technische universität
dortmund

Average Task Execution Time Minimization under (m, k) Soft Error Constraint

Junjie Shi[1], Niklas Ueter[1], Jian-Jia Chen[1], and Kuan-Hsun Chen[2]
[1]TU Dortmund, Deaprtment of Computer Science, Dortmund, Germany
[2]University of Twente, Computer Architecture and Embedded Systems, Twente, Netherlands

Citation: RTAS.2023

BibTeX:

```
@inproceedings{DBLP:conf/rtas/ShiUBC23,
  author       = {Junjie Shi and
                  Niklas Ueter and
                  Jian{-}Jia Chen and
                  Kuan{-}Hsun Chen},
  title        = {Average Task Execution Time Minimization under (m, k) Soft Error Constraint},
  booktitle    = {29th {IEEE} Real-Time and Embedded Technology and Applications Symposium,
                  {RTAS} 2023, May 9-12, San Antonio, Texas.},
  pages        = {},
  publisher    = {{IEEE}},
  year         = {2023},
  url          = {https://doi.org/10.1109/RTASxxxxx.2023.xxxxx},
  doi          = {10.1109/RTASxxxxx.2023.xxxxx}
}
```


CS 12 computer science 12

Average Task Execution Time Minimization under (m, k) Soft Error Constraint

Junjie Shi[1], Niklas Ueter[1], Jian-Jia Chen[1], Kuan-Hsun Chen[2]
[1]Department of Computer Science, TU Dortmund University, Germany
[2]Department of Computer Science, University of Twente, The Netherlands

***Abstract*—Safety-critical systems are often subjected to *transient faults*. Since these transient faults may lead to *soft errors* that cause catastrophic consequences, error-handling must be addressed by design. Full-protection against faults is too costly in terms of resource usage. A common approach to relax the resource demands and limit the impact of errors is to consider (m, k)-constraints, which requires that at least m jobs out of any k consecutive jobs are error-free. To assure (m, k)-compliance, static patterns are widely used to select the job execution modes, i.e., either in an *error-free* mode at the cost of increased worst-case execution time or in an *error-prone* mode with the advantage of less execution time. Although static patterns have been shown to be effective in energy-aware designs, resource over-provision is inevitable due to the relatively low rate of error probability. In this work, we propose two dynamic (and adaptive) approaches that allow the scheduler to opportunistically select execution modes based on the error-history of the past jobs and the actual error probability. We firstly propose a Markov chain based solution if the error-probability is known and static and secondly a reinforcement learning-based approach that can handle unknown error probabilities. Experimental evaluations show that our approaches outperform the state-of-the-art in most of the evaluated cases in terms of average utilization for each task and the overall utilization for multitask systems.**

I. Introduction

Safety-critical embedded systems are often subjected to transient faults due to environmental factors such as cosmic radiation and electromagnetic interference [3]. The sensitivity to such environmental factors is exacerbated by the high integration density of modern systems-on-chips, which leads to non-negligible transient fault-rates. Since these transient faults can lead to soft errors that may cause catastrophic consequences, error-handling must be addressed during system design. Software-based techniques such as explicit output comparison (EOC) [13], control flow checking by using software signatures [33], and redundant multithreading [10], are widely adopted due to their flexibility to trade-off error protection with additional runtime.

In practice, it has been noticed, that some safety-critical applications can tolerate a limited number of errors at the cost of temporarily downgrading the quality of service (QoS), without catastrophic consequences if some constraints of error tolerance are guaranteed. For instance, robotic applications can still successfully finish their tasks under a limited number of errors [9], [45], where an (m, k) robustness constraint is considered. That is, a task must have at least m correct jobs out of any k consecutive jobs. While the original concept of (m, k)-constraints [17] was designed for allowing limited deadline misses [11], [18], the concept of (m, k)-constraints is equally applicable to specify admissible limited numbers of soft errors.

The most prevalent techniques that are currently used to guarantee (m, k)-constraints, rely on static decisions to instantiate jobs using suitable fault-tolerance techniques to assure a reliable job execution such as the deeply red pattern (R-pattern) [24] or the evenly distributed pattern (E-pattern) [37]. To improve the adaptivity of the static pattern based techniques, Chen et al. [9] proposed to track the current resilience during runtime and to adapt the patterns accordingly. More precisely, the pattern-based scheduler defers the time-costly reliable executions to the possible last moment by tracking the number of upcoming jobs that can still be faulty without violating the constraints. Despite that approach allowing for some adaptivity, it might still lead to pessimistic resource usage since the actual soft error probability is not considered.

In the literature of fault tolerant systems, one often stated objective is to minimize the overall system utilization in order to minimize the energy consumption [9], [30], [31], which is due to the processors starkly differing power consumption profiles in the busy and idle state, respectively. As shown in [8], the power consumption of a processor can be modeled by the power consumption P_{busy} in the busy state (a job is executed) and the power consumption P_{idle} in the idle state (nothing is executed). In the busy state, the consumed power can be decomposed into *static* and *dynamic* power consumption in contrast to the idle state, in which (ideally) only static power is consumed. Moreover, keeping the processor in a busy state for a sustained amount of time leads to a temperature increase, which in turn results in higher energy consumption for cooling and increases in the static leakage power consumption, due to the super linear relationship between temperature and static leakage power [40], [29]. The average energy consumption is given by the average amount of time spent in either state weighted with the power consumption associated with each state. Using the the average utilization (U_{avg}), this can be expressed as $P_{idle} \cdot (1 - U_{avg}) + P_{busy} \cdot U_{avg}$ and thus the energy saving is directly linked to the reduction of the average utilization.

In this work, we propose an adaptive state-based algorithm that uses explicit knowledge and/or estimations of the soft error probabilities and assures (m, k)-compliance of each task under real-time constraints, whilst minimizing the expected execution time of each task. Consequently of that mini-

mization, the expected overall system utilization and energy consumption of the system is minimized. We emphasize that even though our studied execution time minimization is more universal than just energy-consumption minimization, we here focus on that use-case due to the practical importance, which is evident in the plethora of published results concerning that problem. We summarize our contributions as follows.

Our Contributions.

- Our fundamental contribution is the enforcement of (m, k) constraints using a finite automata. We formulate all (m, k) compliant states of a task as a minimal automata in Sec. III and only allow transitions between compliant states. Based on the soft error probability and a stochastic transition system, a Markov chain model is derived.
- In Section IV, under the assumption that a stationary error-probability can be accurately estimated, we propose an optimization algorithm based on the Markov chain formulation above, that calculates the stochastic parameters in the job selection strategy to minimize the expected execution time.
- Furthermore, we propose a reinforcement learning (RL)-based approach to aid the job mode selection strategy when soft error probabilities are unknown. We formulate the mapping from a task's execution information to the RL recognizable environment and discuss the barrier function and learning policy in Section V.
- To demonstrate the applicability of our approaches, we provide extensive numerical evaluations in Section VI. The results show that our approaches outperform the state-of-the-art in most of the evaluated cases with respect to the systems average utilization. Furthermore, the overheads of two approaches are discussed in Section VI-C.

II. System Model

In this section, the considered task-, fault-, and error model are explained in detail. In particular, we explain how faults are prevented by job level software-based fault tolerance techniques. Afterwards, we explain and state the here studied problem of minimizing the expected execution time under (m, k)-constraints. Lastly, we define the scheduling problem of the task system under real-time constraints and clarify the impacts of our approach on the worst-case response time analysis.

A. Task Model

We consider a set of periodic and constrained-deadline real-time tasks $\mathbb{T} = \{\tau_1, \ \ldots, \ \tau_n\}$ such that each task τ_i is modeled by a tuple $(C_i, D_i, T_i, m_i, k_i)$ where:

- $C_i = \{C_i^u, C_i^d, C_i^r\}$ is a set of **worst-case execution times** (WCETs) representing the different WCET demands of the different execution modes of *unreliable* mode, *detected* mode, and *reliable* mode task instances. Throughout this work, we assume that $C_i^u < C_i^d < C_i^r$ holds due to the additional overheads for the *detected* and *reliable* modes.
- D_i is the **relative deadline** of τ_i, i.e., a job of τ_i released at time t must finish its execution no later than its absolute deadline $t + D_i$. We consider constrained-deadline task systems, i.e., $D_i \leq T_i, \ \forall \ \tau_i \in \mathbb{T}$.
- The **period** of τ_i is denoted by T_i. Each task releases an infinite number of task instances, called *jobs*, strictly periodically with period T_i. We denote the ℓ-th job of τ_i as J_i^ℓ.
- Each task is subject to an (m_i, k_i) **constraint**, i.e., a task τ_i is required to have at least m_i jobs out of any k_i consecutive jobs to be correctly executed, where $0 < m_i \leq k_i$.

With regards to the different execution modes, we assume that software-based fault tolerance techniques are used to detect and recover fault-induced soft errors. Each task is allowed to instantiate jobs in the *reliable*, *detected*, or *unreliable* mode and the additional composite mode *detected+reliable*, which is an immediate compensation in the same release window, i.e., a reliable execution mode instance can be triggered right after the detected mode instance when it is necessary. The different elementary modes have the following implications for soft errors and overheads; a) In the *unreliable* mode, no additional implementation effort is needed. However it is not known if a error occurred during the execution of this job or not. In consequence, in order to guarantee (m, k) constraints, we have to assume that an soft error occurred by default. b) In the *detected* mode, techniques can be applied to verify the correctness of the executed job, e.g., error detection with special encoding of data, or control flow checking. In this mode it can be observed if a error has occurred during execution or not. c) In the *reliable* mode, it must be assured that no soft error manifests which requires detection and subsequent recovery thereof. To that end, several redundant copies of the job are executed to ensure high reliability of the final result.

The considered system does not allow to skip any job in order to potentially improve the QoS. In the remainder of this work, we omit the task index whenever it is irrelevant what task is referred to.

B. Fault and Error Model

Transient faults can lead to soft errors that cause incorrect results calculated by the affected executed jobs. In this work, we assume that the probability that an executed job is affected from transient faults, which then results in at least one soft error, is given by a stationary probability p_e. Subsequently, the soft error probabilities for a sequence of jobs of the same task is an independent stochastic process. We assume that soft errors can occur at any time during a job's execution, but such error-affected job is assumed to halt after executing for at most its worst-case execution time – even in the unreliable mode – by means of e.g., watchdog timers. Moreover, the error detection in the *reliable* and *detected* mode is certain in the sense that an incurred soft error is detected with the same (very high) probability as other system reliability dependent parameter guarantees. In either the *detected* mode or the *reliable* mode,

errors are detected at the end of a job's execution by using sanity or consistency checks [35]. To guarantee a correct result in the *reliable* mode, either a recovery routine can be issued to guarantee the job's correctness or task replication [19] can be applied to achieve high reliability. For instance, in the simultaneously, and redundantly threaded processors with recovery approach (SRTR) [41], the register values of all redundant threads are compared and are only committed if the register values of all threads agree. Otherwise, the threads are re-executed – for at most a specified number m times – until either the register values of all threads agree or the maximum number of re-executions is reached. Hence, the probability that a correct result is produced in the reliable mode by the SRTR approach after at most m re-executions is given by $1-(p_e)^m$ under the assumption that soft error probabilities of the re-executions are independent and that each soft error results in a disagreement of the compared to register values[1]. The overhead for detection and recovery is henceforth integrated into the WCETs of the corresponding jobs' execution modes.

C. Problem Definition

In this work, we consider a periodic and constrained-deadline task set $\mathbb{T}$ that is scheduled by an arbitrary preemptive scheduling algorithm under some form of temporal constraint, which is further detailed in the following Section II-D. Each task τ_i releases an infinite number of jobs J_i^ℓ for $\ell \in \mathbb{N}$, each of which has an arrival time a_i^ℓ and finishing time f_i^ℓ and is run exclusively in either the *unreliable*, *reliable*, *detected* mode or the composite mode *detected* + *reliable*, which is a *detected* mode instance immediately followed by a *reliable* mode instance in case of a soft error detection during the same interval $[a_i^\ell, f_i^\ell)$. Depending on the mode in which a job J_i^ℓ is executed, a soft error that occurs during the interval $[a_i^\ell, f_i^\ell)$ affects the correctness of that job, i.e., a faulty job in case of the *unreliable* and *detected* mode and a correct job in case of the *reliable* or *detected* + *reliable* mode. Our objective is to devise a job mode selection strategy – for the next-to-be-released job – based on the error history and currently executed job that is reconstructed from the observed soft errors by the *detected* and *reliable* job modes and the additional information of an estimated error probability p_e. In particular, that job mode selection strategy must guarantee that the task under consideration satisfies the specified (m,k)-constraints at all times under the optimization objective that the expected execution time of the task is minimized.

D. Schedulability and Scheduling

We assume an arbitrary preemptive scheduling algorithm to schedule the task set $\mathbb{T}$ that is capable of guaranteeing the temporal requirements such as strict deadline compliance in case of hard real-time systems. Our proposed approach is not strictly limited to hard real-time task systems and focuses on the generation of guaranteed (m,k)-compliant schedules for each task under the objective to minimize the expected execution time. However, the worst-case job mode sequence that can be generated by our approach is identical to the R-Pattern in which the first $k-m$ instances are executed in *detected* mode and the remaining m instances are successively executed in the *reliable* mode. Therefore, any hard real-time schedulability analyses adopting the R-Pattern can be used.

We note that our approach does not improve the worst-case response time analyzability in contrast to static patterns that restrict the possible worst-case generated job mode sequences. Instead, we improve the average case performance, but can still provide hard real-time guarantees. For instance, our approach can be analyzed by adopting the multi-frame task model to analyze the worst-case execution pattern as suggested by Chen et al. [9]. If on the other hand soft real-time or best-effort is required, then a suitable algorithm such as earliest-deadline first (EDF) may be used.

[1]The maximum number of replications m must be determined by the system designer depending on the required confidence and how hardened the considered system must be.

III. Minimal Compliant Automata Construction

In this section, we formalize the problem described in Section II-C. Afterwards, we propose an algorithm to construct (m,k)-compliant automata with the minimal number of states.

A. Automata Construction

Definition 1 (Correctness Indication)**.** *We indicate the correctness of a job at the end of its execution by an element of the set $\Sigma = \{0,1\}$. That is, an error-free executed job is indicated by a 1 and an erroneously executed job is denoted by a 0.*

We indicate the correctness of the ℓ-th job J_i^ℓ of task τ_i by the *character* $c_\ell \in \Sigma$ and use a (possibly infinite) sequence of concatenated characters, i.e., a word $w = c_1 \circ c_2 \circ \cdots \circ c_n$ to indicate the correctness of the job sequence J_i^1 to J_i^n for $n \in \mathbb{N}$. We denote the sub-word of w that starts at index a and ends at index b as $w(a,b) = c_a \circ \cdots \circ c_b$ for $a < b$. The $w(a,:)$ and $w(:,b)$ are used to denote the sub-word starting at index a to the end or from the beginning to the index b inclusively.

To eventually guarantee (m,k)-compliance of a task, i.e., of an infinite sequence of jobs, any sequence of k-consecutive jobs must be analyzed. While there are infinitely many sub-words (since there may be infinite job releases), there are only 2^k many different outcomes $Q := \{00\ldots0, \ldots, 11\ldots1\}$ for which we define a k-error-automata.

Definition 2 (k-Error-Automata)**.** *A k-error-automata $\mathcal{A}_k := (q_s, Q, \Sigma, \delta)$ is defined by a 4-tuple, where $Q := \{0,1\}^k$ denotes the finite set of states of all possible outcomes in any k consecutive job releases. The start $q_s := 11\ldots1 \in Q$ denotes the unique starting state, $\Sigma := \{0,1\}$ denotes the input alphabet, and δ defines the transition system $\delta : (Q,\Sigma) \mapsto Q$ such that for any state $q \in Q := \{00\ldots0, \ldots, 11\ldots1\}$*

$$\delta(q,0) = q(2,:) \circ 0 \in Q \tag{1}$$

$$\delta(q,1) = q(2,:) \circ 1 \in Q \tag{2}$$

An exemplary 3-error-automata $\mathcal{A}_3$ is illustrated in Figure 1. While a k-error-automata models all error sequences in k consecutive jobs, not each sequence is (m,k) compliant.

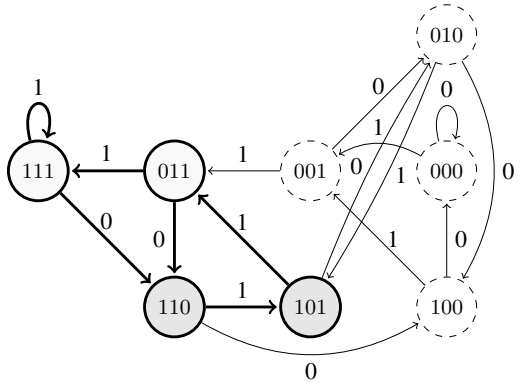

Fig. 1. An exemplary k-error-automata $\mathcal{A}_k$ and the $(2,3)$-compliant automata $\mathcal{A}_k^*$ is highlighted in bold, where the darker states are *critical states* and the lighter states are *nominal states*.

Definition 3 ((m,k)-Compliant State)**.** *A state $q \in Q$ of a k-error-automata $\mathcal{A}_k$ is called (m,k)-compliant if $\mathbb{1}[q] \geq m$ is satisfied, where the operator $\mathbb{1}$ counts the number of 1's in q's representation. The set of all (m,k)-compliant states is called the (m,k)-compliant state-space denoted by $Q^* \subseteq Q$.*

In order to verify if a task satisfies its (m,k) constraint after the finishing of the ℓ-th job given the indication c_ℓ it must be tested if every sub-word of length k in $w = c_1 \circ \cdots \circ c_\ell$ for $\ell \geq k$ contains at least m correct executions.

Definition 4 (Job Sequence Induced State)**.** *Let any concrete word $w = c_1 \circ \cdots \circ c_\ell$ for $\ell \geq k$, that indicates the outcomes of all finished jobs. A sub-word of length k starting at the j-th job $w(j, j+k-1)$ for $j \in \{1, \ldots, \ell-k+1\}$ induces a state $q \in Q$ in the k-error-automata $\mathcal{A}_k$ denoted as $\psi(w(j, j+k-1)) = q \in Q$ if q's binary representation is identical to $w(j, j+k-1)$.*

As the word w, indicating the correctness of all finished jobs, evolves with the finishing of each released job, the state of the k-error-automata transitions accordingly. More precisely, let the $(j+k)$-th job finish at time f_{j+k} and let the sub-word $w(j, j+k-1)$ denote the latest k-consecutive job outcomes prior to time f_{j+k}. Based on the outcomes of the $(j+k)$-th job as indicated by c_{j+k}, the evolved job sequence induced state in $\mathcal{A}_k$ is given by $\delta(\psi(w(j, j+k-1)), c_{j+k})$. The outcome of the $(j+k)$-th job is determined by the occurrence of an error, which is assumed to be stochastic in nature and beyond our control. That is, under the assumption of a stationary soft error probability p_e we have that $\mathbb{P}(c_{j+k} = 0) = p_e$ and conversely $\mathbb{P}(c_{j+k} = 1) = 1 - p_e$. We can however control the execution mode of the $(j+k)$-th job release, i.e., *unreliable*, *detected*, *reliable* or *detected* followed by *reliable*.

As described in Section II-A, in the *unreliable mode*, the correctness of $(j+k)$-th job, i.e., $c_{j+k} = 0$ with probability 1. In *detected mode*, if an error occurred then $c_{j+k} = 0$ with probability p_e and $c_{j+k} = 1$ with probability $1 - p_e$ otherwise. In the *reliable mode*, the execution is guaranteed to be correct, i.e., $c_{j+k} = 1$ with probability 1. In the *detected followed by an optional reliable mode*, a reliable instance is only released if an error was detected and the current instance has to be correct in order to ensure the corresponding (m,k) constraint.

Recall that **our objective** is to devise a state-based execution mode selection such that any infinite sequence of outcomes of jobs as indicated by an evolving word w is firstly (m,k) compliant and secondly minimizes the expected execution time. More precisely for any job sequence induced compliant state $\psi(w(j, j+k-1)) \in Q^*$ of $\mathcal{A}_k$, we devise a mode selection strategy

$$\alpha : Q^* \mapsto \{u, d, r, d+r\} \tag{3}$$

to choose either an *unreliable*, *detected*, *reliable*, or a *detected* job optionally followed by a *reliable* instance for the $(j+k)$-th job release such that $\mathbb{P}(\psi(w(j+1, j+k)) \notin Q^* \mid \psi(w(j, j+k-1) \in Q^*, \alpha(\psi(w(j, j+k-1))) = 0$ for all $j \in \mathbb{N}$. For short, let $x_j := w(j, j+k-1)$ for some $j \in \mathbb{N}$ then

$$\mathbb{P}(c_{j+k} = 1 \mid \alpha(x_j)) = \begin{cases} 0 & \text{if } \alpha(x_j) = u \\ 1 - p_e & \text{if } \alpha(x_j) = d \\ 1 & \text{if } \alpha(x_j) = r \vee (d+r) \end{cases}$$

Conversely, $\mathbb{P}(c_{j+1} = 0 \mid \alpha(x_j)) = 1 - \mathbb{P}(c_{j+1} = 1 \mid \alpha(x_j))$. Please note that while the job may actually execute correctly even in *unreliable* mode, we have to consider it an error to guarantee (m,k) compliance, since the outcome is not observable. From a design perspective, we have to design the transitioning system of $\mathcal{A}_k$ such that only the compliant states Q^* are reachable.

Definition 5 (Compliant Transitions)**.** *A transition system δ of a k-error-automata $\mathcal{A}_k$ is (m,k)-compliant if and only if for any given word w with $j \geq 1$ the following implication holds*

$$\psi(w(j, j+k-1)) \in Q^* \Rightarrow \delta(\psi(w(j, j+k-1)), c_{j+k}(\alpha)) \in Q^*$$

Definition 6 (Critical State)**.** *A compliant state $\psi(w(j, j+k-1)) \in Q^*$ is a critical state with respect to (m,k)-constraints if there are only $(m-1)$ correctly executed jobs in the word $w(j+1, j+k-1)$, i.e., the latest previous $(k-1)$ jobs.*

Definition 7 (Nominal State)**.** *A compliant state $\psi(w(j, j+k-1)) \in Q^*$ is a nominal state with respect to (m,k) constraints if there are at least m correctly executed jobs among the latest previous $(k-1)$ jobs, i.e., $w(j+1, j+k-1)$.*

It can be observed that in order for the transition system to be compliant, we have to enforce an outcome c_{j+k} based on whether $\psi(w(j, j+k-1)$ is a *critical* or *nominal state*. That is if $\psi(w(j, j+k-1)) \in Q^*$ and *critical* then $c_{j+k} = 1$ must be enforced. In the case that $\psi(w(j, j+k-1)) \in Q^*$ and *nominal* then any $c_{j+k} \in \{0, 1\}$ is a feasible outcome. These observations are formalized in the following corollaries.

Corollary 1 (Critical State Transition)**.** *If a compliant state $\psi(w(j, j+k-1)) \in Q^*$ is a critical state then only a correct execution of the $(j+k)$-th job leads to a transition into a compliant state Q^*.*

Proof. The updated word after concatenation of c_{j+1} is given by $w(j+1, j+k)$, i.e., $w(j+1, j+k-1) \circ c_{j+1}$. By definition, the number of correct instances is given by $\mathbb{1}[w(j+1, j+k-$

1)] = m − 1. Clearly |w(j + 1, j + k)| = k and if c_{j+1} = 0 then 𝟙[w(j + 1, j + k)] = m − 1 and 𝟙[w(j + 1, j + k)] = m if c_{j+1}. □

Corollary 2 (Nominal State Transition)**.** *If a compliant state ψ(w(j, j + k − 1)) ∈ Q* is a nominal state then either execution outcome of the (j + k)-th job leads to a transition into a compliant state Q*.*

Proof. The updated word after concatenation of c_{j+k} is given by $w(j+1, j+k)$, i.e., $w(j+1, j+k-1) \circ c_{j+k}$. By definition, the number of correct instances is given by $\mathbb{1}[w(j+1, j+k-1)] = m$. Clearly $|w(j+1, j+k)| = k$ and if $c_{j+k} = 0$ then $\mathbb{1}[w(j+1, j+k)] = m$ and $\mathbb{1}[w(j+1, j+k)] = m+1$ if c_{j+k} each of which complies with the (m, k) constraints. □

Based on these results, we can formulate properties that must be met by any feasible strategy.

Lemma 1 (Compliant Mapping Strategy)**.** *Any mapping strategy α for the k-error-automata $\mathcal{A}_k$ that satisfies the constraints*

$$\alpha(\psi(x_j)) = \begin{cases} r \vee (d + r) & \text{if } \psi(x_j) \text{ is a critical state} \\ u \vee d \vee r & \text{if } \psi(x_j) \text{ is a nominal state} \end{cases} \quad (4)$$

leads almost certainly to a compliant transition system for $x_j := \psi(w(j, j + k - 1)) \in Q^*$ for all j.

Proof. By the results of Corollary 1 and Corollary 2, we know that for any induced state $q := \psi(x_j) \in Q^*$, the strategy $\alpha(q)$ must almost certainly enforce a correct outcome of c_{j+k} if q is a critical state and any outcome if q is a critical state to lead to a compliant transition. Clearly, a *reliable* instance or a *detected* instance followed by an optional *reliable* instance in case of an error in case of q being a critical state enforces that $\mathbb{P}(c_{j+k} = 1 \mid \alpha(q)) = 1$. Conversely, if an *unreliable* or a *detected* instance is chosen if q is a nominal state then $\mathbb{P}(c_{j+k} = 0 \mid \alpha(q)) + \mathbb{P}(c_{j+k} = 1 \mid \alpha(q)) = 1$ which thus almost certainly leads to a compliant state. □

An α-induced (m, k)-compliant subset of a k-error-automata $\mathcal{A}_k$ is denoted by $\mathcal{A}_k^*(\alpha)$ and only contains compliant states $Q^* \subseteq Q$ and a compliant transition system $\delta^* \subseteq \delta$ such that for any $q \in Q^*$ the transition $\delta^*(q, c(\alpha(q))) \in Q^*$, which is exemplified in Figure 1.

B. States Reduction and Minimal Automata Construction

In the remainder of this section, we propose an algorithm to generate a minimal (m, k)-compliant automata $\mathcal{A}_k^*(\alpha)$, which is necessary to improve the computational complexity of our to be designed expected execution time minimization algorithms. We note that the approach to generate minimal finite-state machines as e.g., used in Vreman et al. in [43] is applicable for (m, k) constraints as well. However, their generation algorithm is similar to Hopcroft's algorithm [21], which generates all states and merges *equivalent states* whilst our Algorithm 1 utilizes the specificity of the problem to only generate compliant states right away.

Definition 8. *For given (m, k)-constraints the set of n-step equivalent compliant states of the compliant k-error-automata $\mathcal{A}_k^*$ is given by*

$$[q]_n := \{q, q' \in Q^* \mid (\delta(q, w) = \delta(q', w))\ \forall w \in \{0, 1\}^n\}$$

and we say $q \sim_n q'$ if q and q' are n-step equivalent.

We use the *don't care* notation to denote the representative state $[q]_n$, e.g., $* \circ q(2, :) = * \circ q'(2, :)$ for 1-step equivalent states $q \sim_1 q'$ and $** \cdots * \circ q(n+1, :) = ** \cdots * \circ q'(n+1, :)$ for $q \sim_n q'$.

Lemma 2. *If there exist $q, q' \in Q^*$ such that $q \sim_{n+1} q'$ then there exist $v, v' \in Q^*$ such that $v \sim_n v'$ or conversely if there are not n-step equivalent states then there are no $(n+1)$-step equivalent states.*

Proof. We prove this lemma constructively, i.e., let $q \sim_{n+1} q'$ then $\delta(q, w) = \delta(q', w)$ for all $w \in \{0, 1\}^{n+1}$, which is equivalent to $\delta(q, w(1) \circ w(2, :)) = \delta(\delta(q, w(1)), w(2, :))$ and thus $\delta(\delta(q, w(1)), w(2, :)) = \delta(\delta(q', w(1)), w(2, :))$. Let $v = \delta(q, w(1)) \in Q^*$ and $v' = \delta(q', w(1)) \in Q^*$ then due to the fact that $|w(2, :)| = n$ it must be that $v \sim_n v'$. □

From this lemma it follows that state equivalence must be constructed iteratively until no further n-step equivalent states can be generated from the set of $(n-1)$-step equivalent states for $n \geq 1$. We emphasize that we do not need to consider special constraints on w as e.g., only critical transitions exist for critical states, since only *nominal states* can be equivalent as shown in the following.

Lemma 3. *Only* nominal states *can be equivalent states in a compliant non-minimized automata $\mathcal{A}_k^*$.*

Proof. We prove by contradiction that only *nominal states* can be n-step equivalent. Assume that there exist any $q \sim_n q'$ such that q is a critical state and q' is a nominal state, i.e., by definition $\mathbb{1}[q(n+1, :)] = m - 1$ and $\mathbb{1}[q(n+1, :)] \geq m$. Since q' is equivalent by assumption we have that $q'(n+1, :) = q(n+1, :)$ and thus $\mathbb{1}[q'(n+1, :)] = m - 1$, which implies however that q' is not a nominal state and contradicts the assumption. □

Corollary 3. *The initial set of* nominal states *can be minimized to a set of representatives of the form $* \cdots * \circ v$ where v is the shortest v such that $\mathbb{1}[v] = m$ and the prior $k - |v|$ characters are* don't cares*.*

Proof. This follows from Lemma 2 and Lemma 3, since we know that states q, q' are merged up to n-step equivalence if $\mathbb{1}[q(n+1, :)] \geq m$ and thus $\mathbb{1}[* \cdots * \circ q(n+1, :)] \geq m$ where n is the maximal equivalence found and thus $v = q(n+1, :)$ $|v| = k - (n+1) + 1 = k - n$, i.e., shortest $|v|$. □

Theorem 1 (Minimal Automata)**.** *The minimal number of compliant states Q^* of a (m, k)-compliant $\mathcal{A}_k^*$ is given by*

$$|Q^*| = \frac{k!}{m! \times (k - m)!} \quad (5)$$

Algorithm 1 Generation of minimal compliant $\mathcal{A}_k^*$

Input: Constraint (m, k);
1: $\mathcal{A}_k^* \leftarrow (q_s, Q^* := \emptyset, \delta := \emptyset, \Sigma := \{0, 1\})$;
2: $q_s \leftarrow \{**\ldots 1\ \}$;
3: **for each** $z \in \{0, \ldots, k-m-1\}$ **do**
4: add $q := *_{k-m-z} \circ 1 \circ b(m+z-1, m-1)$ to Q^*;
5: add transition $\delta(q, 1) = q(2, :) \circ 1$ to δ;
6: add transition $\delta(q, 0) = q(2, :) \circ 0$ to δ;
7: **for each** $q \in \{w \in b(k-1, m-1) \mid 1 \circ w\}$ **do**
8: add q to Q^*;
9: add transition $\delta(q, 1) = q(2, :) \circ 1$ to δ;
10: **return** $\mathcal{A}_k^*$;

Proof. The number of compliant states is composed of *critical* and *nominal* states, where the number of *critical* states is given by $\binom{k-1}{m-1}$ since exactly the last $k-1$ characters in a *critical* state q must contain exactly $m-1$ ones.

From Lemma 3, we know that $m \leq |v| \leq k-m$ and thus states with $|v| = \ell$ and $\mathbb{1}[v] = m$ are merged into one representative state for $\ell \in \{m, m+1, \ldots, k-m\}$. The number of combinations for each above class is given by the binomial $\binom{\ell}{m}$. However for each ℓ the number of combinations for $\ell - 1$ must be substracted. This is due to the fact that by Lemma 2, we know that each state is represented by the maximal equivalence representative and the combinations with m ones in the last ℓ characters can be extended to combinations with m ones in the last $\ell + 1$, which should then be covered by the representative of ℓ. In consequence, we have that $|Q^*|$ is given by

$$\binom{k-1}{m-1} + \binom{m}{m} + \sum_{\ell=1}^{k-m} \binom{m+\ell}{m} - \binom{m+\ell-1}{m} = \binom{k}{m}$$

which proves the theorem. □

Let $b(z, n)$ denote all bit strings of length z with exactly n ones which can be recursively defined and computed using dynamic programming. Using the above observations and lemmas, we can generate all *critical* states by $\{w \in b(k-1, m-1) \mid 1 \circ w\}$ and for each *critical* state q, we add a critical transition $\delta(q, 1) = q(2, :) \circ 1$. To generate the minimal set of *nominal* states for (m, k)-constraints, we generate the representatives iteratively using $*_\ell$ to denote a string of ℓ many $*$-characters as follows:

$$\bigcup_{z=0}^{k-m-1} *_{k-m-z} \circ 1 \circ b(m+z-1, m-1) \tag{6}$$

For instance in the case of $(2, 4)$ constraints, the minimal *nominal* states are given by Eq. (6) as $**\circ 1 \circ b(1, 1) = **11$, $* \circ 1 \circ b(2, 1) = \{*110, *101\}$. For each merged *critical* state q the transitions $\delta(q, 0) = q(2, :) \circ 0$ and $\delta(q, 1) = q(2, :) \circ 1$ to the automata.

IV. Minimization of Expected Execution Time

In this section, we explain our mapping strategy of execution modes to jobs by considering different strategies for *critical* and *nominal* states in detail. Afterwards, an optimization strategy based on induced the Markov chain is proposed. In the end, an example is provided to illustrate the work flow of our proposed strategy.

A. Mapping Strategy

Our mapping strategy utilizes the design space of the compliant mapping strategy from Lemma 1 to select the execution mode for next job as follows.

Critical State Action. If the current state q is a critical state then the next job has to be executed correctly and therefore either of the following two actions must be taken:

1) Release a *reliable* task instance, i.e., $\alpha(q) = r$.
2) Release a *detected* task instance and only release an immediate follow-up *reliable* task instance in case of a detected error, i.e., $\alpha(q) = d + r$.

By this mapping, we have enforced that $c(\alpha(q)) = 1$ with probability 1. In the first case, the expected WCET of a job released in state q is either C_r or $(1-p_e) \cdot C_d + p_e \cdot C_r$. It can be seen that for very low error probabilities p_e it is better to first run a *detected* instance followed-up by a *reliable* instance.

Nominal State Action. If the current state q is a nominal state then the next job must not be enforced to be executed correctly. Thus we have the following three options to choose the next job's mode:

1) Release a *reliable* mode instance, i.e., $\alpha(q) = r$ and $c(\alpha(q)) = 1$ with probability 1.
2) Release a *detected* mode instance, i.e., $\alpha(q) = d$ and $c(\alpha(q)) = 1$ with probability $1 - p_e$ and $c(\alpha(q)) = 0$ with probability p_e.
3) Release an *unreliable* mode instance, i.e., $\alpha(q) = u$ and $c(\alpha(q)) = 0$ with probability 1.

Due to the assumed high worst-case execution time of the *reliable* instances, we opt to select either a *detected* or an *unreliable* mode instance in each nominal state q. We draw either a *detected* mode instance at random with probability p_d or an *unreliable* mode instance with probability p_u such that $p_d + p_u = 1$ and the expected execution time is thus given by $p_d \cdot C_d + p_u \cdot C_u$. Based on this randomized mode selection, the outcomes are stochastic in nature, i.e., $\mathbb{P}(c(\alpha(q)) = 1) = p_d \cdot (1 - p_e)$ and $\mathbb{P}(c(\alpha(q)) = 0) = p_d \cdot p_e + p_u$.

B. Induced Markov Chain

Using the mapping strategy α, we can derive an α-induced Markov chain from the automata $\mathcal{A}_k^*$.

Observation 1 (Induced Markov Chain)**.** *The α-induced (m, k)-compliant $\mathcal{A}_k^*(\alpha)$ is a finite discrete-time Markov chain with transition probability determined by the error probability and the mapping strategy α.*

Due to the state-based mode selection strategy, the probability of being in state q' at time $k+1$, i.e., $\mathbb{P}(x_{k+1} = q')$ only depends on the probability of being in a state q at time k for which $(q, q') \in \delta^*$ holds and the probability of taking a specific transition thereof. Therefore the Markov property is trivially

satisfied. Moreover, the specific transition probabilities are derived based on the error probability p_e and the stochastic state-based mode selection, i.e., if q is a nominal state then by our strategy α the following state transitions are given:

$$\mathbb{P}(x_{n+1} = \delta(q,0)|x_n = q) = \mathbb{P}(c(\alpha(q)) = 0) = p_d \cdot p_e + p_u$$
$$\mathbb{P}(x_{n+1} = \delta(q,1)|x_n = q) = \mathbb{P}(c(\alpha(q)) = 1) = p_d \cdot (1 - p_e)$$

and for critical states

$$\mathbb{P}(x_{n+1} = \delta(q,0)|x_n = q) = \mathbb{P}(c(\alpha(q)) = 0) = 0$$
$$\mathbb{P}(x_{n+1} = \delta(q,1)|x_n = q) = \mathbb{P}(c(\alpha(q)) = 1) = 1$$

Definition 9 (Stationary Distribution)**.** *Let a finite, irreducible Markov chain be given by $x_{n+1} = A \cdot x_n$, where $x_n \in Q^{*r}$, $A \in \mathbb{F}^{r \times r}$ and $||x_n||_1 = 1$ for all $n \in \mathbb{N}$ and $|Q^*| < \infty$. A probability distribution ξ is said to be a stationary distribution or invariant distribution if*

$$A \cdot \xi = \xi \tag{7}$$

We then obtain the corresponding stationary distribution ξ according to Eq. (7) by treating ξ as an eigenvector of A with an eigenvalue 1 that can be efficiently numerically solved by e.g., eigenvalue decomposition (spectral theorem [36]). Let the stationary distribution $\xi^T = (\xi_1, \ldots, \xi_r)$ where the ξ_i corresponds to the stationary probability to be in state $q_i \in Q^*$. In consequence the expected execution time is:

$$\mathbb{E}(C) := \sum_{i=1}^{r} \xi_i \cdot (p_d \cdot C_d + p_u \cdot C_u) \cdot [q_i \text{ is nominal}]$$
$$+\xi_i \cdot \min\{C_r, (1-p_e) \cdot C_d + p_e \cdot (C_d + C_r)\} \cdot [q_i \text{ is critical}]$$

Our formal objective is to minimize $\mathbb{E}(C)$ for each task individually with respect to the parameters p_d ($p_u = 1 - p_d$). What is left to show is that each α-induced (m,k)-compliant Markov chain always has a stationary distribution.

Theorem 2 (Renewal Theorem [16])**.** *A finite, irreducible Markov chain has a unique stationary distribution.*

Definition 10 (Irreducibility)**.** *A Markov chain is irreducible if for any two states, i.e., q, q' there exist $n, n' \in \mathbb{N}_0$ such that $\mathbb{P}(x_{i+n} = q|x_i = q') > 0$ and $\mathbb{P}(x_{i+n'} = q'|x_i = q) > 0$ for some $i \in \mathbb{N}$.*

Theorem 3. *The α-induced (m,k)-compliant $\mathcal{A}_k^*(\alpha)$ is a finite and irreducible discrete-time Markov chain.*

Proof. From Theorem 1, it immediately follows that $\mathcal{A}_k^*(\alpha)$ has finite states. Moreover, since the α-induced (m,k)-compliant $\mathcal{A}_k^*(\alpha)$ has non-zero probability for each transition by construction, we only have to prove that any two states q, q' are reachable from one another. We prove this theorem for the non-minimized automata, but since in the minimized automata only equivalent states are merged, it is obvious that the reachability property remains.

Let q, q' any two states in the non-minimized α-induced Markov chain $\mathcal{A}_k^*(\alpha)$ then there always exists a sequence of compliant transitions from $q \rightsquigarrow q'$ by decomposition of the transitions into $q \rightsquigarrow 11\ldots1$ (k ones) and $11\ldots1 \rightsquigarrow q'$. Since

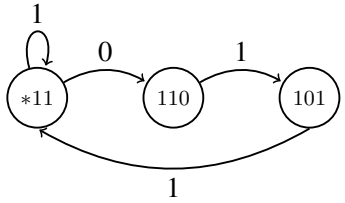

Fig. 2. A minimal $(2,3)$-compliant 3-error-automata $\mathcal{A}_3^*$.

for each feasible state $q \in Q$ the transition $\delta(q,1) \in Q$ is defined for *critical*- and *nominal states*, the state $11\ldots1$ can be reached from any state q by successive 1-transitions. Secondly, for any given (m,k)-constraints, starting from state $11\ldots1$, (by construction of the automata) we can use a 0-transition at most $(k-m)$ times and always be in a compliant state. This allows to reach any compliant state $q \in Q$ with $\mathbb{1}[q] \geq m$ to be reachable from $11\ldots1$, since $q = \delta(11\ldots1, q)$ and $\mathbb{1}[q] + \mathbb{0}[q] = k$ and thus $\mathbb{0}[q] = k - \mathbb{1}[q] \leq k - m$. □

C. An Illustrative Example

To illustrate our approach, we here provide a full example of the previously described task with $(m = 2, k = 3)$-constraints and assume that the execution times of the different job modes are given by $C^u = 1$, $C^d = 1.5$, and $C^r = 3$ and the task has a stationary soft error probability of $p_e = 0.1$. After minimization according to Algorithm 1, the generated automata is shown in Figure 2 The ordered set of the states Q is given by $\langle Q \rangle = \langle *11, 110, 101 \rangle$ where $*11$ is a *nominal* state and $110, 101$ are *critical* states. By using the mapping strategy described in Section IV-A, we derive the following non-zero transition probabilities:

$$\mathbb{P}(x_{n+1} = *11|x_n = *11) = p_d \cdot (1 - p_e) = 0.9 \cdot p_d$$
$$\mathbb{P}(x_{n+1} = 110|x_n = *11) = p_e \cdot p_d + p_u = 1 - 0.9 \cdot p_d$$
$$\mathbb{P}(x_{n+1} = 101|x_n = 110) = 1$$
$$\mathbb{P}(x_{n+1} = *11|x_n = 101) = 1$$

The corresponding transition probability matrix A is:

$$A = \begin{bmatrix} 0.9 \cdot p_d & 0 & 1 \\ 1 - 0.9 \cdot p_d & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{8}$$

where the row and column indexes are referring to the index of $\langle Q \rangle$. By solving the equation $\xi \in ker(A - I)$ such that $||\xi||_1 = 1$, we obtain the values $\xi_1 = 1/(3 - 1.8 \cdot p_d)$ and $\xi_2 = \xi_3 = (1 - 0.9 \cdot p_d)/(3 - 1.8 \cdot p_d)$ which yields the following expected execution time $\xi_1 \cdot p_u \cdot C^u + \xi_1 \cdot p_d \cdot C^d + 2 \cdot \xi_2 \cdot \min\{C_r, C^d + p_e \cdot C^r\}$ and evaluates to

$$\frac{p_u \cdot C^u + p_d \cdot C^d + 2(1 - 0.9 \cdot p_d) \cdot \min\left\{C_r, C^d + 0.1 \cdot C^r\right\}}{3 - 1.8 \cdot p_d}$$

$$\implies \frac{230 - 137 \cdot p_d}{150 - 90 \cdot p_d} \text{ for any } p_d \in [0,1] \tag{9}$$

The function in Eq. (9) is monotonically increasing on the interval $[0,1]$, which implies that the minimum value of the

expected execution time is attained for $p_d = 0$. Therefore, in every *nominal state* q the mapping is given by $\alpha(q) = u$, i.e., to always instantiate an unreliable instance next. Moreover, for each *critical state* q we always select

$$\alpha(q) = \begin{cases} d + r & \text{if } C_d < (1 - p_e) \cdot C^r \\ r & \text{otherwise} \end{cases} \tag{10}$$

In summary, we obtain the following mapping strategy:

$$\alpha(q) = \begin{cases} d + r & \text{if } q \in \{101, 110\} \\ u & \text{if } q \in \{*11\} \end{cases} \tag{11}$$

that results in a minimal expected execution time of 1.533. However, if we decrease the stationary soft error probability to 1%, i.e., $p_e = 0.01$, the expected execution time becomes

$$\frac{1150 - 766 \cdot p_d}{750 - 495 \cdot p_d} \text{ for any } p_d \in [0, 1] \tag{12}$$

In contrast to Eq. (9), Eq. (12) is monotonically decreasing on the interval $[0, 1]$ and thus the minimum expected execution time is attained when $p_d = 1$. This results in the altered strategy

$$\alpha(q) = \begin{cases} d + r & \text{if } q \in \{101, 110\} \\ d & \text{if } q \in \{*11\} \end{cases} \tag{13}$$

Despite the randomized mode selection strategy attempt, due to the linear form of the optimization objective, e.g., in Eq. (9) and Eq. (12), the maximum will always be at one of the interval ends resulting in an off-line calculated deterministic look-up table such as in Eq. (11) or Eq. (13).

V. Reinforcement Learning Based Approach

When the error probability for each task is unknown, the probabilities of state transitions are no longer explicit, which makes the static optimization approach proposed in Sec. IV inapplicable. To this end, we propose an artificial expert (agent) based on reinforcement learning (RL), to optimize the selections of execution modes opportunistically during the runtime. In this section, we first give a short overview of RL. Afterwards, we demonstrate how the execution mode selection with (m, k) constraints problem is formulated to the RL-solvable problem. Furthermore, the barrier function that assures the (m, k) constraint is discussed. Finally, we present the learning policy for RL agent for the studied problem.

A. Overview of Reinforcement Learning

As one of the machine learning paradigms, RL can be reformulated as a Markov Decision Process (MDP). Fig. 3 shows the basic components of a MDP. The *environment* E is defined as a *state* space, which is denoted as S. Each state instance, i.e., $s_t \in S$, is a description of the environment at time t. All the actions that an agent can take formulate the action space, i.e., A. One iteration of MDP is shown in Fig. 3: when an action $a_t \in A$ is taken based on current state s_t, the environment is transited to a new state s_{t+1}. During the transition, a reward r_t is given to the agent according to the reward function R. Therefore, reinforcement learning problem

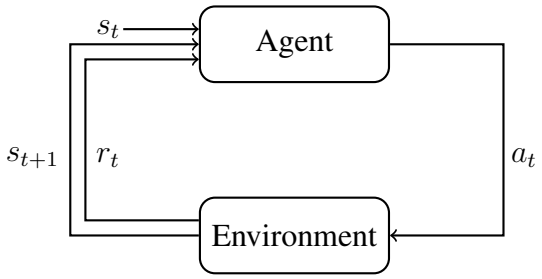

Fig. 3. Schematic of the agent and environment interaction, with the time-dependent states s, rewards r and action a

can be formulated as a four tuple, i.e, (S, A, P, R), where $P: S \times A \times S \mapsto \mathbb{R}$ represents the probability of state transitions $P(s_{t+1} \mid s_t, a_t)$, and $R: S \times A \times S \mapsto \mathbb{R}$ denotes the corresponding reward function. According to the Markov property, the next state s_{t+1} only depends on the current state s_t and current action a_t, and is conditionally independent to all previous states and actions.

The objective of a reinforcement learning task is to let the agent learn a policy π, which is a probability density function to describe the state-to-action mapping. That is, an agent can take an action according to the policy and current state, i.e., $a = \pi(s)$. A policy π can be formed into two different ways: a) the deterministic policy $\pi: S \mapsto A$ is a unique mapping from state to action; b) uniform stochastic policy $\pi: S \times A \mapsto A$ defines the probability distribution of actions according to a given state, i.e., $\pi(a_t|s_t) = \mathbb{P}(a = a_t \mid s = s_t)$ where $\sum_{a_t \in A} \pi(a_t|s_t) = 1$. The learned policy is evaluated by the cumulative future reward, i.e., $W_t = \sum_{i=t}^{\infty} r_i$. Since future reward is often less valuable than present reward, a discount rate $\gamma \in (0, 1]$ is included, i.e., $W_t = \sum_{i=t}^{\infty} \gamma^{i-t} r_i$. In addition, value function is applied to estimate the expected future reward. On the one hand, state-value function is defined as $V_\pi(s_t) = \mathbb{E}_A[Q_\pi(s_t, A)]$, which defines the expected cumulative reward from state s_t by applying policy π. On the other hand, action-value function for a policy π is defined as $Q_\pi(s_t, a_t) = \mathbb{E}[W_t | s = s_t, a = a_t]$, which indicates the quality of the action a_t under state s_t.

B. RL Formulation

We consider each task as an agent, and hence multiple agents are in the same task system. These multiple agents in this work are independent, i.e., they are not the same as multi-agents reinforcement learning. Each agent independently takes its own actions based on its observation without requiring any information from the others. Hence, in the following, we focus on the selection strategy of execution modes for one task.

As shown in Sec. IV-B, the job level execution modes selections for each task is an independent MDP. The action space for each task is, selecting the execution mode for its next job, i.e., $A = \{0: \mathit{unreliable},\ 1: \mathit{detected},\ 2: \mathit{reliable}\}$. The environment state is a set of execution statuses for a tasks' corresponding jobs, i.e., $s_t = [s_t^1, s_t^2, \ldots, s_t^\ell]$. The execution status of each job, i.e., s_t^j, has the following four attributes:

- *Correctness*: it is a binary variable, that indicates the correctness of the corresponding job, i.e., according to Definition 1, 0 denotes error execution and 1 denotes correct execution.
- *Execution mode*: it is used to record the execution mode of a job, i.e., one element from the action space.
- *Expected execution time*: according to the corresponding execution mode, each job has its own expected execution time, e.g., C^u, C^d, or C^r. Please note, the expected execution times here are the same as the WCET of different modes, which are different to the expected execution times in Sec. IV-A.
- *Real execution time*: it is used for recording the execution time of a job. For both *unreliable* and *reliable* modes, a job should have the same real execution time as the expected execution time. However, for the *detected* mode, the real execution time may be much longer than expected when a job is forced to be correct according to (m, k) constraint but is detected as *error*. The job has to re-execute the *reliable* mode to satisfy the (m, k) requirement, where the real execution time equals to $C^d + C^r$.

The length of an environment state ℓ should not be less than k, so that the execution statuses of at least k jobs can be recorded and used to check the (m, k) constraint. Once the length of the environment state is determined and fixed ($\ell = 2k$ in this work), a FIFO stack-like mechanism is applied, i.e., the state matrix only records the latest ℓ jobs' execution statuses.

One example of the environmental state transition is shown in Fig. 4, where a task has the $(m = 3, k = 5)$ constraint. The length of the environment state is set to $k = 5$. The task takes the action to execute the *detected* mode, i.e., $a_t = 1 :$ *detected* for its next new job J^6. The second and third elements of J^6's status have been determined as $\begin{pmatrix} 1 \\ C^d \end{pmatrix}$ by default. By checking the $(m = 3, k = 5)$ constraint, the J^6 has to be executed correctly, therefore, the first element is marked as 1. However, in the real execution, J^6 is detected as *error*, an additional *reliable* mode has to be executed immediately. As a result, the last element of J^6's status is $C^d + C^r$. Afterwards, the state is transited from s_t to s_{t+1} by adding the status of J^6 and deleting the status of J^1.

$$\begin{array}{ccccc} J^1 & J^2 & J^3 & J^4 & J^5 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 2 \\ C^d & C^u & C^d & C^d & C^r \\ C^d & C^u & C^d & C^d & C^r \end{array} \quad \rightarrow \quad \begin{array}{ccccc} J^2 & J^3 & J^4 & J^5 & J^6 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 & 1 \\ C^u & C^d & C^d & C^r & C^d \\ C^u & C^d & C^d & C^r & C^d + C^r \end{array}$$

$$s_t \xrightarrow[a_t = 1]{} s_{t+1}$$

Fig. 4. An example for state transition of a task.

Please note, different environment construction approaches can also be applied. For example, a three dimensional tensor can be applied to record several latest aforementioned two dimensional matrices, so that the more comprehensive information is recorded without discarding.

C. Barrier Function

To achieve the objective of minimizing the average execution time for each task, the reward is intuitively set to be inversely proportional to a job's real execution time, i.e., the longer real execution time a job has, the less reward the agent obtains. Besides the objective of the RL-based approach, i.e., maximize the cumulative reward, the (m, k) constraint has to be satisfied as well. Towards this, we design a barrier function, by which the category of the current state is checked before the agent deploys the job with selected execution mode. If the current state $s_t \in S_{nom}$, the barrier function bypasses the checking, since all three execution modes are legal for next job However, if the current state $s_t \in S_{crt}$, the result of next job has to be correct. If the agent decides to execute the *unreliable* mode, the barrier function forbids the action and limit the options to only execute the *detected* or the *reliable* mode. In addition, an extremely large negative reward is returned to the agent. Such a barrier function thus can: 1) guarantee the satisfaction of (m, k) constraint, and 2) let the agent learn to not select the *unreliable* mode when $s_t \in S_{crt}$.

Although our agent is model free, the barrier function implicitly enforces it follow the R-pattern in the worst case:

Theorem 4. *The worst case execution pattern of RL-based approach is the same as the R-pattern adopted in [9], [24].*

Proof. The worst case execution R-pattern in [9] contains $(k-m)$ incorrect jobs that are executed in the *detected* mode, and m correct jobs that are executed in the *detected* mode and the *reliable* mode, where all the results of k jobs with the *detected* mode are incorrect. In our RL-based approach, the agent can only select one execution mode for the next job. The barrier function is applied to check the feasibility of the selected execution mode and the correctness of the result when a job finishes its execution. If the current state $s_t \in S_{nom}$, no additional effort is needed. Only when the current state $s_t \in S_{crt}$, the execution pattern, i.e., the *detected* mode and *reliable* mode are executed for the same job, can happen. The worst case of the RL approach is that the agent always decides to execute *detected* mode and all the execution results are incorrect. Therefore, m jobs with *reliable* mode are executed right after the *detected* mode, which is the same as the R-pattern in [9], [24]. □

Theorem 5. *The derived schedule from the RL-based approach under the barrier function is schedulable if the schedulability based on the R-pattern has been ensured.*

Proof. Given a static pattern, a task is schedulable if it passes the schedulability test in Lemma 1 from [9]. As shown in Theorem 4, in the worst case the RL-based approach performs the same as the R-pattern. Therefore, if a task has passed the schedulaiblity test in Lemma 1 from [9] based on the R-pattern, the derived schedule for this task from the RL-based approach must be schedulable in the worst case, which concludes the proof. □

D. Learning Policy

In this work, we utilize the deep Q network (DQN) agent as our RL model, where Boltzmann Q Policy is applied to estimate the Q value of each action. While exploring, the agent creates an action distribution. This distribution describes how optimal an action is according to the data gathered by the agent. Afterwards, Boltzmann policy turns the agent's exploration behavior into a spectrum between picking the action randomly (random policy) and always picking the most optimal action (greedy policy). The DQN agent is constructed by a 10-layer neural network, which contains 1 input layer, 1 activation layer, 1 flatten layer, 6 fully connected layers, and 1 output layer.

Please note that the proposed RL-based approach is not limited to any specific learning policy, all the learning approaches support discrete action space are applicable. Finding the best policy to train a DQN agent is considered out of scope.

VI. Evaluation

To evaluate the effectiveness of our proposed approaches, we numerically simulate the task system and compare the performance of the proposed mapping strategy when p_e is known and RL-based approach when p_e is unknown with the state-of-the-art over a wide range of different configurations. The adopted hardware platform was a cache-coherent SMP, consisting of one 64-bit Intel i7-8700k processors running at 3.7 GHz, with 32 GB of main memory. Overall, the following approaches are evaluated, namely:

- Optimized mapping strategy in Sec. IV (OPT): the selection of execution mode for each state follows the optimized mapping between states and execution modes.
- RL-based approach in Sec. V (RL): the environment is constructed by using OpenAI Gym [6]. The implementation of RL agent relies on Keras-rl package [34] and TensorFlow [1].
- Adaptive approach (ADP) [9]: R-pattern is applied, i.e., postpone the forced-correct jobs as late as possible, which can benefit this approach due to the flexibility.
- Static approach (STA) [30]: executes m jobs in the reliable mode and $(k-m)$ jobs in the unreliable mode for any consecutive jobs. Here, R-pattern will be equal to E-patterns in terms of utilization reduction, e.g., energy saving, regardless of the given error probabilities.

A. Single Task Evaluation

We conducted evaluations for one single task with different experimental settings, e.g., (m, k) constraint and error probability. The m was selected from a set, i.e., $m \in \{2, 4, 6, 8\}$, the k was a constant number, i.e., 10, and the error probability was given as $p_e \in \{0.05, 0.15, 0.3\}$. We set $C^d = 1.5 \times C^u$ and $C^r = 3.5 \times C^u$ to emulate the software-based error detection and error recovery. Each task released $10,000$ jobs for one iteration, and 100 iterations were performed.

Fig. 5 shows the results for one single task, where the y-axis represents the normalized average execution time for jobs of one task, i.e., the lower the better. In general, the

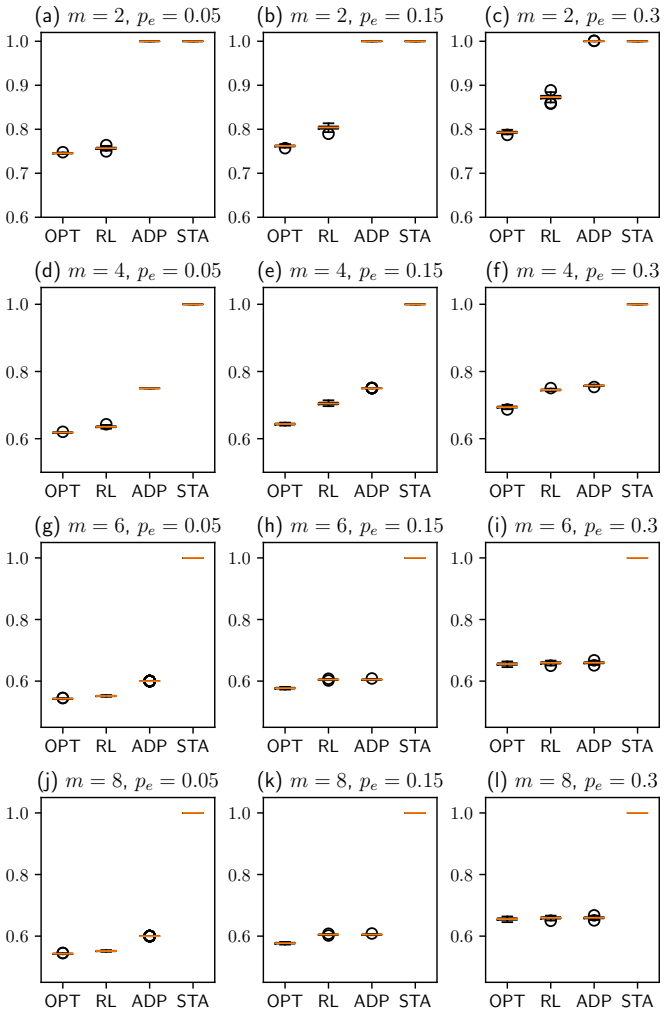

Fig. 5. Results for tasks with different settings.

OPT approach outperforms all the other approaches in all the evaluated cases. When the error probability p_e or the ratio m/k is relatively low, e.g., in Fig. 5 (a)-(h), (j), and (k), the OPT approach outperform other approaches significantly. When both error probability and the ratio m/k increase, e.g., in Fig. 5 (i) and (l), the options to select the execution modes become more limited, which result in negligible difference of the OPT approach, RL approach, and ADP approach.

The RL approach also dominates in most of the evaluated cases of ADP and STA approaches, e.g., in Fig. 5 (a)-(g) and (j), but it always performs worse than the OPT approach (without knowing the error probability in advance). When the error probability is relatively low, e.g., in Fig. 5 (a), (d), (g) and (j), or both error probability and the ratio m/k are relatively high, e.g., in Fig. 5 (i) and (l), the difference between OPT and RL is minor. For a given (m, k) constraint, when the error probability increases, e.g., rows of Fig. 5, or for a given error probability, the number of tolerable error jobs becomes less (m increases with a constant k), e.g., columns of Fig. 5, we can observe that the achievable benefit from the RL approach significantly decreases. It is because the agent tends to execute

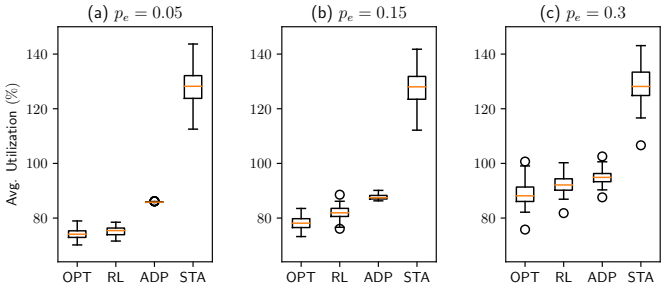

Fig. 6. Results for multitask systems with different error probabilities.

the *unreliable* modes first to maximize the cumulative reward. However, such an intention in fact also reduces the resilience, so that the upcoming jobs executing in the *detected* modes often have to execute the *reliable* mode immediately when a error is detected.

B. Multitask System Evaluation

We conducted the evaluation for multitask on multiprocessor systems as well, where tasks were scheduled by partitioned scheduling. We considered 100 task sets, each of them contained 40 tasks that were scheduled on 4 processors. The total utilization for each task set $U_{\mathbb{T}} \in [20\%, 200\%]$ with each step 20%, when all tasks only executing in the *reliable* modes. For each task, the utilization was generated by applying the Dirichlet-Rescale (DRS) algorithm [15], where utilization for each task was not higher than 50%. The task periods T_i were randomly selected from a set of semi-harmonic periods, i.e., $T_i \in \{1, 2, 5, 10, 20, 50, 100, 200, 1000\}$, which is the periods recommended for automotive systems [25]. The execution time of *reliable* mode for each task was calculated, i.e., $C_i^r = U_i * T_i$, and C_i^u and C_i^d are calculated with the same ratios in Sec. VI-A. We considered the worst-fit partitioning algorithm. That is, tasks are sorted decreasingly at first. Afterwards, each unassigned task (with the largest utilization) was assigned on the processor with lowest utilization. The configurations for (m, k) constraints were the same as described in Sec. VI-A. We set each system with only one unified error probability for all the tasks running on it. We set a hyper-period as 10,000 time units, and obtain the average utilization of the system.

Due to the similarity over results, we selectively show the task systems with total utilization 50% in Fig. 6 to present the trends. In general, the results show that the both OPT approach and RL approach can decrease the utilization for multitask systems in all the evaluated cases. The proposed OPT approach can save 11.7%, 9.56%, and 6.31% in comparison to the state-of-the-art in [9], and save 54.1%, 49.42%, and 40.08% in comparison with the static approach in [30] with different error probabilities respectively. In particular, the OPT approach and RL approach can unleash their power, when the error probability is relatively low, i.e. Fig. 6 (a).

C. Overhead and Applicability Discussion

For each task, the optimal approach generates a lookup table in offline for the mode selection on the fly. The offline computational overhead depends on the (m, k) constraints, e.g., $(m = 2, k = 10)$ takes 10 seconds, $(m = 4, k = 10)$ takes 20.7 hours, $(m = 6, k = 10)$ takes 7.8 hours, and $(m = 8, k = 10)$ takes 0.5 seconds on average. The runtime overhead is only table lookup and negligible.

To evaluate the overhead of training and mode selection, we deployed our RL-base approach on both Intel desktop and Nvidia Jetson AGX Xavier (32G) board. On the Intel desktop, the training process for each task with one configuration took 450 seconds on average. The training process is repeated for 20 times, and the trained DQN with the highest reward is selected, and the overhead for each task to select the execution mode for next job is 300 microseconds. On the Nvidia Jetson AGX Xavier board, two power modes with different power budgets are evaluated, i.e., a) default mode with $15W$ power budget with 4 processors running at 2188 MHz and b) MAXN mode without power budget limitation with 8 processors running at 2265.6 MHz. The detailed configurations can be found in [32].

The overhead for training and online execution mode selection of RL-based approach only depends on the structure of DQN, regardless of given (m, k) constraints. In default mode, the training process took 19.7 minutes, and execution mode selection took 1.06 milliseconds on average. In MAXN mode, the training process took 15 minutes, and execution mode selection took 1 milliseconds on average. We observe that, the number of processors increasing from default to MAXN does not reduce the training time proportionally, and none of the processor is full-loaded in our evaluation. In the inference phase, i.e., selecting execution mode, the MAXN mode slightly outperforms the default mode due to the minor boost of single core frequency.

For the applicability in real world, a lookup table can also be utilized, because the state space in our application, i.e., the minimal legal space S^*, is limited. That is, the trained DQN network can be converted to a table, that shows the mapping between states and corresponding probabilities of different execution modes. In that case, the runtime overhead for selecting execution modes of tasks is negligible.

For systems with unknown probabilities, as one safe policy, the ADP proposed in [9] can be first applied to estimate a safe probability for the current scenario. If the overhead is acceptable, the proposed automata-based approach can derive the selection policy. However, to optimize the policy, recalculating for each scenario is rather expensive. The RL approach could still be more effective in this case.

VII. Related Work

In control theory, controllers are designed to be able to tolerate erroneous input signals and ensure the functionalities of control systems in the environments with uncertainties. Towards this, several techniques are proposed to deal with delayed [38], [26] or dropped signal samples [20], [7], [14]. In case an error is present in the sample input, the sample can be dropped. Successively, a control decision can be computed by using previous inputs to continue the loop [38], [20], [7].

Another series of fault tolerance techniques rely on the (m, k) models, which is originally developed for guaranteeing limited deadline misses for firmed real-time systems, or so-called weakly-hard real-time systems [4], where a task has to meet at least m deadlines, or can miss at most m deadlines, in any of k consecutive jobs[2]. Although the original work [4] applied $\binom{n}{m}$ to describe the any n of m, most of the following works utilize the (m, k) to describe the weakly-hard constraints [11], [18], [39], [43]. Afterwards, such (m, k) models are commonly applied for fault tolerance related domain as well to describe the robustness constraints of (control) systems. That is, a task must have at least m functionally correct instances out of any k consecutive instances. Such a requirement can ensure that a control system is still feasible only if it can satisfy the corresponding (m, k) robustness constraint [9], [45]. In order to comply a given (m, k) constraint, several static patterns are widely applied for different purposes, i.e., deep red pattern (R-pattern) [24], evenly distributed pattern (E-pattern) [38], and reverse E-pattern [37]. Besides the static pattern based approaches, Chen et al. proposed an adaptive approach in [9]. Such an approach tries to minimize the overall execution time of a task by postponing the execution of reliable mode. Liang et al. in [28] developed new methods and an optimization algorithm to analyze and improve control stability and system schedulability under deadline misses, faults, and the application of two different fault-tolerance techniques, where redundant execution is applied of EOC [13] techniques and re-execution are performed in case of a soft error. In addition, to maintain the control quality and ensure the schedulability, skipping certain control computation is considered. AlEnawy et al. in [2] presented an on-line speed adjustment algorithms to exploit the slack time of skipped and completed jobs in order to minimize the number of dynamic failures (in terms of (m,k)-firm deadline constraints) while remaining within the energy budget. Von der Brüggen et al. in [42] determined if the system with dynamic real-time guarantee can provide full timing guarantees or limited timing guarantees without any online adaptation after a fault occurred. Wang et al. in [44] presented a cross-layer approach to improve system adaptability by allowing proactive skipping of task executions. Huang et al. developed an online intermittent-control framework in [23], which combines formal verification with model-based optimization and deep reinforcement learning. The objective of the proposed framework is to opportunistically skip certain control computation and actuation to save actuation energy and computational resources without compromising system safety. Their main constraint is the control safety rather than schedulability and (m, k) robustness.

In recent years, machine learning (ML) has attracted a lot of interests in both academic and industrial areas. However, only a few of works applied ML based approaches in (real-time) embedded systems domain due to the stringent requirements, e.g., timing guarantee, security, and power consumption. Bo et al. in [5] proposed a deep RL based scheduler for multiprocessor real-time systems, which models the real-time scheduling process as a multi-agent cooperative game. In [27], a ML-based approach for priority assignment was proposed. To meet the complex time critical requirements, Dole et al. in [12] advocated the use of Duration Calculus (DC) to express the learning objectives in model-free RL for stochastic real-time systems. In order to improve the efficiency of generating Clock Constraint Specification Language (CCSL) specifications, Hu et al. in [22] combined the merits of both RL and deductive techniques in logical reasoning for efficient co-synthesis of CCSL specifications. In our work, we employ RL to decide the execution modes during runtime to avoid over-provision, without violating the validated schedulability in offline.

[2]Although nonconsecutive situations are considered as well in the original work, this situation is not considered in this work and hence omitted here.

VIII. Conclusion

In this work, we study how to selectively deploy fault-tolerance techniques as different execution modes under (m, k) constraints to reduce the number of expensive execution eventually saving energy, while satisfying the schedulability. Through formulating the mapping between states and selections for execution modes of jobs, we propose two different adaptive approaches. When the error probability is known, we provide a Markov chain based approach to optimize the mapping strategy. When the error probability is unknown, an RL-based agent is trained to aid the selection of the execution mode for its next job. The evaluation results show that both proposed approaches outperform the state-of-the-art in most of the evaluated cases, especially when the error probability is relatively low under the same (m, k) constraint.

Acknowledgement

This work has been supported by Deutsche Forschungsgemeinschaft (DFG), as part of the Collaborative Research Center SFB876, subproject A1 and A3 (http://sfb876.tu-dortmund.de/). This result is part of a project (PropRT) that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 865170).

References

[1] M. Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] T. A. AlEnawy and H. Aydin. Energy-constrained scheduling for weakly-hard real-time systems. In *Proceedings of the 26th IEEE Real-Time Systems Symposium RTSS*, pages 376–385, 2005.

[3] R. Baumann. Radiation-induced soft errors in advanced semiconductor technologies. *IEEE Transactions on Device and Materials Reliability*, 5(3):305–316, 2005.

[4] G. Bernat, A. Burns, and A. Llamosí. Weakly hard real-time systems. *IEEE Trans. Computers*, 50(4):308–321, 2001.

[5] Z. Bo, Y. Qiao, C. Leng, H. Wang, C. Guo, and S. Zhang. Developing real-time scheduling policy by deep reinforcement learning. In *27th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS*, pages 131–142, 2021.

[6] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[7] T. Bund and F. Slomka. Sensitivity analysis of dropped samples for performance-oriented controller design. In *IEEE 18th International Symposium on Real-Time Distributed Computing, ISORC*, pages 244–251, 2015.

[8] J. Chen and C. Kuo. Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms. In *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 28–38. IEEE Computer Society, 2007.

[9] K. Chen, B. Bönninghoff, J. Chen, and P. Marwedel. Compensate or ignore? meeting control robustness requirements through adaptive soft-error handling. In *Proceedings of the 17th ACM Conference on Languages, Compilers, Tools, and Theory for Embedded Systems*, pages 82–91, 2016.

[10] K.-H. Chen, G. v. der Brüggen, and J.-J. Chen. Reliability optimization on multi-core systems with multi-tasking and redundant multi-threading. *IEEE Transactions on Computers*, 67(4):484–497, 2018.

[11] H. Choi, H. Kim, and Q. Zhu. Job-class-level fixed priority scheduling of weakly-hard real-time systems. In *25th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS*, pages 241–253, 2019.

[12] K. Dole, A. Gupta, J. Komp, S. Krishna, and A. Trivedi. Event-triggered and time-triggered duration calculus for model-free reinforcement learning. In *2021 IEEE Real-Time Systems Symposium (RTSS)*, pages 240–252. IEEE, 2021.

[13] Y. Gao, S. K. Gupta, and M. A. Breuer. Using explicit output comparisons for fault tolerant scheduling (FTS) on modern high-performance processors. In *Design, Automation and Test in Europe, DATE*, pages 927–932. ACM, 2013.

[14] S. Ghosh, S. Dey, and P. Dasgupta. Synthesizing performance-aware (m, k)-firm control execution patterns under dropped samples. In *32nd International Conference on VLSI Design and 18th International Conference on Embedded Systems, VLSID*, pages 1–6. IEEE, 2019.

[15] D. Griffin, I. Bate, and R. I. Davis. Generating utilization vectors for the systematic evaluation of schedulability tests. In *IEEE Real-Time Systems Symposium*, 2020.

[16] G. Grimmett and D. Stirzaker. *Probability and random processes*. Oxford university press, 2020.

[17] M. Hamdaoui and P. Ramanathan. A dynamic priority assignment technique for streams with (m, k)-firm deadlines. *IEEE Transactions on Computers*, 44(12):1443–1451, 1995.

[18] Z. A. H. Hammadeh, S. Quinton, and R. Ernst. Dblp:journals/tvlsi/niuq06. *ACM Trans. Embed. Comput. Syst.*, 18(6):121:1–121:25, 2020.

[19] M. A. Haque, H. Aydin, and D. Zhu. On reliability management of energy-aware real-time systems through task replication. *IEEE Trans. Parallel Distributed Syst.*, 28(3):813–825, 2017.

[20] E. Henriksson, H. Sandberg, and K. H. Johansson. Predictive compensation for communication outages in networked control systems. In *Proceedings of the 47th IEEE Conference on Decision and Control, (CDC)*, pages 2063–2068, 2008.

[21] J. Hopcroft. An n log n algorithm for minimizing states in a finite automaton. In *Theory of machines and computations*, pages 189–196. Elsevier, 1971.

[22] M. Hu, J. Ding, M. Zhang, F. Mallet, and M. Chen. Enumeration and deduction driven co-synthesis of ccsl specifications using reinforcement learning. In *2021 IEEE Real-Time Systems Symposium (RTSS)*, pages 227–239. IEEE, 2021.

[23] C. Huang, S. Xu, Z. Wang, S. Lan, W. Li, and Q. Zhu. Opportunistic intermittent control with safety guarantees for autonomous systems. In *57th ACM/IEEE Design Automation Conference, DAC 2020, San Francisco, CA, USA, July 20-24, 2020*, pages 1–6. IEEE, 2020.

[24] G. Koren and D. Shasha. Skip-over: algorithms and complexity for overloaded systems that allow skips. In *Proceedings 16th IEEE Real-Time Systems Symposium*, pages 110–117, 1995.

[25] S. Kramer, D. Ziegenbein, and A. Hamann. Real world automotive benchmark for free. In *6th International Workshop WATERS*, 2015.

[26] P. Kumar, D. Goswami, S. Chakraborty, A. Annaswamy, K. Lampka, and L. Thiele. A hybrid approach to cyber-physical systems verification. In *The 49th Annual Design Automation Conference 2012, DAC*, pages 688–696. ACM, 2012.

[27] S. Lee, H. Baek, H. Woo, K. G. Shin, and J. Lee. ML for RT: priority assignment using machine learning. In *27th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS*, pages 118–130, 2021.

[28] H. Liang, Z. Wang, R. Jiao, and Q. Zhu. Leveraging weakly-hard constraints for improving system fault tolerance with functional and timing guarantees. In *IEEE/ACM International Conference On Computer Aided Design, ICCAD*, pages 101:1–101:9. IEEE, 2020.

[29] Y. Liu, R. P. Dick, L. Shang, and H. Yang. Accurate temperature-dependent integrated circuit leakage power estimation is easy. In *2007 Design, Automation and Test in Europe Conference and Exposition, DATE*, pages 1526–1531. EDA Consortium, 2007.

[30] L. Niu and G. Quan. Energy minimization for real-time systems with (m, k)-guarantee. *IEEE Transactions on Very Large Scale Integration Systems*, 14(7):717–729, 2006.

[31] L. Niu and D. Zhu. Reliable and energy-aware fixed-priority (m, k)-deadlines enforcement with standby-sparing. In *2020 Design, Automation & Test in Europe Conference & Exhibition, DATE*, pages 424–429. IEEE, 2020.

[32] Nvidia. Nvidia jetson agx xavier supported modes and power efficiency, 2022. Available at https://docs.nvidia.com/jetson/archives/l4t-archived/l4t-3261/Tegra%20Linux%20Driver%20Package%20Development%20Guide/power_management_jetson_xavier.html#wwpID0E0XO0HA. Visited on 2023.03.01.

[33] N. Oh, P. P. Shirvani, and E. J. McCluskey. Control-flow checking by software signatures. *IEEE Trans. Reliab.*, 51(1):111–122, 2002.

[34] M. Plappert. keras-rl. https://github.com/keras-rl/keras-rl, 2016.

[35] D. K. Pradhan, editor. *Fault-Tolerant Computing: Theory and Techniques; Vol. 1*. Prentice-Hall, Inc., USA, 1986.

[36] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[37] G. Quan and X. Hu. Enhanced fixed-priority scheduling with (m,k)-firm guarantee. In *Proceedings 21st IEEE Real-Time Systems Symposium*, pages 79–88, 2000.

[38] P. Ramanathan. Overload management in real-time control applications using (m, k)-firm guarantee. *IEEE Trans. Parallel Distributed Syst.*, 10(6):549–559, 1999.

[39] M. Shirazi, M. Kargahi, and L. Thiele. Performance maximization of energy-variable self-powered (m, k)-firm real-time systems. *Real Time Syst.*, 56(1):64–111, 2020.

[40] H. Su, F. Liu, A. Devgan, E. Acar, and S. R. Nassif. Full chip leakage estimation considering power supply and temperature variations. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 78–83. ACM, 2003.

[41] T. N. Vijaykumar, I. Pomeranz, and K. Cheng. Transient-fault recovery using simultaneous multithreading. In Y. N. Patt, D. Grunwald, and K. Skadron, editors, *29th International Symposium on Computer Architecture (ISCA 2002), 25-29 May 2002, Anchorage, AK, USA*, pages 87–98. IEEE Computer Society, 2002.

[42] G. von der Brüggen, K. Chen, W. Huang, and J. Chen. Systems with dynamic real-time guarantees in uncertain and faulty execution environments. In *2016 IEEE Real-Time Systems Symposium, RTSS*, pages 303–314, 2016.

[43] N. Vreman, R. Pates, and M. Maggio. Weaklyhard.jl: Scalable analysis of weakly-hard constraints. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2022.

[44] Z. Wang, C. Huang, H. Kim, W. Li, and Q. Zhu. Cross-layer adaptation with safety-assured proactive task job skipping. *ACM Trans. Embed. Comput. Syst.*, 20(5s):100:1–100:25, 2021.

[45] M. Yayla, K. Chen, and J. Chen. Fault tolerance on control applications: Empirical investigations of impacts from incorrect calculations. In *4th International Workshop on Emerging Ideas and Trends in the Engineering of Cyber-Physical Systems, EITEC@CPSWeek*, pages 17–24, 2018.