Mixed Criticality Scheduling of Probabilistic Real-Time Systems

Jasdeep Singh[1(✉)], Luca Santinelli[1(✉)], Federico Reghenzani[2(✉)], Konstantinos Bletsas[3(✉)], David Doose[1], and Zhishan Guo[4(✉)]

[1] ONERA-DTIS Toulouse, Toulouse, France
{jasdeep.singh,luca.santinelli,david.doose}@onera.fr
[2] DEIB-Politecnico di Milano, Milan, Italy
federico.reghenzani@polimi.it
[3] CISTER Research Centre and ISEP/IPP, Porto, Portugal
koble@isep.ipp.pt
[4] University of Central Florida, Orlando, USA
zhishan.guo@ucf.edu

Abstract. In this paper we approach the problem of Mixed Criticality (MC) for probabilistic real-time systems where tasks execution times are described with probabilistic distributions. In our analysis, the task enters high criticality mode if its response time exceeds a certain threshold, which is a slight deviation from a more classical approach in MC. We do this to obtain an application oriented MC system in which criticality mode changes depend on actual scheduled execution. This is in contrast to classical approaches which use task execution time to make criticality mode decisions, because execution time is not affected by scheduling while the response time is. We use a graph-based approach to seek for an optimal MC schedule by exploring every possible MC schedule the task set can have. The schedule we obtain minimizes the probability of the system entering high criticality mode. In turn, this aims at maximizing the resource efficiency by the means of scheduling without compromising the execution of the high criticality tasks and minimizing the loss of lower criticality functionality. The proposed approach is applied to test cases for validation purposes.

1 Introduction

Real-time applications demand timing guarantees at all of their execution scenarios. Classical approaches apply Worst Case Execution Times (WCET) in order to have safe/pessimistic models of task executions. The actual task execution time may vary but will always lie below the WCET because the instances when the actual execution time is equal to WCET are unlikely [15,17]. Predictability is assured with schedulability analysis that applies worst-case models like WCETs.

This work is also a result of the CISTER Research Unit (UID/CEC/04234), supported by FCT/MCTES (Portuguese Foundation for Science and Technology).

A recent approach to timing analysis involves defining execution times using a probabilistic Worst Case Execution Time (pWCET). The pWCET is a probabilistic distribution which upper bounds all the possible execution times of a task [7]. The pWCET generalizes the notion of WCET with multiple worst-case execution time values, each with the associated worst case probability of being exceeded. The flexibility from pWCET representations allow for probabilistic quantification of pessimism in the WCET deterministic models. Moreover, the probabilistic models are less pessimistic because they are close approximation to the actual task execution. They contain more information about execution time than single-valued deterministic WCET.

The probabilistic schedulability analysis like [4,12,18,19] applies on top of probabilistic models like pWCETs. The results obtained thereafter are also probabilistic as worst-case response time distributions. The probabilistic schedulability analysis exploits the flexibility of probabilistic representations, aiming at reducing the pessimism. There is always an associated cost of complexity when dealing with probabilistic distributions. Operations like convolution add to this complexity. The probabilistic models deal with more information given in the probability distribution than models using WCET. All the possible scenarios of execution of tasks, given by the pWCET as well as schedule, have to be taken into account.

In addition, today's safety critical applications are being approached through the Mixed Criticality (MC) perspective [5]. MC systems operate by switching between various criticality modes depending on the resource requirements by the executing tasks. Whenever there is a higher requirement of resources, the system switches to a mode of higher criticality in order to guarantee the most critical tasks in the least. In this mode, only the safety critical tasks are executed and more resource for their execution is provisioned [22].

The MC problem is supported by the Vestal's model [22] in which a tuple of WCETs describe the task execution behaviour. The WCETs in the tuple are less pessimistic at lower criticalities, and are more pessimistic at higher ones in order to upper bound more execution conditions. The system mode change to higher criticality is classically means that all task in lower criticalities are discarded. This does not take into account if there is room for allowing execution of low criticality tasks in system high mode. Obtaining a solution for applied MC scheduling which is ensured safe as well as resource efficient is a complex problem.

Both MC and probabilistic schedulability analysis tend to have a common characteristic, that is to quantify the existing pessimism and utilize it to maximize resource usage by making intelligent scheduling decisions. This also originates from the fact that safety standards e.g., stemming from the IEC6150, ISO26262 or DO-178C standards, demand a probability or frequency of the system failing at run-time. These standards also extend to mixed criticality systems. Some works are extending the Vestal's model to the probabilistic case [8,14,16]. Works involving probabilities in MC systems [1,2,8,14] can be found among the citations in the MC survey [5]. Scheduling approaches in [10,13] and [3] focus on assigning safe and feasible task priorities.

1.1 Problem Discussion and Contribution

In this work, we look for a probabilistic MC scheduling analysis which provides a reliable probabilistic picture of the system, and safe timing guarantees, especially for high criticality tasks, or low probability of occurrence of critical events. In addition, it must be able to exploit the common objective of the MC and probabilistic approaches, which is leveraging probabilities into maximizing resource usage and minimizing pessimism.

Given a MC periodic non-preemptive task set known beforehand to be executed on a uniprocessor machine, with each task described with a pWCET, instead of a WCET, and given a maximum probability of deadline miss for the tasks, how do we find a schedule such that the probability of the system entering high criticality mode is minimum?

We aim for a minimization of the probability of system entering high criticality because there are certain predefined actions that are taken when the system does enter high criticality. Usually, these actions involve discarding the lower criticality tasks. With our approach, we make such actions least likely. With this objective in mind, we propose a graph-based task execution model. We begin constructing a graph based model to represent the possible job orderings. The possible schedules are then represented in exploration trees. These trees are explored to obtain a resource efficient schedule. Using graphs allows us explore all the possible combinations of job scheduling. Only by exploring all the combinations, we can confidently conclude for a schedule with the least probability of the system switching mode to high criticality among all the possibilities. Using the graph and other structures to model task executions, we are able to extract crucial schedulability information like pWCRT and the probabilities associated. There exist some approaches like [9, 20] which use graphs to express for task execution, but they do so in a non-probabilistic and non-MC domain.

Our approach mainly consists of an offline analysis to obtain a schedule for the jobs. The schedule obtained is a sequence of jobs which are ensured to contain all the high criticality jobs. In addition, the schedule results in the minimum probability that the system enters high criticality mode. The exploration process has exponential complexity. It originates from the complexity of the probabilistic models in addition to the complexity of the MC approaches. However, the relatively high offline complexity is a trade-off we make at the moment where we gain a new application oriented MC approach through the advantages of probabilistic models. The complexity of this exploration is somewhat reduced in parallel of the offline construction of the schedule by discarding unfeasible schedules. The complexity will be further improved in the future works.

The online part of the schedule consists of execution of jobs in the sequence given from the offline analysis. It simply executes the jobs given in the scheduling order. The online complexity is drastically reduced through our method as the schedule is a simple list to follow.

As in most works on MC scheduling, we assume a system with tasks of two criticalities and modes. However, our work can easily extend to multi-criticality system. Moreover, contrary to classical MC we use probabilistic Worst Case

Response Time (pWCRT) to make criticality decisions. Any increased resource demand from a jobs occurs at run-time. The criticality mode change for a job is a reflection of this increased demand. This run-time execution information is represented by the response time and not execution time. With this, we also leverage probabilities into scheduling decisions in order to maximize the resource usage. This is because the pWCRT is affected by the schedule and the schedule defines the resource assignment to the jobs.

Outline of the Paper. The paper is divided as follows: Sect. 1 introduces the context and states the problems which we answer in this paper. Section 2 gives the reader the necessary background to understand this paper as well as the assumptions made. Section 3 presents the graph-based model and the exploration trees. Section 4 details the schedulability analysis from graph and tree representations, together with the offline and online strategies developed. The paper is concluded in the Sect. 5.

2 Notations and Definitions

In this work we consider pWCETs to define task execution which is probability distribution.

For a discrete random variable $\mathcal{C}$, the Probability Mass Function (PMF) $f_{\mathcal{C}}(x)$ of $\mathcal{C}$ gives the probability that x takes a certain value in $\mathcal{C}$, $f_{\mathcal{C}}(x) \overset{def}{=} \mathcal{P}(x = \mathcal{C})$ with the condition $\Sigma_{-\infty}^{\infty}\mathcal{P}(x) = 1$. Alternative representations of $\mathcal{C}$ are the discrete Cumulative Distribution Function (CDF) $F_{\mathcal{C}}(x) \overset{def}{=} \Sigma f_{\mathcal{C}}(c)$, and the discrete Inverse Cumulative Distribution Function (ICDF) $\overline{F}_{\mathcal{C}}(x) \overset{def}{=} 1 - F_{\mathcal{C}}(x)$. In the rest of the paper, calligraphic letters are for random variables, while non-calligraphic letters are for deterministic variables. Figures 1 shows an example of a certain distribution in PMF, discrete CDF and discrete ICDF forms. The discrete cumulative forms show probability increment or decrement in a step-wise fashion.

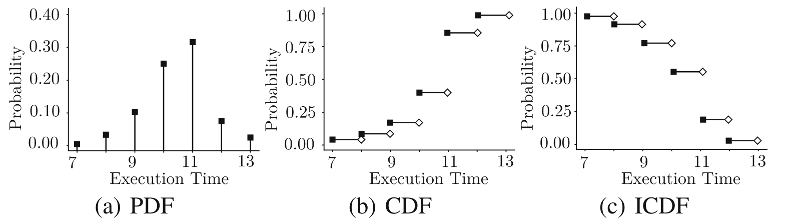


Fig. 1. Example of PDF, discrete CDF and discrete CCDF representations of a certain distribution.

The task pWCET is a discrete random variable $\mathcal{C}$ taking task execution values where PMF $f_{\mathcal{C}}(x)$ represents the probability that the task takes a certain

WCET. In its representation with CDF, $F_{\mathcal{C}}(x)$ is the cumulative probability that the task respects WCET x while executing; in the ICDF representation, $\overline{F}_{\mathcal{C}}(x)$ is the probability that the task overcomes WCET x. The deterministic WCET C from $\mathcal{C}$ is the maximum value of $\mathcal{C}$; for it $F_{\mathcal{C}}(C) = 1$, and $\overline{F}_{\mathcal{C}}(C) = 0$. Figure 2 shows a discrete form of a pWCET with its maximum value as WCET.

The convolution of two PMFs $f_{\mathcal{X}}(x)$ and $g_{\mathcal{Y}}(y)$, denoted by $\otimes$, refers to the summation of the random variables $\mathcal{X}$ and $\mathcal{Y}$, given as: $f \otimes g(u) = \Sigma_{v=-\infty}^{\infty} f(v)g(u-v)$.

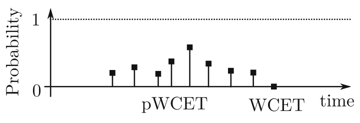


Fig. 2. pWCET PMF with each worst case execution time having a worst case probability.

2.1 Computational Model

We assume a fixed set of periodic tasks executing in a system with each instance of a task called a *job*. All the tasks are known beforehand. We focus our analysis on the jobs. A MC job J_i is represented with a tuple of parameters:

$$J_i = (\mathcal{C}_i, a_i, d_i, L_i, l_i); \tag{1}$$

$\mathcal{C}_i$ is the job pWCET probability distribution; the job arrives at time a_i; has the deadline d_i; and l_i is the criticality execution time threshold. The arrival time and the deadline are deterministic variables. The criticality level of the job is L_i. We assume that the arrival of the first job of each task is always at time zero. The tasks are scheduled periodically and non-preemptively on a uniprocessor machine in which execution of jobs are suspended at their respective deadlines.

For the jobs, we consider pWCET described with discrete distributions, although our method applies to both, continuous and discrete distributions. We analyze the jobs in the hyperperiod because the schedule repeats each hyperperiod. Since the jobs are suspended at the deadline, the execution order does not change across the hyperperiods. The sequence of the jobs remain the same and the execution room for each job is sufficiently given within the hyperperiod. The execution behaviour of the jobs across the hyperperiods can be different and our approach takes that into account. The hyperperiod is defined as the least common multiple of the periods of all the tasks.

Definition 1. *For a probabilistic job J_i as defined in Eq. (1), its probabilistic Worst Case Response Time pWCRT is the PMF $f_{\mathcal{R}_i}(x)$ which gives the probability that J_i will take certain time $\mathcal{R}_i$, to end execution after its release. The CDF is $F_{\mathcal{R}_i}(x) = \Sigma f_{\mathcal{R}_i}(c)$, and the ICDF is $\overline{F}_{\mathcal{R}_i}(x) = 1 - F_{\mathcal{R}_i}$.*

Because the response time is a probability distribution, which in turn comes from pWCET of the jobs, the deadline miss also has an associated probability. This is easily extracted from the response time. We assume a certain allowed maximum probability of deadline miss for any job $\mathcal{P}_{dm}^{max}$ is given. A job J_i is said to have missed its deadline if $1 - \Sigma_{x=0}^{d_i} f_{\mathcal{R}_i}(x) > \mathcal{P}_{dm}^{max}$.

Criticality Levels. We consider two level criticality case, HI and LO, with HI having higher importance than LO. The high criticality job can execute in HI or LO mode, the low criticality job executes only in LO mode. After its release, the high criticality job executes in LO criticality mode until the response time threshold l_i. A job execution exceeding this threshold is said to execute in the high criticality mode. Evidently, l_i is a deterministic single-valued parameter. The job criticality is given by L_i which is its relative importance over others. L_i can take values HI or LO.

Definition 2. *A job J_i is said to have entered* HI *criticality if its response time exceeds a threshold l_i, $f_{\mathcal{R}_i}(x) > l_i$.*

We explain criticality using the response time because we can make scheduling decisions based on actual job execution. This implies, criticality decision takes into account the affect of other job executions as well. Figure 3 shows an example of a job pWCET and pWCRT. It shows both the distributions with criticality thresholds. The criticality obtained from threshold applied on response time contains more run-time information than that obtained from pWCET.

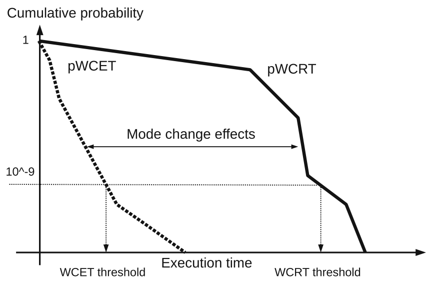


Fig. 3. pWCET and pWCRT representation for criticality mode change using thresholds on pWCET and pWCRT distributions.

In the hyperperiod there are n jobs, and the set of jobs is Λ. There are n^{HI} high criticality jobs in Λ, and n^{LO} low criticality jobs in Λ; Λ^{HI} represents the set of high criticality jobs, and Λ^{LO} represents the set of low criticality jobs; $\Lambda = \Lambda^{\text{HI}} \cup \Lambda^{\text{LO}}$, and $n = n^{\text{LO}} + n^{\text{HI}}$.

Worst-Case Independence. The pWCETs are assumed to be independent [6]; this is because the pWCET represents the worst case execution scenario of the job. Independence implies that the execution of one job does not inherently affect the execution of another (dependence in this case would be in the cases like shared resources by the two jobs). Any dependencies on execution and criticality mode changes due to scheduling of the jobs is not assumed and is taken into account. Any execution delays which are caused apart from the scheduling must already be included in the pWCET distribution.

3 Probabilistic Scheduling Model

We propose an exploration of graph models for possible job execution combinations. We find a schedule which ensures the schedulability all the jobs. Then, it ensures that the probability that the system enters high criticality is minimum. This is when the job criticality is computed from its response time. We begin by defining the graph model. We follow by elaborating the graph model into exploration tree.

3.1 Graph Model

The graph represents the possible job combinations as schedules of the system in each hyperperiod. It uses nodes to represent the jobs and arcs to represent the possible ordering of execution among them. A *directed graph* is defined as a tuple $G = \{V(G), E(G)\}$, where $V(G)$ is a finite set of elements called nodes and $E(G)$ is the finite set of ordered pairs of elements of $V(G)$ called arcs. The graph is acyclic because the schedule repeats each hyperperiod. Since there is no passing of information across the hyperperiods, it is not required to be represented through cyclic graph.

Nodes. Each node $J_i \in V(G)$ represents the execution of a job $J_i \in \Lambda$. Since the elements of $V(G)$ are one-to-one mapped to the elements of Λ, i.e. it represents the jobs, we directly use J_i to represent the node.

The system can begin execution with any of the jobs which arrive at time zero, i.e. first jobs of the tasks. The nodes representing these first jobs are called the *early nodes.* The *early nodes set* $S(G)$ is a subset of the node set $V(G)$, $S(G) \subseteq V(G)$ such that $\forall\, J_i \in S(G) : a_i = 0$. The system can potentially begin execution by any of the jobs in this set. Graphically, we identify the early nodes set $S(G)$ with extra arcs entering in $J_i \in S(G)$ without the source node, see Fig. 5 for an example. These arcs are not considered part of the $E(G)$ set.

Arcs. An arc $\{J_i, J_j\} \in E(G)$ represents a possible ordering of jobs, in particular, that the job J_j executes after the execution of the task J_i. Formally, for $\{J_i, J_j\} \in V(G)$:

$$\{J_i, J_j\} \in E(G) \text{ if } \begin{cases} a_i < d_j & \text{if } i \neq j \\ a_j = a_i + T_i & \text{if } i = j. \end{cases} \tag{2}$$

An arc only exists when the deadline of the next job is greater than the arrival time of the executing job (previous job). This is enforced to prevent the scheduling of a job which has already passed its deadline. Also, in order to prevent scheduling the same job more than once, no self loop (arcs connecting themselves) exist, $\not\exists\{J_i, J_i\} \in E(G)$. It should be noted that, since the graph is directed, $\{J_i, J_j\}$ is not the same as $\{J_j, J_i\}$. Also, the arcs do not represent the time of execution of the jobs, they simply direct to the next job to execute J_j once the executing job J_i finishes. For $\{J_i, J_j\}$, we define a *successor node* as $succ(J_i) = J_j$ and a *predecessor node* $pred(J_j) = J_i$.

3.2 Scheduling Tree

In order to search for a schedule, the graph is unfolded into trees defined as follows.

Definition 3 (Exploration Tree). *The* exploration tree $T_s(G)$ *of a graph* G *with an early node* $J_s \in S(G)$ *is defined as* $T_s(G) = \{V(T_s(G)), E(T_s(G))\}$: $1 - \Sigma_{x=0}^{d_i} f_{\mathcal{R}_i}(x) > \mathcal{P}_{dm}^{max} \forall J_i \in V(T_s(G))$*, where* $V(T_s(G))$ *is the set of nodes of* $T_s(G)$ *and* $E(T_s(G))$ *is the set of arcs of* $T_s(G)$ *such that* $E(T_s(G)) \exists E(G)$ *and* $V(T_s(G)) \exists V(G)$.

The *exploration tree* is constructed from the graph beginning with a root node of the graph as the first job to execute in the schedule. Each node of the tree is labelled with the job J_i it represents. A node is only added to the tree if it does not miss its deadline and a corresponding arc exists in the graph, given that the corresponding node exists the graph. At each time it is added to the tree, the deadline miss is checked for by the condition $1 - \Sigma_{x=0}^{d_i} f_{\mathcal{R}_i}(x) > \mathcal{P}_{dm}^{max}$ for a job J_i. If there is a deadline miss, the node/job is not added because the system should never be scheduled beyond this job.

To relax the notation, from now on we simply write $V(T)$ and $E(T)$ to represent nodes/jobs and arcs respectively of the tree. In particular, since the tree is a rooted tree, its root is defined as the unique node with the label J_s. Because the system can begin with one of the jobs arriving at time zero, there are different possible roots of a tree; this implies the number of possible trees is equal to the number of tasks because each task has a first job arriving at time zero. These trees are collectively called a *forest.*

The set of all *exploration trees* is called the *exploration forest*: $F = \{T_{s_1}(G), T_{s_2}(G), ...T_{s_k}(G)\}$, where $s_1, ..., s_k$ are the indices of early jobs in $S(G)$. The trees represent possible orderings or sequences of jobs beginning with an early job at the root. A *leaf node* in the tree is a node with no outgoing transitions, J_n is a leaf node if $succ(J_n)$ does not exist. Thus, a schedule is a path taken through the tree from a node to a leaf node defined as follows.

Definition 4 (Path). *A path in the T-th tree* $path_T(J_i, J_n)$ *is a unique sequence of connected arcs starting from a node* J_i *to a leaf node* J_n*:* $path_T(J_i, J_n) = \{\{J_i, J_j\}, \{J_j, J_k\}, ..., \{J_l, J_n\}\}$ *with* $\{J_x, J_y\} \in E(T)$ *for any* x, y, $i \neq j \neq k \cdots \neq n$.

Whenever we refer to the notation $path_T(J_i, J_n)$, we always refer to a unique path with a node J_i and leaf node J_n. Two paths are same if their elements, which are the sequences of arcs, are the same. We specify this because there are many possible paths from the root node to a leaf node notated by the same job. Also, a path can begin at any node and it ends at a leaf node.

For any node in the tree, there must not exist a same node in the path between itself and the root node This is done to prevent scheduling the same job more than once, we enforce the following definition. Formally, a node J_i is not added to the tree if it already exists between the root and the desired point of addition:

$$J_j = succ(J_n) \text{ if } \nexists J_j \in path_T(J_s, J_n) : J_n \text{ is leaf node}, J_s \in S(G). \tag{3}$$

Example 1. We use a set of jobs Λ_1 shown in Fig. 4 to explain our method. It consists of five jobs (from two periodic tasks) in the hyperperiod of 30 time units. The jobs are shown with their pWCET PMF in the figure to visualize a probabilistic execution and not deterministic execution, the exact values of PMFs are not yet important. The jobs of task τ_1 $((J_1, J_4))$, are HI criticality and those of task τ_2 $((J_2, J_3, J_5))$ are LO criticality.

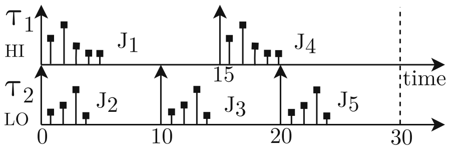


Fig. 4. Jobs of the set Λ_1.

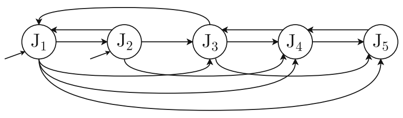


Fig. 5. The graph of the jobs in Λ_1.

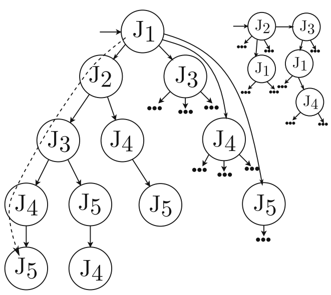


Fig. 6. Graph unfolded into a tree for jobs of Λ_1.

The graph is a direct representation of all the possible ordering of the jobs in the system, see Fig. 5. There are nodes J_1, J_2, etc. for each job which are interconnected by uni-directed arcs which refer to the order of executions; e.g., if J_3 is executing, then it can be followed by the jobs J_1, J_4 or J_5. The system can begin execution with J_1 or J_2. This set of early nodes of the jobs J_1 and J_2 are represented by the extra arcs entering those nodes. In order to explore this graph to look for a schedule, it is unfolded into a forest of trees.

For Λ_1, a portion of the tree is represented in the Fig. 6. The tree begins with a root J_1 followed by possible jobs which can execute, namely, J_2, J_3, J_4, J_5. This goes on until the leaf nodes where any more addition of nodes will mean a repetition of the same job in the schedule on the path. For the tree job set Λ_1 the Fig. 6, one possible path is shown by a dotted line. The dotted line represents a path J_1, J_2, J_4, J_3, J_5. It also shows another possible tree with just its root J_2.

With the construction of the exploration model complete, we now proceed extracting certain metrics from the exploration trees in order to decide for a MC schedule.

4 Evaluation

We first the define the metrics necessary for system criticality and then apply them to our model. We begin with obtaining a non-preemptive pWCRT for a job.

Classically, the pWCETs of the jobs are simply convoluted in order to obtain the pWCRT. The convolution operation does not take different arrival times into account. Thus, performing a convolution contains a hidden assumption that all the jobs arrive at the same time, i.e. at a critical instant. This results in a pessimistic pWCRT. Our approach to obtain the pWCRT involves handling the discrete distributions in a piece-wise manner. To do so, we first define a tail PMF as follows.

Definition 5 (Tail Probability Mass Function). *A tail distribution of the response time $f_{R_i}(x)$ of a job J_i with Worst Case Execution Time $WCET_i$ for some time $a_j(0 < a_j < WCET_i)$, is a PMF $f_{tail_i}^{[a_j,WCET_i]}(x)$ given as:*

$$f_{tail_i}^{[a_j,WCET_i]}(x) = \begin{cases} f_{\mathcal{R}_i}(x + a_j) & \textit{if}\ 0 < x \le WCET_i \\ \Sigma_{y=0}^{a_j} f_{\mathcal{R}_i}(y) & \textit{if}\ x = 0. \end{cases} \tag{4}$$

The tail distribution represents a complete PMF which probabilistically delays the execution of the next job in the schedule where the next job arrives at time a_j. The probability accumulated at the instant a_j ($x = 0$ in the function) in the tail PMF represents the probability that the job J_i has finished execution by then. As in Fig. 7, the tail distribution of job J_1 (from Λ_1 in Example 1) accumulates $0.1 + 0.15 + 0.15 + 0.2 = 0.6$ time zero. By doing so we prevent any loss of information in the distribution. We use this to obtain the pWCRT of the delayed job.

In order to obtain the response time of the jobs, we present the following theorem. The approach we propose is similar to the ones used by [12,14] but applied to every job with respect to their arrival times. It is not same as the classical convolution because convolution does not take the relative arrival times and the deadlines of the jobs into account. Classical convolution between the pWCETs of two jobs contains a hidden assumption that the jobs arrive at the same time. This is easily seen as the tail distribution approaches full pWCET when the arrival times are equal. Since this is not always the case, convolution results in a pessimistic response time distribution.

Theorem 1 (Non-Preemptive Probabilistic Worst Case Response Time). *The pWCRT of a job without preemption is represented by convolution between its pWCET and the Tail PMF of the job executing immediately before.*

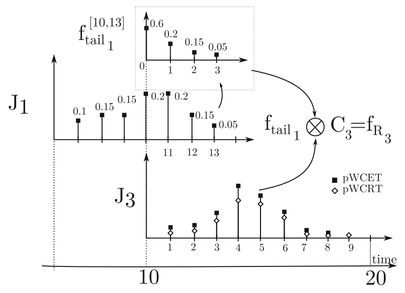


Fig. 7. pWCRT from convolution of Tail and pWCET.

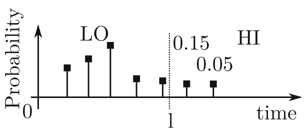


Fig. 8. High criticality from a threshold l_1 on pWCRT of a job J_1 of Λ_1.

Proof. Consider two jobs, J_i arriving at time a_i and J_j arriving at time a_j, $a_i \leq a_j$. A representative example is shown in Fig. 7 as J_1 and J_3 for the jobs in Λ_1 of Example 1. Probabilistically, J_i can continue to execute after the arrival time of J_j. That means, there exists a probabilistic delay to the execution of the job J_j due to the execution of the job J_i. However, the probabilistic delay only exists due to J_i executing between time a_j and its $WCET$ (which is the maximum value of its pWCET). Thus, the state space which affects the execution of J_j is $[a_j, WCET]$. This state space contains the tail of the pWCRT of J_i. In order for the state space to be complete it must respect the property that sum of all the probabilities in the state space is equal to one. This implies, the tail of pWCRT of J_i must be a PMF on its own. This distribution is given by the function $f_{tail_i}^{[a_j,WCET]}(x)$.

The probability that J_j ends execution at time a_j depends on the probability that J_j finishes at time a_j and J_i finishes at time a_j OR J_i finishes at time $a_j + 1$ OR at time $a_j + 2$ OR ... and so on. Similarly, J_j ends execution at time $a_j + 1$ depends on the probability that J_j finishes at time a_j and J_i finishes at time $a_j + 1$ OR at time $a_j + 2$ OR ... and so on. This way we approach the classical convolution operation between the pWCET $f_{C_j}(x)$ of J_j and the tail function $f_{tail_i}^{[a_j,WCET]}(x)$, given as:

$$f_{R_j}(x) = f_{C_j}(x) \otimes f_{tail_i}^{[a_j,WCET_i]}(x).$$

□

Our approach to obtain the pWCRT separates the part of the pWCRT PMF which affects the execution of the next job. We do not lose any information in the distribution as the probabilities before the time a_j are accumulated[1]. Moreover, we retain the information within the time intervals in the distribution.

Using the pWCRT we apply the job criticality Definition 2 on the paths through the exploration tree.

[1] The distribution function accumulates the probabilities in the intervals of discretization at the worst case, e.g. probabilities at execution times $0.2, 0.5, 0.7$, etc. are accumulated the time 1.

Definition 6 (Job Criticality). *A job J_j in a path $path_T(J_i, J_n)$ is said to have entered high criticality if its response time crosses the threshold l_j, $f_{\mathcal{R}_j}(x) > l_j$, the probability of which is given as:*

$$\mathcal{P}_j^{\mathrm{HI}}(path_T(J_i, J_n)) = 1 - \Sigma_{x=0}^{l_j} f_{\mathcal{R}_j}(x), \tag{5}$$

where $l_j < d_j$ and J_n is a leaf node.

Example 2. Figure 7 shows a scenario where job J_3 executes after J_1 from the jobs in Λ_1 in Example 1; for explanation, a pWCET PMF is assumed for J_1. The Tail PMF for the same is shown in the box. To obtain the pWCRT of J_3, this PMF is then convoluted with the pWCET of the job J_3.

A pWCRT of the job J_1 with its threshold is shown in Fig. 8. The threshold is shown by the dotted line labelled $l_i = 12$. From the Figure, the probability that this job enters HI-criticality mode is $0.15 + 0.05 = 0.20$.

From the job criticality defined using its pWCRT, we define the probability of the system entering high criticality as follows.

Definition 7 (System Criticality). *The system enters high criticality mode if at least one high criticality job enters high criticality mode.*

To elaborate the above definition, system enters high criticality if the first high criticality job enters high criticality OR the second high criticality job enters high criticality OR the third..., and so on. This represents a summation of probabilities. However, the probability that a job enters high criticality is given from its pWCRT which has its own sample space. In order to sum the probabilities of all the high criticality jobs involved, we also need to sum the sample space. This is clear from the fact that the summation of probabilities without considering the sample space can result in a probability greater than one[2]. For each job entering high criticality, the probability adjusted for state space is $\Sigma\mathcal{P}_i^{\mathrm{HI}}(path_T)/(\text{number of high criticality jobs})$. We use the law $\mathcal{P}(A \cup B) = \mathcal{P}(A) + \mathcal{P}(B) - \mathcal{P}(A \cap B)$ for any two events A and B with $\mathcal{P}()$ giving their probability of occurrence [21]. Therefore, the probability that the system enters high criticality is given as the summation of probability of each high criticality job entering high criticality minus their product.

We apply these definitions to the graph and tree model. In our model, the schedule is represented as paths. Because the paths represent the schedule, the probability that the system enters high criticality is also a function of paths. However, not all the paths in the exploration tree are available for scheduling because not all paths contain all the high criticality jobs. The set of available paths are defined as follows.

[2] Same reasoning also applies to multiplication of probabilities, however the denominator 1 gets multiplied too.

Definition 8 (Available paths). *Available paths is a set of all possible paths in a T-th tree from the root node $J_s \in S(G)$ to a leaf node J_n, $P_{avail} = \{path_T(J_s, J_n)\}$ such that*

$$\forall k : J_k = \Lambda \implies \exists J_k \in path_T(J_s, J_n); \tag{6}$$
$$and$$
$$\forall J_k \in path_T(J_s, J_n) \implies 1 - \Sigma_{x=0}^{d_k} f_{R_k}(x) < P_{dm}^{max}$$

P_{avail} is a set of possible schedules of the jobs in MC through the trees as the available paths. The available paths are the paths in which contain all the jobs in it and all those jobs meet their respective deadlines. In our context, the set of available paths represent the possible candidates to find a schedule. It should be noted that the criteria of all jobs meeting their deadlines is already met while constructing exploration tree. A node is not added to the tree if it misses its deadline. To quantify the probability of these paths we apply the definition of system criticality on the available paths as follows.

Definition 9 (Probability of system entering high criticality). *For an available path $path_T(J_s, J_n) \in P_{avail}$, the probability $\mathcal{P}_{sys}(path_T(J_s, J_n))$ that the system enters high criticality by taking this path is given as:*

$$\mathcal{P}_{sys}(path_T(J_s, J_n)) = \Sigma_j \frac{\mathcal{P}_j^{\text{HI}}}{n^{\text{HI}}} - \Pi \mathcal{P}_j^{\text{HI}}, \forall J_j \in path_T(J_s, J_n), \forall J_j \in \Lambda^{\text{HI}}; \tag{7}$$
$$path_T(J_s, J_n) \in P_{avail}.$$

Using the available paths and the system criticality metric defined above, we finally obtain a *mixed criticality (MC) schedule* as follows.

Definition 10 (Mixed Criticality Schedule). *A path $P_{MC} = path_T(J_s, J_n)$ is the mixed criticality schedule if $\mathcal{P}_{sys}(path_T(J_s, J_n))$ is the minimum among all possible $path_T(J_s, J_n) \in P_{avail}$ $\forall i, \exists J_i \in path_T(J_s, J_n)$, and $\forall i \exists J_i \in \Lambda$ where $J_s \in S(G)$.*

The MC schedule is a path P_{MC} through the exploration tree which contains all the jobs, no job misses its deadline and the probability of the system entering high criticality is minimum. It represents the schedule that should be taken by the system as the ordered sequence of execution of the jobs. To recall, the probability is minimum given the criticality is defined using the response time. Thus, we have found a solution to our problem which is a schedule represented by the path P_{MC}. The path P_{MC} by definition is from root node to a leaf node, thus we do not need to notate them (unlike jobs J_s and J_n in $path_T(J_s, J_n)$). Since this method is based on complete exploration, the schedule is guaranteed to be the optimal by minimizing the probability of system entering high criticality. Any impossible schedule is the one in which a job does not meet its timing constraints. These impossible schedules are already excluded while building the tree as we calculate response time in parallel.

Offline Schedule. As we see, the offline analysis results in a schedule P_{MC} such that the probability of system entering high criticality is minimum. The schedule ensures that no job missed its deadline. It uses an exploration of tree based on a graph representation of job executions. Because it is an exhaustive exploration, the minimum probability is ensured.

Online Schedule. The online part of our method is straightforward. It takes the schedule P_{MC} obtained in the offline analysis and executes the jobs as given in the sequence. The minimization of the probability of system entering high criticality has already been performed in the offline and is not required to be done in the online. The jobs are suspended if they reach their deadline and the sequence of jobs repeat each hyperperiod.

Example 3. We analyze the set of jobs Λ_2 shown in Table 1 which consists of 4 periodic tasks and 15 jobs with pWCET and implicit deadline as shown. There are 6 high criticality jobs from task τ_1 and 11 low criticality jobs from the rest of the tasks. To recall, the job set is to be scheduled non-preemptively on a uniprocessor system and the jobs are suspended at their deadlines. The threshold for the high criticality jobs of task τ_1 is set at 4 time units. The maximum allowed probability of deadline miss P_{dm}^{max} for any job is set at $1E-03$.

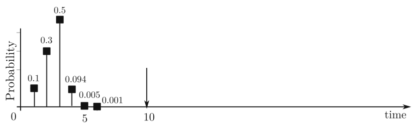
Table 1. Job set Λ_2.

Task	Deadline	pWCET	Criticality
τ_1	10	$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 8 \\ \hline 0.1 & 0.3 & 0.5 & 0.094 & 0.005 & 0.001 \end{bmatrix}$	HI
τ_2	20	$\begin{bmatrix} 1 & 2 & 3 & 4 \\ \hline 0.1 & 0.4 & 0.4 & 0.1 \end{bmatrix}$	LO
τ_3	15	$\begin{bmatrix} 1 & 2 & 3 & 4 \\ \hline 0.1 & 0.4 & 0.3 & 0.2 \end{bmatrix}$	LO
τ_4	30	$\begin{bmatrix} 1 & 2 & 3 \\ \hline 0.1 & 0.7 & 0.2 \end{bmatrix}$	LO

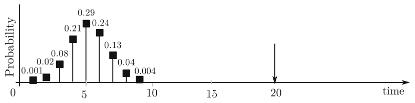
The proposed schedule P_{MC} is: $J_{11}, J_{41}, J_{31}, J_{32}, J_{21}, J_{12}, J_{13}, J_{42}, J_{33}, J_{22}, J_{14}, J_{15}, J_{34}, J_{23}, J_{16}$. The probability that the system enters high criticality is 0.0.00509. The pWCRT of some of the jobs is shown in Fig. 9. The pWCRT jobs of the high criticality task τ_1 remains unchanged as their pWCET as shown in Fig. 9(a). The pWCRTs of jobs which have been affected by probabilistic delay in execution are J_{21}, J_{22}, J_{23}, J_{31}, J_{33}, J_{34} and J_{41} whose pWCRTs are shown in Figs. 9(b), (c), (d), (e), (f) and (g), respectively.

We obtain this result along with other possible schedules in which there is no deadline miss but the probability of the system entering high criticality is higher. For example, another possible schedule is: $J_{11}, J_{21}, J_{12}, J_{31}, J_{41}, J_{32}, J_{13}, J_{14}, J_{22}, J_{42}, J_{15}, J_{33}, J_{34}, J_{23}, J_{16}$. In this case the probability of system entering high criticality is 0.00605.

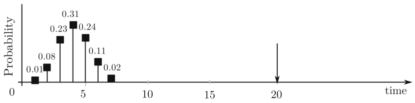
On computation, the tree for the job set Λ_2 contains $716,132$ nodes.



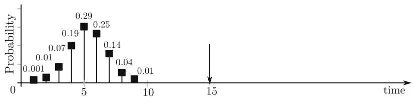
(a) Jobs of task τ_1



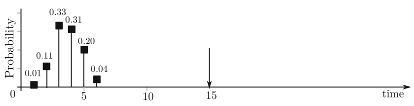
(b) Job J_{22}



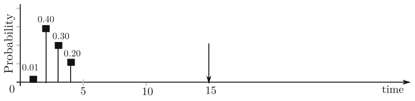
(c) Job J_{23}, J_21



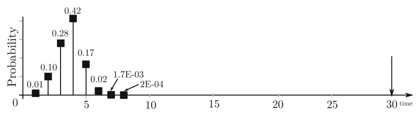
(d) Job J_{31}



(e) Job J_{33}



(f) Job J_{34}



(g) Job J_{41}

Fig. 9. pWCRT PMF of various jobs in Λ_2.

Overall, we observe that we can quantify the probability of something occurring in the system, like deadline miss or entering high criticality. This gives us a global picture of the MC system with pWCETs in terms of risk involved of system entering high criticality when applying such a system. In addition, we can control it through observing the pWCRTs and making the scheduling decisions accordingly.

Complexity. In general, MC problem lies beyond NP and PSPACE complexity and it is NP in uniprocessor case [11]. The complexity of our approach depends on the number of jobs n. The maximum complexity of building the graph is $O(n!)$. However, this might not always be the case because the tree is not built in the direction of a node which missed its deadline. Thus, the complexity also depends on the maximum allowed probability of deadline miss. The complexity of analyzing the tree and finding the paths is linear to the number of leaf nodes in the tree. In addition to this, there exists computational complexity of the convolution operation, which in turn depends on the possible values a random variable can take. Assuming that all jobs release at one critical instant, the convolution complexity is $O(n^n)$.

Actual complexity of the offline process is much less. Firstly, it is rarely the case that all the jobs are release at the same time since they belong to periodic tasks in the hyperperiod. Second, some combinations of sequences of jobs will result in a deadline miss. This means that the exploration tree is not built in that direction.

5 Conclusion

We have utilized a graph base exploratory method to obtain a non-preemptive schedule for MC probabilistic real-time system on uniprocessor machine, where task executions are suspended at the deadline. The task criticalities are defined using the pWCRT. We do this to make criticality decisions based on tasks demanding more resource at run-time. The obtained schedule minimizes the probability of system entering high criticality mode. This way, the actions needed to perform to cope with system high criticality are made less likely for the application.

At the current state of the work, the complexity of searching through enumerations is high. This work is one of the first steps to formalize the mixed criticality domain through response time. At the cost of complexity, we gain an application oriented perspective of the mixed criticality approach by using the response time. We adapt the schedule to the application itself and minimize the probabilities accordingly. We intend to reduce the complexity through methods like ordered searches and merging common and similar paths.

We intend to perform comparisons with real benchmarks in the future once the complexity is feasible for very large task sets. In future work we will extend this model to optimize the decisions when the system does enter high criticality. This future step is a hybrid approach where offline probability minimization has been performed. The online step of the hybrid approach will take care of the real-time response of the tasks and adjust the schedule accordingly to safer scenarios without jeopardizing the system critical functionality. We also aim to perform a sensitivity analysis on the criticality definition using the task response times.

References

1. Alahmad, B., Gopalakrishnan, S.: A risk-constrained Markov decision process approachto scheduling mixed-criticality job sets. In: Workshop on Mixed-Criticality Systems (2016)
2. Alahmad, B., Gopalakrishnan, S.: Risk-aware scheduling of dual criticality job systems using demand distributions. LITES **5**(1), 01:1–01:30 (2018). https://doi.org/10.4230/LITES-v005-i001-a001
3. Baruah, S., Easwaran, A., Guo, Z.: Mc-fluid: Simplified and optimally quantified. In: 2015 IEEE Real-Time Systems Symposium, pp. 327–337, December 2015
4. Bucci, G., Carnevali, L., Ridi, L., Vicario, E.: Oris: a tool for modeling, verification and evaluation of real-time systems. Int. J. Softw. Tools Technol. Transf. **12**(5), 391–403 (2010)
5. Burns, A., Davis, R.: Mixed criticality systems - a review, 12th edn. Technical report, Department of CS, U. of York, UK, March 2019
6. Cucu-Grosjean, L.: Independence - a misunderstood property of and for (probabilistic) real-time systems. In: Real-Time Systems: the Past, the Present and the Future (2013)
7. Davis, R.I., Burns, A., Griffin, D.: On the meaning of pWCET distributions and their use in schedulability analysis. In: In Proceedings Real-Time Scheduling Open Problems Seminar at ECRTS (2017)

8. Guo, Z., Santinelli, L., Yang, K.: EDF schedulability analysis on mixed-criticality systems with permitted failure probability. In: Proceedings of the RTCSA (2015)
9. Huang, H., Gill, C., Lu, C.: Mcflow: a real-time multi-core aware middleware for dependent task graphs. In: 2012 IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp. 104–113, August 2012
10. Huang, S., Zhu, Y., Duan, J.: A new scheduling approach for mix-criticality real-time system. In: 2013 Fourth International Conference on Intelligent Control and Information Processing (ICICIP), pp. 43–46, June 2013
11. Kahil, R., Socci, D., Poplavko, P., Bensalem, S.: Algorithmic complexity of correctness testing in MC-scheduling. In: Proceedings of the 26th International Conference on Real-Time Networks and Systems, RTNS 2018, Chasseneuil-du-Poitou, France, 10–12 October 2018, pp. 180–190. ACM (2018)
12. Kim, K., Diaz, J., Lo Bello, L., Lopez, J., Lee, C.G., Min, S.L.: An exact stochastic analysis of priority-driven periodic real-time systems and its approximations. IEEE Trans. Comput. **54**(11), 1460–1466 (2005)
13. Li, H., Baruah, S.: An algorithm for scheduling certifiable mixed-criticality sporadic task systems. In: Proceedings of the 2010 31st IEEE Real-Time Systems Symposium, RTSS 2010, pp. 183–192. IEEE Computer Society, Washington, DC (2010). https://doi.org/10.1109/RTSS.2010.18
14. Maxim, D., Davis, R.I., Cucu-Grosjean, L., Easwaran, A.: Probabilistic analysis for mixed criticality systems using fixed priority preemptive scheduling. In: Proceedings of the 25th International Conference on Real-Time Networks and Systems, RTNS 2017, pp. 237–246. ACM, New York (2017). http://doi.acm.org/10.1145/3139258.3139276
15. Nélis, V., Yomsi, P.M., Pinho, L.M., Fonseca, J., Bertogna, M., Quiñones, E., Vargas, R., Marongiu, A.: The challenge of time-predictability in modern many-core architectures. In: 14th International Workshop on Worst-Case Execution Time Analysis (2014)
16. Santinelli, L., George, L.: Probabilities and mixed-criticalities: the probabilistic C-space. In: 3rd International Workshop on Mixed Criticality Systems WMC at RTSS (2016)
17. Santinelli, L., Guet, F., Morio, J.: Revising measurement-based probabilistic timing analysis. In: 2017 IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2017, Pittsburg, PA, USA, 18–21 April 2017, pp. 199–208 (2017)
18. Santinelli, L., Meumeu Yomsy, P., Maxim, D., Cucu-Grosjean, L.: A component-based framework for modeling and analysing probabilistic real-time systems. In: 16th IEEE International Conference on Emerging Technologies and Factory Automation (2011)
19. Singh, J., Santinelli, L., Infantes, G., Doose, D., Brunel, J.: Mixed criticality probabilistic real-time systems analysis using discrete time Markov chain. In: 6th International Workshop on Mixed Criticality Systems (WMC) (2018)
20. Stigge, M., Ekberg, P., Guan, N., Yi, W.: The digraph real-time task model. In: Proceedings of the 2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2011, pp. 71–80. IEEE Computer Society, Washington, DC (2011). https://doi.org/10.1109/RTAS.2011.15
21. Soong, T.: Fundamentals of probability and statistics for engineers, p. 391, January 2004. ISBN 9780470868157
22. Vestal, S.: Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In: Proceedings of the 28th IEEE International Real-Time Systems Symposium (RTSS), pp. 239–243. IEEE Computer Society (2007)