Extending Mixed Criticality Scheduling

Tom Fleming
Department of Computer Science,
University of York, UK.
Email: tdf506@york.ac.uk

Alan Burns
Department of Computer Science,
University of York, UK.
Email: alan.burns@york.ac.uk

***Abstract*—As Mixed Criticality work has progressed it has become increasingly clear that considering only 2 levels of criticality will not suffice. Many industrial standards such as IEC 61508 and DO-178B define 4 or 5 levels, whereas the majority of current analytical approaches consider just 2. In this work we evaluate the performance of several fixed priority approaches and how they might be extended to facilitate more than 2 criticality levels. A well-established scheme, Period Transformation is also considered and extended. The effectiveness of the extensions is assessed by way of an evaluation. We show that the schemes maintain their performance relative to each other as the number of criticality levels increases.**

I. Introduction

The field of Mixed Criticality systems is advancing rapidly. Driven by industrial pressure to support new and complex hardware, attention is turning away from the more simplistic uni-processor case. However, in order to further the case for implementation, such systems must be certified and standards must be met. When considering the number of criticality levels that a system might be required to manage it becomes clear that just 2 levels will not suffice. Standards such as IEC 61508 and DO-178B define 4 or 5 Safety Integrity Levels (SIL). It is reasonable to assume that mixed criticality implementations will be required to support at least 5 levels.

Before consideration can be given to an increased number of criticality levels on complex hardware, it seems logical to begin with a single processor approach. Much of the work since Vestal's [4] seminal paper in 2007 has revolved around 2 levels of criticality. Approaches such as AMC (Adaptive Mixed Criticality), presented by Baruah et al. [2] have focused on 2 criticality levels while claiming extendability to many. We focus on single processor analysis and extend both approaches suggested in [2], AMCrtb (Response Time Bound) & AMCmax, to facilitate n potential levels. Additionally we consider Period Transformation [3] as proposed by Vestal [4]. We provide some improvements on the initial analysis and compare its performance with other approaches.

The remainder of the document is organised as follows; Section II considers the original Dual Criticality approaches, Section III considers how such approaches can be extended to include a greater number of criticality levels, Section IV provides an evaluation and Section V closes the work with some concluding remarks.

II. Dual Criticality Analysis

Baruah et al. [2] present a dual-criticality scheme known as Adaptive Mixed Criticality (AMC). AMC monitors, at runtime, the execution of each task and ensures that it remains within its budget for the current criticality level. If a job exceeds its allocated budget, a criticality change is triggered, AMC permanently suspends all tasks at the current criticality level when a change occurs. Two methods are presented in [2]:

A. AMCrtb

The original analysis presented by Baruah et al. [2] is shown in Equations (1), (2) and (3), for a sporadic task model using standard notation (C, T, D, R with $D \leq T$), LO-crit & HI-crit. There are two stages to the approach, the first is to consider the LO and HI criticality levels individually and ensure they are schedulable. The second stage is to consider the schedulability of the criticality change from LO to HI.

Stage 1A: *Check the schedulability of the LO mode for all tasks.*

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

Stage 1B: *Check the schedulability of the HI mode for HI tasks.*

$$R_i(HI) = C_i(HI) + \sum_{j \in hpH(i)} \left\lceil \frac{R_i(HI)}{T_j} \right\rceil C_j(HI) \quad (2)$$

Where hpH is the set of all higher priority HI criticality tasks.

The next step is to assess the schedulability of any HI criticality task executing during a criticality level change.

Stage 2A: *Calculate the schedulability of the criticality change for HI tasks.*

$$R_i^*(HI) = C_i(HI) + \sum_{j \in hpH(i)} \left\lceil \frac{R_i^*(HI)}{T_j} \right\rceil C_j(HI) + \sum_{k \in hpL(i)} \left\lceil \frac{R_i(LO)}{T_k} \right\rceil C_k(LO) \quad (3)$$

Where hpL is the set of all higher priority LO criticality tasks. $R_i(LO)$ represents a static value for higher priority but lower criticality tasks, this allows AMC to place an upper bound upon any potential low-criticality interference during a criticality change. $R_i^*(HI)$ is the value replaced into the equation with each iteration. This is appropriate due to the way in which AMC handles a criticality level change. Under

AMC, all LO criticality tasks are suspended when a criticality change occurs, as such during this time their ability to interfere with the high-criticality tasks is limited. This limit is the LO response time of the HI-crit task as after that time the system will be running in the HI criticality mode, or the task will have completed and no criticality change need occur.

B. AMCmax

There are a finite number of points in time at which a criticality change might take place. It is possible to bound these points as the criticality change must occur sometime between the start of execution, time 0 and the LO response time ($R_i(LO)$). AMCmax uses these points and seeks to determine the point at which the worst-case phasing for a HI criticality task might occur.

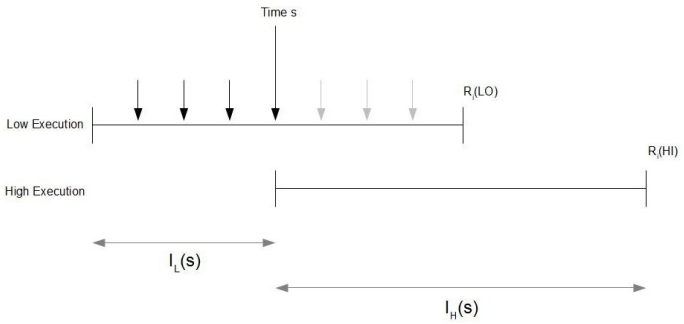

Fig. 1. Example AMCmax criticality change.

Figure 1 shows a criticality change occurring and the system moving into the HI mode. The diagram also shows the Interference suffered in both the LO and HI modes. Baruah et al. [2] illustrate this change with Equation (4), showing the calculations required to determine the response time of a HI criticality task if the change occurs at time s.

$$R_i^s(HI) = C_i(HI) + I_L(s) + I_H(s) \tag{4}$$

From Equation (4) it is easy to see the two segments of interference we must assess, I_L and I_H. These sections are also shown in Figure 1.

The technique used to assess the response time of low-criticality tasks is straightforward, it can be seen in Equation (5):

$$I_L(s) = \sum_{j \in hpL(i)} \left(\left\lfloor \frac{s}{T_j} \right\rfloor + 1 \right) C_j(LO) \tag{5}$$

The floor function is used to ensure that all tasks are accounted for immediately upon release. This algorithm is also used when calculating the LO response times of all tasks, in this case $R_i^s(LO)$ is used rather than s.

Baruah et al. [2] consider high-criticality response times $I_H(s)$. They consider the high mode as an interval of $t - s$ where $t > s$. t is the response time of the task and is the value that is replaced in each iteration. The number of releases in this interval, $t - s$, for a HI criticality task τ_k, can be calculated:

$$\left\lceil \frac{t-s}{T_k} \right\rceil + 1$$

This can be extended for cases where $D_k < T_k$:

$$\left\lceil \frac{t - s - (T_k - D_k)}{T_k} \right\rceil + 1$$

The full calculation is shown in Equation (6) presented in the form of a function M. With input parameters k, s and t, where k is the task, s is time s and t is time t (or the response time replaced into the equation).

$$M(k,s,t) = min\left\{ \left\lceil \frac{t - s - (T_k - D_k)}{T_k} \right\rceil + 1, \left\lceil \frac{t}{T_k} \right\rceil \right\} \tag{6}$$

Equation (6), accounts for all completions of task τ_k, within the interval $s \ldots R_i(HI)$. Rare cases are possible where the calculation is overly pessimistic, the function ensures that the value returned is no greater than the total number of releases.

The number of releases in the LO criticality mode is easily calculable by removing the results of Equation (6) from the total number of releases.

$$\left(\left\lceil \frac{t}{T_k} \right\rceil - M(k,s,t) \right) C_k(LO)$$

Therefore $I_H(s)$ is:

$$\begin{aligned} I_H(s) = \sum_{k \in hpH(i)} \Bigg\{ & (M(k,s,t) C_k(HI)) + \\ & \left(\left(\left\lceil \frac{t}{T_k} \right\rceil - M(k,s,t) \right) C_k(LO) \right) \Bigg\} \end{aligned} \tag{7}$$

And thus the full equation:

$$\begin{aligned} R_i^s = & \sum_{j \in hpL(i)} \left(\left\lfloor \frac{s}{T_j} \right\rfloor + 1 \right) C_j(LO) + \\ & \sum_{k \in hpH(i)} \Bigg\{ (M(k,s,R_i^s) C_k(HI)) + \\ & \left(\left(\left\lceil \frac{R_i^s}{T_k} \right\rceil - M(k,s,R_i^s) \right) C_k(LO) \right) \Bigg\} \end{aligned} \tag{8}$$

And:

$$R_i = max(R_i^s) \forall s$$

Finally they look at which points of s, within $0 \ldots R_i(LO)$ require consideration. Baruah et al. [2] note that the amount of low-criticality interference increases (as a step function), as the value of time s increases. Similarly the high-criticality interference decreases as the low increases. Therefore the response time changes only at the release of a low-criticality job, thus we can limit our search to points of s where a LO criticality job is released.

C. Period Transformation

Vestal [4] proposed a Mixed Criticality Period Transformation (PT) approach. He used PT, not to create a harmonic task set, but to allow for a Criticality and Rate Monotonic based priority ordering. The approach proposes that only those HI criticality tasks with periods greater than that of the shortest

LO criticality period be transformed. This will allow all HI criticality tasks to attain a higher priority than the LO and thus avoid the problem of criticality inversion and allow for criticality monotonic assignment.

This gives us 3 groups of tasks. Those of a LO criticality, these do not need transformation. Those with a HI criticality but a period shorter than the shortest LO criticality task, these do not need transformation. Finally those with a HI criticality with a period greater than that of the shortest LO criticality task, these are the tasks that must be transformed.

The analysis of HI criticality tasks, in the HI mode is calculated via standard response time techniques [1]. The analysis for the LO criticality mode is detailed as follows:

Tasks are transformed by a factor, m.

$$m = \left\lceil \frac{T_j}{T_l} \right\rceil$$

Where τ_l is the LO criticality task with the shortest period and τ_j is a HI criticality task that must be transformed.

At runtime, transformed tasks are expected to execute up to their $C_j(HI)/m$ until they reach their untransformed, $C_j(LO)$, only then can we determine if a task will overrun its LO execution bounds and a mode change would need to occur. Transformed tasks, running in the LO mode execute in $C_j(HI)/m$ time slices until $C_j(LO)$.

The number of transformed dispatches that might interfere with τ_i could be calculated as follows:

$$\left\lceil \frac{R_i}{T_j/m} \right\rceil$$

The number calculated above will contain several complete executions of $C_j(LO)$ and a remainder, this remainder can execute for no longer than $C_j(LO)$, Vestal assumes this value. He calculates the number of transformed executions which complete to $C_j(LO)$:

$$\left\lfloor \frac{R_i}{T_j} \right\rfloor C_j(LO)$$

So including the added pessimism of those transformed executions that do not complete, the total interference from τ_j can be summed as follows.

$$\left\lfloor \frac{R_i}{T_j} \right\rfloor C_j(LO) + C_j(LO)$$

Clearly there are several disadvantages to Vestal's technique. The use of $C_j(LO)$ to account for transformed releases which do not constitute an entire $C_j(LO)$ is overly pessimistic. Vestal's approach also loses one of the key properties of Period Transformation, the ability to create harmonic task sets and, by proxy, the ≤ 1 utilisation bound for RM schedulability. Although not all tasks are transformed it is likely that by transforming the HI tasks the number of context switches will increase significantly. This coupled with the need for additional run-time monitoring causes PT overheads to remain high.

III. Extend to n Criticality Levels

A. AMCrtb

When considering n possible criticality levels for AMCrtb we re-examine the two stages of the analysis. In stage one we examine each level, up to n, and determine schedulability. In stage two we consider the feasability of $n-1$ criticality level changes.

1) Stage One: Consider a system containing 5 distinct criticality levels, $L1 \dots L5$ where $L1 > L5$. The analysis for $L5$ must consider the potential interference of all higher priority tasks, regardless of criticality level (as $L5$ is the lowest level). To calculate the interference suffered from higher priority $L4$ tasks we use the following term:

$$\sum_{j \in hp(i) | L_j = L4} \left\lceil \frac{R_i(L5)}{T_j} \right\rceil C_j(L5)$$

The algorithm looks for those higher priority tasks, τ_j, where the criticality level (L_j) is equal to $L4$. This considers any interference suffered from a task at $L4$, but uses their $L5$ values. The calculation can be completed to account for levels $L3 \dots L1$ as shown in Equation (9).

$$\begin{aligned} R_i(L5) = C_i(L5) + & \sum_{j \in hp(i) | L_j = L4} \left\lceil \frac{R_i(L5)}{T_j} \right\rceil C_j(L5) \quad + \\ & \sum_{k \in hp(i) | L_k = L3} \left\lceil \frac{R_i(L5)}{T_k} \right\rceil C_k(L5) \quad + \\ & \sum_{l \in hp(i) | L_l = L2} \left\lceil \frac{R_i(L5)}{T_l} \right\rceil C_l(L5) \quad + \\ & \sum_{m \in hp(i) | L_m = L1} \left\lceil \frac{R_i(L5)}{T_m} \right\rceil C_m(L5) \end{aligned} \tag{9}$$

This process is repeated for each of the remaining criticality levels to check their schedulability. It is possible to generalise these equations to one that can deal with $2 \longrightarrow n$ criticality levels. We must consider the schedulability of n criticality levels individually.

For each criticality level.

$$\forall L \in 1 \dots n$$

For all tasks where the criticality level is greater than or equal to L.

$$\forall \tau_i | L_i \geq L$$

Calculate the response times for that level.

$$R_i(L) = C_i(L) + \sum_{j \in hp(i) | L_j \geq L} \left\lceil \frac{R_i(L)}{T_j} \right\rceil C_j(L) \tag{10}$$

Equation (10) considers the response time of task τ_i at criticality level L by accounting for any interference from higher priority tasks with a criticality level greater than or equal to L. This test is repeated for each of the n criticality modes. Equations (1) and (2) are the dual criticality application of Equation (10).

2) Stage Two: When assessing the interference suffered during a criticality change we must consider two groups of tasks. The first group are those tasks of a higher priority and with a criticality level greater than or equal to the task in question. This has been considered in Stage One. The second group are those tasks with a higher priority but a lower criticality level. It is clear that under AMC, those tasks with a higher priority but lower criticality will have a bounded effect on a higher criticality task if a criticality change occurs.

The interference suffered by higher criticality task, τ_i from higher priority but lower criticality task, τ_k during a criticality change is bounded by τ_i's response time at τ_k's criticality level, $R_i(L_k)$.

$$\sum_{k \in hp(i) | L_k < L_i} \left\lceil \frac{R_i(L_k)}{T_k} \right\rceil C_k(L_k)$$

For all tasks with a higher priority than τ_i where the criticality level is lower. $R_i(L_k)$ is the response time of τ_i at the criticality level of τ_k. These values are static bounds and do not change upon each iteration.

If we combine the analysis for the higher priority tasks with a criticality level greater than or equal to L_i and the analysis for the higher priority tasks with a criticality level less than L_i we can produce an algorithm to assess the feasibility of the criticality level changes in a system. In a system with n criticality levels we must consider $n-1$ criticality level changes.

For each criticality level.

$$\forall L \in 1 \ldots n$$

For all tasks where the criticality level is greater than or equal to L

$$\forall \tau_i | L_i \geq L$$

Beginning at the lowest criticality level, calculate the schedulability of each criticality change.

$$R_i^*(L) = C_i(L) + \sum_{j \in hp(i) | L_j \geq L} \left\lceil \frac{R_i^*(L)}{T_j} \right\rceil C_j(L) \quad + \sum_{k \in hp(i) | L_k < L_i} \left\lceil \frac{R_i(L_k)}{T_k} \right\rceil C_k(L_k) \tag{11}$$

Equation (11) assesses the schedulability of criticality changes for $2 \longrightarrow n$ criticality levels. This combined with the algorithm in Equation (10) provides an AMCrtb schedulability test generalised to n levels of criticality.

B. AMCmax

Consider the case of two criticality levels A and B, where A is the lowest criticality level in the system, (B > A). We would use the dual criticality analysis shown in Equation 8.

In order to determine the response time of a level B task, AMCmax considers points of s where the criticality change might occur. These points are bounded by the $R_i(A)$ response time of the task in question. If this system were also to include a criticality level C, such that $C > B > A$ then a criticality change might occur at any point (s_2) between the original change from A to B, point s_1, and the task's response time in criticality mode B, $R_i(B)$.

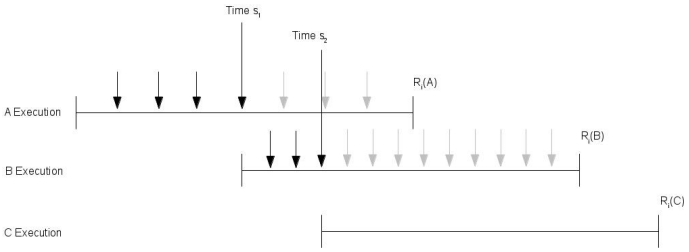

Fig. 2. The system with modes A and B with an additional level, C added.

To calculate the interference suffered between two points of s we define a new function, N:

$$N(k, s_1, s_2) = \left\lfloor \frac{s_2 - s_1 - (T_k - D_k)}{T_k} \right\rfloor + 1 \tag{12}$$

Function N provides the interference suffered between times s_1 and s_2. In this case, this function is used to assess the response time of tasks at criticality level B.

The calculation for the A criticality tasks is similar to the LO calculation, we use s_1 rather than s in order to differentiate between criticality changes.

$$\sum_{j \in hpA(i)} \left(\left\lfloor \frac{s_1}{T_j} \right\rfloor + 1 \right) C_j(A)$$

The calculation for the B criticality tasks changes to make use of the function N (see Equation (12)) as it is now used to determine the interference suffered between two points of s. The criticality A interference is calculated by removing the number of releases, as calculated by function N from the total number of releases.

$$\sum_{k \in hpB(i)} \left\{ N(k, s_1, s_2) C_k(B) + \left(\left\lceil \frac{s_2}{T_k} \right\rceil - N(k, s_1, s_2) \right) C_k(A) \right\}$$

Finally we may consider the calculation for criticality level C. Functions M and N are used in order to calculate the interference a criticality C task might suffer in modes C and B respectively. Both functions M and N are removed from the total number of releases to calculate the criticality A response time.

$$\sum_{l \in hpC} \left\{ M(l, s_2, R_i(C)) C_l(C) + N(l, s_1, s_2) C_l(B) + \left(\left\lceil \frac{R_i(C)}{T_l} \right\rceil - N(l, s_1, s_2) - M(l, s_2, R_i(C)) \right) C_i(A) \right\}$$

Where hpC refers to all higher priority tasks of criticality level C. The complete calculation for $\tau_i(C)$ is shown in Equation

(13).

$$R_i(C) = C_i(C) + \sum_{j \in hpA(i)} \left(\left\lfloor \frac{s_1}{T_j} \right\rfloor + 1 \right) C_j(A) + \sum_{k \in hpB(i)} \left\{ N(k, s_1, s_2) C_k(B) + \left(\left\lceil \frac{s_2}{T_k} \right\rceil - N(k, s_1, s_2) \right) C_k(A) \right\} + \sum_{l \in hpC} \left\{ M(l, s_2, R_i(C)) C_l(C) + N(l, s_1, s_2) C_l(B) + \left(\left\lceil \frac{R_i(C)}{T_l} \right\rceil - N(l, s_1, s_2) - M(l, s_2, R_i(C)) \right) C_i(A) \right\} \quad (13)$$

Where:

$$R_i(C) = max(R_i(C)) \forall s_n$$

Equation (13) shows how $R_i(C)$ can be calculated by considering points of s_1 where the change from A to B might occur and points of s_2 where the change from B to C might occur.

If we were to extend this system to introduce a 4th criticality level, D, we would follow the same steps as we did for criticality level C. Consider points for the criticality change from C to D at time s_3 bounded by the criticality C response time, $R_i(C)$. It is important to note that the function N is always used to calculate the number of releases between two points of s and the function M is always used to calculate the response time at the highest criticality level currently being considered.

The process of adding another set of points to check for each criticality level may be repeated to account for as many criticality levels as desired. However, the computational load increases almost exponentially as the number of criticality levels increases.

C. Period Transformation

In his analysis, Vestal [4] assumes a value of $C_j(LO)$ for the remaining transformed executions, $C_j(HI)/m$ that do not constitute a complete execution of $C_j(LO)$. This value, although an effective upper bound, is undesirably pessimistic. The work below considers a more accurate approach to finding the interference from transformed executions that do not constitute a complete untransformed execution.

We calculate the number of complete executions of $C_j(LO)$ that might interfere with τ_i:

$$\left\lfloor \frac{R_i}{T_j} \right\rfloor C_j(LO)$$

Rather than assuming the value of $C_j(LO)$ for all remaining transformed executions, we seek to determine the size of the incomplete interval. To find the size of the remaining interval, P, we do the following:

$$P = R_i - \left\lfloor \frac{R_i}{T_j} \right\rfloor T_j$$

And thus we use the value P, to calculate the number of transformed executions within the remaining interval, x:

$$x = \left\lceil \frac{P}{T_j/m} \right\rceil \frac{C_j(HI)}{m}$$

Therefore the complete calculation will include the transformed tasks within the incomplete interval and the complete executions of $C_j(LO)$. This is shown in Equation [14].

$$min\{x, C_j(LO)\} + \left\lfloor \frac{R_i}{T_j} \right\rfloor C_j(LO) \quad (14)$$

The interference suffered from the transformed tasks within the incomplete interval will be the minimum of x or $C_j(LO)$.

In keeping with the nature of this work we then considered how the more accurate analysis presented above might be adapted to work in a system with more than two criticality levels. The analysis itself is applicable with little alteration.

The number of complete executions of $C_j(L_i)$ within the interval can be calculated.

$$\left\lfloor \frac{R_i}{T_j} \right\rfloor C_j(L_i)$$

And therefore we calculate the interference in the remaining time period from the transformed executions.

$$P = R_i - \left\lfloor \frac{R_i}{T_j} \right\rfloor T_j$$

$$x = \left\lceil \frac{P}{T_j/m} \right\rceil \frac{C_j(L_j)}{m}$$

The complete calculation for n criticality levels is shown in Equation [15].

$$min\{x, C_j(L_i)\} + \left\lfloor \frac{R_i}{T_j} \right\rfloor C_j(L_i) \quad (15)$$

As can be seen, the analysis is almost directly applicable. The key challenge is the transformation of the tasks in such a way that a criticality monotonic order is created.

Rather than transforming all higher criticality tasks, with a period greater than the shortest period of any LO task, the process for n criticality levels must be iterative. Consider Table I where $HI > ME > LO$.

	T	L
τ_1	80	HI
τ_2	110	ME
τ_3	100	LO

TABLE I
3 CRITICALITY LEVEL PT EXAMPLE, UNTRANSFORMED

Of the three tasks shown in Table I, τ_2 is the only one requiring transformation as it has a period greater than that of τ_3. We can calculate the transformation factor, n, as follows:

$$m = \left\lceil \frac{110}{100} \right\rceil$$

Thus m = 2, this will give τ_2 a transformed period of 55, less than the period of τ_1. The set is not criticality monotonic and, as such, it is clear that this calculation will not suffice. Rather τ_1 must also be transformed to give it a period of 40.

IV. Evaluation

We tested the algorithms above on randomly generated task sets. The task sets were generated the same way as in [2]. We generated 5000 sets of 10 tasks per 2% total utilisation. As MC tasks might have multiple WCETs, we consider the WCET value generated for each task to be at the highest criticality level in the system. The lowest level in the system is half the highest, each additional level is evenly distributed between the highest and lowest.

Our experimental data uses Criticality Dependant Utilisation to assess the percentage of tasks schedulable at any particular utilisation.

$$U_i(L_i) = \frac{C_i(L_i)}{T_i} \tag{16}$$

In order to provide a thorough comparison we considered the performance of AMCrtb, AMCmax and PT (ignoring the inherent overheads). Alongside SMC [2], SMC-NO(Vestal's) [4] and Criticality Monotonic ordering (CrMPO).

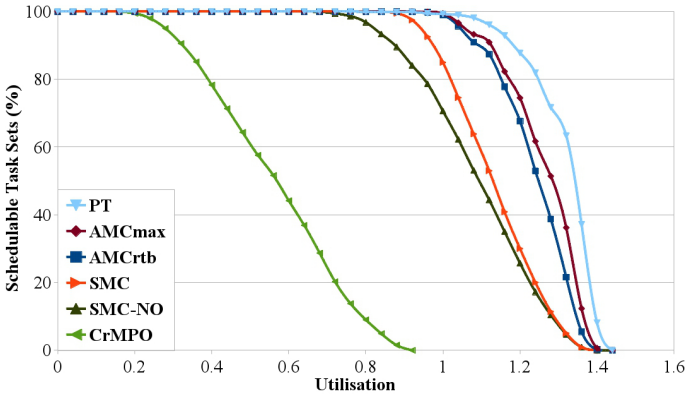

Fig. 3. Two Criticality Levels

Figure 3 shows the performance, at two criticality levels, of each technique at varying utilisations (Criticality Dependant). It is clear that AMCrtb out-performs SMC, CrMPO and SMC-NO. AMCmax performs slightly better than AMCrtb and PT performs well if overheads are ignored.

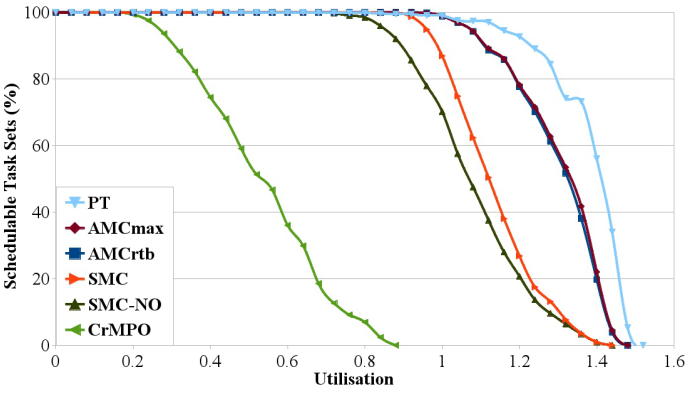

Fig. 4. Three Criticality Levels

As the number of criticality levels is increased each of the algorithms, apart from PT, maintain their performance relative to each other. Period Transformation's performance degrades due to the increasing complexity of the transformations required as the number of criticality levels is increased. Figures 3, 4 and 5 are typical of the relative results obtained for task sets with different characteristics.

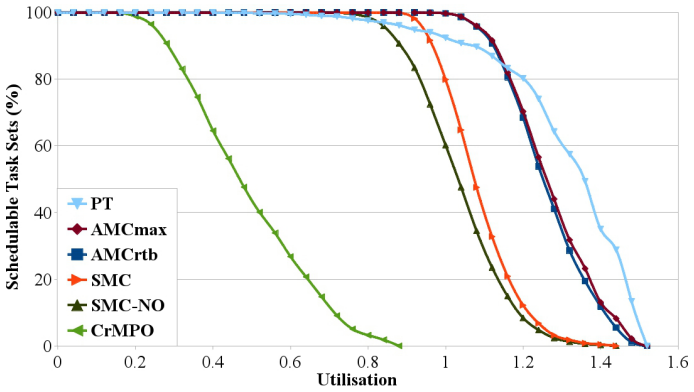

Fig. 5. Five Criticality Levels

V. Conclusion

It is clear that both AMC methods perform well compared to other techniques as the number of criticality levels is increased. Importantly both AMCrtb and AMCmax maintain their strong dominance over SMC. Although AMCmax does out-perform AMCrtb, AMCrtb remains an excellent approximation of AMCmax while keeping computational costs relatively low. As Mixed Criticality systems move forward it may become important to support even greater than 5 levels of criticality. In this case AMCrtb's relatively low computational cost would allow its application.

Period Transformation performs as expected. It provides a schedulability boost and avoids criticality inversion at the cost of vastly increased overheads and the requirement for additional runtime monitoring.

In short, AMCrtb remains dominant over SMC and provides a good approximation of AMCmax at any number of criticality levels. In practice AMCrtb will out perform PT.

References

[1] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[2] S. Baruah, A. Burns, and R. Davis. Response-time analysis for mixed criticality systems. In *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, pages 34 –43, 29 2011-dec. 2 2011.

[3] L. Sha, J. P. Lehoczky, and R. Rajkumar. Solutions for some practical problems in prioritized preemptive scheduling. In *RTSS*, pages 181–191, 1986.

[4] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 239 –243, dec. 2007.