Justifying the Service Provided to Low Criticality Tasks in a Mixed Criticality System

Stephen Law*†
forename.surname@Rolls-Royce.com
*Rolls Royce Control Systems
Birmingham, UK

Iain Bate†
forename.surname@york.ac.uk
†The University of York
York, UK

Benjamin Lesage*†

ABSTRACT

Significant work has been presented over the last decade looking at the application of Mixed Criticality Scheduling. The premise being that if a failure occurs the scheduler performs a mode change from normal mode to high-criticality mode. In high-criticality mode, some low criticality tasks are given a reduced service (e.g. not executed or executed at a different period). Recently work has been performed to bound the number of low criticality jobs that might be skipped while the scheduler operates in high-criticality mode. However a significant gap in the analysis is to understand for how long the service to low criticality tasks may be reduced, i.e. how often the system switches to a high-criticality mode and how long the high-criticality mode is sustained. This is essential as part of supporting software certification. In this paper we consider a process, agnostic to the underlying scheduling strategy, designed to allow a system integrator to address this gap by assessing the level of service provided to low criticality tasks. The result is a safety argument with supporting evidence based on a real life case study, taken from a DAL-A certified aircraft engine control system.[1]

CCS CONCEPTS

• **Computer systems organization → Real-time system architecture**.

KEYWORDS

Mixed-Criticality, Robust scheduling, QoS, Real-time systems

ACM Reference Format:
Stephen Law, Iain Bate, and Benjamin Lesage. 2020. Justifying the Service Provided to Low Criticality Tasks in a Mixed Criticality System. In *28th International Conference on Real-Time Networks and Systems (RTNS 2020), June 9–10, 2020, Paris, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3394810.3394814

[1]This paper is based on a paper accepted for Ada Europe 2019, however it was never published.

1 INTRODUCTION

Real time embedded software tasks developed for safety critical systems, such as civil avionics engine control systems, are typically developed according to a specific Development Assurance Level (DAL) [22]. The DAL indicates a criticality level for a component and is assigned based on the consequence to the system's safety that a failure of this component could cause. This paper considers the model presented in DO-178C [21], that defines DAL-A as the highest criticality level and DAL-E the lowest.

It is typically assumed that the amount of effort assigned to producing enough evidence to prove the correct operation of a software component is directly proportional to its DAL [23]. In practice though, it is still desirable that low DAL software operates as expected; a low DAL component may also be essential to achieve the desired customer capability. Put simply a task's criticality is not necessarily related to its 'importance'. Furthermore, a low DAL task may still be bound by strict temporal requirements; a task's criticality is independent from its temporal properties.

In the literature a Mixed Criticality System (MCS) is a system which combines software of multiple criticalities on the same processor. The dominating model that has emerged in the field of Mixed Criticality Scheduling is that of the double computation time model first introduced by Vestal [23]. The model uses two measures of Worst Case Execution Time (WCET) for each task: one measure for the low criticality mode (C^{LO}), and a higher figure for the high criticality mode (C^{HI}). In practise C^{HI} may be derived from a sound WCET analysis whereas C^{LO} could be derived from a best effort approach. This model essentially allows a system integrator to capitalise on the (strong) assumption that a high criticality task's C^{HI} is pessimistic, while its C^{LO} may be reliable, but optimistic. This is facilitated by allowing low criticality tasks to execute as long as all tasks within the system execute within their C^{LO} bounds. If any tasks execute beyond these bounds, service to low criticality tasks may be reduced or stopped.

Regardless of the scheduling methodology employed, the general assumption in the literature is that low criticality components can be denied service at times of heightened system utilisation. In practise, in a well-designed system this should only occur in extreme cases, if ever. Unfortunately, this potential denial of service cannot be quantified through the schedulability analysis and the performance of the integrated system in operation needs to be considered to understand how long and how often loss of services may occur. This is because it is not known how many tasks, if any, may execute beyond their timing bound within a certain time window without executing the system in a representative environment. In essence, it is difficult to understand the performance afforded to low criticality tasks in a MCS without performing a dynamic assessment in a representative environment. This means it is difficult to obtain concrete proof that a lower DAL component with strict timing requirements will receive a good enough level of service to fulfil its mission requirements without modelling it as a high criticality component with the additional certification effort required.

This paper considers how a system integrator may develop a low DAL task and express its requirements, with a use case taken from a real aircraft engine control system. The aim is to describe a process that could be employed to assess the level of service afforded given to a low criticality component in order to allow an informed decision on system performance to be made, and by providing evidence supporting timing requirements with a sufficient confidence level to support certification arguments.

The key contributions of this paper are as follows:

- To provide a process for assessing the service afforded to a low DAL task, agnostic of scheduling methodology.
- To assess how realistic claims can be expressed as part of a certification case or argument.
- To demonstrate the proposed process in the context of a real aircraft engine control system application.

2 SYSTEM MODEL

We introduce in this section the system model underlying our evaluation. This paper focuses on the Robust Mixed Criticality Model without loss of generality. The reason for focusing on this model is that the concept of being able to carefully manage a graceful degradation to the system, including the introduction of tasks that can be disabled for specific periods of time (encompassing the weakly-hard model), is well suited to the case study investigated here. Whereas the classic MCS models, such as the AMC, offer no graceful degradation and instead subject low criticality tasks to an immediate drop in service, if required.

Note that the low criticality task service assessment conducted in this paper relies on the statistical analysis of simulations of the schedule. As such it is designed to be agnostic to the underlying scheduler methodology, provided the schedule can be accurately simulated. Additional results on the application of the service assessment method on the Robust and AMC+ MCS models are available in [15].

2.1 Task model

A system is defined as a collection of tasks τ_i, where $1 \leq i \leq N$, each denoted by a deadline D_i, a period T_i, a release jitter J_i, a criticality level L_i, and one or many WCETs C^{L_i}. A task DAL level may not be related to its criticality; a low DAL task may be modelled as low or high criticality but the computation of higher criticality C^{L_i} comes at the cost of additional certification effort. Conversely higher DAL tasks require stronger guarantees that their timing requirements are met. The release jitter J_i denotes the maximum permissible variation of the period T_i for the release of the task. Each task is assigned a fixed, unique priority P_i, where $1 \leq P_i \leq N$. Tasks may be clustered in a smaller number of RTOS tasks to reduce the impact of system overheads [17]. Additional constraints between tasks are captured in the form of transactions, if the execution of one task τ_i relies on the completion of another task τ_j. Transactions support ensuring that a set sequence of activities completes in the correct order with a set deadline.

Without loss of generality, we consider only two criticality levels, high and low, with their respective timing bounds C^{HI} and C^{LO}. C^{LO} can be derived from best effort approaches, while C^{HI} is a sound estimate of the task's WCET. A task τ_i is said to be schedulable if its worst case response time (WCRT) R_i, is less than its deadline D_i. A task is said to have a hard deadline ($D_i \leq T_i$) if its execution must meet every deadline. On the other hand, a robust task may drop jobs depending on the current system mode. Switch to different modes may occur after a single or multiple jobs overrun their tasks' C^{LO} as discussed in the next Section.

2.2 Robust Mixed Criticality Systems

The Robust Mixed Criticality Model as presented by Burns et al. [7] introduced the following definitions for a robust mixed criticality system:

Definition 1. A *robust* task is one that can safely drop one non-started job in any extended time interval.

Definition 2. The robustness of a complete system is measured by its F count (how many job overruns can it tolerate without jobs being dropped or deadlines missed) and its M count (the number of job overruns the system can tolerate once each robust task has dropped one job).

Definition 3. A *resilient system* is one that employs forms of graceful degradation that adequately cope with more than M overruns.

A *fault* is measured when one task overruns its C^{LO}, where as an *error* is the manifestation of a one or many faults and represents the point where a task fails to adhere to its timing requirements. A resilient system is designed to cope with one or many faults, while avoiding errors. The resilient system model introduced in [7] recognises 4 different criticality modes:

- low criticality, where all tasks are allowed to execute,
- F-mode or fail robust,
- M-mode or fail resilient,
- and high criticality.

Mode switching occur on successive faults, i.e. when tasks overrun their C^{LO}. The resilient system model is capable of coping with F faults (F-mode or fail robust), before reverting to a mode where robust tasks skip their jobs (M-mode or fail resilient). At this point the system is capable of coping with further faults up to a total of M faults, where $F < M$. This provides a set bound on the size of a robust task skip burst. Once the fault count increases above M, the system reverts to a high criticality mode where no service is provided to low criticality tasks. Once the system reaches the idle state, the failure count is reset and, if required, the system reverts to the normal mode.

The schedulability analysis presented in [7] provides a proof that high criticality, robust tasks, meet their schedulability requirements. The analysis also provides a bound on the number of jobs a robust task may skip between idle points. However, the analysis provides no guarantees on the service given to low criticality tasks, or indeed the time between individual robust task skip bursts.

The aim of the process presented in the remainder of this paper is to assess the performance, and the failure rate afforded to a robust or low criticality task in a representative system. As no service is provided to low criticality tasks in the high criticality mode, it is important to understand how often and for how long the system may sustain the high criticality mode. The statistical analysis is designed to then provide a system integrator with the information, and confidence, required to make an informed decision on whether a robust or low criticality task will meet its temporal requirements.

3 CASE STUDY

The case study investigated as part of this paper is taken directly from a Rolls-Royce Aircraft Engine Control System [17]. The existing system consists of a set of several hundred tasks in the order of tens of thousand lines of code. In order to support this study the code base has been ported to a scheduler designed to implement a robust system [7]. The ported system consists of 17 cluster tasks with support mechanisms provided by a commercial-off-the-shelf RTOS [17]. All tasks in the system are currently certified and ran respectively as high DAL and high criticality ones to satisfy to their timing requirements.

In order to provide a secure record of engine performance, the control system regularly writes system parameters to flash memory. The flash memory is non-volatile and secure, but the time taken to write to this flash memory is considerable. The control system contains a periodic task responsible for writing data to flash memory. This task reads from a memory buffer, written to by other tasks, before copying the buffer to flash. The task's execution time being directly proportional to the amount of data being written. The amount of data written to the data store is thus minimised as far as possible to prevent continuous blocking induced by the task. In order to support future design and maintenance goals, it is desirable to reduce this limitation.

The task is a lower DAL task with strict timing requirements. While failure would have limited safety implications, the shared buffer should be cleared on a regular basis to ensure other tasks can update their parameters without data loss. At present the task is certified as a high DAL, high criticality component and treated as a hard real time task to ensure its strict requirements are met. However the task could more easily be designed to execute for longer, with an assumption that it may periodically drop jobs. The move to a robust system allows such requirements to be expressed, verified, and enforced through necessary mechanisms to protect the wider system.

As part of a transition to a more flexible, mixed-criticality system design, we reassessed the DAL level and criticality of the flash memory task to one reflecting its lower impact on the system overall safety. The newly configured flash memory task is designed to continuously write data when called to do so. If a job of the task is skipped, then it will simply resume writing to memory from the next entry in the memory buffer. The principal requirement is that the memory buffer does not overflow, and so the task is designed to write more data than necessary on each invocation. So following a period of reduced service the task is able to recover and return to normal operation provided it has sufficient time.

The newly configured robust low DAL flash memory task was integrated into the schedulability analysis of the control system. The task execution time budget was increased, while its period was decreased, to reflect the operational change in its behaviour. Overall this increased the permissible utilisation of the task by a factor of 60. The system was shown to be schedulable in the low criticality, fail robust (F-mode), fail resilient (M-mode) and high criticality modes as defined by the robust model [7]. This increase in utilisation was only permitted thanks to the use of a mixed-criticality system exploiting the difference between the analysed (sound, safe and pessimistic) WCET used for the C^{HI} and the (test measured, robust but potentially optimistic) system high water mark time used for each task's C^{LO}.

For this analysis, the following assumptions surrounding the robust task have been defined:

- Due to the task's increased execution budget, if given full service the task is capable of writing data to flash memory at a faster rate than the reporting tasks can write data to the shared memory buffer
- The shared memory buffer is sufficiently large to allow the flash memory task to skip up to four jobs
- Once the flash memory task skips a burst of up to four jobs, the task must execute the following four jobs in order to allow the task to avoid data loss.

Therefore the overriding requirement for analysis is that each time the robust flash memory task suffers a job skip burst, it should have a clear period of at least four successful executions before it can skip a job again. If the task skips a job in less time, the task is said to have suffered an error. The task period itself is 12.5ms, therefore the basic requirement for the task can be defined as follows:

Definition 4. A flash memory task error is recorded when a job skip bursts lasts more than 50ms or two separate job skip bursts occur with a less than 50ms interval.

The following assumptions have been made about the wider system:

- The occurrence of an individual task overrun is very rare. *Rationale: The defined C^{LO} for each task, representing the computation time beyond which a task would register a failure, has been generated from an extensive testing regime and carries with it a high level of confidence. However, being derived from a simple measurement technique it is still assumed to be optimistic.*
- Individual task overruns are independent and are not reliant on the current operation of the control system. *Rationale: an overrun is an event unique to each task, and not a systematic event caused by an error or operation at the system level.*
- Task overruns can be assumed to be independent of hardware operation. *Rationale: The system is designed to be resilient to external hardware failures, secondly the target processor design is compliant with DO-254 as a high criticality device.*

The requirements on the robust task can be expressed as a $(4-8)$-firm deadline [9]. By scheduling the robust task in the high criticality mode at a reduced rate, the schedulability analysis would then verify it is allocated sufficient service. However the contribution of the robust task to the utilisation of the high criticality mode renders the system unschedulable. Instead, we propose a process to assess the service received by the robust task and validate its requirements, by considering how often the system switches to high-criticality mode and how long the high-criticality mode is sustained.

4 CERTIFICATION REQUIREMENTS

This paper focuses on an aerospace application, and so the guidance provided by DO-178C [21] is used as the focus of this work. However the guidelines are considered similar to those detailed in other software domains such as ISO26262 and IEC61508 [10]. The focus of this work is predominately on understanding the performance of a robust and/or low criticality component, the certification requirements surrounding software partitioning and mixed criticality have been assessed in a parallel work and are not discussed further in this paper. Therefore this paper will only consider temporal performance.

DO-178C requires that the certification documentation is able to justify the accuracy, correctness and robustness of the system. This requires an understanding of the temporal performance of the system (with respect to task WCETs and system schedulability), and confirmation that the system's temporal requirements have been met.

When reviewing low criticality and/or robust tasks, this requires the system integrator understand the potential error rate of each task's temporal requirements, and furthermore show that this rate is acceptable given the system wide effect of a temporal requirement error.

The prior work in the field of MCS has presented static analysis models for proving the service afforded to high criticality tasks. However the previous work has not yet offered solutions for system integrators to generate the evidence required to meet the low criticality task certification requirements discussed in this section.

5 ANALYSIS OF ROBUST TASK PERFORMANCE

Regardless of the method chosen to control low criticality or robust tasks, the performance of said tasks is wholly dependant on the actual performance of the system. Therefore the process conducted here is based on a statistical assessment of a set of execution results extracted from either a test rig execution during a system-level test campaign, or from a scheduler simulation of the system in question. This paper predominately follows the results obtained from simulation. The use of which allows a significantly larger data set to be compiled, de-risking the system design early in its life cycle. Ultimately these results are supplemented and improved as testing of the system progresses by test data obtained from a full end to end test campaign.

The simulator is initialised using execution profiles extracted from the system during task-level testing designed to mimic system behaviour while in operation [16]. This ensures that the execution profiles provide a realistic representation of the task's actual performance when in operation. These execution profiles, including RTOS overhead measurements, are input into a bespoke scheduler simulator which is executed on a high performance server over a thousand times in order to build up a comprehensive set of results. The simulator has been validated against the real system, and is designed to be reviewed and improved as additional real system performance data is generated.

The execution time of each task is output by the scheduler, as is information on whether a task executes, or is blocked. The data output by the simulator is analysed to measure the time between each data-set skip. A single execution simulates thirty minutes of scheduler time.

The results are then input into the statistical assessment introduced in this section, that aims to provide a confidence in adherence to the low criticality component requirements, as well as providing an understanding of the probability of the component's requirement being broken. Together these results should allow a system integrator to make a guided decision on whether the low criticality component's performance is acceptable or not. This forms the principal contribution presented in this paper.

The following sections introduce a Goal Structured Notation (GSN) argument for the approach, as well as providing results from applying the analysis to the Rolls-Royce case study introduced in Section 3.

5.1 Goal Structured Notation

Within this and the following sections, we use the Goal Structuring Notation (GSN) [14]. GSN is a widely used approach within the industry [1]. The principal purpose of a goal structure is to show how goals (claims about the system) are successively broken down into sub-goals until a point is reached where claims can be supported by direct reference to available evidence (solutions). As part of this decomposition, using the GSN it is also possible to make clear the argument strategies adopted (e.g. adopting a quantitative or qualitative approach), the rationale for the approach (assumptions, justifications) and the context in which goals are stated (e.g. the system scope or the assumed operational role). The GSN arguments in paper use Goals (G), Assumptions (A), Statements (St), Context (C), and Solutions (S). For further details on GSN see [14].

Figure 1 shows a GSN argument that expands how Definition 4 is assessed. The principal claim (G0) that two job skip bursts will not occur within 50ms, is analysed using a statistical analysis of results obtained by a simulation of the system.

This is in the context that the simulation is a representative example of the real system, and that the overall estimated error rate is acceptable. This error rate can then be taken forward and included as part of a system wide certification case for the low criticality or robust software.

The strategy for the analysis is broken down into four key sub claims, as follows:

- G1 - Confidence - The simulation achieves appropriate coverage
- G2 - Likelihood - The simulation output provides an understanding of the likelihood of an error
- G3 - Correctness - The simulation is a valid representation of a real system
- G4 - Acceptability - The minimum time between data-set skips is acceptable.

The following sections describe each of these goals in more detail.

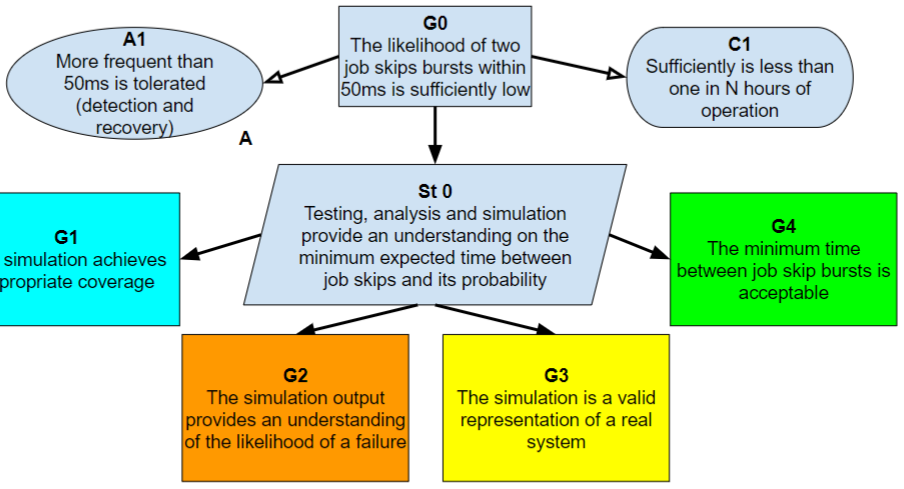
A1
More frequent than 50ms is tolerated (detection and recovery)
A
G0
The likelihood of two job skips bursts within 50ms is sufficiently low
C1
Sufficiently is less than one in N hours of operation
St 0
Testing, analysis and simulation provide an understanding on the minimum expected time between job skips and its probability
G1
The simulation achieves appropriate coverage
G4
The minimum time between job skip bursts is acceptable
G2
The simulation output provides an understanding of the likelihood of a failure
G3
The simulation is a valid representation of a real system

Figure 1: Goal Structured Notation Argument for the Overall low DAL Requirement

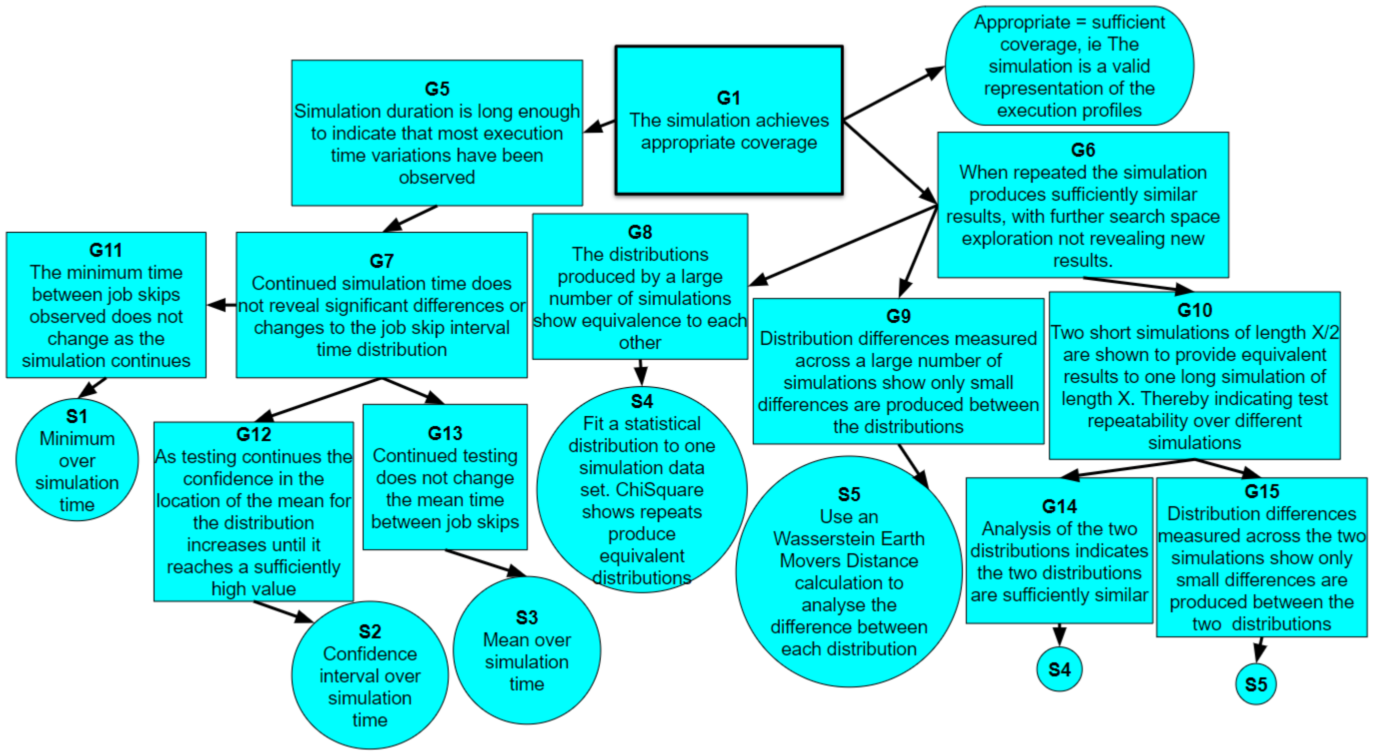
G5
Simulation duration is long enough to indicate that most execution time variations have been observed
G1
The simulation achieves appropriate coverage
Appropriate = sufficient coverage, ie The simulation is a valid representation of the execution profiles
G6
When repeated the simulation produces sufficiently similar results, with further search space exploration not revealing new results.
G11
The minimum time between job skips observed does not change as the simulation continues
G7
Continued simulation time does not reveal significant differences or changes to the job skip interval time distribution
G8
The distributions produced by a large number of simulations show equivalence to each other
G9
Distribution differences measured across a large number of simulations show only small differences are produced between the distributions
G10
Two short simulations of length X/2 are shown to provide equivalent results to one long simulation of length X. Thereby indicating test repeatability over different simulations
S1
Minimum over simulation time
G12
As testing continues the confidence in the location of the mean for the distribution increases until it reaches a sufficiently high value
G13
Continued testing does not change the mean time between job skips
S4
Fit a statistical distribution to one simulation data set. ChiSquare shows repeats produce equivalent distributions
S5
Use an Wasserstein Earth Movers Distance calculation to analyse the difference between each distribution
G14
Analysis of the two distributions indicates the two distributions are sufficiently similar
G15
Distribution differences measured across the two simulations show only small differences are produced between the two distributions
S2
Confidence interval over simulation time
S3
Mean over simulation time
S4
S5

Figure 2: Goal Structured Notation Argument Exploring the Confidence of the Analysis

5.2 G1 - Confidence

In order to properly understand the performance of the system it is vital that the statistical analysis is performed across a significantly large sample that represents the real performance of the system. Claim G1 aims to confirm this is the case and aims to understand whether enough testing has taken place.

Figure 2 shows the extension to claim G1. This claim is fulfilled by ensuring the simulation executes for long enough to indicate that most execution time variations have been observed (G5) and that further exploration of the search space does not reveal new results (G6).

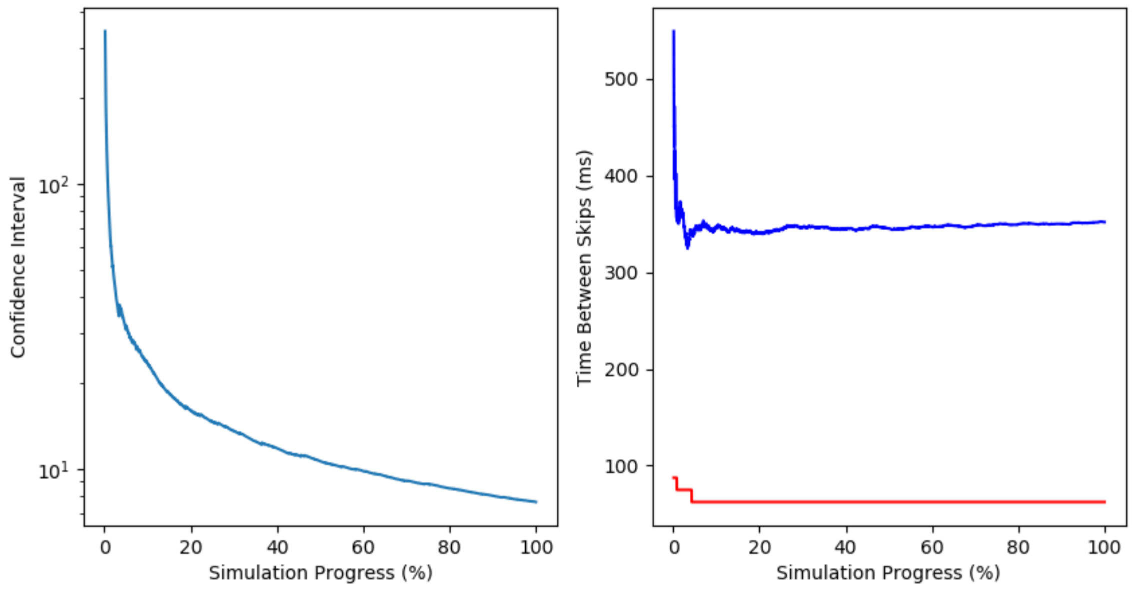

Figure 3: Changes in Confidence Interval (left), Mean (right top) and Minimum (right bottom) of the Time Between Job Skips Over Simulation Time

5.2.1 G5. Claim G5 is concerned with understanding whether a single simulation executes for long enough, and is supported by an assessment that reviews whether continued simulation reveals any additional differences or significant differences in the distribution. This is important to understand as it helps build the argument that the statistical analysis is performed across a fully representative set of execution profiles. This is tested by reviewing the minimum time between job skips (G11), as well as the confidence interval (G12) and the mean (G13). In all cases the aim of the assessment is to review whether, as the simulation continues, the results have converged and significant differences are not expected. Convergence is assessed via a defence in depth approach through different independent routes.

Figure 3 shows the results from one execution of the simulator and illustrates the variation in the confidence interval, mean and minimum as the simulation progresses. The confidence interval in particular captures the range within which there is 95% confidence that the mean resides. As an example, the results show that the mean time between job skips starts around 450±100ms and converges at the end of the simulation to 350±10ms. Despite a significant amount of variability initially, the confidence interval significantly decreases around approximately 10ms, and the mean and minimum converge around 350ms and 60ms respectively. The key to analysing these plots is to identify whether the simulation results are changing as the simulation continues, or in essence do the results indicate that further exploration does not reveal any new or different results.

5.2.2 G6. Claim G6 is concerned with understanding whether a large scale evaluation with numerous simulations produces a similar result to that of a single simulation. This aims to provide further confidence that the search space has been explored sufficiently. The claim confirms firstly whether the analysis is repeatable, that is the results from multiple short simulations create a combined result equivalent to one long-running simulation.

Claim G6 is supported by an assessment of the distributions of time intervals between job skips over 1000 iterations of the simulator using both a χ^2 distribution equivalence test (G8) and an Earth Movers Distribution (EMD) (G9). In both cases the simulation from the first test is used for comparison against the remaining 999. Secondly G10 claims that when two short simulations are appended together they provide equivalent results to one long simulation.

Figure 4 shows the EMD result from executing 1000 simulations. In each case each, the distribution of time interval between job skips was randomly sampled by selecting a slice of the simulation. Increasingly bigger portions of the simulated scheduler time were considered (1%, 5%, 10%, 20%, 40%, 60%, 80%, 100%) to compare increasingly longer simulations. This randomly sampled set was then compared, using an EMD test, to a random sample of the same length taken from the first simulation. As can be seen from Figure 4 the larger the chosen sample, the closer the two randomly selected distributions, secondly the results are shown to converge as more data is appended to the sample.

Secondly the results of the first simulation was fitted to a distribution in order to produce an expected distribution to test against. We fitted the first distribution against an exponential distribution; amongst all considered distributions, the exponential distribution provides the best fit according to a Kolmogorov-Smirnov Goodness of fit test ($p < 10^{-70}$). Each of the subsequent 999 distributions produced by each of the simulations were then compared to this fitted distribution using a χ^2 distribution equivalence test. The job skip

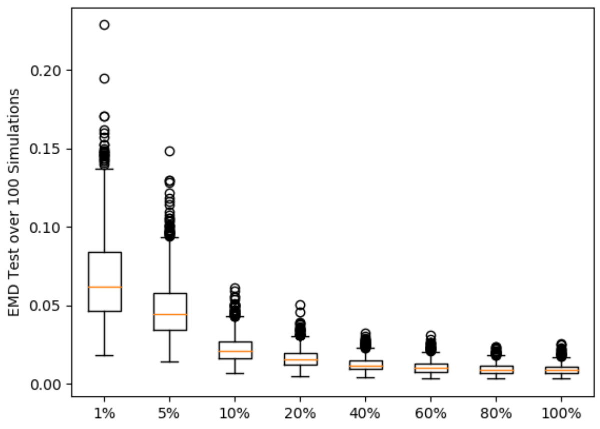

Figure 4: Comparison of EMD over 1000 Simulations

intervals were binned to apply the χ^2 equivalence test, i.e. the samples were placed into intervals of geometrically increasing size from 50ms to 2s such that sufficient observations are available per bin and more precision is achieved close to the analysed task requirement. The equivalent test showed all simulations were produced from the same population (mean result - $\chi^2(12, n = 5080) = 171, p < 0.01)$[2].

5.3 G2 - Likelihood

Once a robust and reliable data set has been generated, the next step in the process is to analyse the results to understand the probability of breaking the requirement. This provides a real measure that can be used to make a decision of whether the service given to the low criticality task is acceptable or not.

Figure 5 shows the process for understanding the probability of the flash memory component suffering a timing requirement error. Claim G2 is split into two parts, the first is an assessment based on the observed performance of the system (G16, G17), and the second is a statistical inference to understand the exceedance probability of the sample (G18).

5.3.1 G16, G17. The principal objective of viewing the range of simulated results is to gain confidence that the minimum, and any results that are close to the minimum requirement, occur with a low probability. That is, denials of service to the low criticality tasks are infrequent.

Figure 6 shows the range of results obtained during one simulation. The main aim of reviewing the figure is to assess how far from the minimum requirement the majority of the inter-quartile range lies. In particular to provide confidence; the majority of results should lie well above the requirement.

To further understand the extreme values in the simulation a percentile test was applied to the full set of 1000 simulation results obtained in Section 2, the results showed that 62.5ms represented the 0.1% percentile. The analysis is expanded in Table 1 which shows the percentage results observed close to the requirement of 50ms.

These results should be used to provide some confidence that the vast majority of results are observed well above the minimum requirement providing confidence that a breach of the requirement is a one-off, rarely seen, event.

5.3.2 G18. Understanding the probability of breaking the requirement is assessed in one of two ways. The probability of exceeding the requirement is preferably estimated using an Empirical Cumulative Distribution Function (ECDF). However, if the requirement has not been broken during the simulation or there is an insufficient number of observed job skips, ECDF might result in inaccurrate estimates. In such scenarios, the exceedance probability is estimated using a fitted distribution. Both techniques require the analysed samples to be independent and identically distributed (i.i.d.). Individual task overruns are assumed to be independent of each other and of the prior behaviour of a task (§ 3). This is modelled in the simulations as the execution time of each job is randomly sampled from its task's execution time profile, a representation verified as correct under G3.

We picked the results of one simulation which did **not** break the requirement and thus cannot be analysed through ECDF. The simulation was fitted to an exponential distribution in order to produce a continuous distribution for analysis ($p < 0.01$). The probability of obtaining a job skip interval of less than the requirement with this distribution was found to be 4.8%.

Secondly the distributions of job skip intervals from all 1000 simulations were combined, as justified by claim G6, thus covering all observed of job skip intervals during our simulations. In this case this extended distribution did include multiple instances where the requirement had been broken. Using an ECDF function, the probability of error was found to be 0.005±0.002% (with 99.99% confidence). While the fitted distribution results in more pessimistic results it operates even with a far smaller sample size.

5.4 G3 - Correctness

The results obtained so far have focused on a simulation of the system. This is advantageous as the simulation can provide a much larger data set to analyse than is possible from execution on a real system test rig, secondly the results can be generated much faster than possible on real hardware. However, it is important to review the statistical results obtained to verify that they provide a valid representation of the real system.

Figure 7 extends the GSN argument and examines how the analysis provides representative results of the actual system performance. Firstly the claim assumes the simulation has been executed for a sufficiently long amount of time, as verified by claim G1 in Figure 2. Secondly the claim is verified using real results obtained from test rig operation (G21 and G22).

5.4.1 G21. Claim G21 concerns the input timing profiles used to generate the simulator results. As noted in the introduction to this section, the simulation is setup using a set of task timing profiles generated through task-level execution in a representative environment, as detailed further in [18]. These timing profiles provide

[2] χ^2 results throughout this paper are denoted using the following terminology - χ^2([*degrees of freedom*], n = [*number of samples*)] = [*result*]), [*statistical significance*]. If the statistical significance (or p value) is less than 0.01, then the two compared distributions can be said to come from the same population, i.e. they are not independent.

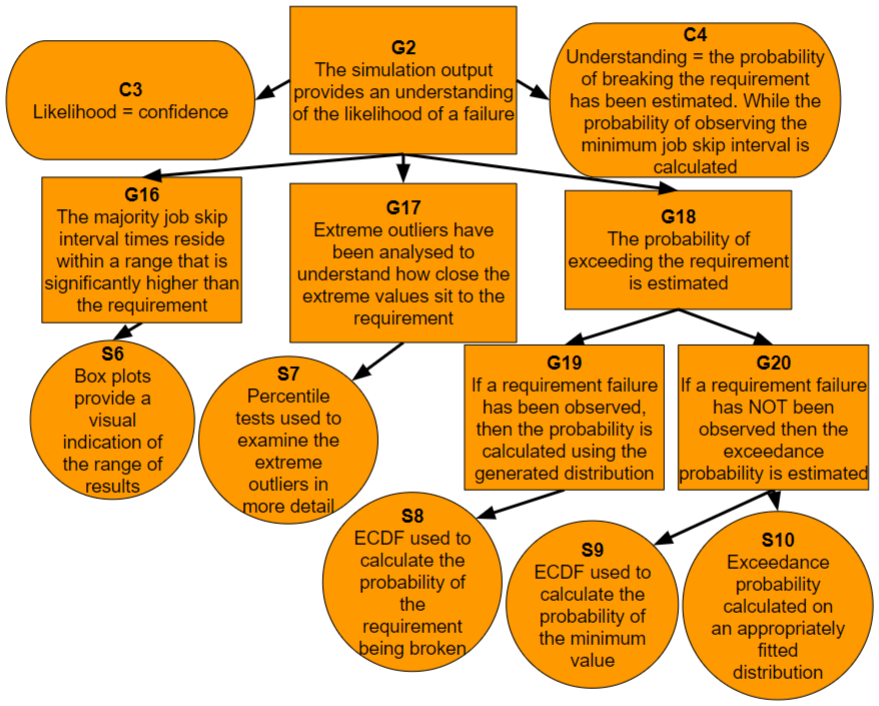
C3
Likelihood = confidence
G2
The simulation output provides an understanding of the likelihood of a failure
C4
Understanding = the probability of breaking the requirement has been estimated. While the probability of observing the minimum job skip interval is calculated
G16
The majority job skip interval times reside within a range that is significantly higher than the requirement
G17
Extreme outliers have been analysed to understand how close the extreme values sit to the requirement
G18
The probability of exceeding the requirement is estimated
S6
Box plots provide a visual indication of the range of results
S7
Percentile tests used to examine the extreme outliers in more detail
G19
If a requirement failure has been observed, then the probability is calculated using the generated distribution
G20
If a requirement failure has NOT been observed then the exceedance probability is estimated
S8
ECDF used to calculate the probability of the requirement being broken
S9
ECDF used to calculate the probability of the minimum value
S10
Exceedance probability calculated on an appropriately fitted distribution

Figure 5: Goal Structured Notation Argument Exploring the Probability Assessment of the Requirement

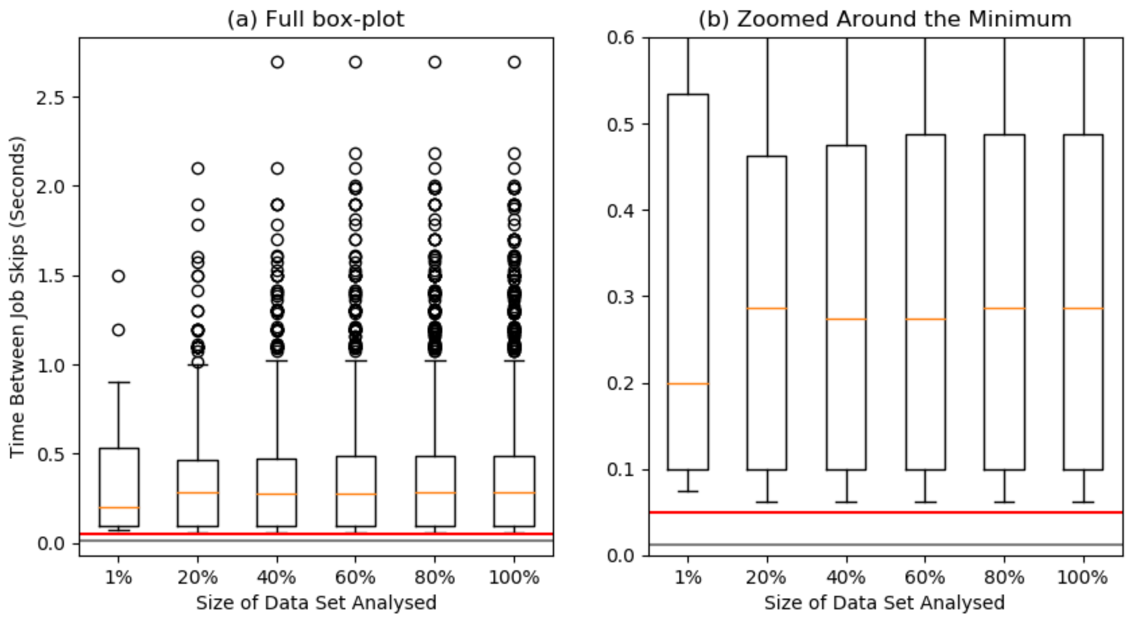

Figure 6: Box Plot Diagrams Showing the Range of Job Skip Interval Times, With a Zoomed-Plot on the Right Around the Minimum Requirement

Time Between Skips	% Results More Frequent Than Time Between Skips
50ms	99.9948%
60ms	99.9947%
70ms	99.76%
80ms	98.73%
90ms	96.76%
100ms	75.43%

Table 1: Percentage Results Observed Close to the Minimum Requirement After 1000 Tests

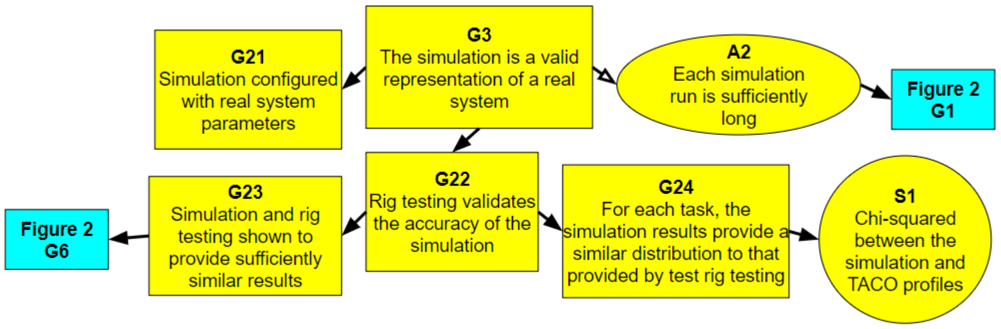
G21
Simulation configured with real system parameters
G3
The simulation is a valid representation of a real system
A2
Each simulation run is sufficiently long
Figure 2
G1
Figure 2
G6
G23
Simulation and rig testing shown to provide sufficiently similar results
G22
Rig testing validates the accuracy of the simulation
G24
For each task, the simulation results provide a similar distribution to that provided by test rig testing
S1
Chi-squared between the simulation and TACO profiles

Figure 7: Goal Structured Notation Argument Exploring the Correctness of the Analysis

a representative set of results for the scheduler simulation to randomly iterate over.

Secondly once a full system test rig campaign has been completed, the results from the real system should be used to both improve the simulator, and to compliment the simulation produced results in order to improve accuracy. This full system test campaign is expected to provide a significantly large set of results to boost confidence in the statistical analysis, arguably approaching a point where the simulation may not be required. However, these results would be expected to take significantly longer to generate, and would be provided at a time in the software design life-cycle too late to allow cost effective improvement.

One risk with this approach is that the test rig campaign may indicate the simulation does not deliver representative results, this is a significant risk with any approach utilising a simulator and is in this case unavoidable. The risks are mitigated by the fact that, as is frequently the case, the software project contains a number of legacy components, for whom timing data should exist, but is also mitigated by an assumption that the simulation can be refined as soon as software testing begins, rather than waiting for its completion. The key is that the simulation provides an easy environment for fast and efficient whole system analysis.

5.4.2 G22. The second step to understanding if the results represent the real system is to compare a set of the produced simulation results against results obtained from the real system to ensure that the results are both sufficiently similar. To do this, a subset of test rig results should be used to repeat the distribution analysis conducted to confirm claim G6 in Figure 2 in order to verify that the sub-set of test rig results produce a similar distribution to the super-set of simulation results. Comparison to a real system test campaign is, at this point, left to future work.

5.5 G4 - Acceptability

The analysis process has so far shown how results can be produced that explore the behaviour of a representative system. This has been executed until sufficient coverage of the system has been produced, resulting in an assessment of likelihood, which has finally been verified against real system behaviour. However the only way to assess whether the results indicate a low criticality task will receive enough service is down to an engineering judgement. This assessment must take into account the resilience, accuracy and correctness of the wider system in the face of low criticality task errors, as well as the individual temporal requirements of the low criticality task.

It is important to remember that the system must be resilient to robust task errors anyway, otherwise the task could not be treated as a robust task. However, should the results prove unacceptable, the next step would be to assess which tasks indicate overruns, altering their C_{LO} figures accordingly. The use of a scheduler simulator facilitates an easy design-analyse-rework cycle that simply could not be achieved with a full test campaign.

5.6 Related Work

Vestal [23] was one of the first publications to consider the schedulability of a MCS. The work draws the comparison that the reliability of the WCET figure used for each task is proportional to its criticality. This is based on the observation that low DAL/criticality tasks are not developed, or verified, to the same extent that high DAL/criticality tasks are, and therefore the output WCET figures cannot be expected to be as reliable.

Building off Vestal's work, Baruah et al. [2] introduced Adaptive Mixed Criticality (AMC). The AMC protocol de-schedules *all* low criticality tasks if *any* high criticality task executes for longer than its C^{LO}. The original AMC algorithm was extended to delay the system's switch to the high criticality mode. The bailout protocol uses a 'Bailout Fund'; a measure of how much slack time is currently in the system [3]. Should the Bailout Fund fall below zero, then the system reverts to a high criticality only mode until such time as the system reaches idle, or the Bailout Fund increases above zero. This protocol essentially delays, potentially suspending completely, entry into the high criticality mode using the assumption that it is unlikely that several tasks will overrun their C^{LO} at the same time.

While low criticality tasks may have less stringent timing or verification requirements, they are still important for the correct operation of the system. Several papers have thus explored approaches to improve low criticality task support and offer more realistic models where low criticality tasks are not abandoned [6]. In particular, the temporal properties of a low criticality tasks can be altered, reducing its service, or execution time, without affecting its overall requirements. Jan et al. [13] looked at applying the elastic task model, originally proposed by Buttazzo et al. [8], to a MCS. Rather than de-scheduling all tasks, this model instead extends the period of low criticality tasks to reduce the utilisation on the system. In contrast, the so-called imprecise mixed criticality model [19] reduces low criticality execution budgets in order to improve wider system performance. Finally, another approach reduces the priority of low criticality tasks as required, effectively executing low criticality tasks during periods of high system utilisation in system slack time only [5]. However, this is wholly dependent on the expressed requirements of the low criticality task being flexible enough the support the proposed models.

Less stringent requirements were introduced to allow for a definition of the Quality of Service (QoS) offered to a task in the form of $(m-k)$-firm deadlines [11], where a dynamic failure occurs only if fewer than m out of k consecutive jobs of a task fail to meet their deadlines. The weakly hard real-time system model proposed by Bernat et al. [4] generalises this model by allowing non-consecutive deadline misses over a k jobs window. Typical Worst-Case Analysis [12] provides for an evaluation of the QoS offered to a task by bounding the number of deadline misses it can suffer in a weakly hard real-time system.

The AMC analysis was extended in [9] to support weakly hard real-time systems such that service is provided to low criticality tasks in high criticality mode by allowing them to run a reduced number of jobs in a given cycle. The analysis effectively determines whether or not all low criticality tasks will at least meet a $(m-k)$-firm deadline requirements. Medina et al [20] also considered a $(m-k)$-firm model but to delay the need to switch to a high criticality mode. No service is provided to low criticality tasks once the switch occurs, and tasks may be skipped if their predecessor exceeds its budget. They rely on a probabilistic process for assessing the availability of low criticality tasks, assuming timing error rates are known. While these approaches can provide some guarantee on the minimum service offered to a task, neither can provide for an evaluation of how often, and how long the system sustains a high criticality mode under different strategies.

The resilient model [7] utilises graceful degradation to improve low criticality task performance, by delaying the switch to the high criticality mode and the loss of service. Similarly to the weakly hard systems, robustness is supported at the task level where certain 'robust' tasks[3] are capable of skipping individual jobs when requested. In addition, the approach employs resilience at the system towards supporting a certain number of timing failures. Service guarantees are provided under each scheduler mode, based on the number of supported timing failures, up to the point where low criticality tasks have to be dropped.

In summary, MCS previous research has focused first on high criticality task requirements, with the static analysis showing that in the worst case low criticality tasks will receive no service. Methods such as elastic scheduling or weak hard real-time aimed to improve low criticality tasks overall service, with graceful degradation aimed to provide some control on the occurrence of service loss for low criticality tasks, or the reduction in service. Even though the move to the high criticality mode may be delayed, it still may occur at some point resulting in no or degraded service for all low criticality tasks in the system. The methods offer many ways to help improve or guarantee low criticality task performance under adverse circumstances, but have not addressed how to assess this performance and service in a meaningful way especially to support certification, i.e. when a mode change may occur, or what impact (by way of duration and frequency of loss of service) this may have on the low criticality, and/or robust, tasks, when all tasks are not continuously assumed to be running to their allocated budgets.

6 CONCLUSION

This paper has presented an approach to verifying the service afforded to low criticality tasks in a MCS. The approach presented is agnostic to the choice of scheduling methodology and instead focuses on a dynamic statistical process based on analysis of task performance within a representative environment. The approach was applied to a real aircraft engine control system. This allowed a formerly low DAL, high criticality task to be re-designated as a robust low DAL, low criticality task with its permissible utilisation being increased by a factor of 60. The analysis aimed to provide a confidence and understanding in the service afforded to the low DAL task, which was found to indicate a job completion rate of 99.995% based on the execution of 1000 tests. This metric, as well as the wider results, provide a system performance understanding which should allow a system integrator to understand compliance to the low DAL task requirements.

Even though the certification strategy, argument and evidence, has been considered in the context of DO-178C by the authors and internal safety experts, as part of our future work the approach is to be discussed with appropriate certification authorities which may lead to changes in the approach.

7 ACKNOWLEDGEMENTS

We would like to acknowledge the invaluable feedback and input to this paper provided by safety experts within Rolls-Royce and NASA. This work was funded by the INNOVATE UK funded Automated Timing framework of Interference in Critical Systems (ATICS) project (KTP010577), and by the HICLASS project, funded by the Aerospace Technology Institute and Innovate UK, as project 113213.

[3]A task's robustness is independent from its criticality

REFERENCES

[1] 2014. *Impact case study: COM04 The Goal Structuring Notation (GSN).* Impact Case study 43445. Research Excellence Framework (REF). https://impact.ref.ac.uk/casestudies/CaseStudy.aspx?Id=43445

[2] Sanjoy K Baruah, Alan Burns, and Robert I Davis. 2011. Response-time analysis for mixed criticality systems. In *32nd IEEE International Real-Time Systems Symposium, (RTSS).* IEEE.

[3] Iain Bate, Alan Burns, and Robert I Davis. 2015. A bailout protocol for mixed criticality systems. *IEEE Transactions on Software Engineering* (2015).

[4] Guillem Bernat, Alan Burns, and Albert Liamosi. 2001. Weakly hard real-time systems. *IEEE Trans. Comput.* 50, 4 (April 2001), 308–321. https://doi.org/10.1109/12.919277

[5] Alan Burns and Sanjoy Baruah. 2013. Towards a more practical model for mixed criticality systems. In *Workshop on Mixed-Criticality Systems (colocated with RTSS).*

[6] Alan Burns and Robert I. Davis. 2017. A Survey of Research into Mixed Criticality Systems. *ACM Comput. Surv.* 50, 6, Article 82 (Nov. 2017), 37 pages. https://doi.org/10.1145/3131347

[7] Alan Burns, Robert Ian Davis, Sanjoy Baruah, and Iain John Bate. 2018. Robust Mixed-Criticality Systems. *IEEE Trans. Comput.* (2018).

[8] Giorgio C. Buttazzo, Giuseppe Lipari, Marco Caccamo, and Luca Abeni. 2002. Elastic scheduling for flexible workload management. *IEEE Trans. Comput.* 51, 3 (March 2002), 289–302. https://doi.org/10.1109/12.990127

[9] Oliver Gettings, Sophie Quinton, and Robert I. Davis. 2015. Mixed Criticality Systems with Weakly-hard Constraints. In *Proceedings of the 23rd International Conference on Real Time and Networks Systems (RTNS '15).* ACM, New York, NY, USA, 237–246. https://doi.org/10.1145/2834848.2834850

[10] Patrick Graydon and Iain Bate. 2013. Safety Assurance Driven Problem Formulation for Mixed-Criticality Scheduling. In *Proceedings of the Workshop on Mixed-Criticality Systems.* 19–24.

[11] Moncef Hamdaoui and Parameswaran Ramanathan. 1995. A dynamic priority assignment technique for streams with (m, k)-firm deadlines. *IEEE Trans. Comput.* 44, 12 (Dec 1995), 1443–1451. https://doi.org/10.1109/12.477249

[12] Zain A. H. Hammadeh, Rolf Ernst, Sophie Quinton, Rafik Henia, and Laurent Rioux. 2017. Bounding deadline misses in weakly-hard real-time systems with task dependencies. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017.* 584–589. https://doi.org/10.23919/DATE.2017.7927054

[13] Mathieu Jan, Lilia Zaourar, and Maurice Pitel. 2013. Maximizing the execution rate of low criticality tasks in mixed criticality system. *Proc. 1st WMC, RTSS* (2013), 43–48.

[14] Timothy Patrick Kelly. 1999. *Arguing safety: a systematic approach to managing safety cases.* Ph.D. Dissertation. University of York York.

[15] Stephen Law. To Appear. *Advancing Mixed Criticality Scheduling Techniques to Support Industrial Applications.* Ph.D. Dissertation. University of York York.

[16] Stephen Law and Iain Bate. 2016. Achieving appropriate test coverage for reliable measurement-based timing analysis. In *28th Euromicro Conference on Real-Time Systems (ECRTS).* IEEE.

[17] Stephen Andrew Law, Iain John Bate, and Benjamin Lesage. 2019. Industrial Application of a Partitioning Scheduler to Support Mixed Criticality Systems. In *Proceedings of the 31st Euromicro Conference on Real-Time Systems (ECRTS).*

[18] Benjamin Lesage, Stephen Law, and Iain Bate. 2018. TACO: An Industrial Case Study of Test Automation for COverage. In *Proceedings of the 26th International Conference on Real-Time Networks and Systems (RTNS '18).* ACM, New York, NY, USA, 114–124. https://doi.org/10.1145/3273905.3273910

[19] Di Liu, Jelena Spasic, Nan Guan, Gang Chen, Songran Liu, Todor Stefanov, and Wang Yi. 2016. EDF-VD Scheduling of Mixed-Criticality Systems with Degraded Quality Guarantees. In *2016 IEEE Real-Time Systems Symposium (RTSS).* 35–46. https://doi.org/10.1109/RTSS.2016.013

[20] Roberto Medina, Etienne Borde, and Laurent Pautet. 2018. Availability enhancement and analysis for mixed-criticality systems on multi-core. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE).* 1271–1276. https://doi.org/10.23919/DATE.2018.8342210

[21] RTCA. 2011. DO-178C - Software Considerations in Airborne Systems and Equipment Certification. (2011).

[22] SAE. 2010. ARP4754A - Guidelines for development of civil aircraft and systems. *SAE International* (2010).

[23] Steve Vestal. 2007. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *28th IEEE International Real-Time Systems Symposium, (RTSS).* IEEE.