Risk-Aware Scheduling of Dual Criticality Job Systems Using Demand Distributions

Bader Naim Alahmad
The University of British Columbia
2366 Main Mall, Vancouver, BC, Canada V6T 1Z4
bader@ece.ubc.ca
https://orcid.org/0000-0002-6409-1277

Sathish Gopalakrishnan
The University of British Columbia
2332 Main Mall, Vancouver, BC, Canada V6T 1Z4
sathish@ece.ubc.ca

Abstract

We pose the problem of scheduling Mixed Criticality (MC) job systems when there are only two criticality levels, Lo and Hi—referred to as Dual Criticality job systems—on a single processing platform, when job demands are probabilistic and their distributions are known. The current MC models require that the scheduling policy allocate as little execution time as possible to Lo-criticality jobs if the scenario of execution is of Hi criticality, and drop Lo-criticality jobs entirely as soon as the execution scenario's criticality level can be inferred and is Hi. The work incurred by "incorrectly" scheduling Lo-criticality jobs in cases of Hi realized scenarios might affect the feasibility of Hi criticality jobs; we quantify this work and call it Work Threatening Feasibility (WTF). Our objective is to construct online scheduling policies that minimize the expected WTF for the given instance, and under which the instance is feasible in a probabilistic sense that is consistent with the traditional deterministic definition of MC feasibility. We develop a probabilistic framework for MC scheduling, where feasibility is defined in terms of (chance) constraints on the probabilities that Lo and Hi jobs meet their deadlines. The probabilities are computed over the set of sample paths, or trajectories, induced by executing the policy, and those paths are dependent upon the set of execution scenarios and the given demand distributions. Our goal is to exploit the information provided by job distributions to compute the minimum expected WTF *below which the given instance is not feasible in probability*, and to compute a (randomized) "efficiently implementable" scheduling policy that realizes the latter quantity. We model the problem as a Constrained Markov Decision Process (CMDP) over a suitable state space and a finite planning horizon, and show that an optimal (non-stationary) Markov randomized scheduling policy exists. We derive an optimal policy by solving a Linear Program (LP). We also carry out quantitative evaluations on select probabilistic MC instances to demonstrate that our approach potentially outperforms current MC scheduling policies.

2012 ACM Subject Classification Mathematics of computing → Markov processes, Software and its engineering → Real-time systems software, Software and its engineering → Real-time schedulability
Keywords and Phrases Real-time scheduling; Mixed-criticality; Probability distribution; Chance-constrained Markov decision process; Linear programming
Digital Object Identifier 10.4230/LITES-v005-i001-a001
Received 2016-02-04 **Accepted** 2018-01-07 **Published** 2018-05-30

1 Introduction

We consider a system comprised of a *finite set of jobs* executing upon a *shared platform* (processor), and a *scheduling policy* that allocates processor time to jobs. A Mixed-Criticality (MC) real-time job system is one that carries out multiple jobs, with each job being of a specific criticality. For example, in an avionics/UAV system, some jobs relate to the flight stability or safety of the aircraft,


© Bader Naim Alahmad and Sathish Gopalakrishnan;
licensed under Creative Commons Attribution 3.0 Germany (CC BY 3.0 DE)

and these jobs have the highest criticality. Other jobs may relate to the mission of the aircraft (gather visual information of a particular region) and these jobs may be of a lower criticality. In the special and important case—that we consider in this article—where every job assumes one of exactly two criticality levels, Lo or Hi, we will refer to the MC system as Dual-Criticality.

From a *job scheduling* perspective, one would like to schedule jobs so that they meet their timing constraints or deadlines. To do so, one needs to know the execution time requirements of these jobs. Using worst-case execution time (WCET) estimates for execution time would lead to infeasibility of low criticality jobs (because the worst-case utilization could saturate the system capabilities) but, since worst-case execution times are rarely realized, one could use the same platform for jobs of all criticality levels provided the scheduler makes suitable choices when the execution duration of a job approaches the worst case or when it exceeds certain thresholds.

Vestal [38] was the first to offer an abstraction for scheduling MC job systems. In Vestal's model, there are $L \geq 2$ distinct criticality levels, and n jobs $J_1, \ldots, J_n$. For notational convenience, we denote as $[n]$ the set $\{1, \ldots, n\}$ for integers $n \geq 1$. Job J_i is characterized by the parameters (χ_i, c_i, d_i), where

- $\chi_i \in [L]$ is job J_i's **criticality**;
- $c_i = \big(c_i(1), \ldots, c_i(L)\big) \in (0, \infty)^L$ is the vector of **WCET estimates** at all criticality levels;
- $d_i > 0$ is job J_i's **deadline**.

For example, consider a triple-criticality MC job system consisting of three jobs J_1, J_2, and J_3 with criticalities $\chi_1 = 1$, $\chi_2 = 3$, and $\chi_3 = 2$, respectively, and with the following WCET estimates:

$$\begin{aligned} J_1: &\quad c_1 = \big(c_1(1) = \mathbf{90}, c_1(2) = 90, c_1(3) = 90\big) \\ J_2: &\quad c_2 = \big(c_2(1) = 10, c_2(2) = 12, c_1(3) = \mathbf{20}\big) \\ J_3: &\quad c_3 = \big(c_3(1) = 1, c_3(2) = \mathbf{500}, c_3(3) = 500\big). \end{aligned}$$

We shall make the following common monotonicity assumption: $c_i(1) \leq \cdots \leq c_i(L)$ for every $i \in [n]$. Moreover, we will assume that $c_i(\ell) = c_i(\chi_i)$ for all $\ell \geq \chi_i$, so that it is sufficient to specify job J_i's WCET estimates by giving $c_i(1), \ldots, c_i(\chi_i)$, $i \in [n]$. An execution **scenario**, or **behavior**, is a particular realization of job demands in a particular run of the system; i.e., it is a vector $b = (b_1, \ldots, b_n)$ in $\prod_{i=1}^{n}(0, c_i(\chi_i)]$. In our example, $(10, 11, 450)$ is a possible execution scenario. In any particular run of the system, the scenario remains unknown until *all* jobs finish execution. The criticality level of behavior b is defined as

$$\mathsf{critDemand}(b) = \min\big\{\ell \in [L] : b_i \leq c_i(\ell) \quad \forall i \in [n]\big\}.$$

For instance, $\mathsf{critDemand}\big((10, 11, 450)\big) = 2$. During a schedule, at time t, say, job J_i is said to be **operating** at criticality level $\ell \in [L]$ if it has been given at least $c_i(\ell - 1)$ but less than $c_i(\ell)$ units of execution, and has not finished execution at time t. We call this time-dependent quantity the **job's operational criticality level** at t. With the monotonicity assumption, the range of execution times that job J_i might demand when operating at criticality level ℓ is the open interval $(c_i(\ell - 1), c_i(\ell)]$, with the convention that $c_i(0) = 0$. In our example, if we take a snapshot of a certain schedule at, say time 63, and observe that jobs J_1, J_2 and J_3 have executed for 50, 10 and 3 time units, respectively, but J_2 has not yet finished execution, then J_2's operational criticality level at time 63 is 2. However, if J_2 finishes execution at time 63 with 10 time units of execution, then its operational criticality level for all $t \leq 63$ is 1. The operational criticality level remains the same from time t until J_i either signals that it has finished execution, or it executes for $c_i(\ell)$ time unit at some $t' > t$ and does not signal completion, at which point its operational criticality level jumps to $\ell + 1$. As such, a job's operational criticality level is an increasing piecewise-constant function of time, demand, and the scheduling policy, with a (random) set of jump points.

At time t, the maximum of all job operational criticality levels is the **system operational criticality level** at time t. We note that the system operational criticality level of an observed allocation snapshot, say b, at some time, is *not* the same as $\mathsf{critDemand}(b)$; the system operational criticality level depends on additional information not encoded in b, namely whether or not jobs finished execution, whereas $\mathsf{critDemand}(b)$ assumes that all jobs finished execution. Since the system operational criticality level is defined in terms of the job operational criticality levels, the former is also an increasing piecewise-constant function. In our example, the system operational criticality level at time 63 with the same execution snapshot $(50, 10, 3)$ is 1 if J_2 finishes execution at or before time 63, and is 2 otherwise. If the scheduler selects J_2 to execute from time 63 to time 67, then at time 65, the system operational critical level makes a jump from 2 to 3 (since then J_2 has executed for $12 = c_2(2)$ time units and has not finished execution), and remains 3 until the end of the schedule.

Once a job signals that it has finished execution, its **demand** is realized. A demand realization is a scenario of execution. Every job demand realization maps naturally to a unique **job criticality level realization**, and the maximum of which across all jobs is the **system criticality level realization**. Different runs, or executions, of the input job system might yield different criticality level realizations, since, generally, a job might demand anything in $(0, c_i(\chi_i)]$, and job demand realizations might differ across different executions.

The Job Dropping Model: Literature and Optimality

In addition to Vestal [38], there has been a substantial body of work that analyzes scheduling policies for deterministic MC systems, wherein low(er) criticality jobs are dropped when a high(er) criticality job demands more execution time. One such approach was studied by Baruah et al. [8, 10]. In this approach, low criticality jobs are dropped when it is deemed necessary to allocate more time to a high criticality job. This decision is based on deterministic thresholds and is conservative in the sense that worst-case assumptions are made about the execution time requirements of the low criticality jobs and other high criticality jobs. As a consequence, low criticality jobs may miss deadlines even when it may be possible to meet the deadlines for high and low criticality jobs. Feasibility of a given Dual-Criticality instance in this model is defined as follows: For every scenario of execution, if the scenario's criticality level is Lo, all jobs should be given enough execution time to complete entirely and should meet their deadlines, but if the scenario's criticality level is Hi, only Hi-criticality jobs need to be given execution budget and must complete before their deadlines. In the latter case, giving any execution time to Lo-criticality jobs is considered as an erroneous allocation, and doing so negatively affects the achievable processor utilization.

A **non-clairvoyant**, or **online**, scheduling policy does not know the scenario of execution in advance, and only an omniscient clairvoyant policy knows the realized scenario at time 0, and is therefore able to decide whether or not to drop Lo-criticality jobs at the beginning of system operation and thus achieve the maximum processor utilization. An instance $I = (J_1, \ldots, J_n; L)$ is said to be **correctly MC-schedulable** by scheduling policy π if for every scenario (behavior) $b \in \prod_{i=1}^{n}(0, c_i(\chi_i)]$, if b has criticality level ℓ, then every J_i with $\chi_i \geq \ell$ can be given b_i units of execution during $[0, d_i]$ under π. An instance I is said to be **MC-feasible** if there is an *online* scheduling policy under which I is correctly MC-schedulable.

Baruah et al. [10] showed that checking MC-feasibility can be reduced to checking its defining condition only for the scenarios that assume the WCET estimates; i.e., for $b \in \{(c_1(\ell_1), \ldots, c_n(\ell_n)) : \ell_i \in [\chi_i]\}$. The MC-feasibility problem was shown to NP-Hard in the strong sense [7]; however, it is not yet clear whether or not MC-feasibility belongs to the class NP.

If an instance I is not MC-feasible, then there is no online scheduling policy under which it is correctly MC-schedulable. Conversely, if instance I is MC-feasible, then an online scheduling policy that correctly MC-schedules I may or may not exist.

A widely used measure of the performance of non-clairvoyant MC-scheduling policies is the processor **speed-up factor** (Baruah et al. [8], Kalyanasundaram and Pruhs [30]). It is defined as follows: If π is a non-clairvoyant scheduling policy, then its speed-up factor is the smallest real number $s > 1$ such that, for *every* MC instance I, if I is MC-feasible on a unit-speed processor, then policy π will correctly MC-schedule I on an s (or more)-speed processor. An **optimal** policy is one that minimizes s.

In the MC context, a non-unit speed-up factor arises because of the following: A non-clairvoyant algorithm has only WCET estimates available, and it does not know the scenario of execution in advance, so in high criticality scenarios, the algorithm might allocate execution time to jobs whose criticality is less than the realized system criticality level. Thus the earlier the time at which the scenario's criticality level is inferred under the scheduling policy *while preserving feasibility* (in the MC sense), the less the "processor time waste" the policy incurs for that scenario. Since the scheduler can drop all LO-criticality jobs as soon as the scenario's criticality is inferred as HI, predicting the earliest such time plays a central role in the MC-scheduling problem. Given an MC job instance, a scenario of execution and a non-clairvoyant scheduling policy, we call the earliest time instant at which the execution scenario's criticality level is inferred with certainty the **Time of Criticality Inference ($T_{\mathbf{CI}}$)** associated with the scheduling policy for the given scenario. Giving any execution time to LO-criticality jobs early on in the schedule will only delay the T_{CI}, and thus delay the time instant at which we can decide whether or not to drop LO-criticality jobs. However, to preserve the schedulability of LO-criticality jobs in case the scenario is of LO criticality, the policy must judiciously give execution time to LO-criticality jobs early on in the schedule. Thus, we are facing conflicting objectives, and the optimal scheduling policy must strike the right allocation balance.

Here is an example to illustrate the situation.

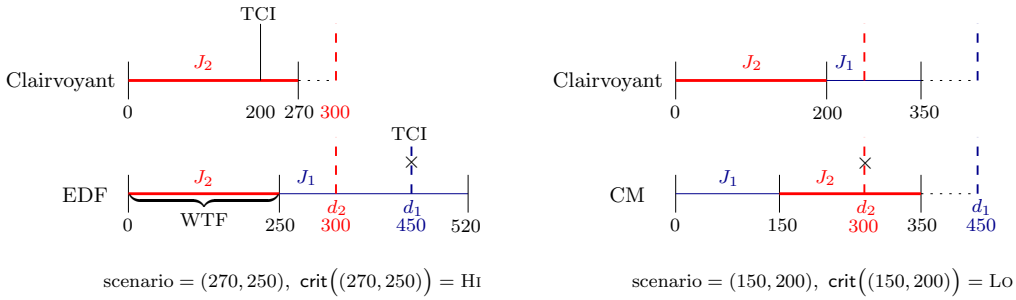
▶ **Example 1.** Consider a dual-criticality MC job system consisting of two jobs J_1 and J_2 with the following parameters:

$$\begin{aligned} J_1: &\quad c_1 = \big(c_1(\text{LO}) = 200, c_1(\text{HI}) = 300\big), &\quad \chi_1 = \text{HI}, &\quad d_1 = 450 \\ J_2: &\quad c_2 = \big(c_2(\text{LO}) = 250\big), &\quad \chi_2 = \text{LO}, &\quad d_2 = 300. \end{aligned}$$

First let us examine how the clairvoyant algorithm would MC-schedule this job instance. If the scenario is of HI criticality, then the clairvoyant policy knows this at time 0 and drops J_2 entirely and schedules J_1, which would then meet its deadline of 450. If the scenario is of LO criticality, then the clairvoyant policy schedules both J_1 and J_2 using the Earliest Deadline First (EDF) policy, and they both meet their deadlines: The worst-case LO-criticality scenario is $(200, 250)$, and under EDF, J_2 is scheduled first and finishes at time $250 < d_2 = 300$, and then J_1 occupies the processor till time 450 ($= d_1$). Now we consider two non-clairvoyant scheduling policies (see Figure 1):

- **EDF.** Suppose that the scenario of execution is $(270, 250)$, which has HI criticality. EDF first schedules J_2 up to time 250, and then selects J_1 to occupy the processor until time 520. At time 450, however, J_1 misses its deadline.
- **Criticality Monotonic (CM)**, which is a fixed-priority scheduling policy that at each instant schedules, among the jobs that have not finished execution, the job with the highest criticality. Suppose that the scenario of execution is the LO-criticality $(150, 200)$. J_1 occupies the processor from time 0 to time 150, then J_2 executes until time 350. J_2, however, misses its deadline at time 300.

The problem with EDF is that it does not consider criticalities, and consequently, in our example, it scheduled J_2 when it should have dropped it altogether. The work done by J_2 affected



$$J_1 : c_1 = \big(c_1(\text{Lo}) = 200, c_1(\text{Hi}) = 300\big), \quad \chi_1 = \text{Hi}, \quad d_1 = 450$$
$$J_2 : c_2 = \big(c_2(\text{Lo}) = 250\big), \quad \chi_2 = \text{Lo}, \quad d_2 = 300$$

Figure 1 Optimal clairvoyant vs. non-clairvoyant EDF (left) and clairvoyant vs. CM (right) schedules for the job set of Example 1. EDF incurs WTF of 250, causing J_1 to miss its deadline at 450. CM does not allocate the Lo-criticality J_2 enough execution time earlier in the schedule so as to guarantee its feasibility if the realized scenario is Lo, which is the case in this example. This causes J_2 to miss its deadline at time 300.

the feasibility of J_1 (which is the only job whose feasibility matters given that the scenario is of Hi criticality). We call this processor time waste—caused by lack of knowledge of the scenario—**Work Threatening Feasibility (WTF)**. In the example, EDF incurred WTF of 250 for the given scenario, caused by scheduling J_2. *The speed-up factor of a given scheduling policy measures its worst case (maximum) incurred WTF across all MC instances that are MC-feasible (on a unit-speed processor).* We note that WTF is only incurred for scenarios that have Hi criticality and, in this case, by giving execution time to Lo-criticality jobs; it is zero for Lo-criticality behaviors. The problem with CM, on the other hand, is that it does not care about the feasibility of Lo-criticality jobs in light of a Lo behavior, although it causes the system criticality level to be realized the soonest possible.

Our example suggests that to both minimize the WTF and guarantee feasibility, the scheduling policy must strive to achieve a balance between the following conflicting objectives:

O1. It should allocate Lo-criticality jobs sufficiently enough execution times early on; in particular, prior to the T_{CI}, so as to guarantee their schedulability in the case where the realized behavior is of Lo criticality, and

O2. It should minimize any WTF, by

- **a.** driving the revelation of the system criticality level sufficiently quickly by scheduling Hi-criticality jobs, so as to decide whether to drop Lo-criticality jobs as soon as possible, and
- **b.** minimizing the allocation in O1 if the scenario is Hi-criticality (it is here where the objectives are conflicting).

Probabilistic MC-Model: Justification

The MC model we consider in this article is a probabilistic variant of the MC model thus described. But *why use a probabilistic MC model?* First, the current MC standards and accreditations express the required performance guarantees of MC software components as failure probabilities. For

Table 1 DO-178B Criticality Specifications (AdaCore [1]).

Level	Failure Condition	Failure Rate Limit (failures/hour)	Example
A	Catastrophic	10^{-9}	Fly-by-wire
B	Hazardous	10^{-7}	Fuel management
C	Major	10^{-5}	Pilot/ATC communication
D	Minor	10^{-3}	Flight data recorder
E	No effect	n/a	Entertainment system

instance, the DO-178B avionics standard[1] lists 5 levels of criticality, and specifies for each criticality level an upper bound on the *failure rate* of software components having that criticality (Table 1). From the scheduling perspective, job failures are deadline misses. Then given how MC systems are specified in practice, we believe that a probabilistic framework is the natural setting in which MC systems ought to be framed and reasoned about.

Note. Failure rate estimation is a research problem in its own right, but is outside the scope of this article. We refer the reader to Shooman [36] for an in-depth account of failure rate estimation in avionics software systems, along with feasibility studies and the associated analysis.

Second, without any additional information about job demands other than WCET estimates, the scheduler is oblivious to job demand realizations prior to job completion, until the realizations present themselves online at one of the system operational criticality level jump instants. As a consequence, working with WCET estimates solely will lead to underutilization of the processor when the WCETs are not realized.

Contribution

Whereas the contribution in this specific article relates to a specific restriction of the MC job model, the overall thrust of our work is to develop a framework for reasoning about workload of different criticality levels and providing probabilistic guarantees about the successful execution of jobs. The one-shot *job* model that we consider—as opposed to the more complex *recurrent task model*—was, as we shall see below, studied extensively in the context of MC scheduling, and it remains highly relevant due to the complexity of MC scheduling problems. We have chosen this particular model as a first step towards reasoning about recurring tasks. One can interpret our work as providing the boundaries for synthesizing feasible policies.

This article is an attempt to reconcile the widely used job-dropping model and the mixed-criticality specifications as instituted by the current standards and the industry requirements. Baruah's work and ours have following in common: We both regard allocating execution times to Lo-criticality jobs in cases of Hi-criticality execution scenarios as undesirable behavior that the scheduling algorithm should avoid. In our model, however, feasibility is defined more generally, and our definition includes Baruah's definition as a special case: We are given upper bounds on the probabilities that jobs at each criticality level miss their deadlines, and one of our goals is to determine a policy under which the probabilities of deadline misses respect the user-supplied failure tolerance parameters. Toward this goal, we introduce the notion of **probably feasible** MC instances in the job dropping model (for the precise definitions, see Definition 4).

[1] Titled *Software Considerations in Airborne Systems and Equipment Certification*, and developed jointly by RTCA SC-167 and EUROCAE WG-12.

We propose an approach for Dual-Criticality job systems that is not deterministic, and uses the probability distribution of job execution times. Our contribution is a model of MC job systems as a **chance-Constrained Markov Decision Process (CMDP)** that then allows us to provide guarantees around jobs meeting their timing constraints with high probability. The chance constraints are sample path constraints on the trajectories of the MDP induced by executing a policy, and they represent the *risk* of missing deadlines at the various criticality levels. We show how to derive a randomized non-stationary Markov scheduling policy that is expected WTF-optimal, by solving a linear program.

This approach can be computationally expensive, but we envisage this as a first step in enabling such probabilistic analysis. Nevertheless, the problem is amenable to approximation, and we briefly outline one method that can be used to obtain approximately optimal and approximately feasible scheduling policies.

More Literature

Before concluding this section, we mention some prior work related to MC-scheduling and to probabilistic analysis of real-time systems.

Baruah and Vestal [11] showed that for recurrent MC task systems, Earliest Deadline First (EDF) does not dominate Rate-Monotonic (RM), and neither are optimal for scheduling MC tasks in the job dropping model. The Own Criticality-Based Priority (OCBP) algorithm was among the first algorithms designed specifically for the scheduling of (deterministic) MC job systems within the job dropping model [10]. OCBP is a fixed-priority scheduling policy, and it utilizes Audsley's priority assignment scheme [6]. OCBP was shown to be optimal in the class of fixed-priority MC-scheduling algorithms in the *speed-up factor*, with a speed-up factor of $(\sqrt{5}+1)/2$ for dual-criticality job system. It was shown that if an instance I is OCBP-schedulable, then it is MC-feasible; thus, correct schedulability by OCBP is sufficient for MC-feasibility, and the correct MC-scheduling policy is given by the OCBP priorities. Conversely, if I is MC-feasible, then OCBP might or might not correctly MC-schedule I; however, if I is MC-feasible, then OCBP can correctly MC-schedule I on a speed $(\sqrt{5}+1)/2$ processor, or, in other words, OCBP is capable of correctly MC-scheduling the (smaller) instance where every given WCET is divided by the speed-up factor $(\sqrt{5}+1)/2$. This quantifies how inexact OCBP is.

The MC-EDF algorithm [37] was shown to dominate OCBP, in the sense that there are (deterministic) MC-feasible instances that are deemed MC-schedulable by MC-EDF but not by OCBP.

Guo and Baruah [22] studied the scheduling of MC jobs (with job dropping) on a single processor with varying speeds. The authors of the latter extended their work to the sporadic task model with implicit deadlines [9]. Chen et al. [15] devised a deadline-tightening technique for scheduling MC *sporadic task* systems on a unit-speed single processor, wherein virtual deadlines that are shorter than the actual deadlines are assigned to the higher criticality jobs. Again, low(er) criticality tasks may be rejected in order to satisfy the demands of high(er) criticality tasks. We refer the reader to the manuscript by Burns and Davis [14] for the most current and comprehensive overview of MC systems and related problems.

The probabilistic analysis of (non-MC) real-time systems is not new. Díaz et al. [16] analyzed the behavior of fixed-priority (e.g., RM) and dynamic-priority (e.g., EDF) scheduling algorithms for recurrent, stochastically independent tasks when execution times are random variables. The goal of their work is to compute the probability of deadline miss as well as the (random) response-time of every task. See also [17, 18, 19, 31]. Maxim and Cucu-Grosjean [33] extended the probabilistic analysis framework of Díaz et al. [16] for fixed-priority scheduling schemes to task systems where also the minimum inter-arrival times between job invocations as well as task deadlines may be

random variables. Their focus was to efficiently compute the response time of each task under the assumption that tasks are stochastically independent. They do so by using convolution of probability distributions as the key underlying mathematical operation.

To the best of our knowledge, there is no work that aims at identifying feasible scheduling policies for MC job systems where job execution times are random. Alahmad et al. [2] were the first to propose the consideration of probabilistic execution times for MC systems. Guo et al. [23] carried out schedulability analysis of EDF applied to recurrent MC task systems, wherein lower priority tasks are given guarantees against failure. The latter is the closest work we are aware of to our efforts in this article. However, our problem is substantially harder, because it is concerned with *synthesizing* MC scheduling policies, as opposed to *analyzing* existing (fixed) scheduling policies.

2 System Model

We will adopt Vestal's model described above, but we will frame it in a probabilistic setting. The system we consider is that of n jobs executing upon a single processor, and all jobs are ready to execute at time 0. We will make use of the sets $\mathbb{N} = \{1, 2, \dots\} \subset \{0, 1, 2, \dots\} = \mathbb{Z}_+$. For ease of reference, we give in Table 2 a listing of most of the notation used in this article.

Note: The purpose of this section is to present as general a probabilistic framework for MC systems. As such, the exposition to follow will be in terms of general probability spaces, arbitrary number of criticality levels, with no assumptions about the random demands except boundedness. *This setting is, however, much more general than the actual problem that we consider, which is a specialization of the framework to be presented to two criticality levels and discrete demands.*

In addition to the parameters (χ_i, c_i, d_i) described earlier, the execution **demand** of job J_i is described by a random variable

$$\zeta_i : \Omega_i \to (0, c_i(1)] \cup \cdots \cup (c_i(\chi_i - 1), c_i(\chi_i)] = (0, c_i(\chi_i)]$$

on a probability space $(\Omega_i, \mathcal{M}_i, \mathbb{P}_i)$, where Ω_i is the scenario space associated with job J_i consisting of all possible execution scenarios, $\mathcal{M}_i$ is the set of possible (observable, measurable) events, and $\mathbb{P}_i$ is a probability measure on Ω_i.

We will assume that the jobs are *independent*; that is, the demand random variables $\zeta_1, \dots, \zeta_n$ are independent. The distribution of ζ_i is the probability measure $\mathbb{P}_{\zeta_i} \equiv \mathbb{P}_i \circ \zeta_i^{-1}$ on $(0, c_i(\chi_i)]$, and $\mathbb{P}_{\zeta_i}$ is known. Accordingly, job J_i is characterized by the tuple $\big((\Omega_i, \mathcal{M}_i, \mathbb{P}_i), \zeta_i, \chi_i, c_i, d_i\big)$, $i \in [n]$. The actual execution time that a job consumes at run-time (upon completion) is a **job demand realization**. The demand realization of a job is not known prior to its completion. A job **completes** execution when it announces, or signals, that it has finished execution; i.e., when the demand realization has presented itself. The latter happens when the job has been allocated enough execution time to produce its output entirely.

To this end, let

$$\Omega = \prod_{i=1}^{n} \Omega_i, \quad \mathcal{M} = \bigotimes_{i=1}^{n} \mathcal{M}_i,$$

where $\bigotimes_{i=1}^{n} \mathcal{M}_i$ is the product σ-algebra; that is, the σ-algebra with respect to which all the projection (coordinate) maps $\mathsf{proj}_i : \Omega \to \Omega_i$ are measurable. Let $\mathbb{P}$ be the product measure on $(\Omega, \mathcal{M})$; i.e., $\mathbb{P}$ is such that for every rectangle $A \in \mathcal{M}$, where $A = A_1 \times \cdots \times A_n$ and $A_i \in \mathcal{M}_i$,

$$\mathbb{P}(A) = \mathbb{P}(A_1 \times \cdots \times A_n) = \prod_{i=1}^{n} \mathbb{P}_i(A_i). \tag{1}$$

Table 2 Notation

Notation		Meaning
$\mathbb{Z}_+$	:	$\{0, 1, 2, \dots\}$
$\mathbb{N}$	:	$\{1, 2, \dots\}$
$[m]$, where $m \in \mathbb{N}$	:	$\{1, \dots, m\}$
$\mathbb{R}$	:	The real numbers
$n \in \mathbb{N}$	:	Number of input jobs
$c_i(\ell) > 0$	:	WCET estimate of job J_i at criticality level ℓ
χ_i	:	Criticality level of job J_i
$d_i > 0$	:	Deadline of job J_i
Ω_i	:	Scenario space of job J_i
$\mathcal{M}_i$	:	Set of events; subsets of Ω_i (σ-algebra)
$\mathbb{P}_i$	:	Probability measure on the scenario space Ω_i of job J_i
$\bigotimes_{i=1}^{n} \mathcal{M}_i$	:	n-fold product σ-algebra
$\mathbb{P}$	:	Probability measure on the product scenario space
ζ_i, Z_i	:	Demand random variables
G_{Z_i}	:	Distribution function of random variable Z_i
$\mathbb{E}$	:	Expectation operator with respect to product scenario space
$\mathbb{1}_E(x)$	:	Indicator function of a set E
$\delta_x(E)$, E is a set, x a point	:	Dirac measure
proj_i	:	Projection (coordinate) map, returns ith component of a given vector
$\mathbf{0}$, $\mathbf{1}$	:	All zeros and all ones vectors
e_i	:	Unit vector whose ith coordinate is 1
A	:	$\{e_1, \dots, e_n\} \cup \mathbf{0}$, Action space of the MDP
S	:	State space of the MDP
a_t	:	n-component vector, action taken at time t
y_t	:	n-component binary vector of job finish signals at time t
x_t	:	n-component vector, cumulative execution time allocations up to time t
r_t	:	scalar error flag
$s_t = (t, y_t, x_t, r_t)$	:	State of the MDP at time t
$\mathsf{A}(s_t)$	:	Admissible actions in state s_t
$\pi(ds_t \mid s_{t-1}, a_{t-1})$	:	Markov policy
$Q(ds_t \mid s_{t-1}, a_{t-1})$	:	State transition kernel of MDP
H_∞	:	Canonical trajectory space of the MDP induced by executing policy π
$\{A_t\}, \{S_t\}, \{Y_t\}, \{R_t\}$, $t \in \mathbb{Z}_+$	:	Action, state, finish signal, and error stochastic processes on H_∞
$\mathsf{crit} : \Omega = \prod_{i=1}^{n} \Omega_i \to \mathbb{N}$	:	Scenario criticality level
$\mathsf{critDemand} : B \equiv \prod_{i=1}^{n} (0, c_i] \to \mathbb{N}$	:	Demand realization criticality level
$\mathsf{critPath} : H_\infty \to \mathbb{N}$	:	System criticality level of trajectory (path)
$\mathsf{critState} : \mathsf{S} \to \{\textsc{Lo}, \textsc{Hi}, \textsc{Unknown}\}$	:	Operational system criticality level given a state
$\mathbb{P}^\pi$	:	The (unique) probability measure on H_∞
$\mathbb{E}^\pi$	:	Expectation with respect to H_∞
$F_i \equiv F_i^\pi : H_\infty \to \mathbb{N}$	:	(Random) Finish time of job J_i with respect to policy π
$T_{\textsc{Lo}} \equiv T_{\textsc{Lo}}^\pi : H_\infty \to \mathbb{N}$	:	Earliest time at which Lo system criticality level is inferred by policy π
$T_{\textsc{Hi}} \equiv T_{\textsc{Hi}}^\pi : H_\infty \to \mathbb{N}$	:	Earliest time at which Hi system criticality level is inferred by policy π
$T_{\textsc{CI}} \equiv T_{\textsc{CI}}^\pi$	:	$\min(T_{\textsc{Lo}}, T_{\textsc{Hi}})$, $T_{\textsc{CI}}$ of a trajectory in H_∞
$w : \mathsf{S} \times \mathsf{S} \to \mathbb{Z}_+$	:	Local (immediate, per stage) objective cost function of MDP
$W \equiv W^\pi : H_\infty \to \mathbb{Z}_+$	:	WTF random variable on trajectory space
$\kappa : \mathsf{S} \to \{0, 1\}$	:	Immediate constraint cost function of MDP
$C \equiv C^\pi : H_\infty \to \mathbb{Z}_+$	:	Constraint cost random variable on trajectory space

We shall denote vectors $\omega \in \Omega$ as $\omega_1, \ldots, \omega_n$, where $\omega_i \in \Omega_i$ is the ith coordinate of ω. We extend every ζ_i to be defined on Ω as follows. Let $Z_i = \zeta_i \circ \mathsf{proj}_i$. Then $Z_i : \Omega \to (0, c_i(L)]$ depends only on the ith coordinate of a given $\omega \in \Omega$; that is,

$$Z_i(\omega) = \zeta_i\big(\mathsf{proj}_i(\omega)\big) = \zeta_i(\omega_i) \qquad (\omega \in \Omega).$$

We define the demand vector $Z = \big(Z_1, \ldots, Z_n\big) : \Omega \to \prod_{i=1}^{n}(0, c_i(L)]$. Then

$$\begin{aligned} Z^{-1}(C_1 \times \cdots \times C_n) &= \big(Z_1, \ldots, Z_n\big)^{-1}(C_1 \times \cdots \times C_n) \\ &= \{\omega \in \Omega : Z_i(\omega) \in C_i \quad \forall i \in [n]\} \qquad [= \textstyle\bigcap_{i=1}^{n} Z_i^{-1}(C_i)] \\ &= \{\omega \in \Omega : \mathsf{proj}_i^{-1}(\omega) \equiv \omega_i \in \zeta_i^{-1}(C_i) \quad \forall i \in [n]\} \\ &= \prod_{i=1}^{n} \zeta_i^{-1}(C_i). \end{aligned} \tag{2}$$

Then the definition of $\mathbb{P}$ implies that the distribution of Z, $\mathbb{P}_Z$, is such that

$$\begin{aligned} \mathbb{P}_Z(C_1 \cdots C_n) &= \mathbb{P}\big(Z^{-1}(C_1 \cdots C_n)\big) \\ &= \mathbb{P}\big(\textstyle\bigcap_{i=1}^{n} Z_i^{-1}(C_i)\big) = \mathbb{P}\big(\prod_{i=1}^{n} \zeta_i^{-1}(C_i)\big) =_{(*)} \prod_{i=1}^{n} \mathbb{P}_i\big(\zeta_i^{-1}(C_i)\big) = \prod_{i=1}^{n} \mathbb{P}_{\zeta_i}(C_i), \end{aligned}$$

where equality $(*)$ follows by (1).

We will let $G_{\zeta_i}(t) = \mathbb{P}_{\zeta_i}\big((-\infty, t]\big)$ denote the **distribution function** of ζ_i. In the probabilistic setting, every $\omega \in \Omega$ is a **scenario** of execution, and $Z(\omega)$ is the corresponding **system demand realization** (contrast these definitions with their counterparts in the deterministic setting described above). Every execution scenario maps to a unique **system criticality level realization** through the function $\mathsf{crit} : \Omega \to [L]$, where

$$\mathsf{crit}(\omega) = \min\Big\{\ell \in [L] : Z_i(\omega) \in (0, c_i(\ell)] \text{ for all } i \in [n]\Big\}. \tag{3}$$

That crit is defined for all scenarios $\omega \in \Omega$ follows by monotonicity of $c_i(\ell)$ with respect to ℓ.

Fix $\ell \in [L]$. For a scenario $\omega \in \Omega$, by (3), $\mathsf{crit}(\omega) = \ell$ if there is *at least* one job, say J_i, such that $c_i(\ell - 1) < Z_i(\omega) \le c_i(\ell)$, while the remaining jobs are such that $Z_j(\omega) \le c_j(\ell)$. For $\ell \in [L]$, by independence of job demands,

$$\mathbb{P}(\mathsf{crit} \le \ell) = \mathbb{P}\big(\textstyle\bigcap_{i=1}^{n}\{Z_i \le c_i(\ell)\}\big) = \prod_{i=1}^{n} G_{\zeta_i}\big(c_i(\ell)\big).$$

Since every scenario has a unique criticality level,

$$\mathbb{P}(\mathsf{crit} \le \ell) = \sum_{k=1}^{\ell} \mathbb{P}(\mathsf{crit} = k).$$

Therefore,

$$\mathbb{P}(\mathsf{crit} = \ell) = \mathbb{P}(\mathsf{crit} \le \ell) - \mathbb{P}(\mathsf{crit} \le \ell - 1) = \prod_{i=1}^{n} G_{\zeta_i}\big(c_i(\ell)\big) - \prod_{i=1}^{n} G_{\zeta_i}\big(c_i(\ell - 1)\big),$$

with the convention that $c_i(0) = 0$. Specializing to the dual criticality case, where $\textsc{Lo} \equiv 1$ and $\textsc{Hi} \equiv 2$, we have

$$\mathbb{P}(\mathsf{crit} = \textsc{Lo}) = \prod_{i=1}^{n} G_{\zeta_i}\big(c_i(\textsc{Lo})\big), \qquad \mathbb{P}(\mathsf{crit} = \textsc{Hi}) = 1 - \prod_{i=1}^{n} G_{\zeta_i}\big(c_i(\textsc{Lo})\big). \tag{4}$$

Now we recast the definitions made earlier in terms of scenarios spaces, random variables, and the functions that we have just defined.

A **scheduling policy** is a rule that at every time instant decides which job, from the set of available jobs (those that have not finished execution), is assigned the processor. At every time instant, a scheduling policy may use the characterizing parameters of all jobs, as well as its previous decisions, in making its next job allocation decision.

▶ **Definition 2** (Correct MC-Schedulability)**.** A policy π is said to **correctly MC-schedule** an instance $I = (J_1, \ldots, J_n; L)$ if for every scenario $\omega \in \Omega$, every J_i with $\chi_i \geq \mathsf{crit}(\omega)$ receives $Z_i(\omega)$ units of execution during $[0, d_i]$ under π.

We stress again that this definition does not require that jobs whose criticality is less than that of the realized system criticality level be given any execution; in fact, we will consider doing so as an undesired allocation scheme that is wasting the processor utilization.

▶ **Definition 3** (MC-Feasibility, Classical)**.** An instance $I = (J_1, \ldots, J_n; L)$ is MC-feasible if there is an online (non-clairvoyant) scheduling policy π under which I is correctly MC-schedulable.

Since our setting is probabilistic, we will be concerned with the notions of probabilistic feasibility and *expected* WTF-optimality. We defer the formal definitions of these notions until we have precisely defined the stochastic process induced by a policy, and the underlying probability space over which the expectation is taken (Definitions 4 and 5 in section 3.4).

3 Problem Definition: Integer Demands and Dual Criticalities

We consider a specialization of the setting discussed in the previous section, in which all demand random variables are integer-valued[2], and the system is dual-criticality. We are given two error parameters: One is a lower bound on the probability that all n jobs finish at or before their deadlines if the system criticality level is realized as Lo, and the other is a lower bound on the probability that Hi-criticality jobs meet their deadlines if the system criticality level is realized as Hi. We are required to compute a scheduling policy that minimizes, in expectation, the time wasted scheduling Lo-criticality jobs if the system criticality level turns out to be Hi, while respecting the deadline miss constraints. That is, we want to compute a policy that minimizes the WTF for the given instance, while respecting the timeliness constraints given by the error parameters. We will make the definition of deadline miss probability precise in sections to follow. Formally, the demand becomes the random variable

$$\zeta_i : \Omega_i \to \{1, 2, \ldots, c_i(\text{Lo}), c_i(\text{Lo}) + 1, \ldots, c_i(\chi_i)\}.$$

Accordingly, all demand realizations are integers, and we will therefore consider scheduling at integer boundaries. We shall assume that a job system is MC if not all the input jobs have the same criticality.

3.1 MDP Setup

Let $\mathsf{Y} = \{0 : \mathsf{finished}, 1 : \mathsf{not\ finished}\}^n$, and let $y_t \in \mathsf{Y}$ be the following variable (n-component vector): $y_t^i = 1$ iff job J_i still requires execution at time t, and $y_t^i = 0$ iff J_i has finished execution. At time 0, all jobs require execution, so we shall assume that $y_0 = \mathbf{1}$, the vector of all 1s. The

[2] One may equally well work with rational times by regarding time as being divided into integer multiples of some fixed rational quantum $q > 0$, and using scaling arguments to convert to integers.

evolution of the system state depends on the policy, so will specify precisely how the system evolves after formally defining policies.

Let A be the set of control actions (here jobs) available to the scheduler. We will let $\mathsf{A} = \{e_1, \ldots, e_n\} \cup \{\mathbf{0}\}$, where $\mathbf{0}$ is the vector of all 0s, and $\{e_1, \ldots, e_n\}$ is the standard basis for $\mathbb{R}^n$; e_i is the unit vector that is 1 at the ith coordinate and 0 elsewhere. If the action taken at time $t \in \mathbb{Z}_+$ is $a_t \in \mathsf{A}$, then $a_t = e_i$ means that job J_i occupies the processor during $[t, t+1]$. If $a_t = \mathbf{0}$, then no job is scheduled and the processor is kept idle. Let $x_t = (x_t^1, \ldots, x_t^n)$ encode the amount of execution time that every job has been allocated up to the beginning of the tth epoch (before acting at time t); that is, $x_0 = \mathbf{0}$ and $x_t = \sum_{m=0}^{t-1} a_m$ for $t \in \mathbb{N}$. Then for every $t \in \mathbb{Z}_+$ and $i \in [n]$, $x_t^i \in \mathsf{X}_i = \{0, 1, \ldots, c_i(\chi_i)\}$. We will let $\mathsf{X} = \prod_{i=1}^n \mathsf{X}_i$. We will utilize a variable r_t to "mark" the state as "error". An error flag stamped on a state signifies a deadline miss. r_t assumes values in

$$\mathsf{R} = \{\mathsf{not\ error}, \mathsf{potential\ error}, \mathsf{error}, \mathsf{error}'\}.$$

The **state** of the scheduling system at time $t \in \mathbb{Z}_+$ is $s_t = (t, y_t, x_t, r_t) \in \mathsf{S}$, where $\mathsf{S} \subset \{0, \ldots, N\} \times \mathsf{Y} \times \mathsf{X} \times \mathsf{R}$. The rationale behind our choice of this particular design of system state will become clear during the derivation of the state process below. For now, we mention how each element comprising our state representation achieves a desirable merit we seek in the system state:

- t: The main reason we include time is that we want to encode job finish times in the state, because we will identify "error" states as those where some job's finish time exceeds its deadline. As a byproduct, augmenting the state with time will result in time-homogeneous (stationary) state transition dynamics (Hernández-Lerma [25], p. 13);
- y_t: Implements the idea that a job *signals* that it has finished execution; this is the only state element that we *observe*, the others we *set* according to y_t;
- x_t: Summarizes all we need to know about the decisions we have made (allocations) up to time t, thus eliminating the need to include all actions up to time t. This is the key to ensure that the state process, which we derive below, is a *Markov chain*;
- r_t: One case where a state s_t becomes error is if some HI-criticality job $i \in \mathcal{I}_{\text{HI}}$ has just missed its deadline, which happens when $t = d_i$ and J_i still requires execution ($y_t^i = 1$). In this case, we will set $r_t = \mathsf{error}$. When $r_t = \mathsf{error}$, we will set $r_{t'}$ to error' for all $t' > t$; we do so to avoid charging the trajectory of execution more than once if more then one job miss their deadlines (see (17) and the discussion thereafter). Another possible error scenario is that when some LO-criticality job $i \in \mathcal{I}_{\text{LO}}$ has just missed its deadline ($t = d_i$) and still demands execution ($y_t^i = 1$), and the system criticality level realization is inferred as LO at or before t; that is, all HI-criticality jobs have already finished execution with LO demand realizations ($y_t^j = 0$ and $x_t^j \leq c_j(\text{LO})$ for all $j \in \mathcal{I}_{\text{HI}}$). However, those are not the only cases where the state becomes error. Consider the more subtle situation where no job has missed its deadline prior to time t, and s_t is such that there is $i \in \mathcal{I}_{\text{LO}}$ that just missed its deadline ($t = d_i$ and $y_t^i = 1$), but the scenario's criticality level realization is not yet determinable; in terms of our control variables, there is a non-empty $F \subset \mathcal{I}_{\text{HI}}$, possibly all of $\mathcal{I}_{\text{HI}}$, such that every job j in F has executed for at most $c_j(\text{LO}) - 1$, and none of the jobs in F have finished execution ($y_t^j = 1$ and $x_t^j < c_j(\text{LO})$ for all $j \in F$), while the other $k \in \mathcal{I}_{\text{HI}} \setminus F$, if any, have finished already with LO demand realizations ($y_t^k = 0$ and $x_t^k \leq c_k(\text{LO})$ for all $k \in \mathcal{I}_{\text{HI}} \setminus F$). In this case, the LO-criticality job J_i that just missed its deadline does not drive the system into an error state at time t since, by our definition of MC feasibility, this cannot be decided until we know the execution scenario's criticality level realization with certainty, which here depends on the (yet unknown) demand realizations of the jobs in F. In such case, we will say that the system is *potentially* in error state at time t, and we set $r_t = \mathsf{potential\ error}$ to "remember" that a LO-criticality job has

missed its deadline at t. Doing so gives us the facility to decide later whether or not the system is in error state—as soon as the scenario's criticality level realization is inferred—and, as a consequence, deduct the penalties correctly in the MDP.

For $t \in \mathbb{N}$, let $(\mathsf{S} \times \mathsf{A})^t$ be the Cartesian product of $\mathsf{S} \times \mathsf{A}$ with itself t times. Define the set of **admissible histories** up to time t as $H_0 = \mathsf{S}$, and

$$H_t = (\mathsf{S} \times \mathsf{A})^t \times \mathsf{S} \qquad (t \in \mathbb{N}).$$

Every element of H_t is called a ***t*-history**, and has the form

$$h_t = (s_0, a_0, \ldots, s_{t-1}, a_{t-1}, s_t).$$

t-histories are the information available to the scheduler before making its job selection decision at time t.

Let $\mathsf{A}(s_t) \subset \mathsf{A}$ be the set of actions that the scheduler is allowed to apply at time t when the scheduling system is in state s_t. We shall call $\mathsf{A}(s_t)$ the set of **admissible actions** in state s_t. A **scheduling policy** is a sequence $\pi = \{\pi_t : t \in \mathbb{Z}_+\}$, where π_t is a stochastic kernel on $\mathsf{A}(s_t)$ given H_t. That is, if we denote the power set of a set X as 2^X, then $\pi_t \equiv \pi_t(da_t \mid h_t)$, where $\pi_t : 2^{\mathsf{A}(s_t)} \times H_t \to [0,1]$ is such that

(i) for every $B \in 2^{\mathsf{A}(s_t)}$, $\pi_t(B \mid \cdot)$ is a function from H_t to $[0,1]$, and

(ii) for every $h_t \in H_t$, $\pi_t(\cdot \mid h_t) : 2^{\mathsf{A}(s_t)} \to [0,1]$ is a probability measure on $\mathsf{A}(s_t)$.

The state s_t summarizes all allocation decisions and remaining demands up to time t. We will restrict our attention to *Markov* policies, where $\pi_t(a_t|h_t) = \pi_t(a_t|s_t)$ for every h_t ([26] Definition 2.3.2 a).

A Note on Terminology: Since our state and action spaces are finite, all the stochastic (transition) kernels here can be represented by *transition matrices*. In this article, however, we will not use any of the matrix algebra machinery used to analyze Markov chains, so we will present our framework in the language of stochastic kernels.

A **work-conserving** scheduling policy always schedules a job that still demands execution; i.e., it never keeps the processor idle whenever there is a job that has not finished execution. Thus a policy is non-work-conserving iff there is $t \in \mathbb{Z}_+$ such that $a_t = \mathbf{0}$ (no job is selected) and there is $i \in [n]$ such that J_i has not finished execution; i.e., $y_t^i = 1$. The epoch $N = \sum_{i=1}^n c_i(\chi_i)$ is an upper bound on our planning horizon. With N fixed, any trajectory induced by executing a work conserving policy satisfies 1) $\sum_{i=1}^n x_t^i = t$ for every $t \in \{0, \ldots, N\}$ for which $y_t^i = 1$ for some $i \in [n]$, and 2) $x_t = x_T$ for all $t \in \{T, \ldots, N\}$, where T is the first time instant at which all jobs finish execution. A non-work-conserving schedule will only delay job completions and the time at which the criticality level realization can be inferred, so we restrict ourselves to work-conserving policies.

We implement the requirement that the scheduling policy be work-conserving by specifying that $\mathsf{A}(s_t)$ includes only vectors e_i for which $y_t^i = 1$, if any. We will *drop* Lo-criticality jobs (temporarily) as soon as the state s_t indicates that the operational system criticality level is Hi, and we will enforce this by placing further restrictions on $\mathsf{A}(s_t)$. Namely, given state s_t, if there is $i \in \mathcal{I}_{\text{Hi}}$ such that both $x_t^i \geq c_i(\text{Lo})$ and $y_t^i = 1$, then the operational system criticality level at time t is Hi and there are Hi-criticality jobs still requiring execution, so we exclude from $\mathsf{A}(s_t)$ all Lo-criticality jobs. Otherwise, we include all Lo-criticality jobs that have not finished yet. If s_t does not satisfy the latter condition, then either the system criticality level realization cannot be inferred at t, or all Hi-criticality jobs finished with Lo demand realizations before or at t, or the

system criticality level was known before or at t as HI, but all HI-criticality jobs have finished execution at t. In the last case, we might have dropped LO-criticality jobs earlier, and, since scheduling LO-criticality jobs at time t in this case is not considered WTF (and does not affect feasibility), we may bring back any LO-criticality jobs that still need to execute. In summary,

$$\mathsf{A}(s_t) = \begin{cases} \{e_i : y_t^i = 1, \chi_i = \text{HI}\} & (*) \text{ if there is } i \in \mathcal{I}_{\text{HI}} \text{ such that } x_t^i \geq c_i(\text{LO}) \text{ and } y_t^i = 1 \\ \{e_i : y_t^i = 1\} & \text{if } (*) \text{ not satisfied and there is } i \in [n] \text{ such that } y_t^i = 1 \\ \{\mathbf{0}\} & \text{otherwise (all jobs finished execution).} \end{cases}$$

Control Model

Scheduling decisions are made at every $t \in \{0, \ldots, N-1\}$ exclusively. If a certain job is chosen to execute at some t, then this job occupies the processor for the duration $[t, t+1]$, without interruption, until the scheduler is invoked again at $t+1$. We call $[t, t+1]$ the tth **control interval**. At any $t > 0$, if job J_i was chosen to occupy the processor during $[t-1, t]$ (i.e., $a_{t-1} = e_i$), then the scheduler knows at time t whether or not job J_i requires more execution by observing the value of y_t^i, which will be set to $\mathsf{finished}$ if job J_i signals that it has finished execution at time t. The other jobs' demands are not affected by scheduling job J_i, and whether or not the other jobs require more execution does not change in $[t-1, t]$. The information available to the scheduler at the beginning of the tth control interval is a_{t-1}, y_t, x_t, and r_t.

Let $s = (t, y, x, r)$ and $\hat{s} = (\hat{t}, \hat{y}, \hat{x}, \hat{r})$. It is necessary for transition $(s, a, \hat{s})$ to be valid that all the following be satisfied:

$$\begin{aligned} \mathbf{NC}: \quad & \hat{t} = t + 1, \\ & \sum_{i=1}^{n} x^i = t, \\ & a = \hat{x} - x = e_i \text{ for some } i \in [n], \text{or } a = \hat{x} - x = \mathbf{0} \\ & y - \hat{y} \in \{\mathbf{0}, e_i\} \text{ for the } \textit{same } i, \text{ and} \\ & (r, \hat{r}) \notin \big\{(\mathsf{error}, \mathsf{not\ error}), (\mathsf{error}, \mathsf{potential\ error}), (\mathsf{not\ error}, \mathsf{error}'), \\ & \qquad (\mathsf{potential\ error}, \mathsf{error}'), (\mathsf{error}', \mathsf{r})\ \forall \mathsf{r} \in \mathsf{R} \setminus \{\mathsf{error}'\}\big\}. \end{aligned}$$

However, not all state transitions satisfying **NC** are valid, as we will describe below. All invalid state transitions have $Q(\{\hat{s}\} \mid s, a) = 0$, however. To this end, we note that the state includes all the information necessary to determine whether or not the system criticality level is inferred, and if so, determine its value. To simplify the exposition, we define a function $\mathsf{critState} : \mathsf{S} \to \{\text{LO}, \text{HI}, \text{UNKNOWN}\}$, where $\mathsf{critState}(s)$ is the system criticality level realization, and is defined as follows: For $s = (t, x, y, r)$,

(1) $\mathsf{critState}(s) = \text{LO}$ if all HI-criticality jobs finished execution with LO demand realizations; that is, if $y^i = 0$ and $x^i \leq c_i(\text{LO})$ for all $i \in \mathcal{I}_{\text{HI}}$;

(2) $\mathsf{critState}(s) = \text{HI}$ if either

- **(i)** there is $i \in \mathcal{I}_{\text{HI}}$ such that $x^i(\text{LO}) = c_i(\text{LO})$ and $y^i = 1$, or
- **(ii)** there is $i \in \mathcal{I}_{\text{HI}}$ such that $x^i(\text{LO}) > c_i(\text{LO})$;

(3) If neither of the above holds, then $\mathsf{critState}(s) = \text{UNKNOWN}$.

Now assuming transition $(s, a, \hat{s})$ satisfies **NC**, we will use monotonicity of $t \mapsto x_t$ and $t \mapsto y_t$, and that s_0 is fixed, to list additional conditions regarding the error flags under which $(s, a, \hat{s})$ is a valid transition in an exact sense. In what follows, for a state $s = (t, x, y, r)$ and $\ell \in \{\text{LO}, \text{HI}\}$, the statement "an ℓ-criticality job misses its deadline at time t" is to be understood formally as "there is $i \in \mathcal{I}_\ell$ such that $d_i = t$ and $y^i = 1$ ($\mathsf{not\ finished}$)."

E1. $(r = \mathsf{not\ error}, \hat{r} = \mathsf{potential\ error})$: If all of the following conditions hold:
- **(i)** no Hi-criticality job misses its deadline at time $\hat{t} = t + 1$,
- **(ii)** a Lo-criticality job misses its deadline at time $\hat{t}$, and
- **(iii)** the system criticality level is not yet determinable at $\hat{t}$; that is, $\mathsf{critState}(\hat{s}) = \textsc{Unknown}$ ($r = \mathsf{no\ error}$ says that no Hi-criticality jobs missed their deadlines up to time t);

E2. $(r = \mathsf{not\ error}, \hat{r} = \mathsf{error})$: Either
- **(i)** a Hi-criticality job misses its deadline at time $\hat{t}$, or
- **(ii)** a Lo-criticality job misses its deadline at time $\hat{t}$ *and* $\mathsf{critState}(\hat{s}) = \textsc{Lo}$;

E3. $(r = \mathsf{potential\ error}, \hat{r} = \mathsf{error})$: Same as 2, except that we dispense with the condition in 22ii that a Lo-criticality job misses its deadline at time $\hat{t}$. $r = \mathsf{potential\ error}$ is saying that no Hi-criticality job missed its deadline till t, and the criticality level could not be inferred till t, but a Lo-criticality job has missed its deadline already;

E4. $(r = \mathsf{potential\ error}, \hat{r} = \mathsf{potential\ error})$: Same as conditions (i) + (iii) of E1;

E5. $(r = \mathsf{potential\ error}, \hat{r} = \mathsf{not\ error})$: If $\mathsf{critState}(\hat{s}) = \textsc{Hi}$ and no Hi-criticality job misses its deadlines at time $\hat{t} = t + 1$;

E6. $(r = \mathsf{not\ error}, \hat{r} = \mathsf{not\ error})$: The combined conditions of $(r = \mathsf{not\ error}, \hat{r} \neq \mathsf{potential\ error})$ and $(r = \mathsf{not\ error}, \hat{r} \neq \mathsf{error})$;

E7. $(r = \mathsf{error}, \hat{r} = \mathsf{error}')$: always;

E8. $(r = \mathsf{error}', \hat{r} = \mathsf{error}')$: always.

The following summarizes the control model:

1. At $t = 0$, all jobs are ready to execute and they all demand execution, and the scheduler needs to pick a job to schedule for exactly one time unit before it is invoked again at $t = 1$ (i.e., a_0 needs to be set). Then $y_0 = \mathbf{1}$, $x_0 = \mathbf{0}$, and $r_0 = \mathsf{no\ error}$;
2. At the beginning of the tth control interval:
   - **2.1 Update** the cumulative system allocation by setting $x_t \leftarrow x_{t-1} + a_{t-1}$;
   - **2.2 Observe** (acquire) y_t;
   - **2.3 Set Error:** Set r_t given r_{t-1} according to one of E1–E8;
   - **2.4 Act**: Set a_t to one of the vectors in $\mathsf{A}(s_t)$.

We will say that a state is **valid** if it can be generated by the control model above. The state space S contains only the valid states; i.e., S is the subset of $\{0, \ldots, N\} \times \mathsf{X} \times \mathsf{Y} \times \mathsf{R}$ that can be generated by the control model. For instance, if $s = (t, x, y, r)$ is such that $t = d_i + 1$ and $y^i = 1$ ($\mathsf{not\ finished}$) for some $i \in [n]$, and $r = \mathsf{not\ error}$, then for no x is s is valid, even if x is such that $\sum_{j=1}^{n} x^j = t$ (necessary for a state to be valid) and $x^j < c_j(\chi_j)$ for all j.

We point out that the state transition diagram has the simple structure of a directed tree of depth at most N (the maximum horizon length), with fixed root s_0, and each node in level t, $t \in \{0, \ldots, N\}$, corresponds to a possible state at time t (i.e., s_t.) Each intermediate node (state) has at most $|\mathsf{A}||\mathsf{Y}| = 2n$ children, each corresponding to a unique current action and next finish signal pair (a_t, y_{t+1}). Note that the next cumulative allocation vector, x_{t+1}, and the next error flag, r_{t+1}, are deterministic once we know a_t and y_{t+1}, so there is only one choice for each given s_t and a_t.

3.2 The Transition Probabilities

We describe the evolution of the system state by a transition kernel (transition matrix) $Q(d\hat{s}|s, a) : 2^{\mathsf{S}} \times (\mathsf{S} \times \mathsf{A}) \to [0, 1]$. Since our state space S is finite, transition kernel Q should satisfy the following for *fixed* action a and previous state s,

Q1. $Q(\emptyset|s, a) = 0$;

Q2. $Q(\mathsf{S}|s, a) = 1$;

Q3. $Q(U|s,a) = \sum_{\hat{s}\in U} Q(\{\hat{s}\}|s,a)$ for every $U \subset \mathsf{S}$;
Q4. $0 \leq Q(U|s,a) \leq Q(V|s,a) \leq 1$ for every $U \subset V \subset \mathsf{S}$.

We shall abuse notation and write $Q(\hat{s}|s,a)$ for $Q(\{\hat{s}\}|s,a)$. Let $(s,a,\hat{s})$ be a valid transition. If $s = (t,y,x,r)$, then $\hat{t} = t+1$, and we shall use the more time-suggestive notation $\hat{s} \equiv s_{t+1}$, where $\hat{y} \equiv y_{t+1}$, $\hat{x} \equiv x_{t+1}$ and $\hat{r} \equiv r_{t+1}$, and we similarly denote s as s_t. If $(s,a,\hat{s})$ is not valid, then $Q(\hat{s}|s,a) = 0$. Fix an action $a_t = e_i$. Then for transition (s_t, e_i, s_{t+1}) to be valid, we must have $y_t^i = 1$ and $x_{t+1}^i = x_t^i + 1$. Also, scheduling J_i does not affect the execution time demands of the other jobs, so s_{t+1} must satisfy $y_t^j = y_{t+1}^j$ for every $j \neq i$. For our fixed state-action pair (s_t, e_i), we know at time t that $Z_i > x_t^i$, and for $j \neq i$, $Z_j \in C_j$ for some $C_j \subset [c_j(\chi_j)]$. The following is a complete list of all the possible next states s_{t+1} and the corresponding transition probabilities for the fixed action-state pair $(s_t, a_t = e_i)$ (it is here where we fully utilize the assumption of independent job demands):

- $y_{t+1}^i = 1$ (not finished): This says that the scenario ω is such that $Z_i(\omega) > x_{t+1}^i = x_t^i + 1$, and since $Z_j(\omega)$ remains in C_j for every $j \neq i$ at time $t+1$, we have

$$\begin{aligned} Q(s_{t+1}|s_t,e_i) &= \mathbb{P}(Z_i > x_t^i + 1, Z_j \in C_j \ \forall j \neq i \mid Z_i > x_t^i, Z_j \in C_j \ \forall j \neq i) \\ &= \frac{\mathbb{P}(Z_i > x_t^i + 1, Z_j \in C_j \ \forall j \neq i)}{\mathbb{P}(Z_i > x_t^i, Z_j \in C_j \ \forall j \neq i)} \\ &= \frac{\mathbb{P}(Z_i > x_t^i + 1)\mathbb{P}(Z_j \in C_j \ \forall j \neq i)}{\mathbb{P}(Z_i > x_t^i)\mathbb{P}(Z_j \in C_j \ \forall j \neq i)} \\ &= \frac{\mathbb{P}(Z_i > x_t^i + 1)}{\mathbb{P}(Z_i \geq x_t^i + 1)} \end{aligned} \tag{5}$$

  if $x_t^i < c_i(\chi_i) - 1$, and $Q(s_{t+1}|s_t,e_i) = 0$ otherwise. The second to last equality follows by independence of job demands.
- $y_{t+1}^i = 0$ (finished): Here the demand of job J_i is realized at time $t+1$; that is, we know that the scenario ω is such that $Z_i(\omega) = x_{t+1}^i = x_t^i + 1$. Using the same reasoning as in the previous case,

$$Q(s_{t+1}|s_t,e_i) = \begin{cases} \frac{\mathbb{P}(Z_i = x_t^i+1)}{\mathbb{P}(Z_i \geq x_t^i+1)} & \text{if } x_t^i < c_i(\chi_i) - 1, \\ 1 & \text{if } x_t^i = c_i(\chi_i) - 1 \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

Then for fixed $(s_t, a_t = e_i)$, summing over all possible next states; i.e., adding (5) and (6), we have

$$\frac{\mathbb{P}(Z_i > x_t^i + 1) + \mathbb{P}(Z_i = x_t^i + 1)}{\mathbb{P}(Z_i \geq x_t^i + 1)} = \frac{\mathbb{P}(Z_i \geq x_t^i + 1)}{\mathbb{P}(Z_i \geq x_t^i + 1)} = 1.$$

That is, our prescribed transition kernel Q satisfies property Q2, and indeed all the others.

3.3 The Underlying Probability Space

In this section we will outline in detail the construction of the probability space that we will be working with. We will shift our attention from scenario spaces (the Ω_is, section 2) to *trajectory spaces*, which consist of the sample paths induced by executing policies. We will need this construction when stating the formal definition of our problem, and we shall make several references to it. Readers acquainted with the theory of Markov decision processes may only wish to familiarize themselves with our notation.

Consider the product space

$$(\mathsf{S} \times \mathsf{A})^\infty = \prod_{t=0}^{\infty} (\mathsf{S}_t \times \mathsf{A}_t),$$

where $\mathsf{S}_0 = \{s_0\}$, $s_0 \equiv (t = 0, x_0 = \mathbf{0}, y_0 = \mathbf{1}, r_0 = \mathsf{not\ error})$ is our *fixed* initial state (all jobs are allocated 0 execution time and they all require execution,) $\mathsf{S}_t \subset \{t\} \times \mathsf{Y} \times \mathsf{X} \times \mathsf{R}$, and A_t is a copy of A. We will consider the subset of $(\mathsf{S} \times \mathsf{A})^\infty$ where each sequence of state-action pairs can be generated by our control model, and we will call such sequences the **valid trajectories**. We denote the set of valid trajectories as H_∞, and we call H_∞ the **trajectory space** induced by all work-conserving scheduling policies. Every $h \in H_\infty$ is a trajectory induced by executing some work-conserving policy, and is of the form

$$h = (s_0, a_0, s_1, a_1, \dots).$$

That is, every h is a realization of a schedule. We endow H_∞ with the product σ-algebra, which we denote as $\mathcal{H}_\infty$. Let $S_t : H_\infty \to \mathsf{S}$ be the projection (coordinate) map on H_∞ such that $S_t(h) = s_t$, $h \in H_\infty$. Define $A_t : H_\infty \to \mathsf{A}(s_t)$ similarly. Given policy $\pi = \{\pi_t : t \in Z_+\}$, transition kernel Q, and initial distribution ν on S, the Ionescu-Tulcea extension theorem ([32], Theorem 14.32; [5], Theorem 2.7.2) asserts that there exists a *unique* probability measure $\mathbb{P}^\pi_\nu$ on H_∞ such that

$$\mathbb{P}^\pi_\nu\big(S_t \in U \mid h_{t-1}, a_{t-1}\big) = Q\big(U | s_{t-1}, a_{t-1}\big) \qquad (U \subset \mathsf{S}).$$

We have $\nu = \delta_{s_0}$ for $s_0 = (0, \mathbf{0}, \mathbf{1}, \mathsf{not\ error})$, where δ_{s_0} is Dirac measure on H_∞, so for brevity we write $\mathbb{P}^\pi \equiv \mathbb{P}^\pi_{\delta_{s_0}}$. We denote expectation with respect to $\mathbb{P}^\pi$ (on H_∞) as $\mathbb{E}^\pi$. Moreover, because every policy is Markov as mentioned above, it follows that the induced state process $\{S_t : t \in \mathbb{Z}_+\}$ is a Markov chain for every policy π. That is, for every $U \subset \mathsf{S}$ and $t \in \mathbb{Z}_+$,

$$\mathbb{P}^\pi\big(S_{t+1} \in U \mid s_t, \dots, s_0\big) = \mathbb{P}^\pi\big(S_{t+1} \in U \mid s_t\big) = Q\big(U | s_t, \pi_t\big),$$

where for fixed s_t,

$$Q\big(U | s_t, \pi_t\big) = \int_{\mathsf{A}} Q\big(U | s_t, a_t\big)\, \pi_t(da_t | s_t) = \sum_{i=1}^{n} Q\big(U | s_t, e_i\big)\, \pi_t(e_i | s_t).$$

From now on, all subsequent random variables will be defined on H_∞.

▶ Remark. For a given s_t, each action random variable A_t of the induced action process $\{A_t : t \in \mathbb{Z}_+\}$ is distributed according to $\pi_t(\,\cdot \mid s_t)$; that is, $\mathbb{P}^\pi(A_t \in C \mid S_t = s_t) = \pi_t(\,C \mid s_t)$ for every $C \subset \mathsf{A}(s_t)$.

3.4 Problem Statement

We start by formally defining the random variables that make up our objective function and constraints, in terms of the induced MDP. First, we note that to every trajectory corresponds at least one scenario in Ω, and that all scenarios that correspond to a trajectory have the same criticality level. Consequently, we define the criticality level of a trajectory as the criticality level of any scenario corresponding to it[3].

[3] To ensure that the trajectory criticality level is well-defined, we must assume that every policy assigns every HI-criticality job the processor for at least $c_i(\text{LO})$ time units. However, this is readily satisfied by every policy since, by the way we specified the set of admissible actions $\mathsf{A}(s_t)$, HI-criticality jobs are never dropped.

To this end, let $\mathcal{I}_\ell = \{i \in [n] : \chi_i \geq \ell\}$. Let $h \in H_\infty$ be a trajectory of execution. If h's criticality level is HI, then the earliest time at which this can be inferred is the first instant at which some job J_i with $\chi_i = \text{HI}$ demands more than $c_i(\text{LO})$ units of execution. We denote this random time as $T_{\text{HI}} : H_\infty \to \mathbb{N} \cup \{\infty\}$ and, in accordance with our control model, we define it as

$$T_{\text{HI}} = \min\big\{t \in \mathbb{N} : X_t^i = c_i(\text{LO}) \text{ and } Y_t^i = 1 \text{ (not finished) } \text{ for } \textit{some } i \in \mathcal{I}_{\text{HI}}\big\}, \tag{7}$$

with the usual convention $\min \emptyset = \infty$.

If the trajectory's criticality level is LO, then this can be inferred with certainty only at the first instant at which *all* HI-criticality jobs finish execution. We denote this random time instant as $T_{\text{LO}} : H_\infty \to \mathbb{N} \cup \{\infty\}$, and we define it as

$$T_{\text{LO}} = \min\big\{t \in \mathbb{N} : Y_t^i = 0 \text{ (finished) } \text{ and } \ X_t^i \leq c_i(\text{LO}) \text{ for } \textit{all } i \in \mathcal{I}_{\text{HI}}\big\}. \tag{8}$$

We note that the events $\{T_{\text{LO}} < \infty\}$ and $\{T_{\text{HI}} < \infty\}$ are mutually exclusive, since a trajectory of execution cannot be both LO and HI criticality. This makes the events $\{T_{\text{LO}} = \infty\}$ and $\{T_{\text{HI}} = \infty\}$ mutually exclusive, since every trajectory *must* have a criticality. There are therefore exactly two possibilities, and one of them must occur: Either $\{T_{\text{LO}} < \infty\}$ and $\{T_{\text{HI}} = \infty\}$, or $\{T_{\text{LO}} = \infty\}$ and $\{T_{\text{HI}} < \infty\}$, exclusively. Accordingly, we define the T_{CI} of a trajectory under policy π as the random (finite) time

$$T_{\text{CI}} = \min(T_{\text{HI}}, T_{\text{LO}}).$$

Recalling the manner in which we specified the set of admissible actions $\mathsf{A}(s_t)$, every policy drops LO-criticality jobs as soon as the scenario's criticality level is realized as HI; moreover, only allocations made to LO-criticality jobs *prior to the* T_{CI} are considered as WTF, and this allocation is regarded as WTF only if the system criticality level realization is HI. Let $\mathsf{critPath}(h)$ denote criticality level realization of trajectory (path) $h \in H_\infty$; $\mathsf{critPath} : H_\infty \to \{\text{LO}, \text{HI}\}$. Then one way to define the WTF of a trajectory $h \in H_\infty$ is as the random variable

$$W(h) = \mathbb{1}_{\{\mathsf{critPath}=\text{HI}\}}(h) \sum_{i:\chi_i=\text{LO}} X^i_{T_{\text{CI}}}(h), \tag{9}$$

where $X^i_{T_{\text{CI}}}(h) \equiv X^i_{T_{\text{CI}}(h)}(h)$ is job J_i's total allocation sampled at T_{CI}. That is, for valid trajectory $h \in H_\infty$, $W(h) = 0$ if $\mathsf{critPath}(h) = \text{LO}$, and $W(h)$ is equal to the total allocation given to the LO-criticality jobs up to the T_{CI} if $\mathsf{critPath}(h) = \text{HI}$.

We define the **finish time** of job J_i with respect to policy π as the stopping time $F_i : H_\infty \to \mathbb{N}$, where

$$F_i = \min\big\{t \in \mathbb{N} : Y_t^i = 0 \text{ (finished)}\big\}.$$

Objective: A dual-criticality probabilistic MC (pMC) instance I is described by the tuple $\big(\{J_1, \ldots, J_n\}, \varepsilon_{\text{LO}}, \varepsilon_{\text{HI}}\big)$, where for every $i \in [n]$, J_i is specified by the tuple $\big((\Omega_i, \mathcal{M}_i, \mathbb{P}_i), \zeta_i, \chi_i, c_i, d_i\big)$. The error parameters ε_{LO} and ε_{HI} are the desired upper bounds on the deadline miss probabilities, and both are in the interval $[0, 1]$. We seek a scheduling policy π such that the expected WTF, $\mathbb{E}^\pi W$, is minimized, while simultaneously guaranteeing probabilistic MC-feasibility in the following sense: (1) Conditioned on $\mathsf{critPath} = \text{LO}$, a job, among all n jobs, may miss its deadline with probability at most ε_{LO}, and (2) conditioned on $\mathsf{critPath} = \text{HI}$, a HI-criticality job may miss its deadline with probability at most ε_{HI}.

We write our problem as the following CMDP:

$$\begin{aligned} \textbf{CMDP}: \underset{\pi \in \Pi}{\text{minimize}} \quad & \mathbb{E}^\pi W \\ \text{subject to} \quad & \mathbb{P}^\pi\big(\textstyle\bigcup_{i\in[n]}\{F_i > d_i\} \mid \mathsf{critPath} = \text{LO}\big) \quad \leq \quad \varepsilon_{\text{LO}} \\ & \mathbb{P}^\pi\big(\textstyle\bigcup_{i\in\mathcal{I}_{\text{HI}}}\{F_i > d_i\} \mid \mathsf{critPath} = \text{HI}\big) \quad \leq \quad \varepsilon_{\text{HI}}. \end{aligned} \tag{10}$$

We note that for any $\ell \in \{\text{Lo}, \text{Hi}\}$, $\{\mathsf{critPath} = \ell\} = \{T_{\text{CI}} = T_\ell\} = \{T_\ell < \infty\} \subset H_\infty$, and that $T^\pi_{\text{CI}} < \infty$ $\mathbb{P}^\pi$-almost surely for any work conserving policy π.

Having formalized the problem, we are now able to give precise definitions of what its means for an instance with probabilistic information to be feasible in the MC setting.

▶ **Definition 4** (Probabilistic MC-feasibility)**.** A pMC instance I is probably MC-feasible (pMC-feasible) if there is a policy π such that the constraints (10) are satisfied.

A scheduling policy under which pMC instance I is pMC-feasible is said to **correctly pMC-schedule** I. Let $\Pi(I)$ denote the set of scheduling policies that correctly pMC-schedule instance I.

▶ **Definition 5** (Expected WTF-Optimality)**.** A scheduling policy π is said to be WTF-optimal for pMC instance I in expectation (or expected-WTF-optimal) if $\pi \in \Pi(I)$ and $\mathbb{E}^\pi W \leq \mathbb{E}^{\pi'} W$ for all $\pi' \in \Pi(I)$.

Accordingly, given pMC instance I, our goal is to compute an expected WTF-optimal scheduling policy for I.

▶ Remark. For $\ell \in \{\text{Lo}, \text{Hi}\}$, $\mathbb{P}^\pi(\,\cdot \,|\, \mathsf{critPath} = \ell)$ is a probability measure on the restriction of H_∞ to trajectories of criticality ℓ. In the special case where H_∞ is finite and $\mathbb{P}^\pi(\,\cdot \,|\, \mathsf{critPath} = \ell)$ is uniform measure, the constraint $\mathbb{P}^\pi\big(F_i > d_i \text{ for some } i \in [n] \mid \mathsf{critPath} = \ell\big) \leq \varepsilon_\ell$ has the following simple interpretation: If we let $H_\infty(\ell)$ be the subset of H_∞ consisting of the trajectories whose criticality is ℓ, then the number of trajectories in $H_\infty(\ell)$ where all n jobs do not miss their deadlines is required to be at least $(1 - \varepsilon_\ell)|H_\infty(\ell)|$.

4 Solution Approach: Risk-Constrained MDP

The WTF as defined in (9) depends on the whole trajectory, so in its current form is not suitable in the MDP framework, where costs are accrued *per stage*. We wish to define the WTF as a sum, over the horizon, of functions that depend at each $t \in \mathbb{Z}_+$ on s_{t-1} and s_t only. To this end, we will use the following

▶ **Proposition 6.** Let $h = (s_0, a_0, s_1, a_1)$ be a valid trajectory (in H_∞). Then

(a) If $\mathsf{critState}(s_t) = \text{Unknown}$ for some $t \in \mathbb{N}$, then $\mathsf{critState}(s_m) = \text{Unknown}$ for every $m < t$; and

(b) If $\mathsf{critState}(s_t) = \ell$ for some $t \in \mathbb{N}$ and $\ell \in \{\text{Lo}, \text{Hi}\}$, then $\mathsf{critState}(s_{m'}) = \ell$ for every $m' > t$.

Proof. Follows readily from monotonicity of $t \mapsto X^i_t(h)$ and $t \mapsto Y^i_t(h)$ for every $i \in [n]$ and $h \in H_\infty$, together with the fact that $x_0 = \mathbf{0}$ and $y_0 = \mathbf{1}$, and that we consider only work conserving policies. ◀

We define the local (per stage) objective cost function $w : \mathsf{S} \times \mathsf{S} \to \mathbb{Z}_+$, where

$$w(s, \hat{s}) = \mathbb{1}\big\{\mathsf{critState}(s) = \text{Unknown}\big\}\mathbb{1}\big\{\mathsf{critState}(\hat{s}) = \text{Hi}\big\} \sum_{i:\chi_i = \text{Lo}} \hat{x}^i$$

for $s, \hat{s} \in \mathsf{S}$. That is, $w(s, \hat{s})$ is equal to the total Lo criticality allocation $\sum_{i:\chi_i=\text{Lo}} \hat{x}^i$ only if the criticality level realization is inferred as a consequence of moving from state s to state $\hat{s}$ and is Hi criticality; otherwise, $w(s, \hat{s}) = 0$. Since we are working with the set valid trajectories exclusively, we may use Proposition 6 to write the WTF as

$$W(h) = \sum_{t=0}^{N-1} w\big(S_t(h), S_{t+1}(h)\big).$$

4.1 The Risk Constraints

We will leverage the ideas of Geibel and Wysotzki [21] to carry out the transformation of the deadline miss probabilities into a single "risk" constraint that takes the form of the expectation of immediate costs.

We may write the first constraint in **CMDP** as

$$\mathbb{P}^\pi(F_i > d_i \text{ for some } i \in [n], \mathsf{critPath} = \textsc{Lo}) \leq \varepsilon_{\textsc{Lo}} \mathbb{P}^\pi(\mathsf{critPath} = \textsc{Lo}), \tag{11}$$

where

$$\mathbb{P}^\pi(F_i > d_i \text{ for some } i \in [n], \mathsf{critPath} = \textsc{Lo}) = \mathbb{P}^\pi\big(\{h \in H_\infty(\textsc{Lo}) : \exists t \in \mathbb{N} \text{ s.t. } r_t = \mathsf{error}\}\big). \tag{12}$$

Following Geibel and Wysotzki [21], constraint (11) defines the **risk** across the Lo-criticality trajectories associated with executing policy π. Similarly to the Lo-criticality case, we write the second constraint in **CMDP** as

$$\mathbb{P}^\pi(F_i > d_i \text{ for some } i \in \mathcal{I}_{\textsc{Hi}}, \mathsf{critPath} = \textsc{Hi}) \leq \varepsilon_{\textsc{Hi}} \mathbb{P}^\pi(\mathsf{critPath} = \textsc{Hi}), \tag{13}$$

where

$$\mathbb{P}^\pi(F_i > d_i \text{ for some } i \in \mathcal{I}_{\textsc{Hi}}, \mathsf{critPath} = \textsc{Hi}) = \mathbb{P}^\pi\big(\{h \in H_\infty(\textsc{Hi}) : \exists t \in \mathbb{N} \text{ s.t. } r_t = \mathsf{error}\}\big). \tag{14}$$

The trajectories

$$H_\infty^{\mathrm{er}} \equiv \big\{h = (s_0, a_1, s_1, \dots) \in H_\infty : \exists t \in \mathbb{N} \text{ s.t. } r_t = \mathsf{error}\big\}$$

are the **error trajectories** that we want to avoid with high probabilities.

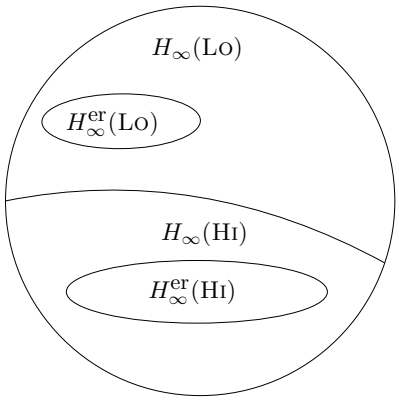


Figure 2 Every policy π induces a probability measure $\mathbb{P}^\pi$ on the trajectory space H_∞, where the latter consists of two disjoint sets: The Lo and Hi-criticality trajectories $H_\infty(\textsc{Lo})$ and $H_\infty(\textsc{Hi})$, respectively. The ovals are graphical representations of the error trajectory sets that we wish to avoid. Given a policy, say π, the "sizes" of the induced error sets $H_\infty^{\mathrm{er}}(\textsc{Lo})$ and $H_\infty^{\mathrm{er}}(\textsc{Hi})$ (relative to the entire trajectory space H_∞) are given by the unique probability measure $\mathbb{P}^\pi$. The smaller the size a policy assigns to the error subsets, the better it is at avoiding trajectories in them. Roughly speaking, each ℓth (criticality-specific) constraint in **ECMDP** is a restriction on the "size" of the corresponding error set $H_\infty^{\mathrm{er}}(\ell)$ *relative to the trajectories of the* ***same criticality***; i.e., relative to the size of $H_\infty(\ell)$ (and not to the whole trajectory space H_∞); hence the conditioning in the constraints.

We now combine the constraints in **CMDP**. Put $p_\ell = \varepsilon_\ell \mathbb{P}(\mathsf{crit} = \ell)$, $\ell \in \{\textsc{Lo}, \textsc{Hi}\}$, and let

$$\Delta = \min(p_{\textsc{Lo}}, p_{\textsc{Hi}}).$$

Since H_∞ is the disjoint union of $H_\infty(\textsc{Lo})$ and $H_\infty(\textsc{Hi})$, we may combine (12) and (14), and require the satisfaction of the more conservative risk

$$\mathbb{P}^\pi\big(H_\infty^{\mathrm{er}}\big) = \mathbb{P}^\pi\big(\{h \in H_\infty : h \text{ is error}\}\big) = \mathbb{P}^\pi\big(\exists t : R_t = \mathsf{error}\big) \le \Delta. \tag{15}$$

For then,

$$\mathbb{P}^\pi\big(H_\infty^{\pi,\mathrm{er}}(\ell)\big) \le \mathbb{P}^\pi\big(H_\infty^{\mathrm{er}}\big) \le \Delta = \min(p_{\textsc{Lo}}, p_{\textsc{Hi}}) \le p_\ell, \quad \text{for all } \ell \in \{\textsc{Lo}, \textsc{Hi}\}. \tag{16}$$

Next we write the risk constraint (15) as an expectation under $\mathbb{E}^\pi$ of immediate costs having a form similar to κ. Inspired by Geibel and Wysotzki [21], we define the per-stage *constraint cost* function $\kappa : \mathbb{K} \to \{0, 1\}$ as

$$\kappa(s, a) \equiv \kappa(s) = \begin{cases} 1 & \text{if } r = \mathsf{error}, \\ 0 & \text{otherwise.} \end{cases} \tag{17}$$

This way, if a trajectory $h = (s_0, a_0, s_1, a_1, \dots)$ is error, then since there is $t \in \mathbb{N}$ such that $r_t = \mathsf{error}$ and $r_{t'} = \mathsf{error}'$ for all $t' > t$, it follows that the sequence of constraint costs corresponding to this trajectory is such that

$$\kappa(s_0) = 0, \dots, \kappa(s_{t-1}) = 0, \underbrace{\kappa(s_t) = 1}, \kappa(s_{t+1}) = 0, \dots, \kappa(s_N) = 0.$$

If h is not error, then $\kappa(s_t) = 0$ for all $t \in \mathbb{Z}_+$. If we let $C = \sum_{t=0}^N \kappa(S_t)$, then $C \in \{0, 1\}$; that is, C is a Bernoulli random variable with probability of success $\mathbb{P}^\pi(C = 1)$. Success of this Bernoulli trial happens if there is $t \in \mathbb{N}$ such that $R_t = \mathsf{error}$. That is,

$$\big\{C = 1\big\} = \big\{\exists t : R_t = \mathsf{error}\big\},$$

from which it follows that

$$\mathbb{P}^\pi(C = 1) = \mathbb{P}^\pi\big(\exists t : R_t = \mathsf{error}\big).$$

Since C is Bernoulli, it follows that

$$\mathbb{E}^\pi C = \mathbb{P}^\pi(C = 1),$$

from which we may write the risk constraint as

$$\mathbb{P}^\pi\big(\exists t : R_t = \mathsf{error}\big) = \mathbb{P}^\pi\big(C = 1\big) = \mathbb{E}^\pi C = \mathbb{E}^\pi \sum_{t=0}^N \kappa(S_t) \le \Delta.$$

As a result of this transformation, all costs are now expectations of immediate costs, and **CMDP** has the form

$$\begin{aligned} \textbf{ECMDP}: \underset{\pi \in \Pi}{\text{minimize}} \quad & \mathbb{E}^\pi W = \mathbb{E}^\pi \sum_{t=0}^{N-1} w(S_t, S_{t+1}) \\ \text{subject to} \quad & \mathbb{E}^\pi C = \mathbb{E}^\pi \sum_{t=0}^{N} \kappa(S_t) \le \Delta. \end{aligned} \tag{18}$$

We denote as V^* the optimal value of **ECMDP**, where

$$V^* = \inf_{\pi \in \Pi} V(s_0, \pi), \qquad V(s_0, \pi) = \mathbb{E}^\pi W,$$

subject to the constraint $\mathbb{E}^\pi C \le \Delta$.

4.2 The Linear Programming Approach

Most of the results that we apply in what follows regarding LP formulations for constrained MDPs are due to Kallenberg [27] and Altman [3]. These formulations are also discussed in several other references [4, 28, 29].

We define the immediate objective cost of taking action a_t in state s_t as the expected cost over all possible next states:

$$w(s_t, a_t) = \mathbb{E}^{\pi} w(s_t, a_t, S_{t+1}) = \sum_{s_{t+1} \in \mathsf{S}} w(s_t, a_t, s_{t+1}) Q(s_{t+1} \mid s_t, a_t) = \sum_{s_{t+1} \in \mathsf{S}} w(s_t, s_{t+1}) Q(s_{t+1} \mid s_t, a_t)$$

(see Puterman [35] equation (2.1.1).) To this end, recall that the schedule concludes when all jobs finish execution. Moreover, costs (both objective and constraint) is (possibly) incurred only until all jobs finish execution. That is, from the costs' perspective, we are concerned with the set of states

$$\mathsf{S}' = \big\{ s = (t, x, y, r) \in \mathsf{S} : y^i = 1 \text{ (not finished) for some } i \in [n] \big\}.$$

Costs keep (possibly) accruing until the state process hits $\mathsf{S} \setminus \mathsf{S}'$. Moreover, the set $U \equiv \mathsf{S} \setminus \mathsf{S}'$ is always reached under any work-conserving policy in finite time that is bounded above by $\max_{i \in [n]} \{F_i\} \leq N$. Since we are considering work-conserving policies only, S' also excludes any states $s = (t, x, y, r)$ for which $t = N$. Once the state process hits the set U, it never departs U; that is; U is absorbing under any work-conserving policy. In this case, our MDP is called **S'-transient** (Altman [3]). In fact, our MDP is in a more restricted class that is a subset of S'-transient MDPs. If we let T_U be the hitting time of set U; i.e.,

$$T_U = \inf\{t \in \mathbb{N} : S_t \in U\},$$

then $\mathbb{E}^{\pi} T_U \leq \mathbb{E}^{\pi} \max_{i \in [n]} \{F_i\} \leq N < \infty$ for any work-conserving policy π, and our MDP is said to be **S'-absorbing**, or absorbing to U.

An optimal policy can be derived by solving the following linear program (Altman [4], equation (8.18)):

$$\begin{aligned}
\textbf{LP}: \text{minimize} \quad & \left[\sum_{s \in \mathsf{S}} \sum_{a \in \mathsf{A}(s)} w(s,a) \rho(s,a) = \sum_{s \in \mathsf{S}} \sum_{a \in \mathsf{A}(s)} \rho(s,a) \sum_{\hat{s} \in \mathsf{S}} w(s, \hat{s}) Q(\hat{s} | s, a) \right] \\
\text{subject to} \quad & \sum_{s \in \mathsf{S}} \kappa(s) \sum_{a \in \mathsf{A}(s)} \rho(s,a) \leq \Delta \\
& \sum_{a \in \mathsf{A}(s)} \rho(s,a) - \sum_{s' \in \mathsf{S}'} \sum_{a \in \mathsf{A}(s')} \rho(s',a) Q(s | s', a) = \delta_{s_0}(s) \quad \forall s \in \mathsf{S} \qquad (*) \\
& \rho(s,a) \geq 0 \qquad \forall s \in \mathsf{S}, a \in \mathsf{A}(s).
\end{aligned}$$

An equivalent formulation is also given by Kallenberg [27], Theorem 6, and Kallenberg [29], equation (57) and Theorem 26, where time is explicit. We note that in constraint $(*)$ of **LP**, for every $s \in \mathsf{S}$, the second summation is taken only over the states in S' that are *predecessors* to state s; that is, over $s' \in \mathsf{S}'$ for which there is $a \in \mathsf{A}(s')$ such that $Q(s|s', a) > 0$. For instance, if the state is $s = (t, x, y, r)$, then $s' = (t', x', y', r') \in \mathsf{S}'$ is a predecessor of s if $t' = t - 1$.

If we let $M = \sum_{s \in \mathsf{S}} |\mathsf{A}(s)|$, then the decision variables involved in **LP** are $\big(\rho(s,a) : s \in \mathsf{S}, a \in \mathsf{A}(s)\big) \in [0, \infty)^M$. If we let $\mathbb{K} = \big\{(s,a) : s \in \mathsf{S}, a \in \mathsf{A}(s)\big\}$ be the set of **admissible state-action pairs**, then the vectors ρ over which the optimization in **LP** is carried out are non-negative finite measures on $\big(\mathbb{K}, 2^{\mathbb{K}}\big)$.

By Altman [4] Theorem 8.2, an optimal policy is the following: When the state is s, if $\rho\big(s, \mathsf{A}(s)\big) > 0$, then π chooses action $a \in \mathsf{A}(s)$ with probability

$$\pi(a|s) = \frac{\rho(s,a)}{\rho\big(s, \mathsf{A}(s)\big)} = \frac{\rho(s,a)}{\sum_{a' \in \mathsf{A}(s)} \rho(s,a')}, \tag{19}$$

and otherwise chooses a arbitrarily from $\mathsf{A}(s)$.

▶ Remark. For $(s_t, a_t) \in \mathbb{K}$, $\rho(s_t, a_t)$ has the interpretation of being the probability that both state s_t is occupied and action $a_t \in \mathsf{A}(s_t)$ is taken at time t under policy π defined by (19). In fact,

$$\rho(s_t, a_t) = \mathbb{E}^{\pi} \sum_{q=1}^{N} \mathbb{1}\big\{S_q = s_t, A_q = a_t\big\}, \tag{20}$$

where $\mathbb{1}\{S_q = s_t, A_q = a_t\} \equiv \mathbb{1}\{S_q = s_t, A_q = a_t\}(h)$ is the indicator function that evaluates to 1 if the trajectory of execution h is such that both $S_q(h) = s_t$ and $A_q(h) = a_t$, and to 0 otherwise. The RHS of (20) is the expected number of times that state-action pair (s_t, a_t) is visited under policy π, and is termed the state-action *occupation measure* (or visitation frequency) associated with policy π. The summation in (20) reduces to $\mathbb{1}\{S_t = s_t, A_t = a_t\}$, from which it follows that

$$\rho(s_t, a_t) = \mathbb{P}^{\pi}(S_t = s_t, A_t = a_t).$$

▶ Remark. Although we embedded time in the state to leverage results for computing stationary policies, the policy defined by transformation (19) is time-dependent (non-stationary); it is Markov nonetheless. However, we do not need to include time in the state; this is because all jobs start at time 0, the schedule concludes when all jobs finish execution, and we consider only work-conserving policies, so for a valid triple (x, y, r), $\sum_{i=1}^{n} x_i$ maps to the unique time instant during the schedule when the state is occupied. All the previous discussion can be modified so that $s = (x, y, r)$, and implicitly using $t = \sum_{i=1}^{n} x^i$. We chose to include t explicitly in the state to make the exposition clearer.

Our main result follows from the previous discussion, and is contained in

▶ **Theorem 7.** *Given a pMC instance I, if **LP** is feasible, then I is pMC-feasible. Moreover, if I is pMC-feasible, then the policy given by transformation* (19) *is expected-WTF-optimal for I.*

4.3 A Less Pessimistic Exact Formulation

Recall that we combined the constraints of **CMDP** into the single risk constraint (15). The combined constraint is more conservative than the original constraints, and it certainly restricts the feasible region of **CMDP**, in the sense that there might be a policy that is feasible for the original problem but not for the one with combined constraints. We did so to simplify the exposition, so as to have a CMDP with a single constraint (and a single cost random variable). Now we detail how to handle the exact case. This will be done at the expense of a constant increase in the size of the state space, and slightly more complicated transition dynamics.

We distinguish two *error types*: Lo-errors and Hi-errors. The Lo-error trajectories are described by the event

$$H_{\infty}^{\mathrm{er}}(\textsc{Lo}) \equiv \{h \in H_{\infty} : F_i(h) > d_i \text{ for some } i \in [n]\} \cap \{\mathsf{critPath} = \textsc{Lo}\},$$

whereas the Hi-error trajectories are precisely

$$H_{\infty}^{\mathrm{er}}(\textsc{Hi}) \equiv \{h \in H_{\infty} : F_i(h) > d_i \text{ for some } i \in \mathcal{I}_{\textsc{Hi}}\} \cap \{\mathsf{critPath} = \textsc{Hi}\}.$$

Thus, if a HI-criticality job misses its deadline under a policy but the trajectory is LO criticality, then this is a LO-error. On the other hand, if a LO-criticality job misses its deadlines but the trajectory is HI criticality, then this is not an error.

We enlarge the set of error flags by adding criticality-specific ones:

$$\mathsf{R} = \big\{\mathsf{not\ error}, \mathsf{potential\ error}, \mathsf{error}, \underbrace{\mathsf{error}_{\mathrm{LO}}, \mathsf{error}_{\mathrm{HI}}}, \mathsf{error}'\big\}.$$

For $h = (s_0, a_0, s_1, \dots) \in H_\infty$ and $\ell \in \{\mathrm{LO}, \mathrm{HI}\}$, $h \in H_\infty^{\mathrm{er}}(\ell)$ iff there is $t \in \mathbb{N}$ such that $r_t = \mathsf{error}_\ell$. Here, the flag error no longer identifies an error trajectory, but $r = \mathsf{error}$ means that a HI-criticality jobs missed its deadline but the system criticality level realization is not yet determinable. If $r_t = \mathsf{error}$ for some t, then an error will happen at some $t' > t$ when the criticality level of the trajectory becomes known, at which point the type of the error will be determined; therefore, it is always the case that $r_{t'} = \mathsf{error}_\ell$ for some ℓ and all $t' > t$. The flags error_ℓ communicate the following information: They indicate the occurrence of an error and the *type* of the error; that is, if $r_t = \mathsf{error}_\ell$ for some $t \in \mathbb{N}$ and ℓ, then the trajectory is an error and it is a type ℓ error (so it is also saying that the system criticality level realization is determined). We will define criticality-specific constraint cost functions, where the ℓ-error cost function charges a unit cost whenever the state's error flag is error_ℓ.

The following is the modification of conditions E1–E8 to accommodate the new setup:

E1'. $(r = \mathsf{not\ error}, \hat{r} = \mathsf{potential\ error})$: If a LO-criticality job misses its deadline at time $\hat{t} = t + 1$ and no HI-criticality job misses its deadline at time $\hat{t}$, but $\mathsf{critState}(\hat{s}) = \textsc{Unknown}$;

E2'. $(r = \mathsf{not\ error}, \hat{r} = \mathsf{error})$: If a HI-criticality job misses its deadline at time $\hat{t} = t + 1$, but $\mathsf{critState}(\hat{s}) = \textsc{Unknown}$;

E3'. $(r = \mathsf{not\ error}, \hat{r} = \mathsf{error}_{\mathrm{LO}})$, $(r = \mathsf{potential\ error}, \hat{r} = \mathsf{error}_{\mathrm{LO}})$: If $\mathsf{critState}(\hat{s}) = \mathrm{LO}$ and either

1. a LO-criticality job misses its deadline at time $\hat{t}$ but no HI-criticality jobs miss their deadlines at $\hat{t}$, or
2. a HI-criticality job misses its deadline at time $\hat{t}$;

E4'. $(r = \mathsf{not\ error}, \hat{r} = \mathsf{error}_{\mathrm{HI}})$, $(r = \mathsf{potential\ error}, \hat{r} = \mathsf{error}_{\mathrm{HI}})$: If a HI-criticality job misses its deadline at time $\hat{t}$ and $\mathsf{critState}(\hat{s}) = \mathrm{HI}$;

E5'. $(r = \mathsf{potential\ error}, \hat{r} = \mathsf{error})$: Same as E2';

E6'. $(r = \mathsf{error}, \hat{r} = \mathsf{error}_\ell)$ for any ℓ: If $\mathsf{critState}(\hat{s}) = \ell$;

E7'. $(r = \mathsf{potential\ error}, \hat{r} = \mathsf{potential\ error})$: Same as before;

E8'. $(r = \mathsf{potential\ error}, \hat{r} = \mathsf{not\ error})$: Same as before;

E9'. $(r = \mathsf{not\ error}, \hat{r} = \mathsf{not\ error})$: Same as before;

E10'. $(r = \mathsf{error}_\ell, \hat{r} = \mathsf{error}')$ for any ℓ: always;

E11'. $(r = \mathsf{error}', \hat{r} = \mathsf{error}')$: always.

The control model is modified so that the step 2.3 that sets the error flags uses rules E1'–E11' instead.

Now we define two immediate constraint-cost functions $\kappa_{\mathrm{LO}}, \kappa_{\mathrm{HI}} : \mathsf{S} \to \{0, 1\}$, where $\kappa_\ell(s) = \mathbb{1}\{r = \mathsf{error}_\ell\}$, $\ell \in \{\mathrm{LO}, \mathrm{HI}\}$. If $h = (s_0, a_0, s_1, \dots)$ is ℓ-error, then there is exactly one $t \in \mathbb{N}$ such that $r_t = \mathsf{error}_\ell$, and $r_{t'} = \mathsf{error}'$ for all $t' > t$. For this trajectory and the t in the preceding sentence, $\kappa_\ell(s_t) = 1$, and $\kappa_\ell(s_{t'}) = 0$ for all $t' \neq t$. Therefore, $\sum_{m=0} \kappa_\ell(s_m) = 1$, and $\sum_{m=0} \kappa_{\ell'}(s_m) = 0$ for $\ell' \neq \ell$. If h is not error, then $\sum_{m=0} \kappa_{\mathrm{LO}}(s_m) = \sum_{m=0} \kappa_{\mathrm{HI}}(s_m) = 0$. If we define the constraint-cost random variables $C_{\mathrm{LO}}, C_{\mathrm{HI}} : H_\infty \to \mathbb{R}_+$ such that $C_\ell = \sum_{t=0}^{N} \kappa_\ell(S_t)$, $\ell \in \{\mathrm{LO}, \mathrm{HI}\}$, then every C_ℓ is a Bernoulli random variable, with probability of success $\mathbb{P}^\pi(C_\ell = 1) = \mathbb{P}^\pi(\exists t \in \mathbb{N} \text{ such that } R_t = \mathsf{error}_\ell) = \mathbb{P}^\pi\big(H_\infty^{\mathrm{er}}(\ell)\big)$, where $\mathbb{P}^\pi(C_\ell = 1) = \mathbb{E}^\pi C_\ell$.

Then, we have the following optimization problem:

$$\begin{aligned} \textbf{ECMDP}' : \underset{\pi\in\Pi}{\text{minimize}} \quad & \mathbb{E}^\pi W = \mathbb{E}^\pi \sum_{t=0}^{N-1} w(S_t, S_{t+1}) \\ \text{subject to} \quad & \mathbb{E}^\pi C_\ell = \mathbb{E}^\pi \sum_{t=0}^{N} \kappa(S_t) \le p_\ell, \qquad \ell \in \{\text{Lo}, \text{Hi}\}. \end{aligned} \tag{21}$$

Finally, an exact WTF-optimal policy may be derived by solving a modification of **LP**, where the constraint $\sum_{s\in\mathsf{S}} \kappa(s) \sum_{a\in\mathsf{A}(s)} \rho(s,a) \le \Delta$ is replaced by the criticality-specific cost constraints

$$\sum_{s\in\mathsf{S}} \kappa_{\text{Lo}}(s) \sum_{a\in\mathsf{A}(s)} \rho(s,a) \le p_{\text{Lo}}, \qquad \sum_{s\in\mathsf{S}} \kappa_{\text{Hi}}(s) \sum_{a\in\mathsf{A}(s)} \rho(s,a) \le p_{\text{Hi}}.$$

Computational Complexity

An optimal satisfying assignment for the variables $\{\rho(s,a) : s \in \mathsf{S}, a \in \mathsf{A}(s)\}$ can be found using any variant of the simplex algorithm, which, in practice, is efficient in the number of decision variables and the number of constraints. However, linear program **LP** can have as many as $n|\mathsf{S}|$ decision variables and $|\mathsf{S}|$ constraints, so it requires explicit enumeration of the state space S, and therein lies the trouble. If we let $c = \max_{i\in[n]}\{c_i(\chi_i)\}$, then a "very" crude estimate of the size of S is $2^n 4(c+1)^n$ (since we do not need to include time in the state as mentioned earlier).

An MC instance might look like the following: $n = 10$ jobs and $c = 100$; for this instance, $|\mathsf{S}|$ might be as large as $2^{10} \times 4 \times 101^{10} \approx 10^{23}$, which is astronomical. In a typical MC instance, $c \gg n$, so our estimate of $|\mathsf{S}|$ indicates that the main cause of this "state space explosion" is the state variable x that records the cumulative execution time allocations, and which introduces the factor $(c+1)^n$ into our estimate of $|\mathsf{S}|$.

A Note on Approximation. We point out that despite the high computational complexity, the problem can be approximated efficiently and accurately using the factored MDP representation framework [13]. In it, instead of explicitly enumerating the states, the transition kernel is stored compactly by considering only the variables on which each state variable depends. To make use of the compact state space representation, we may utilize the symmetric primal-dual LP approximation techniques by Dolgov and Durfee [20], in which the variables of both **LP** and its dual are replaced with linear combinations of basis functions that are defined only on subsets of the state space—the so called *features*. Because jobs are independent, each variable (feature) comprising the state space depends only on a small number of variables, and our MDP falls in a special category of MDPs that are amenable to approximation, namely *loosely (weakly) coupled MDPs* [34]. The basic idea is to decompose the MDP into smaller MDPs—which is possible by the independence assumption—whose *exact* solutions can be obtained in manageable time, and then combine the solutions for the subMDPs in a proper way to construct a solution for the global MDP. This, together with the simple additive form of our cost functions, results in a linear program that contains substantially less decision variables and constraints, which can then be solved efficiently to get an approximate policy that is close to optimal.

5 Quantitative Evaluations

The purpose of this section is to provide intuition on how worst-case based approaches may underperform when worst-case demands are not realized, or when errors may be allowed. Despite the sizeable state space, we were able to compare the performance of our approach to the OCBP algorithm [10] on small instances.

OCBP builds offline a fixed priority table, and then schedules jobs according to their computed priorities. At every iteration of the priority computation procedure, OCBP finds the lowest priority job as follows: Job J_i has the lowest priority among the set of jobs that have not been assigned a priority yet if there is $c_i(\chi_i)$ time in $[0, d_i]$ when the other jobs j have executed for their demand at J_i's criticality; that is, for $c_j(\chi_i)$ each. If a lowest priority job is found at a certain iteration, then it is removed from the set of jobs that have not yet been assigned a priority, and the priority computation procedure is applied to the remaining set. If a total ordering is found on the whole input job set, then the instance is said to be OCBP-schedulable.

In our evaluation, we consider 14 pMC instances, each comprised of 3 or 4 jobs. The instances are handcrafted to showcase different aspects of our approach. For each instance, we generate job J_i's probability mass functions (pmf) over $\{1, 2, \ldots, c_i(\chi_i)\}$, $i \in [n]$, as a uniform vector in the standard $(c_i(\chi_i) - 1)$-simplex. For this purpose, we use the UUniFast Algorithm [12], which generates uniformly distributed task utilization vectors. By setting the target system utilization of UUniFast to unity, we get the desired pmfs. We generate the state space using a recursive procedure, and we use Gurobi optimization software [24] to solve **LP**. The simulation is written in C, and is publicly available at `https://github.com/RADICAL-UBC/mc-simulation.git`.

For each instance, we simulate job execution on $100,000$ demand vectors (behaviors) that are randomly drawn from the job pmfs. Table 3 lists the input job parameters for each instance and the results of job executions. The entry "**# Errors**" is the number of simulation samples—out of $100,000$—where an execution error occurred under the policy derived by solving the more conservative formulation **LP**. An error occurs for a sample if either some job misses its deadline and the sample is Lo-criticality, or a Hi-criticality job misses its deadline and the sample is Hi criticality (the definition of error according to the classical job dropping model.) An error is counted only once for a sample if it happens, even if multiple jobs miss their deadlines. The entry "**# Deadline misses**" in Table 3 is the number of samples per job for which the job missed its deadline. Note that, by definition of error, if a job misses its deadline for some sample, then this does not necessarily mean that an error occurred for that sample.

Putting $\varepsilon_\ell = 0$ for any criticality ℓ results in $\Delta = 0$, which means that not a single error is permitted. We observed that if an instance is OCBP-schedulable, then it is pMC-feasible with $\varepsilon_\ell = 0$ for any ℓ. Instances I_1, I_2 and I_3 are examples of this situation, in which no error happens in any of the simulated demand samples under the policy derived from a solution to **LP**. This is not surprising, however, because worst-case OCBP-schedulability implies worst-case MC-feasibility, so our approach would not make sense if it cannot correctly schedule, with zero errors, pMC instances that are OCBP-schedulable. We note that for I_3, even though J_3 misses its deadline in some samples, none of those deadline misses are errors.

The remaining instances are all not OCBP-schedulable. The job sets comprising instances I_4 and I_5 are identical, and all jobs in both instances have the same execution time pmfs. Moreover, both instances were simulated over the same execution time samples. These instances show how one can control the desired error by supplying different error parameters. Instance I_4 results in $10,874$ error executions (i.e., 10.87% of the samples) when $\varepsilon_{\text{Lo}} = \varepsilon_{\text{Hi}} = 1.0$ (100% allowed error; we do not care about errors,) whereas in instance I_5, where $\varepsilon_{\text{Lo}} = 0.2$ (20%) and $\varepsilon_{\text{Hi}} = 0.4$ (40%), only $7,921$ samples (i.e., 7.92% of the samples) were erroneously scheduled. For the results to make sense, we simulated both I_4 and I_5 on the same demand samples.

Instance I_6 is not even (deterministically) MC-schedulable by the clairvoyant algorithm, so one would not expect this instance to be pMC-feasible for vanishing error parameters. This is indeed the case, and our algorithm reported that I_6 is not pMC-feasible when $\varepsilon_\ell = 0$ for any ℓ. However, when allowing for some (controlled) error, I_6 might become pMC-feasible, and this is the case for I_7, which is identical to I_6 (including its job execution time distribution,) except that

some error is permitted. Instance I_8 has the same jobs and error parameters as I_7, except that the job demand distributions are different. Also here we simulated both I_7 and I_8 on the same demand samples.

Instance I_9 is identical to I_8 (including their job demand distributions,) but instance I_9's error parameters are an order of magnitude less that those of I_8. The number of errors dropped from $1,862$ for I_8 with $\varepsilon_{\text{Lo}} = 0.4$ and $\varepsilon_{\text{Hi}} = 0.5$ to 890 for the same simulation samples in I_9 with $\varepsilon_{\text{Lo}} = \varepsilon_{\text{Hi}} = 0.05$. We observe that the reduction in the number of error executions is not linear with respect to the error parameters.

Instance I_{11} is not OCBP-schedulable but it schedulable by the clairvoyant algorithm, and it turns out that this instance is pMC-feasible when the error parameters are vanishing. There are no error executions out of all the simulated samples. This is one of many MC instances where OCBP overestimates the resources required for correct (deterministic) MC-feasibility, a problem that our approach tackles effectively. Whereas OCBP requires an s-speed processor to schedule this instance for some $s > 1$ (the speed-up factor,) our approach is capable of scheduling this instance on a unit-speed processor without incurring any errors, thus maximizing the processor utilization.

Instance I_{12} and I_{13} are identical (including job demand distributions,) and they are another example where the resulting error executions can be controlled by controlling the error parameters, but for 3 job instances. Finally, instance I_{14} is a pMC-feasible instance whose job execution times are significantly larger than all of the other instances.

In all of the simulated instances, we observed that our algorithm favors Hi criticality jobs; in every simulated instance, the number of Hi-criticality deadline misses is much less than Lo-criticality deadline misses. This can be attributed to the way the set of admissible actions $\mathsf{A}(s)$ is prescribed, where Lo-criticality jobs are always dropped whenever the scenario's criticality level is inferred as Hi.

6 Concluding Remarks

We developed a probabilistic framework for reasoning about dual-criticality MC jobs systems when job demand distributions are available. We transformed the problem of constructing optimal scheduling policies into a risk-constrained MDP, where risk is the probability of missing deadlines at the two criticalities, taken over the relevant trajectories induced by the MDP. We solved the constrained MDP using a Linear Programming formulation, and showed how to construct optimal randomized Markov policies from the solution of the Linear Program. We also provided simulation results of our approach on some representative MC instances to verify and sharpen intuition.

We assumed complete knowledge of job demand distributions, and we did not consider the problem of obtaining and estimating those distributions. The latter is an important problem of investigation, and complements the probabilistic framework that we have developed. In the same vein, it is natural to consider variants of our problem where only samples of the demand distributions are available instead of full fledged distributions. This entails that the state transition probabilities are not known a priori, and the problem becomes that of *adaptive control*. If a sampler that can be queried is available, then sample average approximation (SAA) techniques can be utilized to approximate the expectation-based objectives through suitably defined empirical measures. Moreover, learning techniques, such as unsupervised learning (Q-Learning), can be used to estimate the transition probabilities and construct reasonable heuristics efficiently.

Although our approach is optimal in expectation, it is nevertheless computationally expensive, and its practicality is limited to small instances. This is due to the large state space that grows exponentially in the number of input jobs. Our next-step is to investigate provable approximation schemes that trade optimality for efficiency.

Table 3 Simulation instances and job execution results. The execution of each instance is simulated, using the policy generated by solving **LP**, over 100,000 demand samples (vectors)

Instance								OCBP Schedulability	# Deadline Misses	# Errors (Lo + Hi)
Instance	ε_{Lo}	ε_{Hi}	Job	χ	c(Lo)	c(Hi)	d			
I_1	0.0	0.5	J_1	Hi	70	75	160	YES	0	0
			J_2	Lo	50	50	50		0	
			J_3	Hi	8	20	85		0	
			J_4	Hi	1	15	65		0	
I_2	0.0	0.5	J_1	Lo	100	100	218	YES	0	0
			J_2	Lo	50	50	50		0	
			J_3	Lo	8	8	58		0	
			J_4	Hi	60	61	119		0	
I_3	0.0	0.0	J_1	Lo	60	60	161	YES	0	0
			J_2	Lo	50	50	50		0	
			J_3	Lo	20	20	70		1,447	
			J_4	Hi	30	31	101		0	
I_4	1.0	1.0	J_1	Lo	10	10	140	NO	0	10,874
			J_2	Lo	75	75	75		20,251	
			J_3	Lo	50	50	100		29,931	
			J_4	Hi	5	15	120		0	
I_5	0.2	0.4	J_1	Lo	10	10	140	NO	0	7,921
			J_2	Lo	75	75	75		43,781	
			J_3	Lo	50	50	100		7,109	
			J_4	Hi	5	15	120		0	
I_6	0.0	0.0	J_1	Lo	20	20	70	NO	–	not pMC feasible
			J_2	Lo	30	30	80		–	
			J_3	Hi	27	30	50		–	
			J_4	Hi	8	25	70		–	
I_7	0.4	0.5	J_1	Lo	20	20	70	NO	240	192
			J_2	Lo	30	30	80		1,669	
			J_3	Hi	27	30	50		0	
			J_4	Hi	8	25	70		183	
I_8	0.4	0.5	J_1	Lo	20	20	70	NO	4,640	1,862
			J_2	Lo	30	30	80		10,667	
			J_3	Hi	27	30	50		0	
			J_4	Hi	8	25	70		1,845	
I_9	0.05	0.05	J_1	Lo	20	20	70	NO	5,504	890
			J_2	Lo	30	30	80		9,982	
			J_3	Hi	27	30	50		0	
			J_4	Hi	8	25	70		883	
I_{10}	0.07	0.07	J_1	Hi	20	49	50	NO	0	18
			J_2	Lo	30	30	80		14,822	
			J_3	Hi	1	12	50		0	
			J_4	Hi	5	23	110		13	
I_{11}	0.0	0.0	J_1	Hi	3	10	27	NO	0	0
			J_2	Lo	15	15	17		39,561	
			J_3	Hi	2	5	7		0	
I_{12}	0.05	0.09	J_1	Lo	50	50	90	NO	3,726	2,239
			J_2	Lo	30	30	50		50,563	
			J_3	Hi	19	30	35		0	
I_{13}	0.4	0.5	J_1	Lo	50	50	90	NO	4,493	2,342
			J_2	Lo	30	30	50		33,981	
			J_3	Hi	19	30	35		0	
I_{14}	0.03	0.03	J_1	Hi	49	75	100	NO	0	685
			J_2	Lo	120	120	210		31,883	
			J_3	Hi	125	275	400		588	

We assumed that the given job demands are independent; however, we can drop the independence assumption as long as we are given the joint distribution of job demands. Our framework can handle this with minimal modifications; in particular, the product space construction (that we carried out in section 2) is no longer needed, but one needs to be able to compute the marginal demand distributions in order to compute the MDP's transition probabilities. This, however, places a greater burden on the system designer, because the joint distribution of job demands might be harder to obtain compared to individual job demand distributions. Nevertheless, our framework, as presented, offers great flexibility, because it allows the individual job distributions to be defined on different probability spaces (the Ω_is) that might correspond to different operating environments and settings.

Finally, two important extensions to our model deserve special mention and are due future studies. The first is the case of arbitrary number of criticality levels, and the second is the sporadic *task* model.

References

1 AdaCore. What is do-178b?, 2014. URL: http://www.adacore.com/gnatpro-safety-critical/avionics/do178b/.

2 Bader Alahmad, Sathish Gopalakrishnan, Luca Santinelli, and Liliana Cucu-Grosjean. Probabilities for Mixed-Criticality Problems: Bridging the Uncertainty Gap. In *The Work in Progress session of the 32nd IEEE Real-time Systems Symposium - RTSS 2011*, Wien, Austria, November 2011. URL: https://hal.inria.fr/hal-00646586.

3 Eitan Altman. Constrained markov decision processes with total cost criteria: Lagrangian approach and dual linear program. *Math. Meth. of OR*, 48(3):387–417, 1998. doi:10.1007/s001860050035.

4 Eitan Altman. *Constrained Markov Decision Processes*. Chapman and Hall/CRC, 1999.

5 Robert B. Ash. *Real Analysis and Probability*. Academic Press, 1972.

6 Neil C. Audsley, Alan Burns, Mike M. Richardson, Ken Tindell, and Andy J. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993. doi:10.1049/sej.1993.0034.

7 Sanjoy Baruah. Mixed criticality schedulability analysis is highly intractable. to appear. URL: http://www.cs.unc.edu/~baruah/Submitted/02cxty.pdf.

8 Sanjoy K. Baruah, Vincenzo Bonifaci, Gianlorenzo D'Angelo, Haohan Li, Alberto Marchetti-Spaccamela, Nicole Megow, and Leen Stougie. Scheduling real-time mixed-criticality jobs. In Petr Hlinený and Antonín Kucera, editors, *Mathematical Foundations of Computer Science 2010, 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6281 of *Lecture Notes in Computer Science*, pages 90–101. Springer, 2010. doi:10.1007/978-3-642-15155-2_10.

9 Sanjoy K. Baruah and Zhishan Guo. Scheduling mixed-criticality implicit-deadline sporadic task systems upon a varying-speed processor. In *Proceedings of the IEEE 35th IEEE Real-Time Systems Symposium, RTSS 2014, Rome, Italy, December 2-5, 2014*, pages 31–40. IEEE Computer Society, 2014. doi:10.1109/RTSS.2014.15.

10 Sanjoy K. Baruah, Haohan Li, and Leen Stougie. Towards the design of certifiable mixed-criticality systems. In Marco Caccamo, editor, *16th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2010, Stockholm, Sweden, April 12-15, 2010*, pages 13–22. IEEE Computer Society, 2010. doi:10.1109/RTAS.2010.10.

11 Sanjoy K. Baruah and Steve Vestal. Schedulability analysis of sporadic tasks with multiple criticality specifications. In *20th Euromicro Conference on Real-Time Systems, ECRTS 2008, 2-4 July 2008, Prague, Czech Republic, Proceedings*, pages 147–155. IEEE Computer Society, 2008. doi:10.1109/ECRTS.2008.26.

12 Enrico Bini and Giorgio C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005. doi:10.1007/s11241-005-0507-9.

13 Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. Stochastic dynamic programming with factored representations. *Artif. Intell.*, 121(1-2):49–107, 2000. doi:10.1016/S0004-3702(00)00033-3.

14 Alan Burns and Robert I. Davis. Mixed criticality systems - a review. Preprint, 2015. URL: http://www-users.cs.york.ac.uk/burns/review.pdf.

15 Yao Chen, Qiao Li, Zheng Li, and Huagang Xiong. Efficient schedulability analysis for mixed-criticality systems under deadline-based scheduling. *Chinese Journal of Aeronautics*, 27(4):856 – 866, 2014. doi:http://dx.doi.org/10.1016/j.cja.2014.05.003.

16 José Luis Díaz, Daniel F. García, Kanghee Kim, Chang-Gun Lee, Lucia Lo Bello, José María López, Sang Lyul Min, and Orazio Mirabella. Stochastic analysis of periodic real-time systems. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02), Austin, Texas, USA, December 3-5, 2002*, pages 289–300. IEEE Computer Society, 2002. doi:10.1109/REAL.2002.1181583.

17 José Luis Díaz and José María López. Probabilistic analysis of the response time in a real-time system. *Technical Report*, 2001.

18 José Luis Díaz and José María López. Safe extensions to the stochastic analysis of real-time systems. *Technical Report*, 2004.

19 José Luis Díaz, José María López, Manuel García Vazquez, Antonio M. Campos, Kanghee Kim, and Lucia Lo Bello. Pessimism in the stochastic analysis of real-time systems: Concept and applications. In *Proceedings of the 25th IEEE Real-Time Systems Symposium (RTSS 2004), 5-8 December 2004, Lisbon, Portugal*, pages 197–207. IEEE Computer Society, 2004. doi:10.1109/REAL.2004.41.

20 Dmitri A. Dolgov and Edmund H. Durfee. Symmetric approximate linear programming for factored mdps with application to constrained problems. *Ann. Math. Artif. Intell.*, 47(3-4):273–293, 2006. doi:10.1007/s10472-006-9038-x.

21 Peter Geibel and Fritz Wysotzki. Risk-sensitive reinforcement learning applied to control under constraints. *J. Artif. Intell. Res.*, 24:81–108, 2005. doi:10.1613/jair.1666.

22 Zhishan Guo and Sanjoy K. Baruah. Implementing mixed-criticality systems upon a preemptive varying-speed processor. *LITES*, 1(2):03:1–03:19, 2014. doi:10.4230/LITES-v001-i002-a003.

23 Zhishan Guo, Luca Santinelli, and Kecheng Yang. EDF schedulability analysis on mixed-criticality systems with permitted failure probability. In *21st IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2015, Hong Kong, China, August 19-21, 2015*, pages 187–196. IEEE Computer Society, 2015. doi:10.1109/RTCSA.2015.8.

24 Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2015. URL: http://www.gurobi.com.

25 Onésimo Hernández-Lerma. *Adaptive Markov control processes*. Applied mathematical sciences. Springer, New York, 1989.

26 Onésimo Hernández-Lerma and Jean B Lasserre. *Discrete-Time Markov Control Processes: Basic Optimality Criteria*. Stochastic Modelling and Applied Probability. Springer-Verlag, New York, 1996.

27 L. C. M. Kallenberg. Unconstrained and constrained dynamic programming over a finite horizon. *Technical Report*, 1981.

28 L. C. M. Kallenberg. *Linear Programming and Finite Markovian Control Problems*. Amsterdam : Mathematisch Centrum, 1983.

29 Lodewijk C. M. Kallenberg. Survey of linear programming for standard and nonstandard markovian control problems. part I: theory. *Math. Meth. of OR*, 40(1):1–42, 1994. doi:10.1007/BF01414028.

30 Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000. doi:10.1145/347476.347479.

31 Kanghee Kim, José Luis Díaz, Lucia Lo Bello, José María López, Chang-Gun Lee, and Sang Lyul Min. An exact stochastic analysis of priority-driven periodic real-time systems and its approximations. *IEEE Trans. Computers*, 54(11):1460–1466, 2005. doi:10.1109/TC.2005.174.

32 Achim Klenke. *Probability Theory: A Comprehensive Course*. Universitext. Springer, 2 edition, 2014.

33 Dorin Maxim and Liliana Cucu-Grosjean. Response time analysis for fixed-priority tasks with multiple probabilistic parameters. In *Proceedings of the IEEE 34th Real-Time Systems Symposium, RTSS 2013, Vancouver, BC, Canada, December 3-6, 2013*, pages 224–235. IEEE Computer Society, 2013. doi:10.1109/RTSS.2013.30.

34 Pascal Poupart, Craig Boutilier, Relu Patrascu, and Dale Schuurmans. Piecewise linear value function approximation for factored mdps. In Rina Dechter, Michael J. Kearns, and Richard S. Sutton, editors, *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada.*, pages 292–299. AAAI Press / The MIT Press, 2002. URL: http://www.aaai.org/Library/AAAI/2002/aaai02-045.php.

35 M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, New York, 5 edition, 2005.

36 Martin L. Shooman. Avionics software problem occurrence rates. In *Seventh International Symposium on Software Reliability Engineering, ISSRE 1996, White Plains, NY, USA, October 30, 1996-Nov. 2, 1996*, pages 55–64. IEEE Computer Society, 1996. doi:10.1109/ISSRE.1996.558695.

37 Dario Socci, Peter Poplavko, Saddek Bensalem, and Marius Bozga. Mixed critical earliest deadline first. In *25th Euromicro Conference on Real-Time Systems, ECRTS 2013, Paris, France, July 9-12, 2013*, pages 93–102. IEEE Computer Society, 2013. doi:10.1109/ECRTS.2013.20.

38 Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS 2007), 3-6 December 2007, Tucson, Arizona, USA*, pages 239–243. IEEE Computer Society, 2007. doi:10.1109/RTSS.2007.47.