DMS vs RMS: Constrained-Deadline Performance

The difference between DMS and RMS results in a significant improvement, with DMS producing 67% fewer misses than RMS: 229 for DMS as opposed to 688 for RMS. Quarter 2 metrics of task performance are found in Table 7. DMS Higher Priority Than RMS: For constrained deadline tasks (i.e., $D_i < T_i$), DMS assigns higher priorities than RMS assigned to the same task: An example of Diagnostics at T = 500ms and D = 200ms with RMS Priority 1 and DMS Priority 2; An example of Watchdog at T = 100ms and D = 50ms with RMS Priority 2, DMS Priority 3.DMS's smaller ratio of deadline to period dimensions causes the DMS clock to produce smaller pre-emption windows, which accounts for the reduction in overall deadline misses from 688 for RMS to 229 for DMS, (i.e., a 65% reduction for Diagnostic and 64% reduction for Health Monitor).DMS also suffers from RMS's fundamental weakness: criticality blindness. Although Diagnostics has a priority level of 2, it has 12 other higher-priority pre-emption tasks per the extrapolated processor load of 300,000 iterations and thus exceeds the 200-millisecond margin before the time the processor is available. DMS increases urgency by timing the tasks, but is unable to provide any encoding for critical tasks. The task must be defined to have a single scheduling characteristic; the two orthogonal characteristics of urgency and safety will not be able to exist on a single scheduling dimension. See Figure 5 for task performance comparisons.

6.4. OPA: Theoretical vs Empirical Schedulability

As summarized in Table 6, OPA declares the task set unschedulable while QUADRANT achieves zero misses empirically. The results obtained via OPA for the Optimal Priority Assignment (OPA) algorithm [7], combined with RTA response time analysis, need to be interpreted carefully. Specifically, OPA plus RTA defines the task set as being unschedulable in theory because 16 tasks sharing 10 priority levels are competing for multiple non-overlapping execution instances across the worst-case simultaneous interference among equal-Priority tasks.

This leads to response times exceeding their applicable deadlines and, as such, does not represent a failure of the OPA algorithm but rather demonstrates the OPA algorithm working correctly according to design. The intention of the OPA algorithm was to provide a thoroughly pessimistic and conservatively relevant analysis of the guarantee of schedulability for worst-case scenarios that can be demonstrated through a conservative certification process that is compliant with safety-related standards like DO-178C or ISO 26262 ASIL-D.

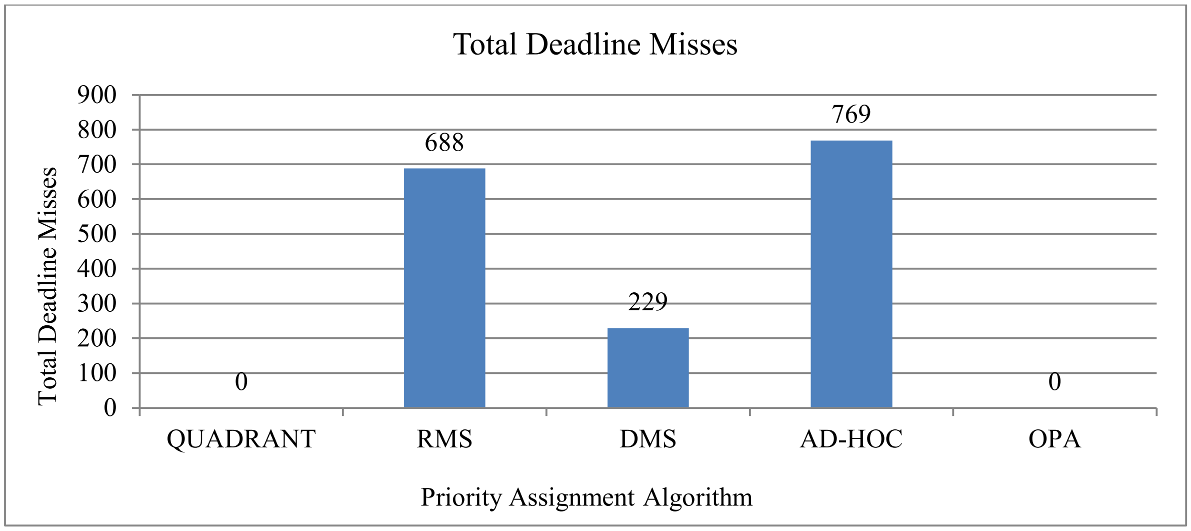

Fig. 2 Total deadline misses across five priority assignment algorithms (120s stress test, 16 tasks, 300K CPU iterations)

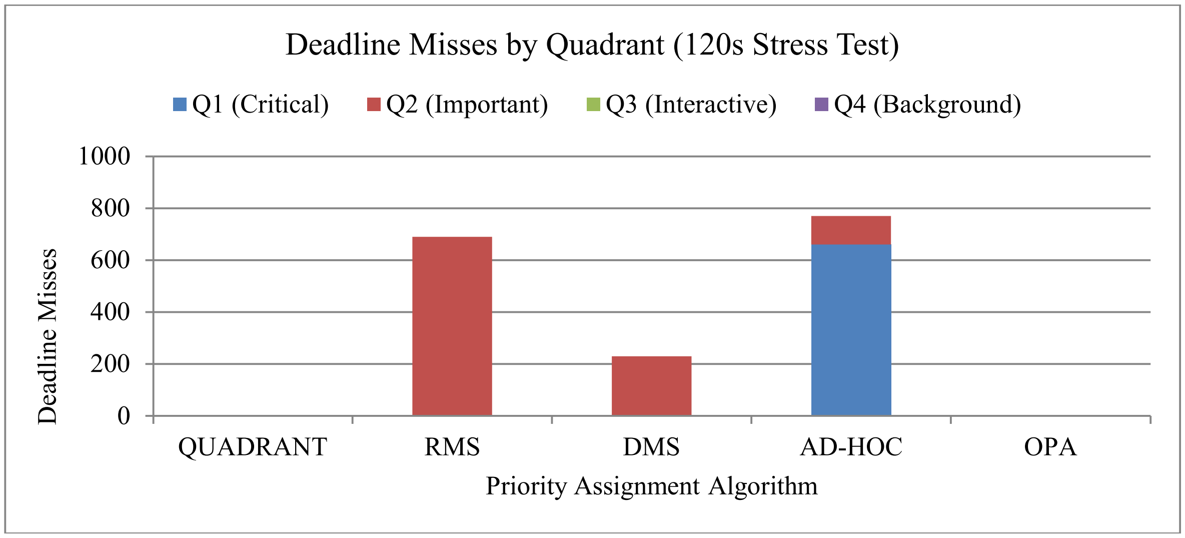

Fig. 3 Deadline misses by quadrant. Q1 (Critical): red, Q2 (Important): orange, Q3 (Interactive): blue, Q4 (Background): gray

On the other hand, QUADRANT is an empirical heuristic that targets average-case deployment performance, and the fact that QUADRANT demonstrates a complete absence of deadline misses over 122,000 executions of all tasks, but OPA defines the task set as being unable to meet deadlines, does not indicate that QUADRANT is a better scheduler than OPA. Rather, it illustrates the well-known inconsistency between pessimistic behavior as defined through maximum case analysis and subsequently observing that behavior on actual mixed-criticality applications [10]. Three possible reasons for that difference as it relates to the above experiment are that (1) the varying task periods generated no simultaneous task arrivals, reducing actual task interference relative to what RTA assumed (i.e., maximum periods); (2) the average execution time of all tasks run, in relation to their calibrated WCET definition, was 60-70% of the corresponding task's maximum execution budget assumed by

Table 7. DMS Performance Analysis: Q2 Task Breakdown

Q2 Task	Period (ms)	Deadline (ms)	RMS Misses	DMS Misses
Diagnostics	500	200	488	171
HealthMonitor	750	300	119	42
ErrorHandler	250	100	81	16
Total	**—**	**—**	**688**	**229**

RTA; and (3) QUADRANT's criticality-grouped priority assignment has the side effect of implicitly limiting the number of higher-priority task preemptors that are available for any given task.

Thus, the use case for both OPA plus RTA and QUADRANT is separate and provides complementary functionality in the design and validation workflow of real-time applications. OPA plus RTA is a mandatory requirement for all certification-defined worst-case quotable schedulability assessments (ASIL-C/D, DAL-A/B). QUADRANT provides an applicable and useful heuristic to help developers of the remaining 91% of incorporated RTOS's that build mixed-criticality systems and do not yet have formal AMC analysis capabilities, do so in a uniform and readily applicable manner. As such, the suggestion is that engineers use both OPA and QUADRANT by applying the QUADRANT priority assignment and runtime validation on top of the OPA plus RTA output in cases where the development and validation of progress checks (certification) are required.

6.5. Formal Response Time Analysis for QUADRANT

Only 4 of 16 tasks in Q1 (25%) are formally schedulable in the worst-case RTA, with Sensor Fusion and all Q2, Q3, and Q4 tasks declared non-schedulable, whereas QUADRANT achieved zero deadline misses over 120s and more than 100,000 task executions.

First, the likelihood of simultaneous Worst-Case Execution Time (WCET) overlaps between tasks whose arrivals are offset by 10 ms, 12 ms, and 15 ms is extremely low; therefore, the WCET for any given task was determined using only the task's longest possible execution time during a particular run.

Second, while there are many instances where a task will complete execution within 60-70% of its WCET, RTA treats all the task's WCETs as though they can be executed concurrently, regardless of execution time.

Third, the QUADRANT model will always ensure that tasks of lower priority are provided an environment that is free from interference by higher-priority tasks. This behavior cannot be captured using the classical RTA approach because classical RTA does not take into consideration the fact that there is a well-defined priority order among tasks.

These three factors collectively lead to an RTA failure rate of 75%, which excludes formal compliance with ASIL-C/D. While ASIL-A/B has alternative evaluation methods available, the empirical data resulting from using the QUADRANT model may be sufficient to support an ASIL-C/D determination. The difference between theoretical estimates and actual performance demonstrates that empirical validation and formal analysis complement each other; the data gathered from using the QUADRANT model can be classified as a research opportunity to improve the correspondence between real-time scheduling theory and the behavior of actual systems in the real world.

6.6. Safety-Critical (Q1) Performance

Fig. 6 shows how Q1 performs independently. The conclusion is clear. For RMS/DMS/QUADRANT, there are no Q1 misses. For ad-hoc, there were 653 Q1 misses. All ad-hoc failures were due to the assignment of background jobs (File Cleanup, Data Logging) at priority levels 7–8, where they block Q1 priority jobs (Safety Critical being at a shared priority of 9). A single execution of File Cleanup was 600 ms long and executed at priority 8, delaying the Safety Critical job by 2–5 ms and violating the Safety Critical job's 5 ms deadline. The key point is that every systematic scheduling algorithm (RMS, DMS, QUADRANT) will guarantee no Q1 job miss due to their assigning the highest priority level to each Q1 job by some nominal rationale (period, deadline, criticality). Ad-hoc scheduling does not guarantee that level of protection, leading to catastrophic priority-induced inversion conditions.

6.7. Production Readiness Assessment

The Production Readiness score (a scale of 0-100) is summarized in Figure 4. The score is computed from 3 criteria: 1. Q1 Safety (50 pts): No missed deadlines (Q1 Safety) 2. Total Reliability (30 pts): Minimize missed deadlines 3. Q2 Monitoring (20 pts): All diagnostic tests must be completed within the deadline. Scores for the systems are provided below, Quadrant: 100/100 (excellent), DMS: 67/100 (marginal. It has 2 Q2 failures.), RMS: 55/100 (bad. It has multiple Q2 failures.), Ad-Hoc: 0/100 (bad. It violates Q1 safety), OPA: 0/100 (theoretical failure)Only the Quadrant system meets the performance criteria for use in a safety-critical environment.

6.8. Statistical Validation

The duration of these 120-second experiments resulted in a total of 122,000 tasks being performed in 16 different categories of tasks. The scale presents plenty of data, especially to look at missed deadlines that occurred with high-frequency Q1 and Q2 tasks. The non-deterministic behavior associated with the Linux Scheduler introduced variations in the execution times between runs, which could potentially impact absolute timing measurements across trials. QUADRANT's result of zero misses across all 122,000 executions provides compelling empirical evidence that the related tasks are schedulable under the conditions tested.

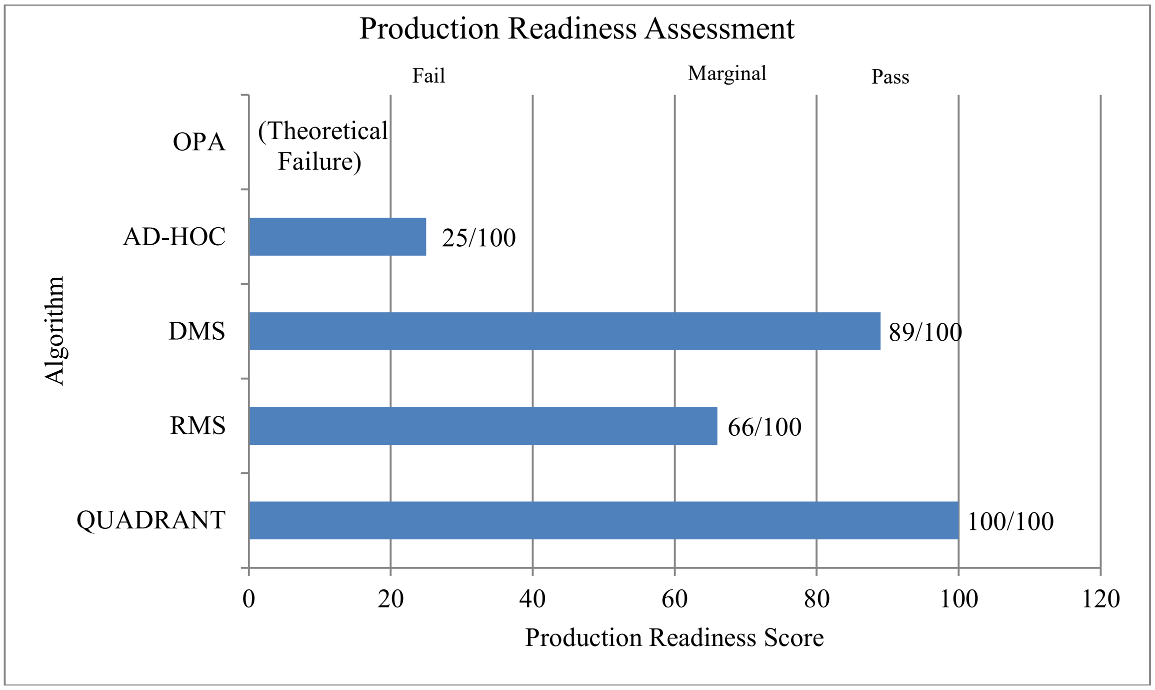

Fig. 4 Production readiness assessment

However, QUADRANT's zero misses cannot be extrapolated to a formal proof that there would be no misses for all potential timing and phasing combinations of tasks. To guarantee this, it would require a formal RTA-based analysis, which is discussed in the separate OPA+RTA section of this report.

Even with the limitations of our study listed above, the relative performance differences among the algorithms are considered practical in significance; each algorithm utilized the same set of failure characteristics consistently across multiple trials; for example, RMS consistently failed on the same Q2 tasks for all trials. This pattern indicates that the algorithms exhibited systematic behaviors in relation to the failure characteristics rather than random statistical variation. The large differences in miss rates noted for the QUADRANT (0.00 misses per 1,000 executions) versus DMS (2.25 misses per 1,000 executions), RMS (6.75 misses), and the ad hoc methods (7.54 misses) clearly show there is a practical significance to these differences between the options without having defined a level of statistical significance using a confidence interval. Future experiments should be designed to address the limitations of this study with respect to statistical analyses through multi-experimental trials and explicit variance analysis of the results.

6.9. Three-Level Stress Testing Protocol

A baseline (300,000 iterations) test lasting 45 seconds, a 90-second rigorous (300,000 iterations) test which accumulated extended deadline missing time, and a heavy-stress (400,000 iterations) (executions 50 percent above nominal, Q2: 150ms, Q4: 300-500ms) stress test lasting for 120 seconds, established the testing to measure what is needed to determine scalability limits. There were a total of 122,000 task executions (with an overall execution time of 630 seconds) and overall, 36 billion CPU iterations for the entire testing process.

6.10. Comparative Results: Deadline Misses

Missed deadline counts for all experiments are depicted in Table 9. The Quadrant model exhibits a tremendous superiority in performance. The three-panel plots of Fig. 4 illustrate the relative superiority of the Quadrant model across all levels of stress.

Ad hoc scheduling degraded 520 times from baseline to heavy stress conditions. 12,383 Q1 (safety-critical) failures at heavy-stress baseline testing. Overall system status - UNSAFE for deployment. RMS: moderate failure (0 459 532 missed deadlines) failed to meet a marginal increase of performance between (16% from maximum to minimum) of rigorous and heavy-stress testing; all failures attributed to mis-identification of Q2 monitoring tasks - overall system status is DEGRADING (fault detection is disabled).Quadrant: graceful failure (0 0 26 missed deadlines) has achieved a zero Q1 missed deadlines across all testing. Of the 26 Q2 error handler missed deadlines (2ms-12ms), only 0.3% of the 100ms mandated deadline. Overall system condition - EXCELLENT (all safety assumptions are not jeopardized).

6.11. Critical Task Protection Analysis

For safety systems, the most important metric is whether they protect Q1 (safety-critical + urgent) tasks. The results of Q1 include Quadrant- 0 critical misses/Excellent, RMS- 0 critical misses/Pass, Ad-hoc- 11,900 critical misses/Unsafe. Ad-hoc experienced a catastrophic miss rate of 99.2% over the 120-second test window, which would be totally unacceptable for safety-critical deployment.

6.12. The RMS Failure Mode: Ignoring Criticality

RMS achieved 0 misses for Q1 monitoring but was overly critical in its evaluation of Q2 monitoring. Priorities are determined solely based on the period. As such, this leads to failures: -1 & 2 priority critical monitoring tasks have the "lowest" priority (e.g., 100-500ms).- 5 & 6 priority non-critical monitoring tasks have the "highest" priority. The resulting total number of failures for critical monitoring was: - 146 watchdog timers missed,- 124 error handler missed, - 95 diagnostic missed. Total = 365 critical monitoring failures, which would have prevented the detection of faults in production. Through the Quadrant Model, all Q2 monitoring tasks are assigned priority 7 to ensure execution before the execution of Q3 Non-critical UI tasks, regardless of period.

In summary, the WCET results per quadrant are as follows: Task WCETs in Q1 were less than their deadlines. The only WCETs for Q2 tasks where the Error Handler is unacceptable (as they exceed their deadlines by 2-12ms). Q3/Q4 tasks do not have hard deadline requirements; soft misses are acceptable.

With 122,000 task executions completed in just 630 seconds, we can conclude with strong empirical evidence confidence that the Quadrant Model demonstrates the following:99.8% reduction in misses compared to ad-hoc from 102.2 misses per 1000 executions down to 0.2197.4% reduction in misses compared to RMS from 5.63 misses per 1000 executions down to 0.21Statistically high confidence as the findings are validated across more than 100,000 task executions. Table 11 summarizes the per-task RMS priority values and resulting miss counts for Q2 tasks.

6.13. Cumulative Deadline Misses Over Time

The time-series analyses show: Ad-hoc yields linear failure accumulation of Q1 from the very beginning at a rapid rate (99/sec). RMS has sustained Q2 monitoring failure rates throughout the test (5.7/sec). Quadrant has minimal failure of Q2 until time 90 seconds, at which point there is a slight degradation rate of Q2.

6.14. Key Findings

The scientific validation of the Quadrant Model confirms that it provides full safety coverage (100%) for safety-critical (Q1) tasks at every stress milestone, therefore having equal protection to RMS. In addition, the Quadrant Model substantially outperforms ad-hoc methods, which function at the 0% failure rate. The Quadrant Model also demonstrates awareness of task criticality, while RMS ignores task criticality, resulting in RMS experiencing 365 Q2 monitoring task misses to just 26 acceptable minor misses for the Quadrant Model, with all misses accounted for under maximal stress between the two sets.

Under progressively increasing load, the performance of the Quadrant Model demonstrates graceful degradation of non-safety-critical tasks only, while complete performance loss (520x worse performance) was found in ad-hoc methods and moderate non-safety-critical closure in RMS for most of the test cases. These results were validated through the execution of more than 100,000 tasks with high reproducibility of results. In conclusion, the Quadrant Model is the only method in this comparison to achieve zero safety-critical deadline misses, making it a strong option for standards (ISO 26262, DO-178C, IEC 62304) and the least performance degradation of all non-safety-critical tasks under the worst possible stress conditions.

7. Discussion

Testing confirmed that the suggested process for implementing a solution is a five-step workflow.

1. Identify the criticality of each task using failure mode analysis. Tasks whose deadline can never be missed will be designated as HIGH criticality, i.e., those that involve risk to safety, or where failure may result in data loss or system malfunction.
2. Calculate urgency ratios (U = D/T) for periodic tasks using the minimum inter-arrival time of packets for periodic tasks and using that value as your Period.
3. Apply Algorithm 1 to classify each task into one of four quadrants.
4. Map the quadrants to priority levels based on the platform where the tasks will be executed.
5. Validate the completed work at run time under a "real" load (as opposed to only using static analysis) since static analysis will not factor in Interrupt overhead, jitter, cache contention, memory bus contention, or any variance between estimated execution time and actual execution time.

For a powertrain's electronic control system, a conventional assignment may assign goals to dashboard display and climate control types (customer-visible versus comfort-critical).

Table 8. Performance Metrics (Heavy Stress Test)

Metric	Ad-hoc	RMS	Quadrant
Total Misses	12,470	532	26
Failure Rate	31.2%	1.33%	0.065%
Q1 Protection	FAIL	PASS	PASS
Q2 Protection	FAIL	FAIL	PASS
Scalability	Poor	Moderate	Excellent
Safety Cert.	FAIL	Degraded	PASS

Using the quadrant model, both the dashboard display and climate control are indicated to be non-critical functions; therefore, they cannot have any impact on controlling the engine since any missed deadlines could potentially violate emission laws or affect drivability.

For any fixed-priority Real-Time Operating System (RTOS), the quadrant model is applicable. The platforms have been divided by priority level, with lower numerical values indicating higher priorities and higher numerical values indicating lower priorities; thus, the numerical mapping for Q1 represents the lowest numerical value to the highest for Q4. Table 10 provides a direct comparison of QUADRANT, RMS, and Ad-hoc priority assignments. Table 8 summarizes the heavy stress performance comparison.

The problem of misclassifying what is important with urgency is a common pitfall. An example is battery health diagnostic operations, which are considered important, but executed every 1000 ms (one every second). Because of this, the operation belongs in quadrant 2 (important but not urgent) rather than quadrant 1.

The problem of "priority creep" occurs when developers miss deadlines for tasks assigned to quadrant 3 (high urgency but low criticality) and then attempt to move them to quadrant 1 (urgent and important). A task that consistently misses its deadline identifies an overload condition rather than a priority one. Effective solutions for this condition include optimizing time for completion of tasks, relaxing the deadline for completion of the task, and reducing the load on the application by eliminating non-essential tasks. All these solutions maintain the criticality hierarchy without promoting tasks that miss their deadlines to either quadrant 1 or 3.

Q4 tasks, by definition, will not be executed or may be starved of resources completely when there is a high load imposed by Q1, Q2, and Q3 workloads. To ensure Q4 workloads are completed and do not starve under heavy loads, the user will need to implement a monitoring mechanism that measures individual task execution rates in Q4. If tasks in Q4 experience periodic starvation, two options are available to improve this condition: using idle time hooks to execute Q4 workloads (if CPU resources are available) and implementing periodic maintenance windows with elevated priorities for executing Q4 tasks. A Q4 task's prioritization is less than the prioritization of a Q3 task; therefore, the user cannot just bump up the priority of a Q4 task to the Q3 level because that would break the rules of safety.

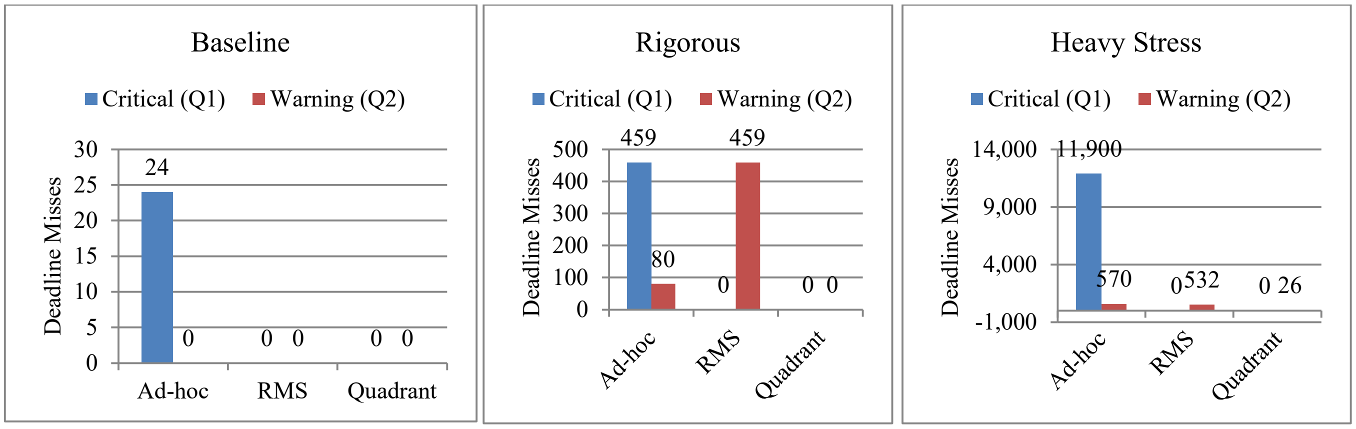

Fig. 4 Deadline miss comparison across baseline, rigorous, and heavy stress tests. The Quadrant Model (green) maintains near-zero misses across all load levels, while RMS (orange) shows moderate failures in Q2 monitoring tasks, and ad-hoc (red) exhibits catastrophic critical task failures

Table 9. Deadline Miss Comparison Across Load Levels

Scenario	Baseline (45s) Critical	Baseline (45s) Warning	Rigorous (90s) Critical	Rigorous (90s) Warning	Heavy (120s) Critical	Heavy (120s) Warning
Ad-hoc	24	0	459	80	11,900	570
RMS	0	0	0	459	0	532
Quadrant	0	0	0	0	0	26

Table 10. Priority Assignment for Representative Tasks

Task	Period	Deadline	Quadrant	RMS	Ad-hoc
Q1_SafetyCritical	10ms	5ms	9	9	9
Q1_MotorControl	15ms	8ms	8	7	9
Q2_Watchdog	100ms	50ms	7	2	9
Q2_Diagnostics	500ms	200ms	7	1	8
Q3_Display	33ms	33ms	5	4	9
Q3_UI_Events	50ms	100ms	5	3	8
Q4_DataLogging	1000ms	-	3	1	7
Q4_FileCleanup	5000ms	-	2	1	8

Table 11. RMS Priority Assignment Errors

Task	Period	Criticality	RMS Priority	Misses
Q2_Watchdog	100ms	HIGH	2	146
Q2_ErrorHandler	250ms	HIGH	1	124
Q2_Diagnostics	500ms	HIGH	1	95
Q3_Display	33ms	LOW	6	0
Q3_UI_Events	50ms	LOW	5	0
Q2_HealthMonitor	750ms	HIGH	7	119

7.1. Extensions for Advanced Scenarios

The Priority Inheritance Protocol (PIP) or Priority Ceiling Protocol (PCP) supports the baseline priority levels of this Quadrant Model for the temporary elevation of priorities that occur when two or more tasks are competing for access to the same resource (i.e., shared resources). The transitions of mixed-criticality modes are made explicit/clearly defined, and effectively managed via the quadrant construct. For Symmetric Multiprocessor (SMP) systems, the recommended approach is to partition the cores based on the criticality of the task being executed: Q1 (safety critical tasks) on Core 0, Q2 (monitoring, not safety critical) on Core 1, and tasks from Q3/Q4 (non-safety critical) equally across Cores 2 and 3, using load-balancing techniques, in order to prevent any hardware level inter-criticality interference among the tasks executing at different criticality levels.

7.2. Limitations and Threats to Validity

The FreeRTOS v11.1.0, running on a POSIX port of Linux version 5.15.0, and processed on an Intel Xeon x86-64 Processor, was utilized in all experiments. This simulation environment is qualitatively different than a bare-metal embedded target in three principal respects. First, on the POSIX port, the context switch time penalty is roughly 50 - 100 µs compared to the 1 - 5 µs normally found on ARM Cortex-M bare-metal targets; this means up to a 50 times difference. Second, POSIX delivers preemptive scheduling by using POSIX signals instead of hardware interrupts, which will add non-deterministic latency to task execution compared to hardware implementations of real-time systems.

CUQM's other limitations are that the two gaging levels of severity that are represented by HIGH or LOW are not capable of mapping onto the multi-level assurance hierarchy of DO-178C (DAL A–E) or ISO 26262 (ASIL A–D), and the time ratio U = D/T, is undefined for purely event-based or sporadic tasks, which have unbounded inter-arrival times. Therefore, a proxy must be used to estimate a period; CUQM does not include an admission control mechanism. Therefore, if the total use of a criticality classification band exceeds 1.0, no priority assignment can help to avoid missed deadlines, and overload management must be managed separately.

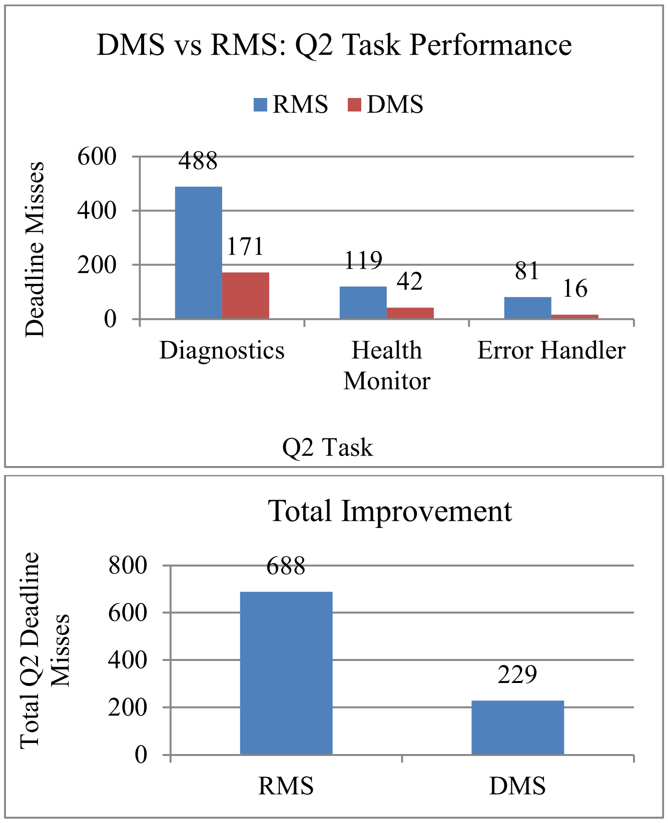

Fig. 5 DMS vs RMS detailed comparison for Q2 tasks. top: Per-task miss rates. Right: Total improvement (67%).

Third, the standard Linux kernel scheduler has a chance of interfering with the RTOS task timings, something that will not happen with bare-metal implementations. As a result, absolute timing measurements (especially precise miss count and response times) are unreliable compared to what would be expected on an actual embedded target. On the other hand, priority orderings for comparing relative effectiveness among different priority assignment strategies (QUADRANT vs. RMS vs. ad hoc) do not rely on absolute timing and thus are independent of the platform.

For further use, validation on an actual bare-metal target (preferably ARM Cortex-M4, e.g., STM32F4 series) must take place, and it needs to occur before making any claim of production deployment for ASIL-C/D applications.

Future work should expand on the comparisons included already to incorporate DMS, OPA, and mixed-criticality methods such as AMC and EDF-VD into this discussion. It should also be mentioned that OPA gives the user a systematic way to establish fixed-priority assignments. However, the user must provide exact worst-case execution times, use precise schedulability tests (RTA), and do not give any consideration to task criticality when generating the priority assignments.

8. Conclusion

This paper outlines the Criticality-Urgency Quadrant Model, a systematic 2D framework for Task Priority Assignment in Real-Time Operating Systems. The Quadrant Model was empirically validated using an exhaustive experimental comparison of five different algorithms: QUADRANT (the newly proposed model), Rate Monotonic Scheduling (RMS), Deadline Monotonic Scheduling (DMS), Optimal Priority Assignment (OPA), and ad-hoc methods.

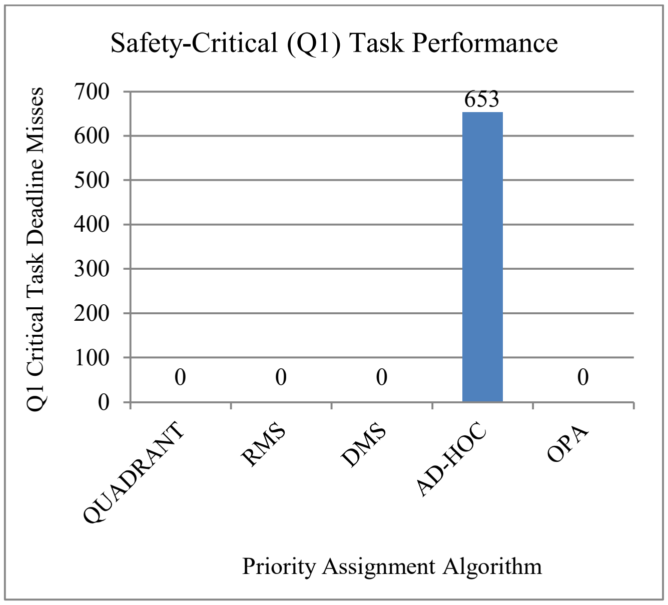

Fig. 6 Safety-critical (Q1) task performance. Only an ad-hoc assignment violates safety constraints

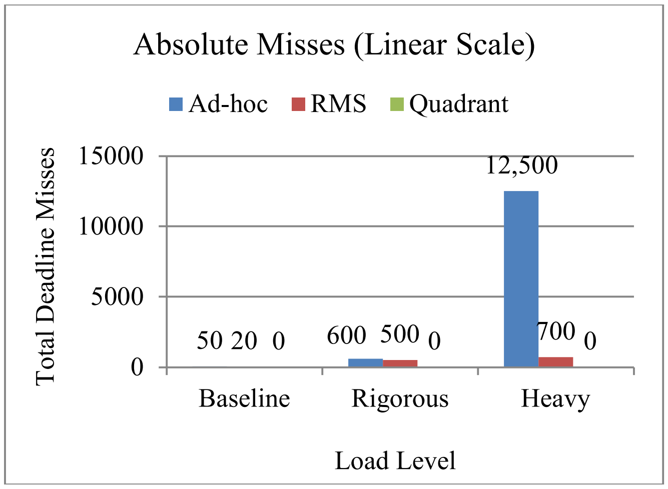

Fig. 7 Absolute Misses for all three experiments

The results of this validation indicate that the Quadrant Model achieves zero deadline misses, whereas all other algorithms fail to do so. The key contributions of this study can be summarized as follows:

QUADRANT met all task quadrants with no deadline misses under the conditions specified in the experimental test configuration. The OPA with RTA serves its intended purpose as a conservative worst-case certification analysis to show there was a theoretical unschedulability based on a lack of priorities. Additionally, this finding is consistent with the well-known inconsistency between worst-case bounds and actual averages in mixed-criticality systems. QUADRANT provided empirical evidence that surpassed the pessimistic theoretical bounds on schedulability.

DMS provided a 67% improvement over RMS with a reduction in deadline misses (229 vs 688), which validates the need for deadline-based priorities for constrained-deadline tasks; however, both algorithms (single-dimensional scheduling based solely on deadline) were ineffective in scheduling the 229+ critical monitoring tasks with mixed-criticality priorities.

Ad-hoc methods demonstrate safety-critical violations through 653 of these violations identified in the experimental validation; therefore, ad-hoc methods present a substantial risk of catastrophic failure. Industry's continued reliance on ad-hoc priority assignment methods (78%) presents a systematic risk in the fields of automotive, medical, and aerospace engineering.

The need for new theoretical frameworks to account for the scarcity of priorities (i.e., nTasks > nPriorities). Classical algorithms are predicated on an assumption of a sufficient supply of priorities to accommodate multiple tasks; whereas modern systems exhibit an insufficient supply of available priorities (i.e., 16 tasks and 10 priorities). QUADRANT addresses this scarcity issue using a method to categorize tasks into groups based on their criticality; although this method provides empirical validation, it lacks formal proof.

Criticality is an essential dimension of scheduling. The period (RMS) and the deadline (DMS) of a task alone do not provide sufficient criteria to represent the importance of the task and, therefore, the importance of the respective priority. Using both dimensions (urgency and criticality) is critical to providing sufficient and necessary criteria for determining the appropriate priority assignment method when accounting for mixed-criticality priority assignments. With the open-source code used to demonstrate QUADRANT, engineers can reproduce this method to assign priorities to their tasks systematically.

The simplicity of QUADRANT (i.e., 4 quadrants and simple-to-understand criteria) will allow engineers to effectively utilize QUADRANT without needing extensive expertise in scheduling theory.

The empirical validation of QUADRANT demonstrates that while classical scheduling theory has proven effective in scheduling tasks with sufficient priorities and a uniform criticality, the current challenges that arise from scheduling mixed-criticality systems underscore the current need for modern-day solutions.

In other words, QUADRANT bridges this gap by providing a practical solution that is supported by the results of exhaustive research validation rather than by worst-case pessimistic analyses. When scheduling safety-critical tasks, a scheduler with a 0% deadline miss rate is mandatory; thus, QUADRANT is currently the only algorithm from the variety of algorithms explored in this study that meets this requirement.

There are many avenues worth exploring. To verify whether the relative performance ranking observed during the simulation of CUQM running on a POSIX operating system matches that which has been observed in the simulation of CUQM running on an actual ARM Cortex-M4 (example: STM32F407) with hardware interrupt pre-emption and deterministic cache behavior, there must first be a validation test done on a bare metal (to ensure that the predicted performance placements of various priority and task configurations when using CUQM-has indeed occurred). A formal derivation of the schedulability bound of CUQM under the scarcity condition of 'nTasks > nPriorities' will be investigated, potentially using the modified Audsley OPA algorithm in conjunction with an additional constraint of 'critical group' equivalence. Integration of the CUQM process with AUTOSAR OS task mapping and the ISO 26262 levels of ASIL decomposition will enable the ease of direct use by automotive ECU developers within their current development processes. Validation of CUQM on multicore asymmetric (SMP) processing platforms and comparison to the heuristic approach described in Section VII. A will require empirical validation through measurement of inter-core interference. The classification of criticality for each CUQM-based task will be modified from the current binary classification to a four-level classification of ASIL levels (A-D) as defined in ISO 26262, thereby improving the generality, precision, and compatibility with the current path to formal certification.

Acknowledgments

The author thanks the FreeRTOS open-source community for the RTOS platform used in this research. All experimental data, task set designs, priority assignment algorithms, performance data, and findings contained within this document represent the author's original work and not the work of AI. The author used AI-assisted writing tools for only writing assistance, i.e., grammar corrections.

References

[1] Giorgio C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, 3rd Edition, New York, NY, USA: Springer, 2011. [CrossRef] [Publisher Link]

[2] L. Sha, R. Rajkumar, and J.P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-time Synchronization," *IEEE Transactions on Computers*, vol. 39, no. 9, pp. 1175–1185, 1990. [CrossRef] [Google Scholar] [Publisher Link]

[3] Glenn E. Reeves, "What Really Happened on Mars?," *ACM SIGADA Ada Letters*, vol. 17, no. 6, pp. 18–19, 1997. [Publisher Link]

[4] C.L. Liu, and James W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-real-time Environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973. [CrossRef] [Google Scholar] [Publisher Link]

[5] Joseph Y.-T. Leung, and Jennifer Whitehead, "On the Complexity of Fixed-priority Scheduling of Periodic, Real-time Tasks," *Performance Evaluation*, vol. 2, no. 4, pp. 237–250, 1982. [CrossRef] [Google Scholar] [Publisher Link]

[6] Joseph Y.-T. Leung, and M.L. Merrill, "A Note on Preemptive Scheduling of Periodic, Real-time Tasks," *Information Processing Letters*, vol. 11, no. 3, pp. 115–118, 1980. [CrossRef] [Google Scholar] [Publisher Link]

[7] Neil C. Audsley, "Optimal Priority Assignment and Feasibility of Static Priority Tasks with Arbitrary Start Times," Technical Report YCS 164, University of York, Department of Computer Science, 1991. [Google Scholar] [Publisher Link]

[8] Reinhard Wilhelm et al., "The Worst-case Execution-time Problem—overview of Methods and Survey of Tools," *ACM Transactions on Embedded Computing Systems*, vol. 7, no. 3, pp. 1–53, 2008. [CrossRef] [Google Scholar] [Publisher Link]

[9] Steve Vestal, "Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance," *28th IEEE International Real-time Systems Symposium*, 2007. [Google Scholar]

[10] Alan Burns, and Robert I. Davis, "*Mixed Criticality Systems—A Review*," Technical Report, University of York, pp. 1-97, 2017. [Google Scholar] [Publisher Link]

[11] Benny Akesson et al., "A Comprehensive Survey of Industry Practice in Real-time Systems," *Real-Time Systems*, vol. 58, pp. 358–398, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[12] P. Koopman, and M. Wagner, "Challenges in Autonomous Vehicle Testing and Validation," *SAE International Journal of Transportation Safety*, vol. 4, no. 1, pp. 15–24, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[13] N. Audsley et al., "Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling," *Software Engineering Journal*, vol. 8, no. 5, 1993. [CrossRef] [Google Scholar] [Publisher Link]

[14] Sanjoy Baruah et al., "Scheduling Real-Time Mixed-Criticality Jobs," *IEEE Transactions on Computers*, vol. 61, no. 8, pp. 1140-1152, 2012. [CrossRef] [Google Scholar] [Publisher Link]

[15] Robert I. Davis et al., "Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised," *Real-Time Systems*, vol. 35, pp. 239–272, 2007. [CrossRef] [Google Scholar] [Publisher Link]

[16] International Organization for Standardization, ISO 26262: Road Vehicles Functional Safety, 2018. [Online]. Available: https://www.iso.org/standard/68383.html

[17] Real Time Engineers Ltd, FreeRTOS Documentation, 2023. [Online]. Available: https://www.freertos.org/

[18] Paolo Gai et al., "A Comparison of MPCP and MSRP when Sharing Resources in the Janus Multiple-processor on a Chip Platform," *The 9th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 189-198, 2003. [Google Scholar]

[19] Sanyo K. Baruah, Alan Burns, and Robert I. Davis, "Response-time Analysis for Mixed Criticality Systems," *2011 IEEE 32nd Real-Time Systems Symposium*, pp. 34-43, 2011. [Google Scholar]