EDF Schedulability Analysis on Mixed-Criticality Systems with Permitted Failure Probability

Zhishan Guo Luca Santinelli† Kecheng Yang
The University of North Carolina at Chapel Hill †ONERA The French Aerospace Lab at Toulouse

***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 paper, 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 the WCET estimations – the likelihood of all jobs of a task finishing their executions within the less pessimistic WCET estimate. An efficient algorithm named LFF-Clustering is derived for scheduling mixed-criticality systems represented by this model. Experimental analyses show our new model and algorithm out-perform current state-of-the-art mixed-criticality scheduling algorithms.**

I. Introduction

Safety-critical systems are 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 which account for any possible behavior, included 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 which 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 [29], 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 [33] 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 the WCET may be significantly large. Instead of completely wasting the processor capacities within the gap, people start to implement functionalities of different degrees of importance, or *criticalities*, upon a common platform, so that the less important tasks may execute in these gaps under normal circumstances, may be dropped in an occasional situation where jobs of higher importance level execute beyond their estimated common case running time.

Much prior research on mixed-criticality scheduling (see [6] for a review) has 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: a LO-WCET determined by a less pessimistic tool, and 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 run-time scheduling strategy which ensures that (i) all jobs of all tasks complete by their deadlines if each job completes upon executing for no more than its LO-WCET; and (ii) 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 are not very likely to be exceeded during run time.

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 $\tau_1 = \{0.4, 0.6\}, \tau_2 = \{0.3, 0.5\}$, 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: (i) absolute certainty of correctness is not required; instead it is specified that the system failure probability should not exceed 10^{-6} per hour; and (ii) it is known that the timing analysis tools used to determine LO-

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

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:

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

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, dropping of less important jobs may not be necessary at all.

Schedulability with Probabilities. In order to formally describe the uncertainty of the WCET estimations and overcome the over-pessimism, many attempts in introducing probability to real-time system model and analysis have been made.

Edgar and Burns [13] 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 probabilistic WCETs (pWCETs) from test data for individual tasks, while providing a suitable lower bound for the overall confidence level of a system. Since then, on 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., [16] [15] [9]. In static probabilistic timing analysis, random replacement caches are applied to compute exact probabilistic WCETs, and probabilistic WCET estimations with preemptions, [11]. More recently, researchers have initiated some pWCET estimation studies [28] [18] in the presence of permanent faults and disabling of hardware elements. On the other hand, there is only one piece of work which proposes probabilistic Execution Time (pET) estimation [10] 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 the worst-case execution time of that task does not exceed a given value.

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. [31] focus on unbalanced heavy loaded system (with maximum utilization larger than 1 and much smaller average utilization) and provide two methods for probabilistic schedulability guarantees. Lehoczky [19] proposes the first schedulability analysis of task systems with probabilistic execution times. This work is further extended to specific schedulers, such as earliest deadline first (EDF, [20]) in [34] and under fixed priority policy in [14]. [12] provides a very general analysis for probabilistic systems with pWCET estimations for tasks. In addition to WCET estimations, statistical guarantees are performed upon the minimum inter-arrival time (MIT) estimation as well [1] [24]. Schedulability analysis based on pETs (instead of pWCETs) is also done in [17] for limited priority level case (quantized EDF), and in [22] where an associated schedulability analysis on multiprocessors is presented. Statistical response-time analysis, e.g., [21], can be further done to real-time embedded systems based upon those probabilistic schedulability analysis.

Unfortunately, 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 probabilistic timing analysis tools (e.g., [4]) that provide WCET bounds and specified confidences. To our best knowledge, there is only one paper presenting scheduling algorithms for probabilistic WCETs of tasks described by random variables [23], which extends the optimality of Audsley's approach [2] in fixed-priority scheduling to the case WCETs are described by distribution functions.

Finally, none of the existing schedulability analysis work regarding mixed-criticality considered pWCET. Since the major goal of both mixed-criticality and introducing probability are the same, which is to better deal with the over-pessimism of running time estimations, we believe a model that considers both aspects would lead us to much more promising results in real-time system design and verification.

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 preemptive uniprocessor 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:

- 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.
- 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.
- We derive a novel scheduling algorithm (and an associated sufficient schedulability test) for a given MC task set and an allowed system failure probability. We seek to schedule the system such that the probability of failing to meet timing constraints during run-time is guaranteed to be no larger than the specified failure probability.

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 dates, deadlines, and WCETs, but also on the probabilities drawn from probabilistic timing analysis tools (see, e.g., [7] [16] [9]).

Organization. Sec. II introduces the model and and show its advantage by a motivating example. Sec. III formally defines probabilistic schedulability and related concepts. In Sec. IV we propose a clustering based scheduling strategy, and the corresponding schedulability test, while Sec. V performs their experimental evaluations and comparisons. Sec. VI 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 contrast to constrained-deadline ones). Throughout this paper, an integer model of time is assumed — all task periods are assumed to be non-negative integers, and all job arrivals are assumed to occur at integer instants in time.

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. However, no prior work on MC analysis has leveraged any information from the confidence of the provisioned WCET.

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: Existing timing analysis work usually analyzes the WCET for a task on a per-job basis; i.e., by focusing on the distribution of WCETs of jobs of a certain task. 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 i) all jobs WCET of a certain task obey the same distribution (identically distributed), and ii) 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 worst-case execution time, 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.

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 pWCET estimate comes from a random variable (the worst-case execution time 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 Cumulative Distribution Functions (CCDFs), $F'_{\mathcal{C}_i}$. In the following, calligraphic upper-case letters are used to refer to probabilistic distributions, while non calligraphic 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 the paper, 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 $p(\text{LO})$, thus the confidence on the upper-bounding worst-case, $c(\text{LO})$ increases.

Given A the event that a job exceeds its threshold and $p^A = P(\mathcal{C}_i > c(\text{LO}))$ its probability of happening; 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 one hour task execution interval, we make use of the joint probability of all the exceeding threshold events within the one hour interval. The joint probability is

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

as the probability of just a task job exceeding its thresholds $c(\text{LO})$, and all the others not exceeding $c(\text{LO})$. With full independence, the probability of exceeding threshold in one hour would be at most $1 - F_{\mathcal{C}_i}(c(\text{LO})) \times \lfloor p_i/3,600,000 \rfloor$, with the task τ_i period p_i expressed in $msec$.

III. Probabilistic Schedulability

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 one hour of execution[3]. F_S may be very close to zero (e.g., 10^{-12} for some safety critical avionics functionalities).

A *failure probability* parameter f_i is added to HI-criticality task τ_i, denoting the probability that the actual execution

[2]The timing analysis that make use of the EVT, by definition provides continuous distributions as pWCET estimates [7]; they are then discretized, to ease their representation, by assigning them a discrete support.

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

requirement of *any* job of the task exceeding $c_i(\text{LO})$ (but still below $c_i(\text{HI})$) in one hour (i.e., the adequate long time interval we assumed in this paper). 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 one hour interval or any another interval, Eq. (2). Thus, f_i can be directly derived from $F_{\mathcal{C}_i}$[4].

A HI-criticality task is represented by: $\tau_i = ([c_i(\text{LO}), c_i(\text{HI})], f_i, p_i, \chi_i)$; 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 one hour, no job of τ_i has an execution greater than $c_i(\text{HI})$ and the probability of *any* job of τ_i has 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 probabilistic WCET 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 $c_i(\text{LO})$ are adopted for them.

Definition 1 (MC Task Instance): A MC task instance I is composed of a 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): A 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 of 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 dependences 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 [8] 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. The 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, HI-WCET) are so far called worst-case execution time 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 [32] for the original definition of the multi-WCET 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, the failure probability f_i of a task τ_i 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 [26], [25] it has been showed that real safety-critical embedded systems have natural variability on the task's execution time, thus it is reasonable to assume the independence (or extremal independence) between jobs.

Since jobs generated from the same task set typically represent execution of the same piece of code, and consecutive such 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 [26], [25] it has been showed that real safety-critical embedded systems have natural variability on the task execution time, thus it is reasonable 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 execution time task dependencies in our framework, Section IV. It is however 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. Furthermore, it is worthy to note that execution times are observed with other tasks executing in parallel, thus the execution time measuring already embeds the dependence effects from other tasks. In future work we plan to further explain task dependence modeling at runtime.

To resume, dependence between jobs of the same task and between tasks are covered by our model.

[4]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).

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

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

Since we consider EDF schedulability instead of fixed priority, we would like to know whether, and *how likely* system utilization may exceed 1: (i) 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 traditional MC model to be probabilistic feasible; (ii) 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 $u_{\text{HI}} = 2/5 + 4/10 = 1$, 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. Recall 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[5]. 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 over-utilized system – a "server" $\tau_s = \{0.2, 1, \text{HI}\}$ 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 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. ∎

[5]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.

IV. Scheduling Strategy

A. The LFF-Clustering Algorithm

In this subsection, 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: (i) 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 (ii) 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 $G_1, G_2, ..., G_M$, and let $y_i \in \{1, 2, ..., M\}$ denote to which cluster (number) task τ_i is assigned.

Definition 4 (Failure probability of a cluster): Failure of a cluster G_m is defined as job 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

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

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 exact one of the tasks exceeding its LO-WCET in an hour.

Note that we are allowing one task per cluster to exceed its LO-WCET during execution, and will assign certain capacity (details to be provided in latter part of the section) to assure such failure won't cause any deadline miss.

Lemma 1: If $g_m < F_S/M$ holds for any cluster G_m, then the probability of having no failure in any cluster 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 to each other, the probability of having no failure in any cluster 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 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.

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.,

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

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

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

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, leading to the Largest Fit First (LFF)-Clustering algorithm.

Algorithm 1: Algorithm LFF-Clustering

Input: F_S, $\{f_i\}_{i=1}^{n_{\text{HI}}}$, $\{\delta_i\}_{i=1}^{n_{\text{HI}}}$
Output: maximum total additional utilization cost Δ

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

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 [12] and [24], we may achieve a precise distribution to the total utilization of all tasks by applying the *convolution* operation '$\otimes$', which results in an exponential ($O(2^{n_{\text{HI}}})$, to be precise) running time (see [24]). The sufficient schedulability test based on the LFF-Clustering algorithm runs in $O(n_{\text{HI}}^2)$ 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*.

Run-Time Strategy. During execution, a HI-criticality *server* τ_s with utilization Δ and a period of 1 tick is added to the task system. 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, the active[6] HI-criticality job, if any, with earliest deadline is executed; if there is no such job, the current job of the server is dropped[7]. All jobs including the server are scheduled and executed in EDF order, and a job is dropped at its deadline if it is not completed by then.

Note that 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:

Situation 1: The job with the earliest deadline is a 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 a 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 a 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 results 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 met always, 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 experimental comparisons).

In this work we make use of servers to implement our algorithms and prove the possibility of proficiently apply failure probability to both MC modeling and MC scheduling. In future

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

[7]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 is no active HI-criticality jobs since there can be no HI-criticality job releases in the current period of the server.

work we will release the server period assumption of 1 unit of time by applying adaptivity to resource reservation [27], [30]. With the analysis of the deadline and task periods we will be able to implement realistic servers which adapt their period and budget to the MC-scheduler needs, while leaving the system predictable at any time interval. Such adaptive behavior will not introduce any overhead, and mostly will allow not to miss task deadline.

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 $\left(\sum_{i=1}^{n} c_i(\text{LO})/p_i + \Delta\right)$ must be no larger than the capacity of the processor (which is 1).

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 $\left(\sum_{i|\chi_i=\text{HI}} c_i(\text{LO})/p_i + \Delta\right)$ 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*

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

Proof: As assumed, the task set is feasible when no job exceeds its LO-WCET; i.e., $\sum_{i=1}^{n} c_i(LO)/p_i \leq 1$. Therefore, if the server does not exist, all task will meet their deadlines under EDF scheduling. Since the server task is not a real task but only executes the earliest-deadline HI-criticality job if exists, introducing this server will never *delay* any HI-criticality task's execution (comparing to no-server circumstance). Thus, the deadlines of all HI-criticality jobs will still be met.

Next, by contradiction, we show if (7) holds, all deadlines of LO-criticality jobs will also met. Suppose t_d is the first time instant when a deadline of a LO-criticality job is missed. Let t_0 denote the last idle instant for jobs with deadlines at or before t_d[8], then $[t_0, t_d)$ is a busy interval. Let Ψ denote the set of the HI-criticality jobs that are released at or after t_0 and with deadlines at or before t_d, and Ψ' denote the complement (i.e., HI-criticality jobs with deadlines after t_d).

Let W denote the total demand created by jobs in Ψ within $[t_0, t_d)$, then

$$W \leq \sum_{i|\chi_i=\text{HI}} \left\lfloor \frac{t_d - t_0}{p_i} \right\rfloor \cdot c_i(LO). \tag{8}$$

We have shown that all HI-criticality jobs will meet their deadlines (in the first paragraph of this proof), which implies that there must be a processor supply of W allocated to those jobs in Ψ. Since the server has a period of 1, no job will be released during each server period. Moreover, the server has the highest scheduling priority, and will execute the earliest-deadline HI-criticality job (when exists). Thus for any unit-length period, if jobs in Ψ are executed for a cumulative length of w, at least a server budget of $\Delta \cdot w$ will be consumed by those jobs. Thus, within $[t_0, t_d)$, at least $\Delta \cdot W$ server budget must execute jobs in Ψ. On the other hand, the server (by its definition) could have at most $\Delta \cdot (t_d - t_0)$ budget in $[t_0, t_d)$. Thus, within the period $[t_0, t_d)$, jobs in Ψ' will consume server budget of at most $\Delta \cdot (t_d - t_0) - \Delta \cdot W$. Moreover, since there will always be active jobs with deadline at or before t_0 throughout the interval, and we are using pure EDF "outside" the server, jobs in Ψ' (with later deadlines) can only execute within $[t_0, t_d)$ by consuming server budget.

Also, within the busy interval $[t_0, t_d)$, a LO-criticality task τ_i can only release $\lfloor (t_d - t_0)/p_i \rfloor$ jobs with deadlines at or before t_d. Thus, and by the definition of t_d and t_0, we have

$$\begin{aligned} &\left(\Delta \cdot (t_d - t_0) - \Delta \cdot W\right) + W + \\ &\sum_{i|\chi_i=\text{LO}} \left\lfloor \frac{t_d - t_0}{p_i} \right\rfloor \cdot c_i(LO) > t_d - t_0. \end{aligned} \tag{9}$$

Moreover,

$$\begin{aligned} &\left(\Delta \cdot (t_d - t_0) - \Delta \cdot W\right) + W + \sum_{i|\chi_i=\text{LO}} \left\lfloor \frac{t_d - t_0}{p_i} \right\rfloor \cdot c_i(LO) \\ =\ & \Delta \cdot (t_d - t_0) + (1 - \Delta) \cdot W + \sum_{i|\chi_i=\text{LO}} \left\lfloor \frac{t_d - t_0}{p_i} \right\rfloor \cdot c_i(LO) \\ \leq\ & \{\text{by (8) and } \Delta \leq 1\} \\ & \Delta \cdot (t_d - t_0) + (1 - \Delta) \cdot \sum_{i|\chi_i=\text{HI}} \left\lfloor \frac{t_d - t_0}{p_i} \right\rfloor \cdot c_i(LO) + \\ & \sum_{i|\chi_i=\text{LO}} \left\lfloor \frac{t_d - t_0}{p_i} \right\rfloor \cdot c_i(LO) \\ \leq\ & \{\text{by } \lfloor (t_d - t_0)/p_i \rfloor \leq (t_d - t_0)/p_i \text{ for all } i \text{ and } \Delta \leq 1\} \\ & \Delta \cdot (t_d - t_0) + (1 - \Delta) \cdot \sum_{i|\chi_i=\text{HI}} \frac{t_d - t_0}{p_i} \cdot c_i(LO) + \\ & \sum_{i|\chi_i=\text{LO}} \frac{t_d - t_0}{p_i} \cdot c_i(LO) \\ =\ & \Delta \cdot (t_d - t_0) - \Delta \cdot (t_d - t_0) \cdot \sum_{i|\chi_i=\text{HI}} \frac{c_i(LO)}{p_i} + \\ & (t_d - t_0) \cdot \sum_{i|\chi_i=\text{HI}} \frac{c_i(LO)}{p_i} + (t_d - t_0) \cdot \sum_{i|\chi_i=\text{LO}} \frac{c_i(LO)}{p_i} \\ =\ & \Delta \cdot (t_d - t_0) \cdot \left(1 - \sum_{i|\chi_i=\text{HI}} \frac{c_i(LO)}{p_i}\right) + \\ & (t_d - t_0) \cdot \sum_{i=1}^{n} \frac{c_i(LO)}{p_i}. \end{aligned} \tag{10}$$

By (9) and (10),

$$\begin{aligned} &\Delta \cdot (t_d - t_0) \cdot \left(1 - \sum_{i|\chi_i=\text{HI}} \frac{c_i(LO)}{p_i}\right) + \\ &(t_d - t_0) \cdot \sum_{i=1}^{n} \frac{c_i(LO)}{p_i} > t_d - t_0, \end{aligned} \tag{11}$$

[8] If at a instant there is no active job with deadline at or before t_d, it is considered *idle* in this proof.

Canceling $(t_d - t_0)$ on both sides contradicts our theorem assumption, (7).

Thus, such t_d does not exist and therefore no LO-criticality job will miss its deadline. □

Theorem 1 yields the schedulability test pMC (Algorithm 2), while Theorem 2 below 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: From Lemma 1 and Theorem 1, we may conclude that the possibility of HI-criticality tasks altogether requiring an additional utilization of no more than Δ is less than F_S, and thus they can still meet their deadlines with probability no less than $1 - F_S$ upon the assigned server task.

The utilization-based test of EDF is run twice. If the first test succeeds; i.e., total utilization (including the server) is less than 1, then all tasks will meet their deadlines with a probability no less than $(1 - F_S)$ — this ensures strongly probabilistic schedulable. If not, we need to check two other conditions which together ensure *weakly probabilistic schedulable*: (i) a utilization test involving HI-criticality tasks and the server, which guarantees that the probability of all HI-criticality tasks meeting their deadlines is no less than $(1 - F_S)$ should some jobs exceed their LO-WCETs; and (ii) a utilization based condition involving all tasks and the server, which guarantees *correctness* for all tasks when no HI-criticality one exceeds its LO-WCET (Theorem 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 either we have set a too high safety requirement to the system; 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's are not small enough comparing to F_S (and n_{HI}), and/ or the $c_i(\text{LO})$'s are still not differentiable enough with respect to $c_i(\text{HI})$'s.

We show how our algorithm works by applying it to an example.

Example 3: Consider the MC task system consisting of six tasks shown in Table I, and a specified allowed system failure probability of $F_S = 3.2 \times 10^{-4}$. For simplicity, tasks are ordered decreasingly by δ_i values. (The δ_i's for each task are calculated according to (3).)

The LFF-Clustering algorithm initially assigns each task a single cluster, and try to expand the one (of the largest Δ_i value) with task τ_1. τ_2 can be combined into Cluster G_1 since

TABLE I
A SET OF MC TASKS.

-	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6
$[c_i(\text{LO}), c_i(\text{HI})]$	$[0.5, 0.5]$	$[1, 3]$	$[1, 2]$	$[3, 5]$	$[3, 6]$	$[1, 2]$
p_i	5	10	25	50	20	4
δ_i	0.2	0.2	0.08	0.06	0.05	-
f_i	0.0001	0.0001	0.01	0.01	0.001	-
χ_i	HI	HI	HI	HI	HI	LO

$g_1 < F_S/M$ holds ($g_1 = 1 - (1 - f_1)(1 - f_2) - f_1(1 - f_2) - f_2(1 - f_1) = f_1 f_2 < F_S/4$). Similarly, combining τ_3 will results in a smaller number of total remaining clusters ($M = 3$), and Inequality $g_1 < F_S/M$ continues to hold.

However, this inequality no longer holds as we further expand G_1 for τ_4 (g_1 becomes greater than $F_S/2$). Thus we assign Task τ_4 a second cluster G_2. Similar to the situation of τ_4, τ_5 cannot be combined into cluster G_1. However, combining τ_5 with τ_4 is allowed since $M = 2$ and $g_2 = f_4 f_5 < F_S/2$.

Finally we have visited all HI-criticality tasks, and the value to be returned by the LFF-Clustering algorithm is $\Delta_1 + \Delta_2 = u_1 + u_4 = 0.26$.

Since the total system utilization (including the LO-criticality task τ_6) remains less than 1 with a server of utilization 0.26. The schedulability test pMC returns *strongly probabilistic schedulable*. During run-time, an additional server $\tau_s = \{0.26, 1, \text{HI}\}$ will be added to the task system, on which active HI-criticality jobs will execute (also in EDF order). When there is no active HI-criticality job, current job of the server will be dropped. ▪

Algorithm 2: Schedulability Test pMC

Input: τ, F_S
Output: schedulability

begin
  Calculate the δ_i values for all HI-criticality tasks in τ;
  $u_{\text{LO}} \leftarrow \sum_{i=1}^{n} c_i(\text{LO})/p_i$;
  $u'_{\text{LO}} \leftarrow \sum_{i|\chi_i=\text{HI}} c_i(\text{LO})/p_i$;
  $\Delta \leftarrow$ *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

V. SCHEDULABILITY EXPERIMENTS

We have conducted schedulability tests on randomly-generated task systems, comparing our proposed method with 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 [3].

Since this is the first work that combines pWCET and schedulability with mixed-criticality, it is hard to find a fair

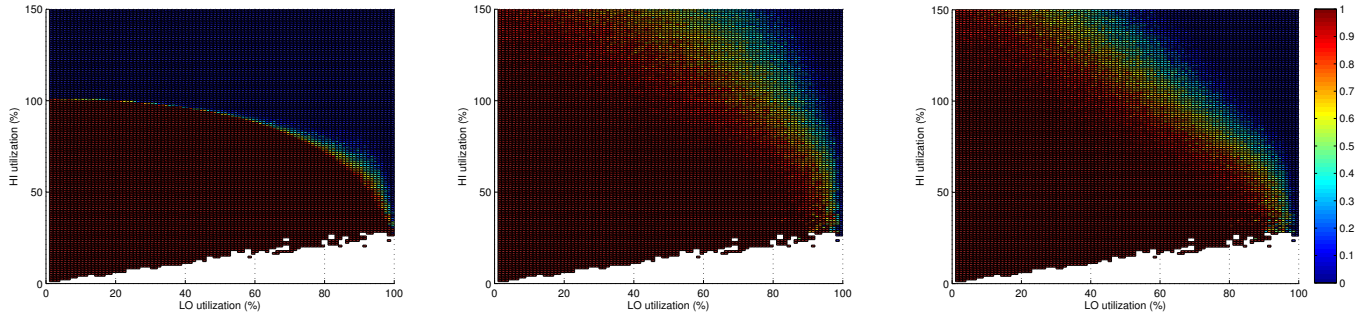

Fig. 1. Schedulability comparison of the EDF-VD algorithm (left), our algorithm when returning partial correct or correct (middle), and our algorithm returning only correct (right), where color of each block represents the percentage of schedulable sets within certain *utilization* ranges. (*Please enlarge these figures enough, and/or use colored printer for better view.*)

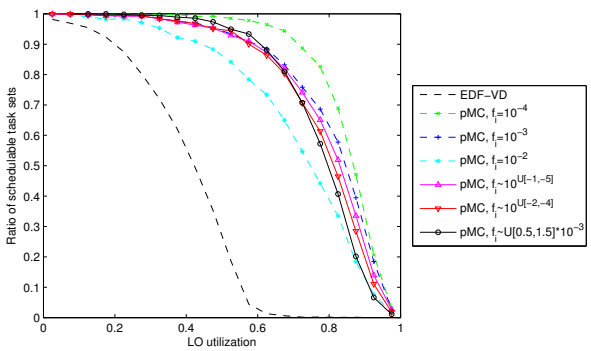

Fig. 2. Schedulability ratio comparison of EDF-VD and pMC, where HI utilization varies from 0.9 to 1 in a uniform manner.

base line to compare with. The reason EDF-VD is selected here since (i) it is a widely accepted MC scheduling strategy; (ii) it is the most general algorithm in the whole VD family; and (iii) 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 a 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* [5] to generate task sets for various values of cumulative LO utilization ($u(\text{LO}) = \sum_{i=1}^{n} c_i(\text{LO})/p_i$) and HI utilization ($u(\text{HI}) = \sum_{i|\chi_i=\text{HI}} c_i(\text{HI})/p_i$). 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 438787 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. Figure 1 depicts the schedulability results for the two algorithms, where $f_i = 10^{-3}$ of all tasks τ_i and $F_s = 10^{-6}$. Instances with similar $u(\text{LO})$ and $u(\text{HI})$ values are put into a same small block. The color of each small block represents the percentage of schedulable sets[9]. As shown in Figure 1, although EDF-VD and pMC do not dominate each other, pMC generally significantly outperforms EDF-VD, particularly upon task-sets with large HI-utilization.

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. Figure 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 [3]. The introduced parameter f_i is assigned to tasks in different ways; i.e., all sharing a same value, following uniform distribution, or following 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 probabilistic timing analysis tools.

VI. Context & Conclusions

In this paper we have presented our initial research into scheduling MC systems that account for probabilistic information. Existing MC task models are generalized with an additional parameter specifying the distribution information of the WCET. We require that it is *a priori* determined how *likely* jobs may exceed their LO-WCETs per hour. We proposed a novel EDF-based scheduling algorithm, which exploits the probabilistic information to make mode-switching and LO-criticality-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*.

[9]Since we randomly assign criticality levels to all tasks, the LO utilization of HI-criticality tasks is expected to be $u(\text{LO})/2$. It is unlikely to generate tasks with $u(\text{HI}) < u(\text{LO})/2$, and thus the right lower triangle regions are left blank in Figure 1.

Experimental results show the advantages of the novel model and the proposed scheduling schemes.

Future Work. The solution provided in this paper requires a server with period of 1. We are currently working on applying the idea of adaptive servers (with dynamic periods or budgets) [30] to avoid too many preemptions.

Our study only targeted the scheduling of implicit deadline sporadic task sets (including both synchronous and asynchronous periodic systems) on uniprocessor platforms. We believe the arbitrary deadline case cannot be solved by simply extending or modifying the proposed method. While for multiprocessor systems, dependencies between WCET of tasks widely arises due to cache sharing, which requires investigation on more sophisticated model and strategy.

So far we only considered systems with two criticality levels. For multiple levels of criticality, the number of WCET estimations per task will also increase, which results in multiple probability thresholds. Probabilistic correctness for multiple probability thresholds per task need to be defined, and the scheduling problem (which is likely to be much more complicated) is worth studying.

ACKNOWLEDGEMENTS

We would like to thank Dr. Sanjoy Baruah and Dr. Dorin Maxim for their suggestive comments. Work supported by NSF grants CNS 1115284, CNS 1218693, CNS 1409175, and CPS 1446631, AFOSR grant FA9550-14-1-0161, and a grant from General Motors Corp.

REFERENCES

[1] L. Abeni and G. Buttazzo. QoS guarantee using probabilistic deadlines. In *the 11th Euromicro Conference on Real-Time Systems (ECRTS'99)*, 1999.

[2] N. Audsley. On priority assignment in fixed priority scheduling. *Information Processing Letters*, 79(1):39–44, 2001.

[3] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie. The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *the 24th Euromicro Conference on Real-Time Systems (ECRTS'12)*, 2012.

[4] G. Bernat, A. Colin, and S. Petters. pWCET: a tool for probabilistic worst-case execution time analysis of real-time systems. Report - University of York Department of Computer Science YCS, 2003.

[5] E. Bini and G. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[6] A. Burns and R. Davis. Mixed-criticality systems: A review. `http://www-users.cs.york.ac.uk/~burns/review.pdf`, 2013.

[7] F. Cazorla, E. Quinones, T. Vardanega, L. Cucu-Grosjean, B. Triquet, G. Bernat, E. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo, and D. Maxim. PROARTIS: Probabilistically analysable real-time systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 12(2):1–26, 2013.

[8] L. Cucu-Grosjean. Independence - a misunderstood property of and for probabilistic real-time systems. In *N. Audsley and S. Baruah, editors, Real-Time Systems: the past, the present and the future*, 2013.

[9] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quinones, and F. Cazorla. Measurement-based probabilistic timing analysis for multi-path programs. In *the 24th Euromicro Conference on Real-Time Systems (ECRTS'12)*, 2012.

[10] L. David and I. Puaut. Static determination of probabilistic execution times. In *the 16th Euromicro Conference on Real-Time Systems (ECRTS'04)*, 2004.

[11] R. I. Davis, L. Santinelli, S. Altmeyer, C. Maiza, and L. Cucu-Grosjean. Analysis of probabilistic cache related pre-emption delays. *Proceedings of the 25th IEEE Euromicro Conference on Real-Time Systems (ECRTS'13)*, 2013.

[12] J. Díaz, D. Garcia, C. Lee, L. Bello, J. López, and O. Mirabella. Stochastic analysis of periodic real-time systems. In *the 23rd IEEE Real-Time Systems Symposium (RTSS'02)*, 2002.

[13] S. Edgar and A. Burns. Statistical analysis of WCET for scheduling. In *the 22nd IEEE Real-Time Systems Symposium (RTSS'01)*, 2001.

[14] M. Gardner and J. Liu. Analyzing stochastic fixed-priority real-time systems. In *the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, 1999.

[15] D. Griffin and A. Burns. Realism in statistical analysis of worst case execution times. In *the 10th International Workshop on Worst-Case Execution Time Analysis (WCET'10)*, 2010.

[16] J. Hansen, S. Hissam, and G. Moreno. Statistical-based WCET estimation and validation. In *the 9th International Workshop on Worst-Case Execution Time Analysis (WCET'09)*, 2009.

[17] J. Hansen, J. Lehoczky, H. Zhu, and R. Rajkumar. Quantized EDF scheduling in a stochastic enviroment. In *the 16th IEEE International Parallel and Distributed Processing Symposium (IPDPS'02)*, 2002.

[18] D. Hardy and I. Puaut. Static probabilistic worst case execution time estimation for architectures with faulty instruction caches. In *the 21st International Conference on Real-Time and Networked Systems (RTNS'13)*, 2013.

[19] J. Lehoczky. Real-time queueing theory. In *the 17th IEEE Real-Time Systems Symposium (RTSS'96)*, 1996.

[20] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[21] Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean. A statistical response-time analysis of real-time embedded systems. In *the 33rd IEEE Real-Time Systems Symposium (RTSS'12)*, 2012.

[22] S. Manolache, P. Eles, and Z. Peng. Schedulability analysis of applications with stochastic task execution times. *ACM Transactions on Embedded Computing Systems (TECS)*, 3(4):706–735, 2004.

[23] D. Maxim, O. Buffet, L. Santinelli, L. Cucu-Grosjean, and R. Davis. Optimal priority assignment algorithms for probabilistic real-time systems. In *the 19th International Conference on Real-Time and Networked Systems (RTNS'11)*, 2011.

[24] D. Maxim and L. Cucu-Grosjean. Response time analysis for fixed-priority tasks with multiple probabilistic parameters. In *the 34th IEEE Real-Time Systems Symposium (RTSS'13)*, 2013.

[25] A. Melani, E. Noulard, and L. Santinelli. Learning from probabilities: Dependences within real-time systems. In *the 8th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'13)*, 2013.

[26] L. Santinelli, J. Morio, G. Dufour, and D. Jacquemart. On the sustainability of the extreme value theory for WCET estimation. In *the 14th International Workshop on Worst-Case Execution Time Analysis (WCET'14)*, 2014.

[27] Luca Santinelli, Giorgio Buttazzo, and Enrico Bini. Multi-moded resource reservation. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2011.

[28] M. Slijepcevic, L. Kosmidis, J. Abella, E. Q. Nones, and F. J. Cazorla. Dtm: Degraded test mode for fault-aware probabilistic timing analysis. In *the 25th Euromicro Conference on Real-Time Systems (ECRTS'13)*, 2013.

[29] 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 *the 5th International Workshop on Worst-Case Execution Time Analysis (WCET'05)*, 2005.

[30] Nikolay Stoimenov, Lothar Thiele, Luca Santinelli, and Giorgio Buttazzo. Resource adaptations with servers for hard real-time systems. In *International Conference On Embedded Software (EMSOFT)*, 2010.

[31] 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 *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'95)*, 1995.

[32] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *the 28th IEEE Real-Time Systems Symposium (RTSS'07)*, 2007.

[33] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution-time problem – overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):1–53, 2008.

[34] H. Zhu, J. Hansen, J. Lehoczky, and R. Rajkumar. Optimal partitioning for quantized EDF scheduling. In *the 23rd IEEE Real-Time Systems Symposium (RTSS'02)*, 2002.