Mixed-Criticality Scheduling Upon Permitted Failure Probability and Dynamic Priority

Zhishan Guo, Sudharsan Vaidhun, Luca Satinelli, Samsil Arefin, Jun Wang, Kecheng Yang

Mixed-Criticality Scheduling Upon Permitted Failure Probability and Dynamic Priority

Zhishan Guo, *Senior Member, IEEE*, Sudharsan Vaidhun, *Graduate Student Member, IEEE*,
Luca Satinelli, *Member, IEEE*, Samsil Arefin, Jun Wang, *Senior Member, IEEE*,
and Kecheng Yang, *Member, IEEE*

***Abstract*—Many safety-critical real-time systems are considered certified when they meet failure probability requirements with respect to the maximum permitted incidences of failure per hour. In this article, the mixed-criticality task model with multiple worst case execution time (WCET) estimations is extended to incorporate such system-level certification restrictions. A new parameter is added to each task, characterizing the distribution of WCET estimations—the likelihood of all jobs of a task finishing their executions within the less pessimistic WCET estimates. Efficient algorithms are derived for scheduling mixed-criticality systems represented using this model for both uniprocessor and multiprocessor platforms for independent tasks. Furthermore, a 0/1 covariance matrix is introduced to represent the failure dependency between tasks. An efficient algorithm is proposed to schedule such failure-dependent tasks. Experimental analyses show our new model and algorithm outperform current state-of-the-art mixed-criticality scheduling algorithms.**

***Index Terms*—Dependency, failure probability, mixed criticality, multicore real-time scheduling.**

I. Introduction

SAFETY-CRITICAL systems are as failure prone as any other system, and today's system certification approaches recognize this and specify *permitted system failure probabilities*. The underlying idea is to certify considering more realistic system models that account for any possible behavior, including faulty conditions, and the probability of these behaviors occurring. The gap that still exists is between such enhanced models and the current conservative deterministic analyses that tend to be pessimistic.

The worst-case execution time (WCET) abstraction plays a central role in the analysis of real-time systems. The WCET of a given piece of code upon a specified platform represents an upper bound to the duration of time needed to finish execution. Unfortunately, even when severe restrictions are placed upon the structure of the code (e.g., known loop bounds), it is still extremely difficult to determine the absolute WCET. An illustrative example is provided in [2], which demonstrates how the simple operation "`a = b + c`" on integer variables could take anywhere between 3 and 321 cycles upon a widely used modern CPU. The number of execution cycles highly depends upon factors such as the state of the cache when the operation occurs. WCET analysis has always been a very active and thriving area of research, and sophisticated timing analysis tools have been developed (see [3] for an excellent survey).

Traditional rigorous WCET analysis may lead to a result of much pessimism, and the occurrence of such WCET is extremely unlikely, unless under highly pathological circumstances. For instance, although a conservative tool would assign the "`a = b + c`" operation, a WCET bound of 321 cycles, a less conservative tool may assign it a much smaller WCET (e.g., 30) with the understanding that the bound may be violated on rare occasions under certain (presumably highly unlikely to occur) pathological conditions.

Mixed-Criticality Systems: The gap between the actual running time and WCET may be significantly large. Instead of completely wasting the processor capacities within the gap, recent research focused on implementing functionalities of different degrees of importance, or *criticalities*, upon a common platform, so that the less important tasks that may execute in these gaps under normal circumstances may be dropped in occasional situations where jobs of higher importance level execute beyond their estimated common case running time.

Much of the prior research on mixed-criticality scheduling (see [4] for a review) have focused upon the phenomenon that different tools for determining WCET bounds may be more or less conservative than one another, which results in multiple WCET estimations for each individual task (piece of code). Typically, in the two-criticality-level case, each task is designated as being of either higher (HI) or lower (LO) criticality, and two WCETs are specified for each HI-criticality task: 1) a LO-WCET determined by a less pessimistic tool and 2) a larger HI-WCET determined by a more conservative one, which is sometimes larger than the LO-WCET by several orders of magnitude. The scheduling objective is to determine a runtime scheduling strategy, which ensures that: 1) all jobs

Manuscript received May 30, 2020; revised September 17, 2020 and December 2, 2020; accepted January 16, 2021. Date of publication January 21, 2021; date of current version December 23, 2021. This work was supported in part by the National Science Foundation under Grant CNS-1850851 and Grant SPX-2028481. A preliminary version of this article with early results on uniprocessor was published in RTCSA 2015 [1]. This article was recommended by Associate Editor C.-L. Yang. *(Corresponding author: Zhishan Guo.)*

Zhishan Guo, Sudharsan Vaidhun, and Jun Wang are with the Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL 32816 USA (e-mail: zsguo@ucf.edu).

Luca Satinelli is with the Department of DTIM-LAPS, ONERA, 31055 Toulouse, France, and also with Department of DTIM-LAPS, Airbus Defence and Space, 82024 Taufkirchen, Germany.

Samsil Arefin is with the Intune Team, Microsoft New England Research and Development Center, Cambridge, MA 02142 USA.

Kecheng Yang is with the Department of Computer Science, Texas State University, San Marcos, TX 78666 USA.

of all tasks complete by their deadlines if each job completes upon executing for no more than its LO-WCET and 2) all jobs of tasks designated as being of HI criticality continue to complete by their deadlines (although the LO-criticality jobs may not) if any job requires execution for more than its LO-WCET (but no larger than its HI-WCET) to complete.

Under the current mixed-criticality model, it is assumed that *all* HI-criticality jobs may require executions up to their HI-WCETs in HI mode simultaneously. However, since WCET tools are normally quite pessimistic, LO-WCET is not very likely to be exceeded during runtime.

Example 1: Consider a system comprised of two *independent*[1] HI-criticality tasks τ_1 and τ_2, where each task is denoted by two utilization estimations $u_{\text{LO}} \leq u_{\text{HI}}$. The two tasks $\tau_1 = \{0.4, 0.6\}$ and $\tau_2 = \{0.3, 0.5\}$, represented by utilizations in different modes, are to be scheduled on a preemptive unit-speed uniprocessor. It is evident that this system cannot be scheduled correctly under the traditional model, since the HI-criticality utilization, at $(0.6 + 0.5)$, is greater than the processor capacity which is 1.

However, suppose that: 1) absolute certainty of correctness is not required; instead it is specified that the system failure probability should not exceed 10^{-6} per hour and 2) it is known that the timing analysis tools used to determine LO-criticality WCETs ensure that the likelihood of any job of a task exceeding its LO-WCET is no larger than 10^{-4} per hour.

Based on the task independence assumption, the probability of jobs from both tasks exceeding their LO-WCETs is $10^{-4} \times 10^{-4} = 10^{-8}$ per hour. Thus, we know that it is *safe* to ignore the case that both tasks simultaneously exceed their LO-WCETs. Hence, the system is *probabilistically feasible*, since the total remaining utilization will not exceed

$$\max\{0.4 + 0.3, 0.4 + 0.5, 0.6 + 0.3\} \;=\; 0.9 \;\leq\; 1.$$

Example 1 gives us an intuition that with the help of probabilistic analysis, we may be able to ignore some extremely unlikely cases, and come up with some *less pessimistic* schedulability analysis—if we have the prior knowledge that there will be at most a fixed number of HI-criticality tasks with execution exceptions per hour, then dropping of less important jobs may not be necessary at all.

In traditional MC models, each HI-criticality task is characterized by two WCETs, c_{LO} and c_{HI}, which could be derived with different timing analysis tools. By the level of pessimism and/or other properties in the timing analysis, such a tool usually provides a confidence for its resulting WCET estimates. *Confidence level* of the execution budget estimates is the likelihood that the estimate is true, given the information available. However, very few work on MC analysis have leveraged any information from the confidence of the provisioned WCET.

The existing MC analysis usually makes the most pessimistic assumption that *every* HI-criticality task may execute beyond its LO-WCET and reach its HI-WCET *simultaneously*. In real applications, the industry standards usually only require the expected probability of missing deadlines within a specified duration to be below some specified small value, as the deadline miss can be seen as a faulty condition. Instead, our work aims at leveraging probabilistic information from the timing analysis tools (i.e., confidence) to rule out the too pessimistic scenarios and to improve schedulability of the whole system under a probabilistic standard.

Our work also differs from most prior work on WCET analysis as follows. The existing timing analysis works usually analyze WCET for a task on a per-job basis; i.e., by focusing on the distribution of WCETs of jobs of a certain task [3]. When it comes to analyzing a series of consecutive jobs generated from the same task, the distribution is directly applied. It is usually assumed that: 1) all jobs of WCET of a certain task obey the same distribution (identically distributed) and 2) the WCET of a job is probabilistically drawn from the distribution with no dependence on other jobs of the same task (independence). While the independence assumption holds for the WCET, as we will see in the next section, it may not hold for the task execution time. For example, in many applications such as video frames processing, the execution times of processing consecutive frames of a certain video are usually dependent. However, the event that a certain task has ever over run its provisioned execution time in time intervals of a certain adequate large length (e.g., an hour) is independent from the scenario in other such intervals, and the probability of such event should be derived from the confidence of corresponding timing analysis tools only.

Contributions: In addition to the existing mixed-criticality task model, this work introduces a new parameter to each task that represents the distribution information about its WCET. This work aims to provide schedulability analysis to instances with this additional probability information, with respect to the given safety certification requirement of the whole system, which is the permitted system failure probability per hour.

We consider the scheduling of dual-criticality task systems upon both preemptive uniprocessor and multiprocessor platforms. As stated above, dual-criticality tasks are traditionally characterized with two WCET estimations—a LO-WCET and a larger HI-WCET. Our contributions are as follows.

1) We propose a supplement to current MC task models: an additional parameter for each HI-criticality task, denoting the *probability* of no job of this task exceeding its LO-WCET *within an hour* of execution.
2) We further generalize our notion of system behavior by allowing for the specification of a *permitted system failure probability per hour*, denoting an upper bound on the probability that the system may fail to meet its timing constraints during any hour of running.
3) We derive a novel scheduling algorithm (and an associated sufficient schedulability test) for a given MC task set and an allowed system failure probability on uniprocessor platforms. We seek to schedule the system such that the probability of failing to meet timing constraints during runtime is guaranteed to be no larger than the specified allowed system failure probability.

[1] Two events are independent if the occurence of one event does not have any impact on the other.

4) We further extend our uniprocessor scheduling technique to multiprocessor platforms by combining with the partitioned scheduling techniques.
5) While our initial scheduling technique focuses on independent task sets, we further introduce a covariance matrix in the system model to represent the failure dependencies between tasks in a task set, and we propose an efficient scheduling technique to schedule task sets with given failure dependencies.

We emphasize that our algorithm, in the two-criticality-level case, requires just one probabilistic parameter per task—the probability that the actual execution requirement will exceed the specified LO-WCET in an hour. We believe our scheduling algorithm is novel in that it is, to our knowledge, the first MC scheduling algorithm that makes scheduling decisions (e.g., when to trigger a mode switch) based not only on release times, deadlines, and WCETs but also on the probabilities drawn from probabilistic timing analysis (PTA) tools (see [5]–[7]).

Organization: Section II introduces the model and shows its advantage by a motivating example. Section III formally defines probabilistic schedulability and related concepts. In Section IV, we propose a clustering-based scheduling strategy, and the corresponding schedulability test, while Section V presents the multiprocessor scheduling algorithm and the corresponding schedulability test. In Section VI, we introduce the covariance matrix to represent the failure dependencies between the tasks and present the scheduling technique for such task sets, and finally, Section VII performs their experimental evaluations and comparisons. Section VIII elaborates the existing results, and Section IX concludes and suggests future work.

II. Model

We start out considering a workload model consisting of *independent implicit-deadline sporadic tasks*, where the deadline and the period of a task share the same value. In Section VI, we extend this model to allow for pairwise dependencies between tasks. Throughout this article, an integer model of time is assumed—all task periods are assumed to be nonnegative integers, and all job arrivals are assumed to occur at integer instants in time.

Before detailing our task model, a few statistical notions need to be introduced in order to clarify previous and next observations. Given a task τ_i, its probabilistic WCET (pWCET) estimate comes from a random variable (the WCET distribution), notably continuous distributions[2] denoted by $\mathcal{C}_i$. Equivalent representations for distributions are the probabilistic density functions (pdfs) $f_{\mathcal{C}_i}$, the cumulative distribution functions (CDFs) $F_{\mathcal{C}_i}$, and the complementary CDFs (CCDFs) $F'_{\mathcal{C}_i}$. In the following, calligraphic uppercase letters are used to refer to probabilistic distributions, while noncalligraphic letters are used for single value parameters.

The CCDF representation relates *confidence* to *probabilities*; indeed, from $F'_{\mathcal{C}_i}(c(\text{LO}))$, we have the probability of exceeding c_{LO}. The confidence is then for $c(\text{LO})$ being an upper bound to task execution time. The WCET threshold, simply named pWCET or WCET in the rest of this article, is a tuple $\langle c(\text{LO}), p(\text{LO})\rangle$, where the probability $p(\text{LO})$ sets the confidence (at the job level) of exceeding $c(\text{LO})$, $p(\text{LO}) = F'_{\mathcal{C}}(c(\text{LO})) = P(\mathcal{C} > c(\text{LO}))$. By decreasing the probability threshold $p(\text{LO})$, the confidence on the upper bounding worst case $c(\text{LO})$ increases.

Given the event A that a job exceeds its threshold and its probability of happening $p^A = P(\mathcal{C}_i > c(\text{LO}))$; given B the event that another job exceeds its threshold (in a different execution interval) with $p^B = P(\mathcal{C}_i > c(\text{LO}))$ its probability of happening. With separate jobs as well as separate execution intervals, and considering WCETs, the conditional probability $P(A|B)$ is equal to $P(A)$, thus the joint probability is

$$P(A, B) = P(A|B) \times P(B) = P(A) \times P(B) \tag{1}$$

due to the independence between WCETs. Projecting the per job probability threshold $p(\text{LO}) = F'_{\mathcal{C}_i}(c(\text{LO}))$ to 1-h task execution interval, we make use of the joint probability of all the exceeding threshold events within the 1-h interval. The joint probability is

$$1 - P(\mathcal{C}_i \leq c(\text{LO}), \mathcal{C}_i \leq c(\text{LO}), \mathcal{C}_i \leq c(\text{LO}), \ldots, \mathcal{C}_i \leq c(\text{LO})) \tag{2}$$

as the probability of at least one task job exceeding its thresholds $c(\text{LO})$. With full independence, the probability of exceeding threshold in 1 h would be at most $1 - F_{\mathcal{C}_i}(c(\text{LO})) \times \lfloor T_i/3600000 \rfloor$, with the task τ_i period T_i expressed in millisecond.

III. Probabilistic Schedulability

System Failure Probability F_s: In our model, an *allowed system failure probability F_S* is specified. It describes the permitted probability of the system failing to meet timing constraints during 1 h of execution.[3] F_S may be very close to zero (e.g., 10^{-12} for some safety critical avionics functionalities).[4]

Failure Probability: A *failure probability* parameter f_i is added to HI-criticality task τ_i, denoting the probability that the actual execution requirement of *any* job of the task exceeding $c_i(\text{LO})$ [but still below $c_i(\text{HI})$] in 1 h (i.e., the adequate long time interval we assumed in this article). f_i depends on a failure distribution $F_i(t)$ that describes the task τ_i probability of failure up to and including time t. Since $F_i(t)$ would refer to time (interval) and to task execution, it is going to be the one we computed for 1-h interval or any another interval (2). Thus, f_i can be directly derived from $F_{\mathcal{C}_i}$.[5]

[2] The timing analysis that makes use of the EVT, by definition provides continuous distributions as pWCET estimates [5]; they are then discretized, to ease their representation, by assigning them a discrete support.

[3] Failure probability is easily referable to failure rate, being careful at considering the failure rate as a probability.

[4] From DO-178B/C at the highest DAL—Level A, the acceptable failure rate is below 10^{-9}/h [8].

[5] It is possible to apply existing timing analysis tools to determine f_i – by monitoring executions of a piece of code for enough length, one may derive a stable pWCET, or may need to adapt EVT in case there are significant changes of execution time (to guarantee the safety of pWCET).

A HI-criticality task is represented by: $\tau_i = ([c_i(\text{LO}), c_i(\text{HI})], f_i, T_i, \chi_i)$, where T_i is the task period and $\chi_i \in \{\text{LO}, \text{HI}\}$ is the criticality level of the task; LO-criticality tasks continue to be represented with three parameters as before. This enhanced model is essentially asserting, for each HI-criticality task τ_i, within a time interval of 1 h, no job of τ_i has an execution greater than $c_i(\text{HI})$ and the probability of *any* job of τ_i having an execution greater than $c_i(\text{LO})$ is f_i—we would expect f_i to be a very small positive value. In our work, we assume $c_i(\text{HI})$ the deterministic WCET, $\langle c_i(\text{HI}), 0\rangle$, while $\langle c_i(\text{LO}), f_i > 0\rangle$ the pWCET with $c_i(\text{LO}) \leq c_i(\text{HI})$. Normally, we do not guarantee higher assurance for LO-criticality tasks (than HI-criticality ones), and thus only $c_i(\text{LO})$ is adopted for them. In the traditional MC model, $c_i(\text{LO})$ and $c_i(\text{HI})$ values are chosen from the WCET distribution. However, the corresponding probabilities, f_i and 0, respectively, are omitted from the task model. The proposed task model retains the probability information associated with the WCET estimates.

Definition 1 (MC Task Instance): An MC task instance I is composed of an MC task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ and a system failure requirement $F_S \in (0, 1)$. (Although F_S may be arbitrarily close to 0, $F_S = 0$ is not an acceptable value—"nothing is impossible.")

Let $n_{\text{HI}} \leq n$ denote the number of HI-criticality tasks in τ. We assume that the tasks are indexed such that the HI-criticality ones have lower indices, i.e., the HI-criticality tasks are indexed $1, 2, \ldots, n_{\text{HI}}$.

We seek to determine the *probabilistic schedulability* of any given MC task instance.

Definition 2 (Probabilistic Schedulability): An MC task set is *strongly probabilistic schedulable* by a scheduling strategy if it possesses the property that upon execution, the probability of missing any deadline is less than F_S. It is *weakly probabilistic schedulable* if the probability of missing any HI-criticality deadline is less than F_S. (In either case, all deadlines are met during system runs where no job exceeds its LO-WCET.)

That is, if a schedulability test returns strongly schedulable, then all jobs meet their deadlines with a probability no less than $1 - F_S$, while weakly schedulable only guarantees (with probability no less than $1 - F_S$) that HI-criticality jobs meet their deadlines. Moreover, similar to all MC works, for either strongly or weakly probabilistic schedulable, all deadlines are met when all jobs finish upon executing their LO-WCETs. Again, F_S comes from the natural need of some system certifications, while f_i is the additional information for each task that we need to derive from WCET estimations to achieve such probabilistic certification levels.

A. On the WCET Dependencies

In our model, the failure probability per hour of each task f_i represents the probability of *any* job of the task τ_i exceeding its LO-WCET. Thus, dependencies between tasks and task executions could have a strong impact on f_i. We hereby detail how we intend to cope with statistical dependence.

In [9], it has been shown that *neither* probabilistic dependence among random variables *nor* statistical dependence of data implies the loss of independence between tasks' pWCETs or WCET estimates. WCET is an upper bound to any execution time, which makes the important consequence on the independence between WCETs: jobs and tasks modeled with WCETs are independent because WCETs already embed dependence effects. Although both execution bounds (LO-WCET and HI-WCET) are so far called WCET estimations, the LO-WCET may also serve as an execution time upper bound, where dependence between tasks and within tasks needs to be more carefully accounted for (see [10] for the original definition of the MC task model).

Each MC task may generate an unbounded number of jobs. Since jobs generated from the same task set typically represent execution of the same piece of code, and such consecutive jobs could experience similar circumstances, in the definition of the failure probability f_i (of a task τ_i), we naturally assume dependence among jobs of the *same* task, i.e., it represents the likelihood that the required execution time of *any* job generated within an hour by τ_i will exceed $c_i(\text{LO})$. In [11] and [12], it has been shown that real safety-critical embedded systems have natural variability on the task execution time, thus it is reasonable to assume independence or extremal independence between jobs.

Concerning task dependencies, we can cope with the dependence by specifying the task pairwise dependence model. Assuming we are given a list of pairs (τ_i, τ_j) indicating that (WC)ETs of these two tasks may be dependent on each other. It means that the probability of them both exceeding their LO-WCET is no longer the product of their individual probabilities. By knowing $P(\mathcal{C}_i > c_i(\text{LO}), \mathcal{C}_j > c_j(\text{LO}))$, we are able to model (τ_i, τ_j) dependence including the execution time task dependencies in our framework, (Section IV-A). For many real-world systems, it is reasonable to assume that many (or most) task pairs do not have such dependencies to each other (although at the execution time level), since the limited impact of one task to another in a mixed-critical partitioned system. In Sections IV and V, we consider intertask independence (no pairwise failure-dependencies) to schedule the MC task set on uniprocessor and multiprcessor platforms, respectively. Furthermore, in Section VI, we introduce a covariance matrix to represent the pairwise failure dependencies and present the corresponding scheduling technique.

To summarize, intratask as well as intertask job dependences are covered by our model.

B. Utilization Costs

The notion of *additional utilization cost*, defined below, helps quantify the capacity that must be provisioned under HI-criticality mode.

Definition 3 (Additional Utilization Cost): The *additional utilization cost* of HI-criticality task τ_i is given by

$$\delta_i = (c_i(\text{HI}) - c_i(\text{LO}))/T_i. \tag{3}$$

Since we consider EDF schedulability instead of fixed-priority, we would like to know whether and *how likely* system utilization may exceed 1: 1) if it is extremely unlikely that the total HI-criticality utilization exceeds 1 (weakly probabilistic schedulable), we could assert a system that is infeasible in the traditional MC model to be probabilistic feasible and 2) if it is extremely unlikely that total system utilization exceeds 1 (strongly probabilistic schedulable), we could decide not to drop any LO-criticality task even if some HI-criticality tasks accidentally suffer from failures (that they require more execution time than expected).

Example 1 has shown an infeasible task set (under traditional MC scheduling) being weakly probabilistic schedulable under our model. As seen from the definitions, existing mixed-criticality systems are often analyzed under two modes—the HI mode and the LO mode, and mode switch is triggered when any HI-criticality job exceeds its LO-WCET without signaling finishing. Upon such a mode switch, deadlines of all LO-criticality jobs will no longer be guaranteed. A natural questions arises—is such sacrifice (dropping *all* LO-criticality jobs) necessary whenever a HI-criticality job requires execution for more than its LO-WCET? The following example illustrates the potential benefits in terms of enhanced schedulability of the proposed probabilistic MC model.

Example 2: Consider a system composed of the three independent MC tasks that $\tau_1 = \{[2, 3], 0.1, 5, \text{HI}\}$, $\tau_2 = \{[3, 4], 0.05, 10, \text{HI}\}$, and $\tau_3 = \{[1, 1], 10, \text{LO}\}$, to be scheduled on a preemptive uniprocessor, with desired system failure probability threshold of $F_S = 0.01$.

Since HI-utilization of the system is $u_{\text{HI}} = 3/5 + 4/10 = 1$, any deterministic MC scheduling algorithm will prioritize τ_1 and τ_2 over the LO-criticality task τ_3, and drop τ_3 if *any* HI-criticality job exceeds its LO-WCET.

With the additional probability information provided in our richer model, however, more sophisticated scheduling and analysis can be done. Recalling from the definition of f_i, τ_1 has a probability of no larger than 0.1 to exceed a 2-unit execution within an hour, while the probability of any job in τ_2 exceeding a 3-unit execution within an hour is 0.05. Under the task-level independence assumption, the probability of jobs from both HI-criticality tasks requiring more than their LO-WCETs in an hour $[P(x_1 = x_2 = 1) = P(x_1 = 1)\times P(x_2 = 1) = 0.1\times 0.05 = 0.005]$ is smaller than F_S.[6]

Hence, in the schedulability test, we do not need to consider the case that *both* HI-criticality tasks exceed their LO-WCETs simultaneously. Moreover, either one of them exceeding its LO-WCET will not result in an overutilized system—an HI-criticality "server" with budget 0.2 and period 1 can be added to provide the additional capacity (over and above the LO-WCET amount). This server will be scheduled and executed as a virtual task, and both HI-criticality tasks may run on the server. The additional budget of 0.2 is sufficient to handle either $\delta_1 = 0.2$ or $\delta_2 = 0.1$, which is necessary when one of the HI tasks exceed their LO-WCET. The server needs not provide $\delta_1 + \delta_2 = 0.3$ budget, since the probability of such an event is less than F_s.

The total system utilization thus provisioned for the HI-criticality tasks is $2/5+3/10+0.2/1 = 0.9$; upon provisioning an additional utilization of $1/10 = 0.1$ for the LO-criticality task τ_3, the total utilization becomes 1. Thus, under any optimal uniprocessor scheduling strategy, e.g., EDF, the failure (any deadline miss) rate of the system in any hour will be no greater than F_S, and the MC instance is *strongly probabilistic schedulable* under this scheduling strategy (EDF plus the HI-criticality server) for the specified threshold F_S.

IV. Scheduling Strategy

A. LFF-Clustering Algorithm

In this section, we present our strategy for scheduling independent preemptive MC task instances, by combining HI-criticality tasks into clusters intelligently, and provide a sufficient schedulability test for it. Consider what we have done in Example 2 above. We essentially: 1) conceptually combined the HI-criticality tasks τ_1 and τ_2 into a single cluster, provisioning an additional server into the system to accommodate their possible occasional HI-mode behaviors (execution beyond their LO-WCETs) and 2) performed two EDF schedulability tests: one considering only HI-criticality tasks (with LO-WCETs) and the server, and the other also considering the LO-criticality task (τ_3). Since both tests succeed, we declare *strongly probabilistic schedulable* for the given instance; we would have declared *weakly probabilistic schedulable* if the second schedulability test had failed while the first one succeeded.

The technique that was illustrated in Example 2 forms the basis of the scheduling strategy that we derive in this section. To obtain a good upper bound to HI-criticality utilization of the system, we combine tasks into *clusters*—suppose that the n_{HI} HI-criticality tasks have been partitioned into M clusters $G_1, G_2, \ldots, G_M$, and let $y_i \in \{1, 2, \ldots, M\}$ denote to which the cluster (number) task τ_i is assigned.

Definition 4 (Failure Probability of a Cluster): Failure of a cluster G_m is defined as jobs generated by more than one tasks in a single cluster exceeding their LO-WCETs within an hour. The probability of a failure occurring in cluster m is denoted as g_m and is given by

$$g_m \overset{\text{def}}{=} 1 - \prod_{i|y_i=m} (1 - f_i) - \sum_{j|y_j=m} f_j \frac{\prod_{i|y_i=m}(1 - f_i)}{1 - f_j} \tag{4}$$

where the second term of right-hand side is the probability of no task (in the cluster) exceeding its LO-WCET, and the last term represents the probability of exactly one of the tasks exceeding its LO-WCET in an hour.

Lemma 1: If $g_m < F_S/M$ holds for every cluster G_m, then the probability of having no failure in all clusters is greater than $(1 - F_S)$.

Proof: Since clusters do not overlap with each other (each HI-criticality task belongs to a single cluster) and thus are independent of each other, the probability of having no failure in all clusters is given by the product of each cluster being failure free, which is: $\prod_{m=1}^{M}(1 - g_m) > \prod_{m=1}^{M}(1 - F_S/M) = (1 - F_S/M)^M \geq 1 - F_S$ (from the binomial theorem). ■

Lemma 1 provides a safe *failure threshold F_S/M* for each cluster, i.e., the rule for forming *clusters* is $g_m < F_S/M$, where M is the current number of clusters.

[6] In general, we cannot simply ignore an event when its failure probability is below F_S. Instead, we do not need to consider a set of events only when the *sum* of their failure probability is below F_S. More details on this can be found in Section III.

Algorithm 1: Algorithm LFF-Clustering

```
Input: $F_S$, $\{f_i\}_{i=1}^{n_{\mathrm{HI}}}$, $\{\delta_i\}_{i=1}^{n_{\mathrm{HI}}}$
Output: maximum total additional utilization cost $\Delta$
begin
    Sort the tasks in non-increasing order of $\delta_i$;
    $m \leftarrow 1$, $M \leftarrow n_{\mathrm{HI}}$, $y_i \leftarrow 0$ for $i = 1, ..., n$;
    while $\prod_{i=1}^{n_{\mathrm{HI}}} y_i = 0$ (an unassigned task exists) do
        $\Delta_m \leftarrow 0$ (additional utilization of each cluster);
        for $i \leftarrow 1$ to $n_{\mathrm{HI}}$ do
            if $y_i > 0$:  continue;
            $y_i \leftarrow m$, $M \leftarrow M - 1$;
            if $g_m \geq F_S/(M+m)$:  $y_i \leftarrow 0$, $M \leftarrow M + 1$;
        end
        $\Delta_m \leftarrow \max_{i|y_i=m} \delta_i$; $m \leftarrow m + 1$, $M \leftarrow M + 1$;
    end
    return $\sum_{m=1}^{M} \Delta_M$;
end
```

The *additional utilization cost of a cluster G_m* is defined to be equal to the additional utilization cost (δ_i) of the task within the cluster with the largest δ_i value, i.e.,

$$\Delta_m \overset{\mathrm{def}}{=} \max_{i|\tau_i \in G_m} \delta_i. \tag{5}$$

The *total system additional utilization cost* is given by the sum of additional utilization cost of all M clusters

$$\Delta \overset{\mathrm{def}}{=} \sum_{m=1}^{M} \Delta_m. \tag{6}$$

A critical observation is that if a task τ_i with additional utilization cost δ_i has been assigned to a cluster, assigning any other task τ_j with $\delta_j \leq \delta_i$ to the cluster will not increase the additional utilization cost. To minimize the total additional utilization cost of the entire task set, we, therefore, greedily expand existing clusters with tasks of larger additional utilization cost while ensuring that the relationship $g_m < F_S/M$ continues to hold, which leads to the largest fit first (LFF)-clustering algorithm.

This algorithm greedily expands each existing cluster with unassigned tasks while the condition $g_m < F_S/M$ holds; while a new cluster is created only if it is not possible to assign a task to any current cluster without violating the condition ($g_m < F_S/M$).

Remark 1: Similar to what has been done in [13] and [14], we may achieve a precise distribution to the total utilization of all tasks by applying the *convolution* operation "⊗," which results in an exponential [$O(2^{n_{\mathrm{HI}}})$, to be precise] running time (see Appendix B). The sufficient schedulability test based on the LFF-Clustering algorithm runs in $O(n_{\mathrm{HI}}^2)$ time, where n_{HI} is the number of HI-criticality tasks.

Remark 2: In the case that *all tasks share the same f_i value*, the schedulability test based on LFF-clustering becomes *necessary and sufficient*.

Runtime Strategy: During execution, an HI-criticality *server* τ_s with utilization Δ and a period of 1 tick is added to the task system. The server is represented as $\tau_s = \{\Delta, 1, \mathrm{HI}\}$.

We need the server period as 1 tick because the mechanism and the analysis will not work if there is release or deadline within a server period. At any time instant that the server is executing, if there are active[7] HI-criticality jobs, they are executed following the earliest deadline first policy; if not, then the current server job is dropped.[8] All jobs including the server are scheduled and executed in the EDF order, and a job is dropped at its deadline if it is not completed by then.

Although we introduce a server task with period of 1, preemption does not necessarily happen that often. The goal of the sever task with utilization Δ is to preserve a "bandwidth" of at least Δ for HI-criticality jobs if the HI-criticality ready queue is not empty. There are three situations to be considered as follows.

Situation 1: The job with the earliest deadline is an HI-criticality job. In this situation, we execute the HI-criticality job with 100% processor share, and no more preemption is incurred by the server.

Situation 2: The job with the earliest deadline is an LO-criticality job *and* the HI-criticality ready queue is empty. In this situation, we execute the LO-criticality job with 100% processor share, and hence, there is no additional preemption in this situation either.

Situation 3: The job with the earliest deadline is an LO-criticality job *and* the HI-criticality ready queue is *not* empty. In this situation, we want to preserve a processor share of Δ for HI-criticality jobs and to execute the LO-criticality ones with the rest $1 - \Delta$ of the processor capacity. Therefore, the server creates preemptions every time unit.

That is, only in situation 3, our algorithm "introduces" extra preemptions due to the server scheme, and normal EDF scheduling is applied in other cases. One may claim that such server allocation scheme may result in more preemptions than the approaches where the server capacity is only used for overruns. Actually this is because that the goal here is trying *not to drop* LO-criticality tasks even when a few HI-criticality ones exceed their LO-WCETs. Thus, in order to guarantee HI-deadline being always met, we have to make certain use of the server even when no HI-criticality behavior is detected – simply taking "precautions." Alternative way such as assigning HI-criticality jobs virtual deadlines may lead to fewer preemptions, at a cost of losing the performance of schedulability ratio (see Section VII).

B. Schedulability Test

It is evident that for *strongly probabilistic schedulable* (i.e., to ensure that the probability of missing *any* deadline is no larger than the specified system failure probability F_S – see Definition 2), it is (necessary and) sufficient that $(\sum_{i=1}^{n} c_i(\mathrm{LO})/T_i + \Delta)$ must be no larger than the capacity of the processor (which is 1).

[7] A job is *active* if it is released and incomplete at that time instant.

[8] Since an integer model of time is assumed (i.e., all task periods are integers and all job arrivals occur at integer instants in time), and the server has a period of 1, it is safe to drop the current job of the server if there are no active HI-criticality jobs since there can be no HI-criticality job releases in the current period of the server.

```
Algorithm 2: Schedulability Test pMC
Input: $\tau, F_S$
Output: schedulability
begin
    Calculate $\delta_i$ values for all HI-criticality tasks in $\tau$;
    $u_{\text{LO}} \leftarrow \sum_{i=1}^{n} c_i(\text{LO})/T_i$;
    $u'_{\text{LO}} \leftarrow \sum_{i|\chi_i=\text{HI}} c_i(\text{LO})/T_i$;
    $\Delta \leftarrow$ LFF-Clustering($F_S$, $\{f_i\}_{i=1}^{n_{\text{HI}}}$, $\{\delta_i\}_{i=1}^{n_{\text{HI}}}$);
    if $u_{\text{LO}} + \Delta \leq 1$ then
        return strongly probabilistic schedulable;
    else if $u'_{\text{LO}} + \Delta \leq 1,\ \Delta \cdot (1 - u'_{\text{LO}}) + u_{\text{LO}} \leq 1$ then
        return weakly probabilistic schedulable;
    return unknown;
end
```

For *weakly probabilistic schedulable* (i.e., to ensure that the probability of missing any HI-*criticality* deadline is no larger than F_S – again, see Definition 2), it is necessary that $(\sum_{i|\chi_i=\text{HI}} c_i(\text{LO})/T_i + \Delta)$ must be no larger than 1 as well. The following theorem helps establish a sufficient condition for ensuring weakly probabilistic schedulable.

Theorem 1: If no job exceeds its LO-*WCET, then no deadline is missed if*

$$\Delta \cdot \left(1 - \sum_{i|\chi_i=\text{HI}} \frac{c_i(\text{LO})}{T_i}\right) + \sum_{i=1}^{n} \frac{c_i(\text{LO})}{T_i} \leq 1. \qquad (7)$$

Proof: Refer to the conference version [1]. ■

Theorem 1 yields the schedulability test pMC (Algorithm 2), while the following theorem establishes its correctness.

Theorem 2: The schedulability test pMC is *sufficient* in the following sense. If it returns *strongly probabilistic schedulable*, the probability of any task missing its deadline is no greater than F_S; and if it returns *weakly probabilistic schedulable*, the probability of any HI-criticality task missing its deadline is no greater than F_S, and no deadline is missed when all jobs finish upon execution of their LO-WCETs.

Proof: Refer to the conference version [1]. ■

The schedulability test pMC returns *strongly probabilistic schedulable* if we are able to schedule the system such that the probability of missing any deadline is at most the specified threshold F_S, or *weakly probabilistic schedulable* if we are able to schedule the system such that the probability of missing any HI-criticality deadline is at most F_S. We will then use EDF to schedule and execute the task set with LO-WCETs and the additional server task $\tau_s = \{\Delta, 1, \text{HI}\}$.

In the case that the schedulability test pMC returns *unknown*, we are not able to schedule the system using the proposed probabilistic analysis technique. Normally, it is because either the safety requirement of the system is too high (i.e., the threshold F_S is too small), or the WCET estimations are not precise enough for HI-criticality tasks [i.e., the f_i values are not small enough compared to F_S (and n_{HI}), and/or the $c_i(\text{LO})$ values are not differentiable enough against $c_i(\text{HI})$ values].

V. Multiprocessor Case

Multiprocessor devices are becoming more and more popular, while it is becoming more efficient to schedule real-time tasks in multiprocessor platforms to achieve better throughput. Considering the pragmatic application of applying probabilistic scheduling, implementing a similar technique in multiprocessors will dissipate the pessimistic assumption of the existing scheduling mechanism and improve the resource efficiency. This section proposes a multiprocessor scheduling technique of MC tasks considering the failure probability based on an partitioned-based approach.

The partitioned-based scheduling of implicit-deadline sporadic task system can be converted into a bin-packing problem [15]. Hence, each processor is modeled as a bin of capacity one, and each task τ_i has the capacity of size u_i (its utilization). As bin packing is NP-hard [15], heuristics can be applied for solving the problem. As our system model considers MC tasks with failure probability, we need to modify the task sets to fit into a traditional bin packing heuristics. Here, we briefly discuss three most-common heuristics [first fit (FF), best fit (BF), and worst fit (WF)] and then present an algorithm to schedule our system model in multiprocessor environments.

Different Allocation Heuristics: For partitioned scheduling, most common heuristics are FF, BF, and WF. Note that reasonable allocation decreasing (RAD) algorithms are proven to provide optimal utilization bound [16]. Hence, we adapt such approach, and thus, the three algorithms discussed above would have been called FF decreasing (FFD), BF decreasing (BFD), and WF decreasing (WFD) partitioned algorithms.

Task Allocation: To schedule tasks in our proposed system model in multiprocessor platforms, we present a slightly different scheduling approach than the one for uniprocessor in Section IV. Initially, all the tasks are grouped into different clusters using the same LFF-clustering algorithm presented before. Then, we schedule the clusters on different processors by using different RAD partitioned heuristics. Note that for every cluster, there is an additional utilization cost Δ_m, which is also needed to be allocated in case any task of the cluster exceeds its LO-criticality WCET. For every processor, we need to allocate a server that has a utilization equal to the sum of additional utilization cost Δ_m of all the clusters allocated in that processor. For example, while considering scheduling three clusters on two processors, if the Clusters 1 and 2 are allocated on Processor 1 and the Cluster 3 is allocated on Processor 2, then we also need to allocate a server with utilization $(\Delta_1 + \Delta_2)$ to Processor 1, and another server with utilization Δ_3 to Processor 2. In short, while applying the partitioned heuristic, we need to accommodate the demand of server utilization as well. Specifically, the LO-criticality tasks are considered for allocation once all the HI-criticality clusters and their Δs are allocated, using the same partitioning techniques used for the HI tasks.

A. Schedulability Analysis

The schedulability of the task set is determined in two different steps. First, we have to check whether the tasks can be properly allocated to the available processors. Upon successful allocation, we need to check whether the correctness of each uniprocessor MC task system can be guaranteed in runtime even under worse conditions. In each step, there is a possibility that either only HI tasks or all the tasks are allocated/schedulable. Based on the allocation, the task sets can be strongly allocated or weakly allocated. If all the HI-task clusters, Δs, and LO-tasks are allocated, then we call it strongly allocated task set. If only the clusters and Δs are allocated properly but not the LO-tasks, then we call it weakly allocated task set. If neither, then the task sets cannot be considered for further schedulability test. Upon successful allocation, we need to check the schedulability of the allocated task set on all the processors. The schedulability test is done by following the similar technique used for the uniprocessor scheduling.

Before going into the schedulability conditions, it is necessary to introduce the parameter α and β. α is the utilization factor of a task set, i.e., the maximum utilization among all tasks. β is the maximum number of tasks of α, which fit into one processor under EDF scheduling. β can be expressed as a function of α

$$\beta = \lfloor 1/\alpha \rfloor. \tag{8}$$

López *et al.* [16] proved that for multiprocessor partitioned scheduling using EDF, FFD, BFD, and WFD algorithms provide the optimal upper bound, which is the following.

Lemma 2 [16]: Let $U(N, \alpha)$ denote the utilization bound to schedule N tasks using RAD algorithms on m processors and α represent the utilization factor for the task set, then $U(N, \alpha) = (\beta N + 1/\beta + 1)$ when $m > \beta N$.

Task Allocation Conditions: For the HI-tasks we need to allocate each cluster with its Δ_m in the same processor as in the partitioned scheduling, a task is always needed to be executed on the same processor, which it was initially allocated. Let us assume there are M clusters with utilization $UC_1, UC_2, \ldots, UC_M$ with the corresponding delta values as $\Delta_1, \Delta_2, \ldots, \Delta_M$. As each cluster is to be allocated to a processor with its corresponding Δ, let us define the HI mode utilization factor α_h and the overall utilization factor α_s as follows:

$$\alpha_h = \max_{i \in 1,2,\ldots,M}(UC_i + \Delta_i); \quad \alpha_s = \max(\alpha_h, \alpha_l) \tag{9}$$

where α_l is the utilization factor for the LO-tasks.

Furthermore, corresponding β values can be defined

$$\beta_h = \lfloor 1/\alpha_h \rfloor; \;\; \beta_s = \lfloor 1/\alpha_s \rfloor. \tag{10}$$

Theorem 3: All the HI-criticality clusters along with their server allocation (Δ) and all the LO-criticality tasks can be allocated (i.e., strongly allocated task-set) to K processors if the following two conditions hold:

$$K > \beta_s(M + n); \quad U_s \le \frac{\beta_s(M + n) + 1}{\beta_s + 1} \tag{11}$$

where U_s is the sum of the utilization of all clusters, their Δs and LO-criticality tasks.

Proof: Here, each cluster and LO-criticality tasks can be seen as a single entity with specific utilization demand. The total number of entity here is $(M+n)$. Thus, according to Lemma 2, the maximum utilization bound can be $[(\beta_s(M + n) + 1)/(\beta_s + 1)]$. So for a successful allocation, the system utilization U_s must be no greater than the utilization bound. ■

Algorithm 3: pMCMP Algorithm

```
Data: Allocation of HI-criticality $\Delta_i$s and LO-criticality task-set
      $\tau_i$ on each processor $\kappa_i$
Result: The schedulability of the task set
if $\forall\kappa_i$, pMC returns strongly-schedulable then
    return strongly-schedulable;
else if $\forall\kappa_i$, pMC returns weakly-schedulable then
    return weakly-schedulable;
else
    return non-determined;
end
```

Theorem 4: All the HI-criticality clusters along with their server allocation (Δ) can be allocated (i.e., weakly allocated task-set) to K processor if the following two conditions hold:

$$K > \beta_h M; \quad U = \frac{\beta_h M + 1}{\beta_h + 1}. \tag{12}$$

Proof: The proof is very similar to Theorem 3 (omitted). ■

Partition Approach and Its Correctness: Upon successful allocation (either strongly or weakly allocated), the schedulability of the task set can be determined by running the probabilistic mixed-criticality on multiprocessor (pMCMP) algorithm (presented in Algorithm 3). The pMCMP algorithm basically uses the pMC algorithm presented in Section IV on all the processors to check the schedulability. The result of pMCMP can be strongly schedulable, weakly schedulable, or nondetermined. If a task set is only weakly allocated on the available processors, the task set can never be considered as strongly schedulable.

Theorem 5: Algorithm pMCMP is correct. That is, if pMCMP returns strongly probabilistic schedulable, the probability of any task missing its deadline is no greater than F_S; while if pMCMP returns weakly probabilistic schedulable, the probability of any HI-criticality task missing its deadline is no greater than F_S, and no deadline is missed when all jobs finish upon execution of their LO-WCETs.

Proof: After a successful allocation, each processor has a specific set of Δ's and LO-criticality tasks assigned for scheduling. Let $\overline{UC}_1, \overline{UC}_2, \ldots, \overline{UC}_M$ denote the number of clusters assigned to K processors (note that there are total M number of clusters). Hence, each processor assignment can be seen as a subset problem of uniprocessor scheduling presented in [1].

For an arbitrary processor i, similar to the proof of [1, eq. (5.2)], the failure probability of processor i is no greater that $(\overline{UC}_i \times F_S)/M$. As the tasks are independent, the total failure probability of the system is no greater than $\sum_{i=1}^{M}(\overline{UC}_i \times F_S)/M = F_S$. ■

VI. Considering Failure Dependency

In previous sections, we have considered only independent tasks, i.e., the failure of a task τ_i is independent to whether another task fails or not. In other words, whether a HI task fails to complete within its LO-WCET budget does not affect the

TABLE I
Covariance Matrix of a Set of Eight Tasks

	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8
τ_1	0	0	0	1	0	0	0
τ_2	-	1	0	0	1	0	0
τ_3	-	-	0	0	1	1	0
τ_4	-	-	-	0	0	0	0
τ_5	-	-	-	-	0	0	1
τ_6	-	-	-	-	-	0	0
τ_7	-	-	-	-	-	-	0

completion of other HI tasks. On the contrary, most existing work in MC scheduling assumed that all HI tasks may exceed their LO-WCET budgets at the same time. That means failure probabilities of all HI tasks are dependent on each other, which is rather a pessimistic assumption in certain scenarios. However, in practical systems, not every failure probability is independent. It is possible that while most of the HI tasks are independent, a certain amount of task is directly dependent to each other with respect to their probability of exceeding LO-WCET, i.e., once a HI task exceeds the LO-WCET, other dependent tasks may also exceed their LO-WCET budgets, *regardless of their own failure probability*. Techniques such as fault tree analysis [17], [18] can be used to analyze tasks for failure dependency. With the dependency information provided as input, we propose a covariance matrix representation and propose a graph-coloring-based partitioning approach to handle such a scenario.

A. Covariance Matrix

We introduce a covariance matrix to represent the dependencies between each pair of tasks regarding their potential failures to complete before LO-WCETs. In Table I, a sample covariance matrix is shown for eight HI-criticality tasks. Here, the covariance matrix is binary, where for each item, 0 represents the independence between two tasks while 1 represents that there is dependency on failure probability between two tasks. Note that 1 does not mean fully dependant, while it is a safe (although pessimistic) measurement in terms of schedulability guarantees. As a result, if there is a dependency between two tasks, we assume that upon exceeding the LO-WCET budget of one task, the other dependent task/s will also exceed their LO-WCET budget simultaneously.

Remark 2: Failure dependency is not statistical dependence.

Definition 5 (Failure-Dependent Tasks): When scheduling a pair of tasks on the same processor, if one of those tasks exceeds its LO-WCET and, the other one's failure likelihood becomes larger than the given value, or the other way around, then two tasks are considered failure dependent.

In other words, it is safe to assume that both of the tasks exceed their LO-WCET simultaneously. Taking the task set shown in Table I as an example, τ_2 and τ_3 are failure dependent but τ_1 and τ_2 are not. Note that two tasks are failure dependent only if those are scheduled on the same processor. Upon scheduling on different processors, those tasks can be executed independently since we conduct partitioned scheduling and assume isolation between processors.

B. Task Isolation Using Graph Model

Similar to the clustering problem without covariance, finding the optimal clustering for the new problem is also NP-Hard. Thus, we propose a heuristic to find an efficient solution to the problem. From the covariance matrix, we can visualize a task set as a graph problem. We can assume the tasks as a node of the graph and the failure dependencies as edges between the nodes, i.e., if there is a 1 between two tasks, we can consider an edge between those two nodes (tasks). Note that if the graph is fully connected, the problem will become a traditional MC scheduling problem as in our strategy, we will need to create a cluster for each task. In practical scenarios, the graph is assumed to be a disconnected graph as there can be both independent and failure dependent tasks in a system. Hence, there will be multiple islands in the graph, which consist of interdependent tasks.

Definition 6 (Transitive Failure Dependency): If two tasks are not directly dependent but have a common failure-dependent task, then we call the failure dependency between the first two tasks transitive failure dependency.

For example, τ_1 and τ_8 in Table I are not directly failure-dependent, but they both are dependent with τ_5. Hence, if we partition these three tasks into a same processor, they may exceed their LO-WCET budgets simultaneously. However, if we schedule τ_5 in a separate processor, τ_1 and τ_8 will have no failure dependency and can act as independent task pairs.

Upon transforming the covariance matrix into a graph, we apply our clustering heuristics. In the aforementioned LFF-clustering algorithm, we sorted all the tasks in descending order based on their additional utilization δ_i value and we keep adding the tasks one by one to clusters until Lemma 1 is not violated. While scheduling the task set including failure-dependent tasks, we isolate the tasks of each island into m (number of processors) number of groups in a way such that there are no failure-dependent tasks in any group. By doing that we ensure that no two interdependent tasks are grouped into a single processor.

C. Isolating Tasks of Each Island

Once we convert the task set into a graph with different islands, m number of groups of independent tasks are created. One can visualize this problem as a m-coloring graph problem, where the nodes of a graph are colored with at most m number of colors with no two nodes of same color adjacent to each other (i.e., there is no edge between them). If we can properly color the graph with m or less number of colors, then we can easily allocate nodes with the same color into the same processor. Unfortunately, the m-coloring problem is an NP-Complete problem [19]. As a result, we will need an approximate algorithm to solve such problems. Furthermore, it may not always be possible to color the subgraph in each island with m colors by using the approximate algorithm (even by using an optimal algorithm). Hence, in this section, we propose a modified approximate algorithm to color each subgraph with m colors. To accomplish this goal, we adapt a modified greedy coloring algorithm. As our base greedy coloring algorithm, we use the

Algorithm 4: m-Coloring Algorithm

```
Data: $G = \{V, E\}$, $m$
Result: $m - colored$ graph
Sort all nodes $V_i$s in non-increasing order or their degrees;
for $i \leftarrow 1$ to $m$ do
    /* All nodes are colored                          */
    if $\forall V_i$ is colored then
        return;
    end
    /* If at last processor, there exists
       some connected nodes                           */
    if $(i == m)$ and $(\exists E_i)$ then
        forall Connected subgraph do
            $V_{new} \leftarrow \{connected\_nodes\}$;
            $V_{new}.u_i^{\text{LO}} \leftarrow \{connected\_nodes\}.u_i^{\text{LO}}$;
            $V_{new}.\delta_i \leftarrow \{connected\_nodes\}.\delta_i$;
        end
    else
        Color the nodes following LF [20] Algorithm;
    end
end
```

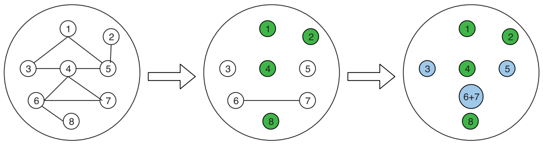


Fig. 1. Graph coloring with $m = 2$.

Welsh Powell Algorithm [20] [also known as largest-first (LF) coloring algorithm].

In LF coloring, node with the highest degree[9] will be colored first, and then the adjacent nodes will be colored and corresponding edges are deleted. By doing this, we remove the dependencies of other nodes with the colored nodes. At any step, if the sum of the utilizations exceeds one, we stop coloring for that particular color and leave that node for the next coloring iteration. However, when we already use $m - 1$ colors and only one color is left to use, we need to contract (merge) the remaining failure-dependent tasks. When at the last processor, there are still some nodes which are connected (failure dependent), we contract those nodes in a single node and use u_i^{LO} and δ_i of the node as the sum of the corresponding values of all the connected nodes. The steps of the coloring technique are shown in Algorithm 4 and further demonstrated in Example 3.

Example 3: In Fig. 1, the tasks of one island is shown. Here, we need to allocate the tasks on two processors, i.e., we have to color the graph with two colors. To do this, we first take the node with the highest degree (τ_4 with degree 4) and color it with green. Then, we color the nonadjacent nodes of τ_4 (i.e., τ_1, τ_2, and τ_8) with green and remove the edges of those nodes. We can no longer use green in this graph. Now, we have one processor (color) left but there are still two tasks τ_6 and τ_7, which are failure dependent to each other. So we merge these two tasks and all the nodes become independent. Finally, we color all the nodes with blue (τ_3, τ_5, and τ_{6+7}).

Algorithm 5: Task Allocation Algorithm With Covariance

```
Data: $F_S$, $\{f_i\}_{i=1}^{nHI}$, $\{u_i^{\text{LO}}\}_{i=1}^{n}$ $\{\delta_i\}_{i=1}^{n}$, covariance matrix
Result: Task allocation result
Run DFS on covariance matrix and get islands $\{I_j\}_{j=1}^{l}$;
Color the nodes of each island using Algorithm 4;
Sort $\forall \tau_i \in \forall I_j$ in descending order w.r.to $\delta_i$;
Sort $\forall I_i$ based on $\max(\delta_i) \in I_i$;
$\mathcal{K} = \{\kappa_1, \kappa_2, \ldots, \kappa_K\}$;
forall multi-task islands $I_i$ do
    $allocated = \phi$;
    forall $\forall \tau_j \in I_i$ do
        allocate $\tau_j$ into $\kappa_k \in \{\mathcal{K} - allocated\}$ following WF and
          update $\Delta_k$;
        $allocated = allocated \cup \kappa_k$
    end
end
if all LO tasks are allocated using WF then
    return $strongly - allocated$;
end
return $weakly - allocated$;
```

D. Task Allocation and Scheduling

Given a task set, we first run depth-first-search (DFS) over the covariance matrix and convert them to different number of islands. Then, we use Algorithm 4 to color the nodes of each island with at most m number of distinct colors. Then, the nodes in each island are sorted in descending order with respect to their δ_i values and the islands themselves are then sorted in descending order based on $\max(\delta_i)$ values. HI-criticality tasks are allocated by the following rules.

1) Assign tasks of each color of each island to a distinct processor with the WF heuristics on processor capacity.
2) While assigning a task to a processor, we create a cluster following the LFF-Clustering algorithm. First, we keep adding the same colored tasks in an island to the existing task/s assigned to that processor. If we cannot assign the new task to an existing cluster, we create a new cluster. Once all nodes of the same color are allocated, we allocate the next color tasks to a different processor.
3) Every time we add a new task to a processor, we update the Δ value of the processor.
4) Once all islands with multiple tasks are assigned, we assign the remaining single-task islands by following the WF partitioning heuristic.

If all the tasks are allocated, the task set becomes strongly allocated, while only the allocation of HI-criticality tasks results in a weakly allocation. Similar to the multiprocessor case with no failure-dependent tasks, successful allocation does not imply schedulability of the allocated task sets. If a task set is not strongly allocated, then the task set cannot be strongly schedulable, because of the lack of guarantees to the LO-tasks. Upon successful allocation, either strong or weak, we run the pMCMP algorithm to check the schedulability of the task set. Algorithm 5 details the task allocation procedure.

VII. Schedulability Experiments

In this section, we present extensive experimental evaluations to show the performance of our proposed algorithms. We present our results in two different categories. First, we

[9] Degree of a node is the number of edges connected to the node.

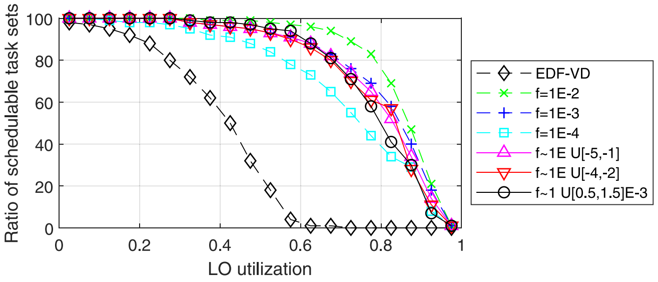


Fig. 2. Schedulability ratio comparison of EDF-VD and pMC, where HI utilization varies from 0.9 to 1 uniformly.

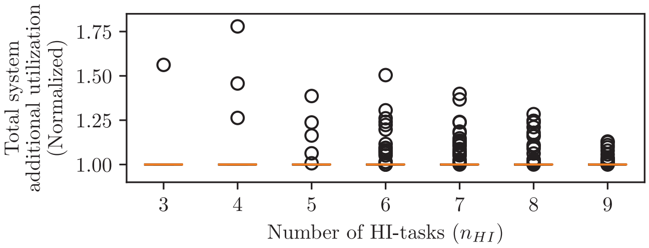


Fig. 3. Comparison of additional utilization Δ calculated by LFF-clustering normalized by the precise value.

present the schedulability result for uniprocessor platforms to show the efficiency of our algorithm. We performed simulation for different constraints and also compared with other state-of-the-art MC scheduler. For the next part, we present the schedulability performance on multiprocessor platforms. We have performed a number of experiments by varying different important factors to observe the efficiency of our algorithm.

A. *Results for Uniprocessor Platforms*

We have conducted schedulability tests on randomly generated task systems, comparing our proposed method with the existing one. The objective was to demonstrate the benefits of our model: by adding a probability estimation f_i to each task, our algorithm may successfully schedule (return *probabilistically correct* or *partial probabilistically correct*) many task sets that are unschedulable according to existing MC-scheduling algorithms, e.g., the EDF-VD algorithm [21].

Since this is the first work that combines pWCET and schedulability with mixed-criticality, it is hard to find a fair base line to compare with. The reason EDF-VD is selected here since: 1) it is a widely accepted MC scheduling strategy; 2) it is the most general algorithm in the whole VD family; and 3) HI-criticality tasks are treated as a whole in both algorithms—EDF-VD sets virtual deadline according to a common factor, while we make use of an HI-criticality server. We need to point out that EDF-VD assumes unknown f_i for each task (not simply 0 or 1), and thus our algorithm has privilege naturally.

We use the algorithm *UUniFast* [22] to generate task sets for various values of cumulative LO utilization ($u(\text{LO}) = \sum_{i=1}^{n} c_i(\text{LO})/T_i$) and HI utilization ($u(\text{HI}) = \sum_{i|\chi_i=\text{HI}} c_i(\text{HI})/T_i$). The parameter $u(\text{LO})$ is ranged from 0 to 1, while $u(\text{HI})$ is ranged from 0 to 1.5, each with step 0.01.

Each task set contains 20 tasks, each of which is assigned LO or HI criticality with equal probability. LO-criticality utilizations are assigned according to *UUniFast*; given an expected HI utilization $u(\text{HI})$, we inflate the LO-criticality utilizations of the HI-criticality tasks using random factors chosen to ensure that the cumulative HI utilization of the task set equals the desired value with high probability.

Among the 626 200 valid task sets that we generated, EDF-VD succeeds to schedule 306 299 (48.9%) of them, and the proposed pMC reports probabilistic schedulable for a total of 438 787 sets (70.1%), and only 121 426 sets (19.4%) are reported unknown. Even when focused only upon systems for which HI-criticality utilization is less than 1, EDF-VD fails to schedule 18.0%, while pMC returns unknown for only 8.4% of the sets.

Although EDF-VD and pMC do not dominate each other, pMC generally significantly outperforms EDF-VD, particularly upon task sets with large HI-utilization. Due to space constraints, the detailed schedulability comparison with EDF-VD algorithm is omitted in this article, but is available in the conference version of this work [1].

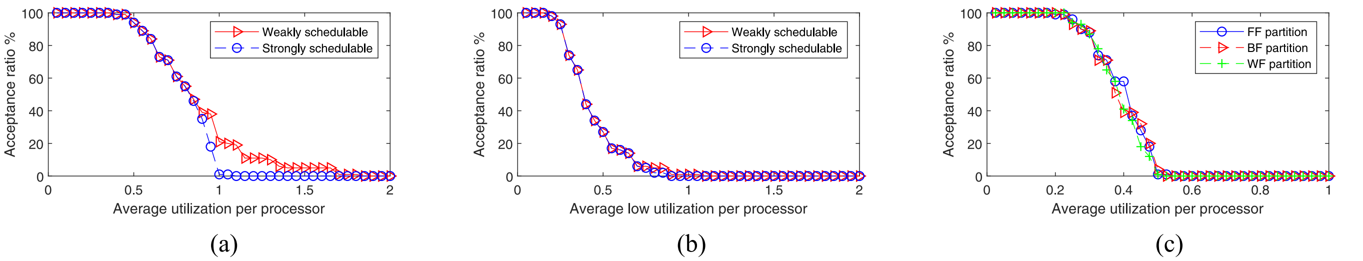
To show the robustness of our algorithm with respect to different f_i distributions, we focus on task sets with HI utilization between 0.9 and 1. Fig. 2 reports the ratios of schedulable (i.e., weakly probabilistic schedulable) sets over different LO utilizations. With the additional probability information, the schedulable ratio is significantly improved for heavy tasks comparing to EDF-VD [21]. The introduced parameter f_i is assigned to tasks in different ways; i.e., all sharing the same value, following uniform or log-uniform distribution ($f_i = 10^x$, where x is uniformly chosen). Generally speaking, smaller average f leads to higher ratio of acceptance, and there is no significant difference between different distributions of f_i with the same average, which indicates that our algorithm is *robust* to different combinations of output measurement probabilities from PTA tools.

The boxplot in Fig. 3 shows the total additional utilization resulting from LFF-Clustering normalized by the precise value is calculated using exhaustive search. Due to high complexity for calculating precise values, 100 task sets under each n_{HI} setting is considered. The mean (orange line) is close to 1 under each setting, meaning over 75% of the task sets were precisely clustered. However, the outliers get closer to 1 as n_{HI} increases—LFF-Clustering is particularly beneficial when there are large number of HI-tasks and calculating precise clustering becomes prohibitively expensive.

B. *Results for Multiprocessor Platforms*

Workload Generation: To conduct the experiments, we have generated MC tasks based on the following parameters.

1) *m:* The number of processor cores.
2) *U_a:* The average utilization for the task set. The average is calculated by averaging the LO and HI-criticality utilization of the task set.
3) *$P_{\text{HI}=0.5}$:* Probability of a task to be an HI-criticality one.
4) *$R = 4$:* Denotes the maximum ratio of u_i^{HI} to u_i^{LO}. u_i^{HI} is generated uniformly from $[u_i^{\text{LO}}, R \times u_i^{\text{LO}}]$.

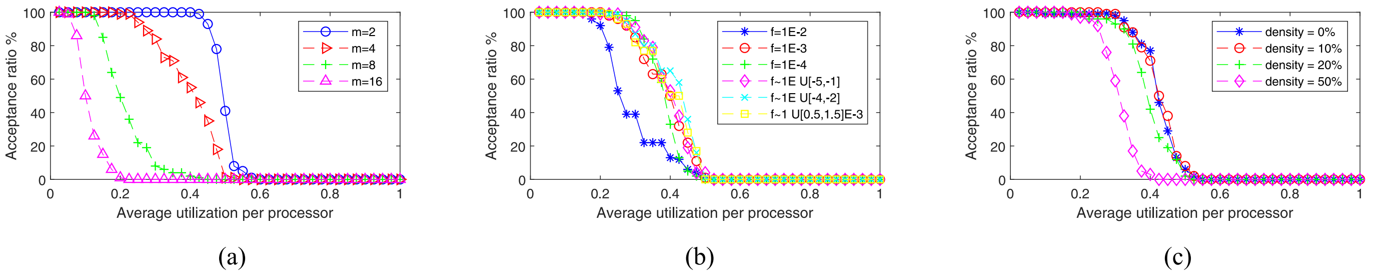


(a)

(b)

(c)

Fig. 4. Acceptance ratio for pMCMP in a 4-core platform under different utilizations and different partition heuristics. (a) Varying U_a. (b) Varying U_{LO}. (c) Varying partition heuristics.



(a)

(b)

(c)

Fig. 5. Acceptance ratio for pMCMP upon a quad-core platform. (a) Varying number of processors (m). (b) Varying distribution assumptions on f_i. (c) Varying densities of covariance.

5) *F_S:* The system-wide permitted failure probability, default set as 10^{-6} for our experiments.

We performed the simulation for average utilization ranging from 0.05 to 2 m with increasing at step size 0.05 m. For every average utilization, we generate 100 task sets that consist of 20 tasks each. Note that for most of the experiments, we have measured the performance with respect to average utilization as we wanted to show the improved quality of service for generated MC task sets.

First, for a specific average utilization, we use the *UUniFast algorithm* [23] to generate a lognormal distribution of U_a for all the tasks in a task set. The values of u_i^{LO} are uniformly generated from $[(2 \times u_i^a)/(R+1), u_i^a]$ so that the value of u_i^{HI} is always in the range $[u_i^{\text{LO}}, R \times u_i^{\text{LO}}]$.

Evaluation Results: We execute a set of MC tasks under our proposed algorithm by varying different parameters. The simulation results for various scenarios are presented in Figs. 4 and 5. We perform the following simulations.

The schedulability performance of pMCMP is shown in Fig. 4. Fig. 4(a) shows the acceptance ratio (ratio of successfully scheduled task sets over total number of task sets) with respect to average utilization U_a, while Fig. 4(b) shows the acceptance ratio with respect to u_i^{LO}. In both figures, we show the acceptance ratio for both strongly schedulable and weakly schedulable task sets. In Fig. 4(b), we can see that a good number of task sets is schedulable when the average utilization of the task set is one or even higher as the average utilization is calculated based on both u_i^{LO} and u_i^{HI} but pMCMP does not need to allocate the full HI-WCET budget. To understand the schedulability with respect to u_i^{LO}, we further performed the experiment presented in Fig. 4(b) where the u_i^{LO} is generated following the lognormal distribution using the *UUniFast algorithm* [23].

Fig. 4(c) presents the acceptance ratio for all three heuristics (FF, BF, and WF) discussed in Section V. The results match the discussion that all RAD algorithms share the same utilization bound while task partitioning. We use only strongly schedulable task set to calculate the acceptance ratio from this experiment as only the strongly schedulable task sets provide the graceful degradation to LO-criticality tasks.

Fig. 5 first presents the percentage of successful strongly schedulable task set under different numbers of processors [Fig. 5(a)] and different numbers of f_i values [Fig. 4(b)]. As expected, the performance of schedulability decreases with the increase of the number of the processors by following the performance of the partition heuristics. On the other hand, with the lower f_i values, we get better acceptance ratio as the algorithm can create more clusters and thus needs to allocate a smaller Δ. We also evaluated the effect of the density of the covariance matrix on the acceptance ratio, calculated as the density of the undirected graph interpreted from the covariance matrix. The number of edges of the covariance graph (1 in the covariance matrix) is randomly generated based on the density of edge. The simulation result is presented in Fig. 5(c). Under lower densities, our algorithms performed surprisingly well. With the increase of density, the possibility of merging also increases and the acceptance ratio decreases.

VIII. Related Works

There is a gap between the current conservative deterministic analysis and the richer models, which include the probabilistic information about the WCET estimates. Besides the resource underutilization issue due to over-pessimism, LO-critical tasks do not receive guarantees in HI-critical modes. Graceful degradation techniques and imprecise computation have been proposed to provide guarantees to the LO-critical tasks as well. Specifically, reduced utilization budget [24] to scale budgets in HI-mode as well as precise computation techniques, such as providing asymptotic rate guarantees [25], guaranteed completion rate [26], and QoS guarantees for LO-critical tasks [27]. While these approaches address the underprovision of resources to LO-critical tasks, they do not leverage the information from the WCET estimates.

Real-Time Models With Probabilities: In order to formally describe the uncertainty of the WCET estimations and overcome the overpessimism, many attempts in introducing probability to real-time system model and analysis have been made. Edgar and Burns [28] made a major step forward in introducing the concept of *probabilistic confidence* to the task and the system model. Their work targets the estimation of pWCETs from test data for individual tasks, while providing a suitable lower bound for the overall confidence level of a system. Since then, on the one hand, much work has been done to provide better WCET estimations and a predicted probability of any execution exceeding such estimation alongside the usage of extreme value theory (EVT), e.g., [6], [7], and [29]. In static PTA, random replacement caches are applied to compute exact pWCETs, and pWCET estimations with preemptions [30]. More recently, researchers have initiated some pWCET estimation studies [31], [32] in the presence of permanent faults and disabling of hardware elements. On the other hand, there is only one piece of work that proposes probabilistic execution time (pET) estimation [33]-based upon a tree-based technique. The pET of a task describes the probability that the execution time of the job is equal to a given value, while the pWCET of a task describes the probability that WCET of that task does not exceed a given value.

Schedulability With Probabilities: Based upon the estimated pWCET and pET parameters (often as distributions with multiple values and associated probabilities), studies aim to provide estimations that the probability of missing a deadline of the given system is small enough for safety requirements; e.g., of the same order of magnitude as other dependability estimations. Tia *et al.* [34] focused on unbalanced heavy loaded system (with maximum utilization larger than 1 and much smaller average utilization) and provided two methods for probabilistic schedulability guarantees. Lehoczky [35] proposed the first schedulability analysis of task systems with pETs. This work is further extended to specific schedulers, such as earliest deadline first (EDF, [36]) in [37] and under fixed-priority policy in [38]. Díaz *et al.* [13] provided a very general analysis for probabilistic systems with pWCET estimations for tasks. In addition to WCET estimations, statistical guarantees are performed upon the minimum interarrival time (MIT) estimation as well [14], [39]. Schedulability analysis based on pETs (instead of pWCETs) is also done in [40] for limited priority-level case (quantized EDF), and in [41], an associated schedulability analysis on multiprocessors is presented. Statistical response-time analysis, e.g., [42], can be further done to real-time systems based upon those probabilistic schedulability analysis. Unfortunately, to the best of our knowledge, most existing studies have only shown probabilistic schedulability analysis (e.g., estimating the likelihood for a system to miss any deadline) or probabilistic response time analysis to existing algorithms, such as EDF and fixed-priority scheduling, instead of incorporating probabilistic information into the scheduling strategy. In other words, current research has not addressed the possibility of making *smarter scheduling decisions* with probabilistic models from existing powerful PTA tools (e.g., [43]) that provide WCET bounds and specified confidences.[10]

Mixed-Criticality Probabilistic Scheduling: Few recent works have applied probabilistic models to mixed-criticality scheduling. In [46], the probabilistic models are applied to LO-criticality modes and a scheduling algorithm is developed to leverage probabilities into schedulability analysis. In particular, it is quantified how LO-criticality jobs behave whenever HI-criticality jobs overrun their optimistic LO-criticality reservation. In [47] and [48], the probabilistic models are applied into classical mixed-critical scheduling policies for fixed-priority task scheduling. These works have proven the advantages in resource utilization from the use of flexible probabilistic worst case representations. Draskovic *et al.* [49] introduced a new probabilistic model for mixed-criticality systems; safety metrics are defined, and an analysis is developed to quantify safety levels according to the considered criticality level. Santinelli and Guo [50] discussed the accuracy and the flexibility of probabilistic models as advantages for mixed-criticality schedulability analysis in efficient resource usage. Davis and Cucu-Grosjean [51] depicted a survey of recent probabilistic scheduling approaches, with a dedicated section of the above introduced mixed-criticality scheduling approaches.

IX. Conclusion

This article presented some efforts into scheduling MC systems that account for probabilistic information. The existing MC task models are generalized with an additional parameter specifying the distribution information of WCET. We require that it is *a priori* determined how likely jobs may exceed their LO-WCETs. We proposed a novel EDF-based scheduling algorithm, which exploits the probabilistic information to make mode-switching and LO-task-dropping decisions. Given a system failure probability threshold, the goal is to derive more precise schedulability analysis, which may deem a system that is infeasible under the traditional MC model as feasible, and will not drop any task unless it is probabilistically necessary. The experimental results show the advantages of the model and the proposed scheduling schemes.

The provided solution requires a server with period of 1—applying the idea of adaptive servers (with dynamic periods or budgets) may avoid many preemptions. So far we only considered systems with two criticality levels. Probabilistic correctness for multiple probability thresholds per task needs to be defined, and the scheduling problem is worth studying.

References

[1] Z. Guo, L. Santinelli, and K. Yang, "EDF schedulability analysis on mixed-criticality systems with permitted failure probability," in *Proc. IEEE 21st Int. Conf. Embedded Real Time Comput. Syst Appl. (RTCSA)*, 2015, pp. 187–196.

[10] To our best knowledge, there is only one paper presenting scheduling algorithms for pWCETs of tasks described by random variables [44], which extends the optimality of Audsley's approach [45] in fixed-priority scheduling to the case WCETs are described by distribution functions.

[2] J. Souyris, E. Pavec, G. Himbert, V. Jegu, and G. Borios, “Computing the worst case execution time of an avionics program by abstract interpretation,” in *Proc. 5th Int. Workshop Worst-Case Execution Time Anal. (WCET)*, 2005, pp. 1–4.

[3] R. Wilhelm *et al.*, “The worst-case execution-time problem—Overview of methods and survey of tools,” *ACM Trans. Embedded Comput. Syst.*, vol. 7, no. 3, pp. 1–53, 2008.

[4] A. Burns and R. Davis. (2018). *Mixed-Criticality Systems: A Review*. [Online]. Available: http://www-users.cs.york.ac.uk/ burns/review.pdf

[5] F. Cazorla *et al.*, “PROARTIS: Probabilistically analysable real-time systems,” *ACM Trans. Embedded Comput. Syst.*, vol. 12, no. 2, p. 94, 2013.

[6] J. Hansen, S. Hissam, and G. Moreno, “Statistical-based WCET estimation and validation,” in *Proc. 9th Int. Workshop Worst-Case Execution Time Anal. (WCET)*, 2009, pp. 1–11.

[7] L. Cucu-Grosjean *et al.*, “Measurement-based probabilistic timing analysis for multi-path programs,” in *Proc. 24th Euromicro Conf. Real-Time Syst. (ECRTS)*, 2012, pp. 91–101.

[8] RFS, *Software Considerations in Airborne Systems and Equipment Certification, RTCA Document DO-178C*, RTCA, Washington, DC, USA, 1992.

[9] L. Cucu-Grosjean, “Independence—A misunderstood property of and for probabilistic real-time systems,” in *Real-Time Systems: The Past, the Present and the Future*, N. Audsley and S. Baruah, Eds. 2013. [Online]. Available: https://scholar.google.com/scholar?cluster=4922492882824036056&hl=en&as_sdt=40005&sciodt=0,10#d=gs_cit&u=%2Fscholar%3Fq%3Dinfo%3A2H4dSjI1UEQJ%3Ascholar.google.com%2F%26output%3Dcite%26scirp%3D0%26scfhb%3D1%26hl%3Den

[10] S. Vestal, “Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance,” in *Proc. 28th IEEE Real-Time Syst. Symp. (RTSS)*, 2007, pp. 239–243.

[11] L. Santinelli, J. Morio, G. Dufour, and D. Jacquemart, “On the sustainability of the extreme value theory for WCET estimation,” in *Proc. 14th Int. Workshop Worst-Case Execution Time Anal. (WCET)*, 2014, pp. 21–30.

[12] A. Melani, E. Noulard, and L. Santinelli, “Learning from probabilities: Dependences within real-time systems,” in *Proc. 8th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, 2013, pp. 1–8.

[13] J. Díaz, D. Garcia, C. Lee, L. Bello, J. López, and O. Mirabella, “Stochastic analysis of periodic real-time systems,” in *Proc. 23rd IEEE Real-Time Syst. Symp. (RTSS)*, 2002, pp. 289–300.

[14] D. Maxim and L. Cucu-Grosjean, “Response time analysis for fixed-priority tasks with multiple probabilistic parameters,” in *Proc. 34th IEEE Real-Time Syst. Symp. (RTSS)*, 2013, pp. 224–235.

[15] B. Korte and J. Vygen, “Bin-packing,” in *Kombinatorische Optimierung*. Berlin, Germany: Springer, 2012, pp. 471–488. [Online]. sAvailable: https://doi.org/10.1007/978-3-642-24488-9_18

[16] J. M. López, J. L. Díaz, and D. F. García, “Utilization bounds for EDF scheduling on real-time multiprocessor systems,” *Real-Time Syst.*, vol. 28, no. 1, pp. 39–68, 2004.

[17] I. E. Commission *et al.*, *Fault Tree Analysis (FTA)*, IEC Standard 61025, 2006.

[18] IT Committee, *Analysis Techniques for System Reliability-Procedure for Failure Mode and Effects Analysis (FMEA)*, IEC Standard 60812, 2006.

[19] R. W. Irving, “NP-completeness of a family of graph-colouring problems,” *Discr. Appl. Math.*, vol. 5, no. 1, pp. 111–117, 1983.

[20] D. J. Welsh and M. B. Powell, “An upper bound for the chromatic number of a graph and its application to timetabling problems,” *Comput. J.*, vol. 10, no. 1, pp. 85–86, 1967.

[21] S. Baruah *et al.*, “The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems,” in *Proc. 24th Euromicro Conf. Real-Time Syst. (ECRTS)*, 2012, pp. 145–154.

[22] E. Bini and G. Buttazzo, “Measuring the performance of schedulability tests,” *Real-Time Syst.*, vol. 30, nos. 1–2, pp. 129–154, 2005.

[23] M. Bolado *et al.*, “Platform based on open-source cores for industrial applications,” in *Proc. IEEE DATE*, vol. 2, 2004, pp. 1014–1019.

[24] D. Liu *et al.*, “EDF-VD scheduling of mixed-criticality systems with degraded quality guarantees,” in *Proc. 37th Real-Time Syst. Symp. (RTSS)*, 2016, pp. 35–46.

[25] M. S. Branicky, S. M. Phillips, and W. Zhang, “Scheduling and feedback co-design for networked control systems,” in *Proc. 41st IEEE Conf. Decis. Control*, vol. 2, 2002, pp. 1211–1217.

[26] Z. Guo, K. Yang, S. Vaidhun, S. Arefin, S. K. Das, and H. Xiong, “Uniprocessor mixed-criticality scheduling with graceful degradation by completion rate,” in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, 2018, pp. 373–383.

[27] D. Matschulat, C. A. Marcon, and F. Hessel, “ER-EDF: A QoS scheduler for real-time embedded systems,” in *Proc. 18th IEEE/IFIP Int. Workshop Rapid Syst. Prototyping (RSP)*, 2007, pp. 181–188.

[28] S. Edgar and A. Burns, “Statistical analysis of WCET for scheduling,” in *Proc. 22nd IEEE Real-Time Syst. Symp. (RTSS)*, 2001, pp. 215–224.

[29] D. Griffin and A. Burns, “Realism in statistical analysis of worst case execution times,” in *Proc. 10th Int. Workshop Worst-Case Execution Time Anal. (WCET)*, 2010, pp. 44–53.

[30] R. I. Davis, L. Santinelli, S. Altmeyer, C. Maiza, and L. Cucu-Grosjean, “Analysis of probabilistic cache related pre-emption delays,” *Proc. 25th IEEE Euromicro Conf. Real-Time Syst. (ECRTS)*, 2013, pp. 168–179.

[31] M. Slijepcevic, L. Kosmidis, J. Abella, E. Q. Nones, and F. J. Cazorla, “DTM: Degraded test mode for fault-aware probabilistic timing analysis,” in *Proc. 25th Euromicro Conf. Real-Time Syst. (ECRTS)*, 2013, pp. 237–248.

[32] D. Hardy and I. Puaut, “Static probabilistic worst case execution time estimation for architectures with faulty instruction caches,” in *Proc. 21st Int. Conf. Real-Time Netw. Syst. (RTNS)*, 2013, pp. 35–44.

[33] L. David and I. Puaut, “Static determination of probabilistic execution times,” in *Proc. 16th Euromicro Conf. Real-Time Syst. (ECRTS)*, 2004, pp. 223–230.

[34] T. Tia, Z. Deng, M. Storch, J. Sun, L. Wu, and J. Liu, “Probabilistic performance guarantee for real-time tasks with varying computation times,” in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, 1995, pp. 164–173.

[35] J. Lehoczky, “Real-time queueing theory,” in *Proc. 17th IEEE Real-Time Syst. Symp. (RTSS)*, 1996, pp. 186–195.

[36] C. Liu and J. Layland, “Scheduling algorithms for multiprogramming in a hard real-time environment,” *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[37] H. Zhu, J. Hansen, J. Lehoczky, and R. Rajkumar, “Optimal partitioning for quantized EDF scheduling,” in *Proc. 23rd IEEE Real-Time Syst. Symp. (RTSS)*, 2002, pp. 212–222.

[38] M. Gardner and J. Liu, “Analyzing stochastic fixed-priority real-time systems,” in *Proc. 5th Int. Conf. Tools Algorithms Construction Anal. Syst. (TACAS)*, 1999, pp. 44–58.

[39] L. Abeni and G. Buttazzo, “QoS guarantee using probabilistic deadlines,” in *Proc. 11th Euromicro Conf. Real-Time Syst. (ECRTS)*, 1999, pp. 242–249.

[40] J. Hansen, H. Zhu, J. Lehoczky, H. Zhu, and R. Rajkumar, “Quantized EDF scheduling in a stochastic environment,” in *Proc. 10th Int. Workshop Parallel Distrib. Real-Time Syst.*, Apr. 2002, pp. 1–7.

[41] S. Manolache, P. Eles, and Z. Peng, “Schedulability analysis of applications with stochastic task execution times,” *ACM Trans. Embedded Comput. Syst.*, vol. 3, no. 4, pp. 706–735, 2004.

[42] Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean, “A statistical response-time analysis of real-time embedded systems,” in *Proc. 33rd IEEE Real-Time Syst. Symp. (RTSS)*, 2012, pp. 351–362.

[43] G. Bernat, A. Colin, and S. Petters, “pWCET: A tool for probabilistic worst-case execution time analysis of real-time systems,” Dept. Comput. Sci., Univ. York, York, 2003.

[44] D. Maxim, O. Buffet, L. Santinelli, L. Cucu-Grosjean, and R. Davis, “Optimal priority assignment algorithms for probabilistic real-time systems,” in *Proc. 19th Int. Conf. Real-Time Netw. Syst. (RTNS)*, 2011, pp. 129–138.

[45] N. Audsley, “On priority assignment in fixed priority scheduling,” *Inf. Process. Lett.*, vol. 79, no. 1, pp. 39–44, 2001.

[46] M. Kuttler, M. Roitzsch, C.-J. Hamann, and M. Volp, “Probabilistic analysis of low-criticality execution,” in *Proc. WMC RTSS*, 2017, pp. 168–179.

[47] Y. Abdeddaïm and D. Maxim, “Probabilistic schedulability analysis for fixed priority mixed criticality real-time systems,” in *Proc. Design Autom. Test Europe Conf. Exhibit. (DATE)*, Mar. 2017, pp. 596–601.

[48] D. Maxim, R. I. Davis, L. Cucu-Grosjean, and A. Easwaran, “Probabilistic analysis for mixed criticality systems using fixed priority preemptive scheduling,” in *Proc. 25th Int. Conf. Real Time Netw. Syst. (RTNS)*, Grenoble, France, Oct. 2017, pp. 237–246.

[49] S. Draskovic, P. Huang, and L. Thiele, “On the safety of mixed-criticality scheduling,” *Proc. Workshop Mixed Criticality (WMC) RTSS*, 2016, pp. 1–7.

[50] L. Santinelli and Z. Guo, “On the criticality of probabilistic worst-case execution time models,” in *Proc. Depend. Softw. Eng. Theories Tools Appl. 3rd Int. Symp. (SETTA)*, Changsha, China, Oct. 2017, pp. 59–74.

[51] R. I. Davis and L. Cucu-Grosjean, “A survey of probabilistic schedulability analysis techniques for real-time systems,” *LITES*, vol. 6, no. 1, pp. 1–4, 2019.