Evaluating Mixed Criticality Scheduling Algorithms with Realistic Workloads

David Griffin[1] and Iain Bate[1,2] and Benjamin Lesage[1] and Frank Soboczenski[1]
[1]Department of Computer Science
University of York
York, United Kingdom
[2]School of Innovation, Design, and Engineering
Mälardalen University
Västerås, Sweden
Email: {david.griffin, iain.bate, benjamin.lesage, frank.soboczenski}@york.ac.uk

***Abstract*—Most work on mixed-criticality scheduling has considered timing-related failures to be independent of one another. In reality this is not true as in many systems the state that caused the original failure will be similar to the state in the next release (job) of the task. Therefore when arguing about the number of jobs that do not meet their deadlines, it is crucial tasks have an appropriate fault model incorporated into the tool framework (i.e. task set generators and simulators) used to evaluate scheduling policies. The second issue that affects the tool framework is the choice of Worst-Case Execution Times (WCET) for different criticality modes of tasks. In the current literature it has been argued that a WCET should be chosen which would only be exceeded incredibly rarely, e.g. 1 in 10^{16} jobs. This leads to WCET values much greater than the High WaterMark (HWM). The needs of certification and the consideration of how safety is argued leads to the conclusion that the probability of a job not meeting its deadline can be much greater. This would greatly impact the WCETs and hence the results of the evaluation. The contributions of this paper are thus a more realistic tool framework, and hence more realistic results than those previously reported, which we claim gives a better insight into how the scheduling policies would behave in practice and hence better evidence for any safety case.**

I. Introduction

Mixed-Criticality Scheduling (MCS) is basically where tasks of different criticality levels share processing resources. There are many motivations for MCS including the ability to efficiently use resources whilst giving appropriate timing guarantees and reducing the cost of certification by allowing software sharing resources to be developed and certified by different processes [1]. The basic model of MCS consists of a system with N_m modes of operation.

In each mode, M, different tasks are executed and each task (denoted i) may have a different execution time per mode $C_{i,M}$. The task itself may not change between modes, however level of confidence desired in the Worst-Case Execution Time (WCET) may differ between modes. To date most of the work has been giving definitive knowledge of the schedulability of tasks in each mode of operation. This knowledge is very useful for the safety case that is produced as part of the certification of systems [2], [3], [4], however the safety case is also interested in the loss of service when a failure occurs. More specifically, if a High-Criticality Task (HCT) exceeds the expected WCET in a particular mode then a mode change will occur which means some Lower-Criticality Tasks (LCT) are not executed for a period of time. It is important to know how often this happens and for how long the LCTs are not executed.

In [5] a simulator framework and scenario-based evaluation was used to provide information about how many LCTs would miss their deadlines in "situations of interest". The results of the evaluation gave some interesting insights including showing that a newly proposed scheduling policy, the Bailout Model (BM), caused LCTs missing their deadlines less frequently than previous state of the art algorithms. The insight gained though is only useful if the situations of interest reflect reality. The simulator framework had two main components: a Task Set Generator (TSG) that used UUnifast [6] to generate task set profiles according to specified characteristics (e.g. target utilisation) and the simulator itself. However, two issues make the presented framework unrealistic:

1) *Phasing of Failures* – Most work, including in [5], assumes that failures are independent. In reality this is not true. For example in a feedback-based control system, each job execution reuses much of the state from the previous one and the inputs to successive jobs are likely to be similar. Hence if a job exceeds the WCET in the current mode the next few jobs are also likely to exceed this value.
2) *Probability Distribution for Execution Times* – Real software doesn't conform to the "standard" distributions of execution times used in UUnifast and are more likely to be some form of extreme-value distribution [7].

In [8], Griffin used observations from real software on actual platforms to learn a model of timing-related failures. Specifically the model contained information about the likely number of successive failures and the magnitudes of these successive failures. Using this model, the DepET algorithm was proposed which was able to generate task sets with appropriate failure characteristics.

The first contribution of this paper is to use the models created as part of DepET to create DepET-RND that can generate and simulate individual jobs with dependent failures. The second contribution of this paper is to assess how generating jobs with dependent failures may affect our previously presented simulation results, in [5],where different Mixed-Criticality Scheduling (MCS) policies were evaluated with jobs that only had independent failures. Given appropriate execution time profiles, realistic results still depends on an appropriate choice of the initial probability of failure. The initial probability of failure F_i is defined as the likelihood the execution of a job exceeds its WCET for the

current mode when previous jobs have not exceeded this WCET, or in other words the likelihood a sequence of dependent failures will commence. Many works, e.g. [9], on probabilistic WCET (pWCET) have claimed that values of F_i could typically be 10^{-16}. The stated reason is that some certification standards require the likelihood of hazardous events to be once in every 10^9 operating hours and that a typical task may execute over 10^6 times an hour. The final contribution of this paper is to explain why this type of value is completely inappropriate and to propose more realistic values that can be used to guide pWCET analysis, configure DepET and evaluate MCS.

The structure of the paper is as follows. Section II presents a more realistic model of exceedance probability. Section III introduces DepET in more detail. An evaluation is presented in section IV before finally the conclusions are given in section V.

II. Choice of Exceedance Probability

As stated earlier, an important decision is to choose an exceedance probability that when combined with other sources of uncertainty[1] is suitable for the integrity level of the task such that the task's deadline is missed with an acceptable pattern. More specifically for each task and for each mode, an exceedance probability and associated WCET is needed. For example, consider a system that has the following two modes.

1) *normal mode* – All tasks are executed and the schedulability analysis is performed with a WCET referred to as C_{LO}.
2) *high-criticality mode* – This mode is triggered when any HCT exceeds its C_{LO} and where just the HCTs are executed and the schedulability analysis is performed with a WCET referred to as C_{HI}.

Given a distribution of execution times, obtaining C_{LO} and C_{HI} corresponding to a given P_{LO} and P_{HI} is straightforward, but choosing those target probabilities is not. As an example in Figure 1, assuming values P_{HI} and P_{LO} of 10^{-4} and 10^{-6}, C_{LO} and C_{HI} amount respectively to 2300 and 2800 cycles. However, knowing the effect of the choice of P_{HI} and P_{LO} on the system is not trivial; The value P_{LO} should be chosen such that the Low Criticality Tasks (LCTs) receive sufficient service, which must take into account how often these tasks are stopped from executing as well as how long the system can cope with them not being available. Schedulability analysis to date does not provide useful information for either of these. The scenario-based assessment in [5] provides a partial answer but more work is needed. Specifically statistical analysis of the results in [5] is needed to provide the information at the necessary level of confidence. It is noted the smaller the value of P_{HI}, the more C_{HI} will exceed the true WCET. The degree of this pessimism is defined as how much greater the WCET is than the actual WCET. The level of pessimism affects how much functionality can be put onto the resources whilst guaranteeing the timing requirements are met. To date most work on MCS has assumed P_{HI} is zero, however in [3] it is explained how safety-critical systems are designed to tolerate this. The rest of this section concentrates on how based on standard safety analysis the values of P_{LO} and P_{HI} could be chosen and what appropriate values for these might be.

[1]It is noted the other sources of uncertainty mean the exceedance probability used in pWCET analysis does not directly relate to the probability the WCET is exceeded. Other sources of uncertainty include having incomplete observations of execution time that feed into the pWCET analysis.

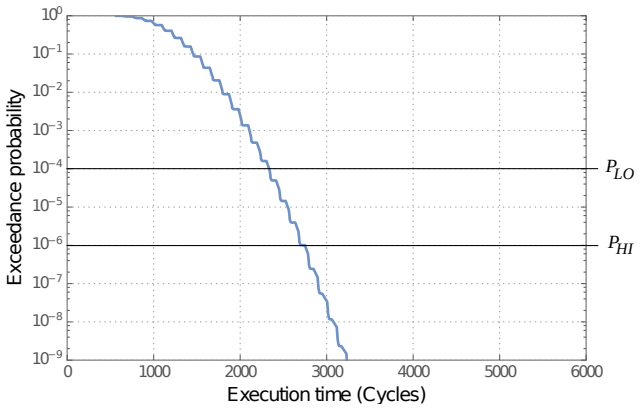

Fig. 1: Example illustrating how to choose P_{LO} and P_{HI} from an EVT execution time distribution corresponding to a task

For the purposes of this paper's discussion, we are interested in the likelihood of the conditions needed for a hazardous event (i.e. an event that might lead to death or injury) occurring and not how long these conditions are maintained. The classical approach for understanding how a hazardous event could occur is fault tree analysis [10] and the resulting fault tree can be used to give an understanding of the associated probabilities. It is also noted in [10] that these probabilities should only ever be used as a guidance in the safety case and cases are highlighted where over reliance on these figures have lead to serious incidents. A simplified example of a fault tree is given in Figure 2. Fault tree analysis considers how a hazardous event occurs (e.g. *Engine stops working* at the top of the tree through the logical combination of basic events (e.g. *Task exceeds WCET* at the bottom of the tree).

The fault tree presented is a simplified one as in fact there would be many more events involved between the basic events and the hazardous events. For example, the fault tree does not show how for many examples a single deadline miss would not be a problem, i.e. it may be we would have to stop control signals to the engine and fuel system for a period of time before it would have to stop not least due to inertia. Despite the simplifications, it is considered sufficient to illustrate the following. Its worth noting that removing the simplifications would likely mean that the points below are even more influential.

1) No single point of failure leads to the hazardous event. Wherever feasible this should be avoided and where it cannot regulatory authorities demand extra levels of rigour.
2) If the target probability for a hazardous event E is X (such as *Function Late*, then there is little benefit to a contributing event, (such as *Tasks WCRT is exceeded*) being lower than X. This can be seen due to the following facts. The behaviour of the *AND* operator is

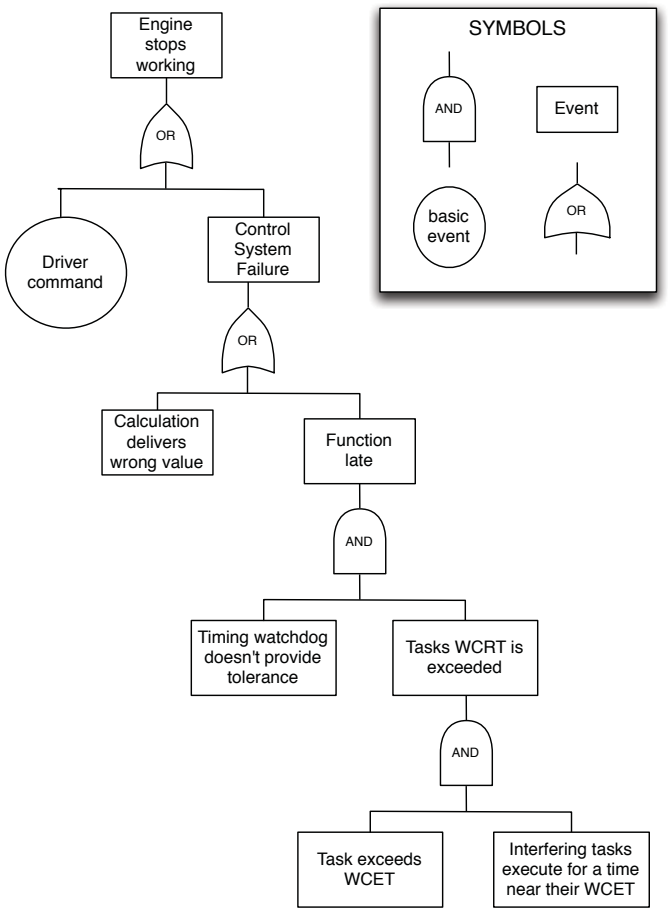

Fig. 2: Fault Tree for Car Engine Hazard - Engine Stops

such, then X provides a lower bound for the probability of events in the subtree [11]. Next, observe that when $lim_{X \to 0} X \ll 1$, *OR* gates tend to sum the probabilities of their input events [11]. As the probability of any failure due to WCET/WCRT in the fault tree is expected to be close to 0 (i.e. $\leq 10^{-4}$), then nX, where n is the number of input events to E, provides a lower bound to the probability of events in the subtree. As the number of input events n is expected to be relatively low (due to the hierarchical nature of a fault tree), and X is small, $nX \approx X$. Therefore it can be concluded that there is little or no benefit in the lower events in the tree having a smaller probability than the target probability of the hazardous event E. This leads to the corollary that as the highest target for the likelihood of a hazard in any certification standard is 10^{-9}, the likelihood of the hazardour event *Function Late* is upper bounded by 10^{-9}. Therefore, 10^{-9} provides a lower bound on the two causes of *Task exceeds WCET*, *Timing Watchdog does not provide tolerance* and *Tasks WCRT is exceeded*. Assuming that the failure rate of the timing watchdog is known, then due to the nature of *AND*, the probability of *Tasks WCRT is exceeded* can be significantly higher than 10^{-9}.

3) The probability of *Tasks WCRT is exceeded* would be low as it requires interfering tasks to execute for a time close or equal to their WCET as well as one task exceeding its WCET. This suggests *Task exceeds WCET* can be significantly greater than 10^{-9}.
4) There is little benefit in the probability of *Tasks WCRT is exceeded* being much less likely than *Calculation delivers wrong value* as the parent event *Function late* is combined with *Calculation delivers wrong value* by an *OR* gate, and hence the event *Control System Failure* has a probability at least as great as *Calculation delivers wrong value*. Therefore, *Calculation delivers wrong value* provides an effective lower bound to the values for *Function late* which significantly impact the probability of *Control System Failure*. Assuming that the software program is correct, then hardware failure is the only cause of *Calculation delivers wrong value*. In most safety-critical systems, the rate of hardware failure tends to be limited to a of 10^{-6} per hour of operation [12], and therefore for any hour of operation hardware has a probability of no greater than 10^{-6} of failure. Hence, this probability provides an effective lower bound on the *Calculation delivers wrong value*. Other factors that could increase this value include confidence on the functional testing of the task, but as the level of testing required is domain specific this is not considered in this paper. Hence there is little benefit to reducing the fault rate of *Function late* below that of *Calculation delivers wrong value*, 10^{-6}.

The second of these observations provides an upper limit to any failure probability in the fault tree. The third observation suggests that a new form of probabilistic analysis is necessary to determine the exceedance probability for the WCET given a desired exceedance probability for the WCRT. However, the final observation suggests that there is little benefit for P_{HI} being lower than 10^{-6} per hour, as any lower values would be negated by the 10^{-6} per hour rate of *Calculation delivers wrong value*, as this is an effective upper bound. To convert this failure rate to the failures per instance required when setting an exceedance probability for WCRT is difficult due to point 1; a single WCRT exceedance is not sufficient cause to failure of the system. In order to calculate the exact probability of failure it would be necessary to model the dependencies in execution times and find the probability of a sufficient number of faults (at both the execution and response time levels) occurring. It is clear that the value of P_{HI} should be quite a bit less than 10^{-16}. Using such a figure would be overly pessimistic and bring more uncertainty into the analysis as its harder to fit a curve at that type of probability level especially as it entails a significant extrapolation from the observations fed into the analysis [13]. Given an appropriate value of P_{HI}, then P_{LO} for each HCT should be chosen such that the LCT get sufficient service to support their associated hazardous events in the safety case.

In summary, this discussion shows that techniques are needed that help System Safety Engineers to convert the system-level reliability and availability targets into requirements on the computer system. The real-time systems engineers would then need a *translation method* to take these

lower-level reliability and availability requirements for tasks meeting their deadline and convert these into requirements for the pWCET community. We would suggest the real-time requirements for both deadlines and WCETs are split into the likelihood of the initial failure and the maximal duration of the failures once they have first occurred. The following section summarises a method for modelling dependent failures and then using these models when simulating task sets executing. This work could form the basis for any translation method.

III. Model of Dependent Failures

As illustrated in previous sections, limited research has been carried out as to how MCS algorithms behave when presented with realistic workloads. In particular, the dependencies between deadline overruns can create transitionary periods of high load which would not be present in randomised experiments. Recent work by Griffin et al. [8] defined the DepET algorithm, which is capable of modelling the dependencies of job execution times by utilising exceedance models; after operation, DepET returns a set of dependent execution times for the tasks it is asked to generate. In order to accomplish this, DepET defines each task as having a number of execution time $bands$ with the following properties that the user can configure in addition to the DepET algorithm internal properties $duration$, $prev$, $next$ and cET:

- mn, mx: The minimum and maximum values within this band
- d: The maximum value that an execution time may be displaced from its previous instance
- p: The probability of leaving the band
- EM: An exceedance model, used to determine the duration of the higher band

However, as previously employed, DepET is only capable of utilising an existing failure model and therefore is not usable for randomised testing. For reference, a pseudo-code implementation of DepET is given in Algorithm 1. Full details can be found in [8].

Therefore an extension to DepET is proposed, DepET-RND. DepET-RND utilises simple randomised exceedance models to control exceedance duration. The random exceedance models proposed simply selects, at random, the duration of a fault from a pre-specified randomly selected list. As DepET exposes a large number of variables, which may make targeting specific failure rates difficult, DepET-RND, produces a randomised configuration and then samples the values from that configuration. If the values are not sufficiently close to the target failure rate, the configuration is rejected and the process repeated. To hasten the search, user knowledge can provide a range of values for each parameter which are likely to produce configurations close to the desired failure rate. A pseudo-code implementation of the algorithm is given in Algorithm 2.

IV. Evaluation

In order to evaluate the effects of dependent failures with regard to the effectiveness of mixed criticality scheduling algorithms, experiments were carried out using the

```
Function DepET(tasks)
    ETs ← []
    for task ∈ tasks do
        band ← task.current
        add randomnormal() * band.d to band.cET
        ETs.append(band.cET)
        clamp band.cET within band.mn and band.mx
        if band.duration = 0 then
            task.current ← band.prev
        end
        else if random() < band.p then
            task.current ← band.next
            band.next.duration ← band.EM.sample()
        end
        while band is not None do
            decrement band.duration
            band ← band.prev
        end
    end
    return ETs
end
```

Algorithm 1: The *DepET* algorithm

```
Function DepET-RND(number_of_tasks,
number_of_bands, target_failure_rate)
    tasks ← []
    for n ∈ range(number_of_tasks) do
        repeat
            random_ems ← a list of number_of_bands
            randomised exceedance models
            task ← a DepET task with random
            parameters and exceedance models
            random_ems
        until Failure rate of DepET([tasks])
        ≈ target_failure_rate;
        append task to tasks
    end
    return DepET(tasks)
end
```

Algorithm 2: The *DepET-RND* algorithm

simulation framework used by Bate et al. [5]. This simulator was extended to implement the DepET-RND algorithm. The algorithm was set up in a similar manner to Bate et al., with a simulation duration of 10^{11} time units of 0.1ms. Tasks were defined using UUniFast [6] targeting 90% maximum utilisation in low criticality mode, with tasks having harmonic periods chosen randomly from the base frequencies of 20, 40, 80, 200, 400, 800ms, as commonly found in automotive systems [14]. Deadlines were implicit. Given the number of time units simulated, the duration of the simulation was sufficient to simulate 10^5 instances of the longest task. Tasks were chosen at random to be either low or high criticality, with 50% of tasks being high criticality. As low criticality mode targeted 90% worst case utilisation, once high criticality mode is taken into account, many of the task sets exceeded 100% maximum utilisation. However, a benefit of mixed criticality algorithms is that these systems are still acceptable. With regard to generating the utilisations of each job, three configurations were tested:

1) A control simulation with an independent failure rate of 0.1%, without DepET-RND.
2) A simulation using DepET-RND with dependent failures with overall (average) failure rate of 0.1% and a

maximum number of consecutive failures 200.

3) A simulation using DepET-RND with dependent failures with initial failure rate of 0.1% and a maximum number of consecutive failures 200.

Three different state of the art mixed criticality scheduling algorithms are compared under each configuration:

1) FPPS: Fixed Priority Pre-emptive Scheduling
2) AMC+: An extended version of Adaptive Mixed Criticality scheme [15] proposed in [5] where the execution of LCTs resumes following an idle instant.
3) BM: The Bailout Mode algorithm [5] which uses the slack associated with individual high-criticality jobs to determine when it is okay to return to normal mode whilst preserving the schedulability of all necessary tasks.

We also consider enhancements to both the AMC+ and BM approaches. AMC+S and BMS respectively make use of offline computed slack to increase the budgeted C_{LO} values for individual tasks. AMC+SG and BMSG extend AMC+S and BMS by also using gain time (on-line computed slack) to increase the budgeted C_{LO} values for individual jobs. These protocols are defined in more detail in [5].

Figures 3, 4 and 5 each present, under a given configuration, the percentage of tasks not scheduled by the MCS algorithms evaluated. To capture the variance of this percentage across all the performed simulations, the plots presents for each algorithm the median, quartiles, 9^{th} and 91^{st} percentiles of the dataset.

The differences between Figures 3 and 4, assuming independent then dependent failures with the same overall failure rate, clearly shows that all the algorithms considered perform substantially better on dependent failures than independent failures. Assuming independent failures, a fair portion of simulations resulted in at least 1% to 4% of tasks not being executed (denoted by the red median in the boxes). The worst performing algorithm under dependent failures (AMC+ in Figure 4) still resulted in less than 0.3% of not-executed tasks in the majority of simulations (90% as captured by the top whisker of the plot). This is to be expected as the overheads of entering high criticality mode due to a deadline failure in all the considered algorithms are high, and by introducing dependencies the deadline failures are clustered, resulting in fewer criticality mode transitions. In addition, the relative performance of the scheduling algorithms remains the same as was observed in the independent case.

Figure 5 illustrates the performance of the algorithms when the initial failure rate is 0.1%. Under that configuration, the different algorithms still perform better than with independent failures at the same rate (as shown in Figure 3). As an example when 3% to 4% of the tasks fail to execute for most simulations (denoted by the blue quartile box) using BM and independent failures, this figure falls around 0.5% to 1% under dependent failure. This is despite the fact that due to the initial failures being 0.1%, the total number of failures observed in Figure 5 is greater than in Figure 3. However, this can be explained as follows: due to the clustering effect of DepET, the chances of observing two

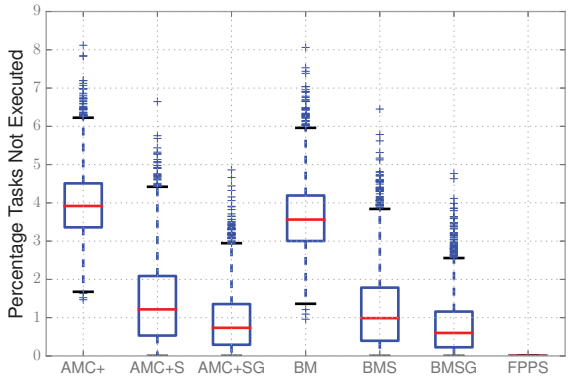

Fig. 3: Percentage of tasks not scheduled, independent failure rate 0.1%

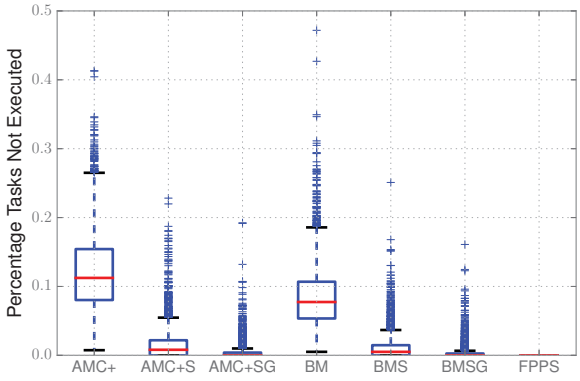

Fig. 4: Percentage of tasks not scheduled, dependent overall failure rate 0.1%

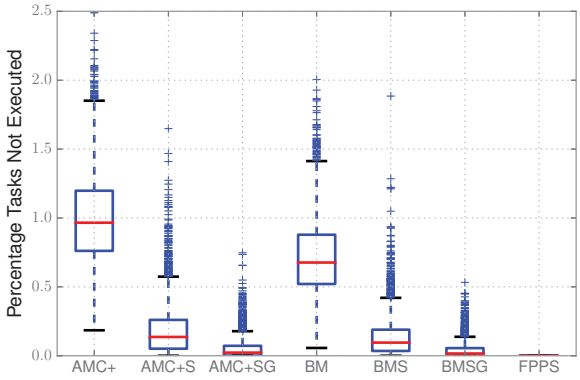

Fig. 5: Percentage of tasks not scheduled, dependent initial failure rate 0.1%

or more faults simultaneously is increased. In turn, this decreases the number of criticality mode transitions of the algorithm, and therefore decreases the overheads allowing more tasks to be scheduled.

Figures 6 and 7 examine the characteristics of two of the algorithms considered, respectively AMC+SG and BMSG, specifically examining how the maximum duration of a fault

observed effects the number of tasks scheduled. While both algorithms exhibit a spike in the percentage of tasks not executed at approximately 200 (the maximum duration of a single fault), the spike resulting from the AMC+SG approach (in Figure 6) is more defined than that seen in the BMSG approach (in Figure 7). Preliminary analysis of this effect suggests that it is caused by the harmonic periods meaning the frequency of idle periods have a regular pattern. For systems of a high utilisation these can be quite small and when C_{LO} is exceeded some of these idle periods can disappear. For the AMC-based algorithms, this could lead to longer times before a return to normal mode whereas with the BM-based policies the normal mode can be returned to any time.

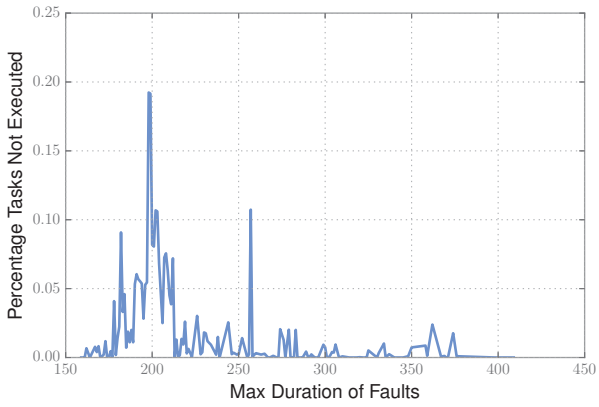

Fig. 6: Percentage of tasks not scheduled vs Max Duration of Faults, using AMC+SG

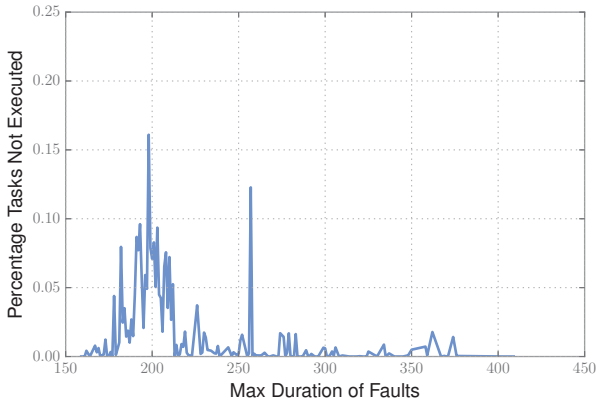

Fig. 7: Percentage of tasks not scheduled vs Max Duration of Faults, using BMSG

V. Conclusions

In this paper, three contributions outlined in the introduction have been made. Firstly an explanation has been provided as to why currently proposed exceedance probabilities for pWCET may be excessively small and different values have been proposed. Secondly, a simulation framework for MCS has been adapted to use the results of a more realistic fault model that has been previously learned using observations from "real" systems. Finally, a scenario-based evaluation of different scheduling policies have been performed. The evaluation has shown that having a dependent fault model does not affect the trends previously seen between different scheduling policies, i.e. the improvement one policy gives over another is approximately the same, however it does affect the sizes of the loss of service to LCTS.

Further, this paper has presented an argument that WCET exceedance probabilities seen in literature on probabilistic real-time systems are unrealistically low, given other components in the system and their interactions in the causes of failures. As minimising the amount of extrapolation required in pWCET from the observed data reduces the inaccuracies, and hence the uncertainty, resulting in tighter and more useful results.

Acknowledgments

This work was funded by the EPSRC grant, MCC (EP/K011626/1), the EU FP7 IP PROXIMA (611085), and the Swedish Foundation for Strategic Research (SSF) SYNOPSIS Project. EPSRC Research Data Management: No new primary data was created during this study.

References

[1] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Proceedings of the 28th Real-Time Systems Symposium*, 2007, pp. 239–243.

[2] I. Bate, P. Conmy, T. Kelly, and J. McDermid, "Use of modern processors in safety-critical applications," *The Computer Journal*, vol. 44, no. 6, pp. 531–543, 2001.

[3] P. Graydon and I. Bate, "Realistic safety cases for the timing of systems," *The Computer Journal*, vol. 57, no. 5, pp. 759–774, 2014.

[4] ——, "Safety assurance driven problem formulation for mixed-criticality scheduling," in *Proceedings of the 1st International Workshop on Mixed Criticality Systems*, 2013, pp. 19–24.

[5] I. Bate, A. Burns, and R. Davis, "A bailout protocol for mixed criticality systems," in *Proceedings of the 27th Euromicro Conference on Real-Time Systems*, 2015, pp. 259–268.

[6] E. Bini and G. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems Journal*, vol. 30, pp. 129–154, 2005.

[7] A. Burns and S. Edgar, "Predicting computation time for advanced processor architectures," in *Proceedings of the 12th EUROMICRO Conference on Real-Time Systems*, 2000, pp. 89–96.

[8] D. Griffin, B. Lesage, I. Bate, F. Soboczenski, and R. Davis, "Modelling fault dependencies when execution time budgets are exceeded," in *Proceedings of the 23rd International Conference on Real-Time Networks and Systems*, 2015.

[9] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quinones, and F. Cazorla, "Measurement-based probabilistic timing analysis for multi-path programs," in *Proceedings of the Euromicro Conference on Real-Time Systems*, 2012, pp. 91–101.

[10] N. Leveson, *Safeware: System Safety and Computers*. Addison Wesley, 1995.

[11] N. Roberts, W. Vesely, D. Haasl, and F. Goldberg, "Fault tree handbook nureg-0492," *US Nuclear Regulatory Commission*, 1981.

[12] AMSC, "Military handbook, mil-hdbk-338b," *US Department of Defense*, vol. 1, 1998.

[13] D. Maxim, F. Soboczenski, I. Bate, and E. Tovar, "Study of the reliability of statistical timing analysis for real-time systems," in *Proceedings of the 23rd International Conference on Real-Time Networks and Systems*, 2015.

[14] I. Bate and A. Burns, "An integrated approach to scheduling in safety-critical embedded control systems," *Real-Time Systems Journal*, vol. 25, no. 1, pp. 5–37, Jul 2003.

[15] S. Baruah, A. Burns, and R. Davis, "Response-time analysis for mixed criticality systems," in *Proceedings of the 32nd Real-Time Systems Symposium*, 2011, pp. 34–43.