Cross-Layer Adaptation with Safety-Assured Proactive Task Job Skipping

ZHILU WANG, Northwestern University, USA
CHAO HUANG, University of Liverpool, UK and Northwestern University, USA
HYOSEUNG KIM, University of California, Riverside, USA
WENCHAO LI, Boston University, USA
QI ZHU, Northwestern University, USA

During the operation of many real-time safety-critical systems, there are often strong needs for adapting to a dynamic environment or evolving mission objectives, e.g., increasing sampling and control frequencies of some functions to improve their performance under certain situations. However, a system's ability to adapt is often limited by tight resource constraints and rigid periodic execution requirements. In this work, we present a cross-layer approach to improve system adaptability by allowing *proactive skipping* of task executions, so that the resources can be either saved directly or re-allocated to other tasks for their performance improvement. Our approach includes three novel elements: 1) formal methods for deriving the feasible skipping choices of control tasks with safety guarantees at the functional layer, 2) a schedulability analysis method for assessing system feasibility at the architectural layer under allowed task job skippings, and 3) a runtime adaptation algorithm that efficiently explores job skipping choices and task priorities for meeting system adaptation requirements while ensuring system safety and timing correctness. Experiments demonstrate the effectiveness of our approach in meeting system adaptation needs.

CCS Concepts: • **Computer systems organization** → *Embedded and cyber-physical systems*; *Robotic control*; *Real-time systems.*

Additional Key Words and Phrases: Cross-layer, adaptation, safety, weakly hard

ACM Reference Format:
Zhilu Wang, Chao Huang, Hyoseung Kim, Wenchao Li, and Qi Zhu. 2021. Cross-Layer Adaptation with Safety-Assured Proactive Task Job Skipping. *ACM Trans. Embedd. Comput. Syst.* 1, 1, Article 1 (January 2021), 25 pages. https://doi.org/10.1145/3477031

1 INTRODUCTION

Many real-time embedded systems, such as automotive and robotic systems, closely interact with the physical environment that they operate within. During runtime, there are often strong needs to adapt the system for better responses to unpredictable disturbances, dynamic environment, and changing missions [51, 52]. For instance, when a sporadic external perturbation occurs and affects

This article appears as part of the ESWEEK-TECS special issue and was presented in the International Conference on Embedded Software (EMSOFT), 2021.
Authors' addresses: Zhilu Wang, Northwestern University, Evanston, IL, USA, zhilu.wang@u.northwestern.edu; Chao Huang, University of Liverpool, Liverpool, UK, Northwestern University, USA, chao.huang2@liverpool.ac.uk; Hyoseung Kim, University of California, Riverside, Riverside, CA, USA, hyoseung@ucr.edu; Wenchao Li, Boston University, Boston, MA, USA, wenchao@bu.edu; Qi Zhu, Northwestern University, Evanston, IL, USA, qzhu@northwestern.edu.

1539-9087/2021/1-ART1 $15.00
https://doi.org/10.1145/3477031

a control loop, increasing the sampling and control frequency of that loop [15] or assigning it with high quality-of-service resources [40, 41] may help reject the perturbation faster; when the environment becomes challenging to navigate, deploying more sophisticated sensing and control strategies could help improve the system performance, stability and safety (but also consume more resources); or when the environment becomes less challenging, some of the control actuation may be skipped to save actuation energy [27, 47].

However, many of the practical embedded systems are resource limited and have little space for accommodating adaptations that require more resources (e.g., for increasing sampling periods). This is further exacerbated by the fact that many of those systems are also safety-critical and employ rigid execution requirements for ensuring functional correctness – in particular, their tasks are often periodically executed and every task instance (job) is required to be performed successfully. The combined effect of tight resource constraints and rigid execution requirements often makes it difficult to perform effective adaptation in traditional real-time embedded systems.

Our Approach: To address the above challenges and facilitate runtime adaptation, we present an approach that allows proactive skipping of task executions (i.e., not executing some of the task instances at all), so that resources can be re-allocated for improving other tasks during adaptation or saved directly for energy efficiency. Our approach is inspired by the observation that many system functions can in fact tolerate a certain degree of task instance failures. For instance, recent works [20, 24, 26, 34, 37] have studied control performance, stability, and safety under weakly-hard constraints, where occasional deadline misses are allowed, and demonstrated that it is still possible to formally ensure the correctness of those functional properties under bounded deadline misses.

To realize this vision, we need to address three technical challenges: 1) how to formally verify whether the functional correctness can be preserved under certain task skipping choices; 2) how to assess whether the adapted system is feasible at the architectural level, i.e., meeting scheduling and resource constraints, under skipping choices and other changes such as increased frequencies for other tasks; and 3) how to automatically and efficiently reconfigure the system at runtime, e.g., deciding which task skipping choice to make and how to reconfigure other parameters such as task priorities, for meeting adaptation requirements. Correspondingly, we develop a **novel cross-layer approach for improving system adaptability that includes three elements**:

- A formal *state-aware safety verification method* that verifies control safety under various task job skipping patterns based on computing invariant sets and identifies the feasible skipping choices (e.g., the maximum number of jobs that are allowed to be skipped within a given time window). Compared with previous formal methods that verify control safety under deadline misses (task instance failures) [24, 26], our method considers current system state and is able to provide a much larger feasible space for task skipping.
- An *event-based schedulability analysis method* that considers task job skipping choices and other system changes, and evaluates whether all remaining task instances can meet their deadlines in the worst case.
- An *efficient runtime adaptation algorithm* that leverages the safety verification method at the functional layer and the schedulability analysis method at the architectural layer for exploring feasible adaptation space, including various task skipping choices and task priority assignment, to meet adaptation requirements and optimize adaptation goals (e.g., increasing certain task frequencies). The adaptation solutions found by our algorithm are guaranteed to ensure system safety and scheduling feasibility. For efficiency purpose, our approach may pre-compute feasible task skipping choices for each possible state and directly check the results during runtime.

We evaluate our adaptation framework with a case study of a robotic system. The experimental results demonstrate that our approach can significantly outperform a no-skipping adaptation

approach that only adjusts task periods but does not skip jobs, in terms of meeting adaptation goals, improving control performance, and maintaining system safety.

The rest of the paper is structured as follows. Section 2 introduces some of the most relevant work. Section 3 presents our system model in both functional layer and architecture layer. Section 4 introduces our cross-layer adaptation approach, including the safety verification in the functional layer, the scheduling analysis in the architecture layer, and the cross-layer adaptation algorithms. Section 5 presents the experimental results and Section 6 concludes the paper.

2 RELATED WORK

Control system verification: The key idea of our work is to *safely* skip task jobs and make other changes at functional and architectural layers to address adaptation needs, with guarantees on control safety and task schedulability. This is related to a rich literature on fault-tolerant control systems. Those works consider a variety of fault types (e.g., sensing, actuation, or execution errors) and fault models [28], and address system safety [16, 18, 26] or stability [30, 43] under *passive* faults. In contrast, our work considers skipping control operations *proactively* at runtime to save resources for addressing adaptation needs (e.g., improving performance of other tasks). In [27], the authors design a verification approach to determine the safety of actively skipping control actuation based on the system state. The work however only focuses on the functional layer, and assumes that the control task runs on a single-task platform without any resource limitation or timing constraints. In [45], the author proposes an event-triggered control system where the lower bound of the inter-arrival time of control instances is formally derived to maintain control stability and be used for analyzing schedulability. Different from it, our work addresses the safety property of control and the system adaptation across layers at runtime.

Real-time systems with adjustable task periods: One important aspect of our adaptation framework is to allow some tasks adjust their periods at runtime. Such idea is related to several period-adjustable task models in the literature. In elastic scheduling [7], the period of each task is modelled to be adjustable within a given range and a coefficient of elasticity is proposed to model the cost of the task to adjust its period. Several elastic scheduling policies [8, 12, 17] are proposed to find an optimal period selection of the tasks, while ensuring the schedulability for either earliest-deadline-first (EDF) or fixed-priority preemptive (FPP) scheduling. In [8], the utilization of all tasks is jointly compressed according to elastic coefficients, and shared resources are also considered. In [17], the selected periods optimizes the total control performance. And the period selection problem of elastic scheduling is generalized into an optimization problem in [12]. Rhythmic tasks [29], or adaptive variable-rate (AVR) tasks [3–5] consider tasks whose periods can accelerate or decelerate within a certain rate. AVR schedulability analyses are proposed for FPP in [5, 29] and for EDF in [3]. In [4], the workloads of AVR tasks are optimized for different periods. Different from these works, our approach formally ensures reachability-based safety, and proactively explores job skipping based on the current system state. Moreover, with strong emphasis on both safety and schedulability under job skippings, our work develops a novel runtime adaptation framework. We assume that an adaptation goal of task periods is given to meet changing system needs. The works above on optimizing task periods may be leveraged to provide such adaptation goal.

Other cross-layer adaptation methods: There are also other approaches that adapt task execution with cross-layer consideration. For instance, in [15], the authors propose an online adaptation approach for hard real-time systems to temporarily increase control sampling frequency under disturbances while maintaining schedulability. Feedback schedulers [9–11] assign new sampling periods to control tasks during runtime to optimize the control performance under EDF scheduling. In [40], an approach is proposed to adaptively minimize tasks' usage of high quality-of-service resources while meeting control performance requirements. In [42], the authors propose

the simplex control architecture, where multiple controllers are being switched at runtime based on the system state and a safety controller is in charge of keeping the system safe. Our work employs a backup configuration that is inspired by such safety controller design. However, different from these adaptation approaches that are based on traditional hard timing constraints, our approach explores proactive task job skippings based on the dynamic system state for state-aware tasks and static weakly-hard constraints for other state-unaware tasks. This significantly increase system's adaptation capability, especially for resource-constrained systems.

Weakly-hard real-time systems: Our consideration of job skipping relates to the concept of weakly-hard timing constraints, where occasional deadline misses are allowed in a bounded manner. In [2], the concept of (m, K) weakly-hard constraint is introduced, where at most m deadline misses are allowed within any K consecutive executions of a task, to specify the degree of the task's tolerance in deadline misses. A similar concept is called (m, K)-firm real time systems [39], where only m out of every K consecutive executions are mandatory and required to meet their deadline. The weakly-hard paradigm provides more flexibility on the system design comparing to the traditional hard real-time constraint, while still allows the possibility of formally guaranteeing functional correctness that soft deadlines cannot provide. In the literature, a number of approaches have been proposed to address system schedulability under weakly-hard constraints [1, 2, 13, 14, 22, 23, 31, 38, 44, 50]. There are also methods for analyzing and optimizing control stability and performance with weakly-hard constraints [18, 19, 35, 37, 39, 48], showing that some control tasks could remain stable when execution jobs are skipped/missed in a bounded manner. In recent works [34, 36], the weakly-hard control systems are formulated into switched control systems for stability verification. A few methods have addressed the safety verification problem under weakly-hard constraints for linear [16, 18] and non-linear [24, 26] systems, considering system safety as a reachability problem (i.e, whether the system may reach unsafe states during execution). Finally, recent works in [32, 33] leverages the scheduling slack obtained from allowing deadline misses under weakly-hard constraints to improve system security and fault tolerance.

In these approaches for weakly-hard systems, deadline misses or skipped executions are mostly considered offline and not adapted to the changing system needs. Different from them, our work considers skipping jobs proactively at runtime based on the *current system state and adaptation needs*. Moreover, previous adaptation methods for real-time systems focus on control stability and rejection of sporadic disturbances, while recent works for weakly-hard systems separately consider the reachability-based safety verification and schedulability analysis, without any runtime adaptation. This work is the first to conduct safety-assured adaptation by ensuring both *reachability-based safety* at functional layer and schedulability at the architecture layer. Our approach includes novel and efficient techniques for state-aware safety verification of control tasks under job skippings, event-based analysis to assess system schedulability under job skippings and other changes, as well as runtime exploration of adaptation space. As stated above, we do leverage static weakly-hard constraints for those tasks that are state-unaware, and integrate their consideration with the state-aware tasks in a holistic framework.

3 SYSTEM MODEL

Our adaptation framework considers a system model that crosses the functional layer and architecture layer, as shown in Fig. 1. At the architecture layer, a set of independent, safety-critical tasks are periodically executed on a single-core processor using preemptive fixed-priority scheduling policy. At the functional layer, the functionalities of tasks (e.g., safety, performance) are considered under possible job skippings or period changes.

The entire set of tasks is denoted as $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$, which includes control tasks and other types of safety-critical tasks. For some control tasks, their safety property is considered based on

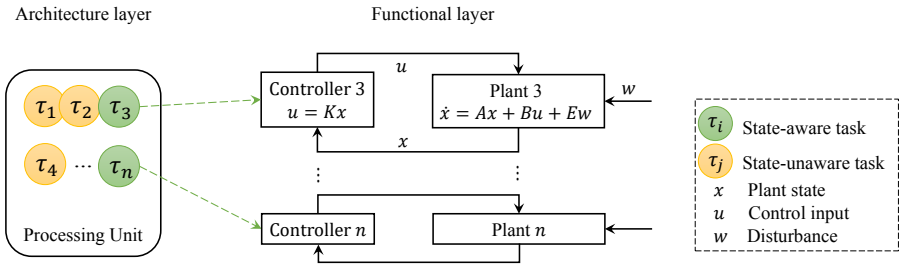


Fig. 1. System model across functional and architectural layers.

the runtime system state. Such tasks are denoted as *state-aware (SA) tasks*. The rest of the tasks, whose requirements for safety are pre-defined offline without considering their runtime state, are called *state-unaware (SU) tasks*. $\mathcal{T}_{SA} \subseteq \mathcal{T}$ and $\mathcal{T}_{SU} = \mathcal{T} \backslash \mathcal{T}_{SA}$ represent the sets of SA tasks and SU tasks, respectively.

For each task τ_i, its timing properties are represented by its period T_i, relative deadline $D_i \leq T_i$, and worst-case execution time C_i. Each task τ_i has a unique priority p_i. The notation $hp(\tau_i)$ (and $hep(\tau_i)$) denotes the set of tasks with priorities higher than (higher than or equal to) τ_i. Task τ_l represents the lowest priority task in $\mathcal{T}$. At a given time t, each task τ_i is associated with an offset o_i; and its j-th job, denoted as J_{ij}, has the release time $s_{ij} = t + o_i + jT_i$ and the absolute deadline $d_{ij} = s_{ij} + D_i$.

In this work, we consider skipping task jobs for addressing adaptation needs. The skipping choice of job J_{ij} is denoted as δ_{ij}, where $\delta_{ij} = 0$ if J_{ij} is skipped, and $\delta_{ij} = 1$ if J_{ij} is executed. In addition to skipping jobs, the periods of SA tasks are also assumed to be adaptable. Specifically, for each SA task $\tau_i \in \mathcal{T}_{SA}$, its period can be adapted within a given range $T_i \in [T_i^{\min}, T_i^{\max}]$ during runtime. For SU tasks, we assume that their periods are pre-defined and not adaptable. Our proposed adaptation framework makes the adaptation decision at runtime, while ensuring both architectural-level schedulability and functional-level safety.

Architectural-level Schedulability. To guarantee the functionality of each task, its non-skipped jobs must satisfy their deadline constraints. Thus, we define the system schedulability as follows:

Definition 3.1 (Schedulability). The system taskset $\mathcal{T}$ is schedulable if and only if, for each job that is not skipped, the worst-case response time (WCRT) r_{ij} is no late than its deadline:

$$\delta_{ij}(r_{ij} - s_{ij}) \leq D_i \quad \forall J_{ij}, \forall \tau_i \in \mathcal{T}$$

The WCRTs are evaluated during adaptation to guarantee schedulability, as explained in Section 4.2.

Functional-level Safety. In this work, the functional-level safety is considered for all tasks in the system. For SU tasks, we assume that their safety (or functional correctness) can be guaranteed as long as their timing properties are satisfied at the architecture layer. In this work, we allow task $\tau_i \in \mathcal{T}_{SU}$ to have a weakly-hard constraint (m_i, K_i) associated with its deadline constraint. Note that some tasks may not allow any skipped jobs, i.e., same as traditional hard real-time tasks. For the consistency of presentation, they will be associated with an $(m_i, K_i) = (0, 1)$ constraint.

The safety for SU tasks is defined as:

Definition 3.2 (SU task safety). For a schedulable system (i.e., all non-skipped jobs can meet their deadline), the SU task $\tau_i \in \mathcal{T}_{SU}$ is guaranteed safe (functionally correct) under its (m_i, K_i) constraint if for any K_i consecutive jobs of τ_i, at most m_i jobs are skipped.

For SU tasks, their (m_i, K_i) safety constraints can be derived and verified offline, using techniques from works such as [24, 26]. In this work, we assume that such safety constraints are given.

For an SA task $\tau_i \in \mathcal{T}_{SA}$, we guarantee its safety with a state-aware online verification process, which provides a much larger feasible adaptation space than offline analyses that consider the worst-case among all possible states. In this work, we assume that each SA task independently controls a physical plant under its own control law. The plant models we consider are linear time-invariant (LTI) systems with bounded external disturbance, which can be represented by the following discrete system dynamics:

$$x[t+1] = Ax[t] + Bu[t] + Ew[t], \tag{1}$$

where $x[t]$, $u[t]$ and $w[t]$ are, respectively, the system state, control input and external disturbance at time step t. The disturbance is assumed to be bounded $w[t] \in W$. A safe set X defines the set of safe system states. The control task computes a control input by a feedback control law $u = Kx$ in each sampling and control period. When the job execution is skipped, the controller applies a constant control $u = \kappa_0$ (usually $\kappa_0 = 0$). The safety of SA tasks is then defined as:

Definition 3.3 (SA task safety). For a schedulable system, the SA task $\tau_i \in \mathcal{T}_{SA}$ is safe if and only if its controlled system state always belongs to the safe set X (under possible job skippings), i.e., $x[t] \in X, \forall t$.

Note that our adaptation framework is not limited to linear controllers on LTI plant models. More discussion on extending our approach to general control laws $u = \kappa(x)$ and non-linear plant models can be found in Section 4.1.

4 OUR ADAPTATION FRAMEWORK

In this section, we present our cross-layer runtime adaptation framework that adaptively skips jobs and assigns task priorities to achieve given adaptation goals, while maintaining architectural-level schedulability and functional-level safety. An overview of our framework is shown in Fig. 2. We assume that the system initially runs under a backup configuration that guarantees schedulability and safety (it is called "backup" as the system can go back to it if the adaptation fails, as explained later). During runtime, an adaptation goal, i.e., a set of target sampling periods of SA tasks, can be given by an external party to our runtime adapter. The adapter then explores the configuration space to search for a feasible solution that makes the SA tasks executed at their target periods, while ensuring schedulability and safety within a finite time horizon Δt from the current time. If a solution is found, the system will run at this new configuration for the next Δt time interval; otherwise, it will stay at the backup configuration. Such exploration of the configuration space will be performed again after Δt, or when the system is under the backup configuration and there is no remaining workload on the core (i.e., between two busy windows). If the adaptation goal is changed within this Δt time interval, the system will first finish the planned execution for the Δt time interval, and then switch back to the backup configuration, before it tries to explore the configuration space for meeting the new adaptation goal. Such choice is to avoid the possibility of entering unsafe states when switching between two adaptation goals.

Adaptation goal. In this work, we assume that the target periods of SA tasks are given from an external source as the adaptation goal. Such goal can be updated during runtime and the adapter will attempt to schedule the SA tasks at their latest target periods if possible. The adaptation goal is decided based on the changing system needs and dynamic environment. For instance, some task periods may be reduced to improve their quality of service [7], control performance [17], or security level [33]. The objective of our framework is to find feasible configurations that can meet such goals while ensuring system safety and schedulability.

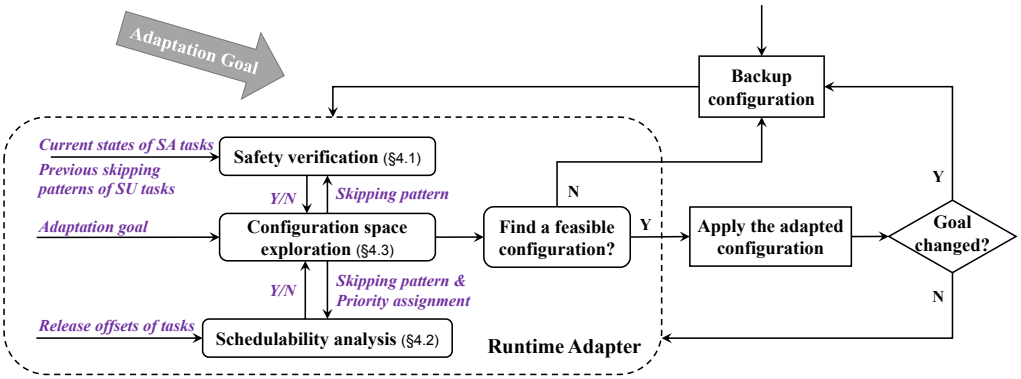


Fig. 2. Overview of our adaptation framework.

As a switching system, to ensure the system schedulability defined in Definition 3.1 and the safety of all tasks defined in Definition 3.2 and 3.3, the feasibility of the backup configuration should be considered together with the feasibility of the finite time adaptation solution, as explained below.

Backup configuration. The backup configuration is a static configuration that is designed offline. It should guarantee that the system is scheduleable without any job skipping and the SA tasks are always safe, under the assumptions that at the moment when the system switches to the backup configuration: 1) there is no remaining workload on the core, and 2) the state of each SA task satisfies a safety requirement of the backup configuration.

These assumptions will be guaranteed by the feasibility of the adaptation solution. The details of the safety requirement and guarantee will be introduced in Section 4.1. The schedulability guarantee of the backup configuration can be verified with hard real-time schedulability analysis (as there is no job skipping). The no workload assumption makes the critical-instant based analysis sound. The reason for not allowing job skipping in the backup configuration is that verifying weakly-hard system schedulability with unknown prior job skippings is challenging and too restrictive (in the worst case, m prior jobs are all skipped so the upcoming $(K - m)$ jobs are not allowed to skip).

Runtime adaptation. At runtime, when an adaptation goal is given, the adapter will run every time when the previous Δt time interval has passed (note that the length of Δt could be different each time) or the system is at the backup configuration and there is no workload on the core. When the adapter runs at t, it will search for a finite-time feasible configuration with a length of Δt that the configuration can remain feasible. This exploration is based on the current state of each SA task, the past job skipping pattern of each SU task, and the offset of each task. The configuration includes the priority assignment and which jobs to skip during $[t, t + \Delta t]$. A configuration is feasible within interval $[t, t + \Delta t]$ if:

- The skipping strategy of each SU task $\tau_i \in \mathcal{T}_{SU}$ satisfies its weakly-hard constraint (m_i, K_i) in $[t, t + \Delta t]$;[1]
- The skipping strategy of each SA task $\tau_i \in \mathcal{T}_{SA}$ ensures that its state x_i is safe during the interval $[t, t + \Delta t]$ and satisfies the safety requirement of the backup configuration at the end of the interval;
- There is no remaining workload at the end of the interval $[t, t + \Delta t]$;
- All non-skipped jobs in the interval $[t, t + \Delta t]$ must meet their deadline.

[1]The skipping pattern of $K_i - 1$ previous jobs before the busy window also needs to be considered.

Given the past job skipping pattern and the (m_i, K_i) constraints of SU task $\tau_i \in \mathcal{T}_{SU}$, the feasibility of the skipping choice in $[t, t+\Delta t]$ can be determined by checking every K_i consecutive jobs starting from the $(K_i - 1)$-th job before t.

For an SA task $\tau_i \in \mathcal{T}_{SA}$, the maximum number of jobs to be skipped in $[t, t+\Delta t]$ under the above safety constraint can be determined based on its current state $x_i(t)$. Details will be introduced in Section 4.1.

The requirement of no remaining workload is to ensure that, at $t + \Delta t$, if there is no feasible configuration for another time interval and the system needs to switch to the backup configuration, the assumption of no remaining workload of the backup configuration is satisfied. This requirement implies that the end of the time interval $t + \Delta t$ must not overlap with any busy window of the core, where the concept of busy window is formally defined as in Definition 4.1. Therefore, the adaptation time interval $[t, t + \Delta t]$ needs to be at least one busy window.

Definition 4.1 (System busy window). A system busy window $[t_0, t_1)$ is a time interval during which the core is always busy. In this busy window, at least one job is released but not finished yet, which satisfies:

$$\sum_{\forall J_{ij}: s_{ij} \in [t_0, t)} \delta_{ij} C_i > t - t_0, \quad \forall t \in [t_0, t_1) \tag{2}$$

Given the job skipping patterns, priority assignment and task offsets, we propose a finite-time schedulability analysis in Section 4.2 to determine the length of the upcoming busy window (i.e., length of the adaptation interval) and whether there are non-skipped jobs missing their deadline.

Our runtime adapter explores the configuration space of job skipping and priority assignment and verifies the schedulability and safety requirements to search for a feasible solution. In this work, we proposed an efficient heuristic for the configuration space exploration in Section 4.3.

4.1 Task Job Skipping with Safety Guarantees

In this section, we focus on deriving safety guarantee for SA tasks, and propose a method to formally verify the safety of an SA task with or without job skipping. Specifically, for an SA controller without job skipping (e.g., in backup configuration), its infinite-time safety is verified. And for an SA controller with job skipping, a finite-time safety is verified based on its current state. As this safety verification problem is at the task level for each SA task $\tau_i \in \mathcal{T}_{SA}$, we omit the subscript in the remaining of the section for simplicity.

Consider a general discrete system $x[t+1] = f(x[t], u[t], w[t])$ with a bounded disturbance $w \in W$ and the control law $u = \kappa(x)$. The safety of the system is defined on the safe set X. First, a *safe invariant set* X_I is defined as:

$$X_I = \{x \in X | \forall w \in W, f(x, \kappa(x), w) \in X_I\}. \tag{3}$$

If there is no job skipping and $x[t_0] \in X_I$, for the infinite time interval $t \in [t_0, +\infty)$, the system will always be safe, i.e., $x[t] \in X_I \subseteq X$. The infinite-time safety of the backup configuration can be verified based on the initial state and the safe invariant set.

To determine the set of states that is safe with job skipping, we leverage the idea of *backward reachable set* [27], and define two new concepts of *one-step hit set* $\mathcal{H}(\Omega)$ and *one-step skip set* $\mathcal{M}(\Omega)$ of set Ω:

$$\mathcal{H}(\Omega) = \{x \in \Omega | \forall w \in W, f(x, \kappa(x), w) \in \Omega\}. \tag{4}$$

$$\mathcal{M}(\Omega) = \{x \in \Omega | \forall w \in W, f(x, \kappa_0, w) \in \Omega\}. \tag{5}$$

For any state $x[t] \in \mathcal{H}(\Omega)$, if the job is not skipped, the state at the next step will be $x[t+1] \in \Omega$. Similarly, for any state $x[t] \in \mathcal{M}(\Omega)$, if the job is skipped, the state at the next step will be $x[t+1] \in \Omega$.

Building upon the safe invariant set X_I and the one-step hit/skip sets, we define an (m,K) safe set, denoted as $X^{(m,K)}$, which is the set that ensures safety for the upcoming K sampling periods if there is no more than m skipped jobs. $X^{(m,K)}$ can be derived recursively from X_I by:

$$X^{(m,K)} = \mathcal{M}(X^{(m-1,K-1)}) \bigcap \mathcal{H}(X^{(m,K-1)}), \tag{6}$$

where $X^{(0,K)} = X_I$ for $m = 0, K \geq 0$, and $X^{(m,m)} = \mathcal{M}(X^{(m-1,m-1)}) \cap \mathcal{H}(X^{(m-1,K-1)})$ for $K = m > 0$. Because of the invariant property of X_I, we have $\mathcal{H}(X_I) = X_I$. Thus, $X^{(1,1)} = \mathcal{M}(X_I)$, $X^{(1,K)} = \mathcal{H}(X^{(1,K-1)})$.

THEOREM 4.2. *If the current system state is in the (m,K) safe invariant set, i.e., $x[t] \in X^{(m,K)}$, and the control task skips at most m jobs in the next K ($K \geq m$) steps, the following conditions will hold:*

(1) *The state will be in the safe invariant set in the next K steps, i.e., $x[t+k] \in X_I, \forall k \in [0,K]$.*
(2) *The system will remain safe if no job skipping occurs after the next K steps.*

PROOF. Assume that the state at t is $x[t] \in X^{(m,K)}$. For any K-step sequence of job skipping pattern with at most m jobs being skipped, we denote $M(k)$ as the number of jobs skipped between $[t, t+k)$. We prove that $x[t+k] \in X^{(m-M(k),K-k)}$ for $\forall k \in [1,K]$ by induction:

For $k = 1$, if the first job at time t is skipped, $M(1) = 1$. Since $x[t] \in X^{(m,K)} \subseteq \mathcal{M}(X^{(m-1,K-1)})$, the next state will be

$$x[t+1] \in X^{(m-1,K-1)} = X^{(m-M(1),K-1)}.$$

If the first job is not skipped, $M(1) = 0$. Since $x[t] \in X^{(m,K)} \subseteq \mathcal{H}(X^{(m,K-1)})$, the next state is

$$x[t+1] \in X^{(m,K-1)} = X^{(m-M(1),K-1)}.$$

Thus, $x[t+k] \in X^{(m-M(k),K-k)}$ is true for the base case $k = 1$. For any $k \in [1, K-1]$, assume that $x[t+k] \in X^{(m-M(k),K-k)}$. If the job at time $t+k$ is skipped, $M(k+1) = M(k) + 1$. Since $x[t+k] \in X^{(m-M(k),K-k)} \subseteq \mathcal{M}(X^{(m-M(k)-1,K-k-1)})$, we have the following:

$$x[t+k+1] \in X^{(m-M(k)-1,K-k-1)} = X^{(m-M(k+1),K-(k+1))}.$$

If the job at time $t+k$ is not skipped, $M(k+1) = M(k)$. Since $x[t+k] \in X^{(m-M(k),K-k)} \subseteq \mathcal{H}(X^{(m-M(k),K-k-1)})$, the next state is

$$x[t+k+1] \in X^{(m-M(k),K-k-1)} = X^{(m-M(k+1),K-(k+1))}.$$

Thus, the inductive step is also proven as true.

As $M(k) \leq m$ for $\forall k \in [1,K]$, $X^{(m-M(k),K-k)} \subseteq X_I$. Thus, the first condition $x[t+k] \in X_I, \forall k \in [1,K]$ is proved.

For the second condition, as $x[t+K] \in X_I$ and no further job skipping occurs, for $\forall k > K$, the system will remain in the safe invariant set $x[t+k] \in X_I, \forall k > K$. □

Note that $X^{(1,1)}$ is identical to the notion of the strengthened safe set in [27], which ensures that one-step skipping will keep the system state inside X_I at the next step.

In this work, we consider the discrete linear time-invariant (LTI) system[2] modeled as in Equation (1), with a linear feedback controller $u = Kx$. We assume that the safe set X and disturbance range W are both polyhedrons. When a job is skipped, the controller will apply a constant control $u = \kappa_0$. For such a discrete LTI system, the one-step hit/skip sets $\mathcal{H}(\Omega)$ and $\mathcal{M}(\Omega)$ of a polyhedron Ω can be written as:

[2]Our approach is applicable to any control system if there exist methods for deriving its invariant set and backward reachable set and hence computing its (m,K) safe set. In this paper we use LTI system to demonstrate the derivation of (m,K) safe sets, but our framework is not limited to linear controllers: There are methods in the literature to compute invariant sets for nonlinear model-based controllers or even neural network controllers [25, 46].

$$\mathcal{H}(\Omega) = [(A+BK)^{+}(\Omega \ominus EW) \oplus \mathcal{N}(A+BK)] \bigcap \Omega \tag{7}$$

$$\mathcal{M}(\Omega) = [A^{+}((\Omega \ominus EW) - B\kappa_0) \oplus \mathcal{N}(A)] \bigcap \Omega \tag{8}$$

where $\ominus$ and $\oplus$ is the Minkowski difference and Minkowski sum operations. A^{+} is the pseudo inverse of the matrix A and $\mathcal{N}(A)$ is the null space of A.

As the safe set X of the discrete LTI system (1) is a polyhedron, the safe invariant set X_I will also be a polyhedron [6]. As introduced in [6], the largest invariant set inside the safe set X can be determined by

$$X_I = \lim_{k \to \infty} X_{-k}, \tag{9}$$

where X_{-k} is computed recursively by $X_{-k} = \mathcal{H}(X_{-k+1})$, and initially $X_{-1} = \mathcal{H}(X)$.

With the invariant set X_I in Equation (9) and the one-step hit/skip sets in Equations (7) and (8), the (m,K) safe set $X^{(m,K)}$ can be determined by Equation (6).

As X_I and $X^{(m,K)}$ does not depend on the system state $x[t]$, they can be evaluated offline. There could be multiple candidate periods for each SA task. The invariant set and (m,K) safe sets of each period need to be evaluated separately, as the discrete system dynamics and the control laws are different for each sampling period. Meanwhile, as the online adaptation can switch to the backup configuration, the controller should preserve the safety during the switching. Since the safety of the backup period can only be guaranteed if the state is in its safe invariant set X_I^{bk}, we enforce the state to be always within X_I^{bk} by setting the safe set X of all other periods to be X_I^{bk}.

At runtime, based on the current state $x[t]$, the adapter can determine the number of allowed skipping jobs in the next K jobs according to the (m,K) safe set with the maximum m that $x[t]$ is in. Specifically, we assume that the system state can only be observed at each job release time. Since there are multiple tasks in the system, the run time of the adapter cannot always align with the SA job release time. The current state $x[t]$ is actually the state of the last released job. Thus, $x[t] \in X^{(m,K)}$ actually means that in the upcoming $K-1$ jobs, $m-1$ jobs can be skipped if the last released job is skipped, and m jobs can be skipped if otherwise.

When switch from the backup period to a different period, the last released job will still have the backup period. Thus, a special type of (m,K) safe sets is needed, which is not recursively derived from X_I, but from $X_I \bigcap \mathcal{H}^{bk}(X_I^{bk})$, where $\mathcal{H}^{bk}(X_I^{bk})$ is the one-step hit set of the backup period.

4.2 Schedulability Analysis with Job Skipping

In this work, we developed an event-based analysis to determine both the length of the upcoming busy window, and whether the response time of all non-skipped jobs are within their deadline. Without loss of generality, the beginning of the busy window is denoted as $t = 0$, and the release time of the first job of task τ_i released no earlier than $t = 0$ is denoted as the offset $o_i \geq 0$. The skipping choices of the upcoming jobs of task τ_i is denoted by vector $\vec{\delta}_i = \{\delta_{ij} | s_{ij} \geq 0\}$.

At the moment when the adapter runs, it determines the offsets $\{o_i\}$, proposes the priority assignment $\{p_i\}$ and job skipping choices $\{\vec{\delta}_i\}$, and calls the event-based analysis to derive the schedulability for the upcoming busy window.[3] The event-based analysis simulates the scheduling process and determines the worst-case response time for each non-skipped job. The analysis terminates when either the first deadline is missed or the busy window ends. The details of the event-based analysis are shown in Algorithm 1, which takes the skipping choices $\vec{\delta}_i$, priority p_i and

[3] As the schedulability analysis routine is called whenever the adapter runs during runtime, unlike offline analysis approaches that need to ensure schedulability for infinite time horizon, the online event-based analysis only needs to determine the schedulability for a finite time interval, i.e., a busy window.

Algorithm 1 Schedulability analysis for a busy window

Input: $\{\vec{\delta}_i\}, \{p_i\}, \{o_i\}$
1: $Q_{future} \leftarrow futureJobQueue(\{o_i\}, \{T_i\}, \{\vec{\delta}_i\})$
2: $Q_{released} \leftarrow \emptyset$ // *Priority queue of released jobs*
3: $J_{ij} \leftarrow Q_{future}$.pop()
4: $t \leftarrow s_{ij}$
5: **repeat**
6: **while** $s_{ij} = t$ **do**
7: $wl_{ij} \leftarrow C_i$ // *Workload of* J_{ij}
8: $Q_{released}$.push(J_{ij})
9: $J_{ij} \leftarrow Q_{future}$.pop()
10: **if** $Q_{released}$ is not empty **then**
11: $J_{hp} \leftarrow Q_{released}$.pop()
12: **if** $t + wl_{hp} > d_{hp}$ **then**
13: **return** Unschedulable
14: **if** $t + wl_{hp} > s_{ij}$ **then**
15: $wl_{hp} \leftarrow wl_{hp} - (s_{ij} - t)$
16: $Q_{released}$.push(J_{hp})
17: $t \leftarrow s_{ij}$
18: **else**
19: $t \leftarrow t + wl_{hp}$
20: **until** $Q_{released}$ is empty
21: **return** Schedulable

release offset o_i for each task $\forall \tau_i \in \mathcal{T}$ as inputs, and determines the schedulability of the upcoming busy window.

During the event-based analysis, the releases and the response times (i.e., finishes) of non-skipped jobs are denoted as events. Between two consecutive events, only one job is executing. Two priority queues, Q_{future} and $Q_{released}$, keep tracking the unreleased jobs and the released jobs with remaining workloads. Q_{future} contains all unreleased jobs, which is initialized by function $futureJobQueue()$. Jobs in Q_{future} are sorted by their release time. $Q_{released}$ is sorted by the job priority, where the released job with the highest priority is always at the top. Based on the chronological order of events, the analysis simulates the fixed-priority scheduling procedure from the start of the busy window until either the end of the busy window (no released job has remaining workload) or a deadline is missed. At each event, the analysis will first check whether there are new jobs to be released and pick the highest-priority released job J_{hp}, determine whether J_{hp} is going to miss deadline and whether J_{hp} can finish before the next event, and update the remaining workload of J_{hp}. The correctness of such schedulability analysis process is proven below.

THEOREM 4.3. *Given the release offsets* $\{o_i\}$, *if for each task* τ_i, *any jobs released before* o_i *are finished before* $t = 0$ *and Algorithm 1 terminates its simulation at* $t = \Delta t$ *with a 'Schedulable' flag, the next busy window will be* $[0, \Delta t)$ *and all non-skipped jobs in* $[0, \Delta t)$ *will meet their deadlines.*

PROOF. A job is only possible to be scheduled for execution on the core if either it is just released or a higher priority job is finished. The simulation in Algorithm 1 checks and schedules the highest priority job every time when a new job is released or a scheduled job is finished. Thus, Algorithm 1 exactly mimics the fixed-priority preemptive scheduling for the tasks released at offsets $\{o_i\}$ and executed with their WCETs, which 1) derives the WCRT of each non-skipped job released in $[0, \Delta t)$, and 2) guarantees that there is no remaining workload (i.e., released-but-not-finished jobs) at $t = \Delta t$ if no scheduled job before Δt misses deadline. Note that the length of the busy window, Δt, is determined by repeating the simulation until $Q_{released}$ is empty, following Def. 4.1. □

4.3 Feasible Configuration Exploration

As shown in Fig. 2, every time when the adapter runs, it will explore the configuration space of task priority assignment and job skipping pattern to look for a feasible configuration that satisfies the adaptation goal. In this section, we will introduce an efficient heuristic we developed for the configuration space exploration.

Algorithm 2 Skipping Decision Making

Input: Priority assignment $\{p_i\}$
Input: Busy window beginning time t
1: $\tau_l \leftarrow lowestPriority(\mathcal{T}, \{p_i\})$
2: **for** $r = 1 : K_{\max}$ **do**
3: $\Delta t \leftarrow d_{lr} - t$
4: $\delta_{ij} \leftarrow 1, \forall J_{ij} \in \{J_{ij} | s_{ij} \in [t, t + \Delta t]\}$
5: $N_i \leftarrow numJobs(\tau_i, [t, t + \Delta t]), \forall \tau_i \in \mathcal{T}$
6: $n_i^{\max} \leftarrow nAllowSkip(\tau_i, [t, t + \Delta t]), \forall \tau_i \in \mathcal{T}$
7: **repeat**
8: $J_{lr}.isSkipped \leftarrow False$
9: **if** $isSchedulable(t, \{p_i\}, \{\vec{\delta}_i\})$ **then**
10: **return** $\{\vec{\delta}_i\}$
11: **else**
12: $J_{ik} \leftarrow$ the deadline missed job
13: **while** $r_{ik} > d_{ik}$ **do**
14: $J_{jp} \leftarrow toSkip\left(J_{ik}, \{\vec{\delta}_i, N_i, n_i^{\max} | \forall \tau_i\}\right)$
15: **if** J_{jp} not exist **then**
16: **return** No feasible skipping decision
17: **if** J_{jp} is J_{lr} **then**
18: $J_{lr}.isSkipped \leftarrow True$
19: **break**
20: **if** J_{jp} is J_{ik} **then**
21: reset all skipped jobs in this while loop
22: $\delta_{jp}(t) \leftarrow 0$
23: $r_{ik} \leftarrow updateResponseTime(J_{ik})$
24: **until** $J_{lr}.isSkipped = True$

Our exploration heuristic has a two-level hierarchical structure. At the lower level, a *Skipping Decision Making* algorithm is used to determine whether there exists a feasible job skipping strategy for a given task priority assignment. A feasible skipping strategy is guaranteed to satisfy all feasibility requirements (including safety and schedulability) for the upcoming busy window. At the upper level, a *Priority Decision Making* heuristic searches through different priority assignments to find one with feasible skipping strategy. If such priority assignment is found, the adapter will update the task priorities, assign job skipping, and set its next wake-up time as the end of the busy window. Otherwise, the system will switch to the backup configuration.

4.3.1 Skipping Decision Making. Algorithm 2 presents the skipping design making algorithm for a given priority assignment. To determine the feasible skipping pattern, we first assume that no job is going to skip, and check the schedulability of the corresponding busy window by the event-based analysis introduced in Section 4.2. If there is a deadline missed job J_{ik}, eligible jobs are going to be picked for skipping one by one, until either J_{ik} meets its deadline or there is no eligible jobs to select. When the deadline miss of J_{ik} is eliminated, the algorithm will continue to resolve the next deadline-miss job in the same way. This process will terminate until either there is no deadline miss in the busy window (i.e., found the skipping strategy) or there is no eligible job to select during the elimination of certain deadline miss (i.e., fail to find a feasible skipping pattern). During the process of eliminating the deadline miss of J_{ik}, if the selected job is J_{ik} itself (lines 20-21), all skipped jobs, which are previously selected to prevent the deadline miss of J_{ik}, will not need to be skipped.

From safety perspective, the functions $numJobs()$ and $nAllowSkip()$ determine the total number of jobs and maximum allowed number of skipped jobs in the busy window $[t, t + \Delta t]$. For SA task $\tau_i \in \mathcal{T}_{SA}$, given the latest available state $x(s_{i(-1)})$, the maximum allowed skipping is:

$$n_i^{\max} + (1 - \delta_{i(-1)}) = \max\{m \geq 0 | x(s_{i(-1)}) \in X_i^{(m,N_i+1)}\}.$$

Note that N_i and $n_i^{\max}$ are related to the busy window size Δt, which will impact the job skipping choices. As we assume $D_i \leq T_i$, the deadline of the first non-skipped job of the lowest priority task τ_l is an upper bound of the busy window size Δt. Since $X^{(m,K_1)} \subseteq X^{(m,K_2)}$ for $\forall K_1 \geq K_2$, $n_i^{\max}$ derived from the upper bound of the busy window is a lower bound for the allowed number of skipped jobs. In Algorithm 2, starting with $r = 1$, we assume that the r-th job is the first non-skipped job of τ_l. If this assumption is violated as this job is assigned to be skipped (lines 17-19), the first

Algorithm 3 Priority Decision Making

Input: Period assignment $\{T_i\}$
Initialize $\{p_i\}$
$\{\vec{\delta}_i(t)\} \leftarrow SkipDecision(\{p_i\})$
if Found feasible $\{\vec{\delta}_i(t)\}$ **then**
  return $\{p_i\}, \{\vec{\delta}_i\}$
repeat
  $\tau_{dm} \leftarrow$ the deadline-miss task
  $newAttempt = False$
  for $\tau_j \in hp(\tau_{dm})$ **do**
    $p_j \leftarrow demotePriority(\tau_j, \tau_{dm})$
    $\{\vec{\delta}_i(t)\} \leftarrow SkipDecision(\{p_i\})$
    if Found feasible $\{\vec{\delta}_i(t)\}$ **then**
      return $\{p_i\}, \{\vec{\delta}_i\}$
    $\tau_{dm}^{new} \leftarrow$ the deadline-miss task
    if $\tau_{dm}^{new} \neq \tau_j$ **then**
      $newAttempt = True$
      break
    reset τ_j's priority p_j
until $newAttempt == False$
return No feasible solution

non-skipped lowest-priority job will be assumed to be the next one. In that case, $n_i^{\max}$ of each task needs to be re-evaluated and the skipping decision of all jobs needs to be reset.

From scheduling perspective, the function $toSkip()$ picks an eligible non-skipped job to skip by leveraging the idea of level-i busy window [44].

Definition 4.4 (Level-i busy window). For task τ_i, a level-i busy window $[t_0, t_1)$ is a time interval during which the core is always occupied by the jobs of τ_i or higher priority tasks.

To eliminate the deadline miss of J_{ik}, only the jobs in the same level-i busy window will impact the response time of J_{ik}. Therefore, a non-skipped job is useful to skip only if it belongs to the level-i busy window that J_{ik} is in.

Specifically, the function $toSkip()$ selects a "skippable" and useful job from an eligible task. A job is "skippable" if it is currently non-skip; and for SU tasks, skipping it will not violate the (m, K) constraint. A job is useful if it is released before d_{ik} and is in the level-i busy window of the deadline-miss job J_{ik}. A task τ_j is eligible if its total number of skipped jobs in the busy window is still less than $n_j^{\max}$. If there are multiple "skippable" and useful jobs, the function will select the latest one. If multiple tasks are eligible with "skippable" and useful jobs, the function will choose the least impacted task that has the least skipping percentage in the busy window, where the skipping percentage is the ratio between the total number of skipped jobs and the maximum allowed one.[4]

4.3.2 Priority Decision Making. In Algorithm 2, we have introduced the algorithm to determine a skipping strategy with certain priority assignment. It is an NP-hard problem to optimize the priority assignment with respect to finding a feasible skipping strategy. The complexity of enumerating all possible priority assignments is exponential to the number of tasks, which is unacceptable for the adapter because of its online nature. Thus, we develop an efficient heuristic to search for a priority assignment with feasible skipping strategy in Algorithm 3.

In Algorithm 3, the heuristic first initializes the priority assignment. If the period assignment is the same as before, the initial priority is also set as the same as before, which led to feasible skipping strategy in the previous busy window. Otherwise, when some periods (and deadlines) are changed, the priority assignment will be initialized by the deadline monotonic (DM) policy, a reasonable starting point of the priority assignment exploration.

The heuristic then explores the priority assignments starting from the initial solution, until a feasible skipping strategy is found by the $SkipDecision()$ function, i.e., Algorithm 2. Every time

[4]This order may be adjusted based on specific adaptation objective. For instance, certain tasks can have a higher priority than others to skip jobs or not to skip jobs.

when a priority assignment leads to no feasible solution (i.e., a job's deadline miss cannot be eliminated, where the corresponding task is denoted as τ_{dm}), the heuristic attempts to pick a task with higher priority than τ_{dm} and demote its priority to be just lower than τ_{dm} by the function $demotePriority()$. Besides finding a feasible skipping strategy, the attempted priority demotion will also be accepted if the new deadline-miss task is not the demoted one. As long as there is an accepted priority demotion, the exploration will continue until a priority assignment leads to a feasible skipping strategy. If none of the higher priority tasks of the unschedulable task τ_{dm} is accepted for the priority demotion, the heuristic will terminate with no feasible solution and the system will switch to the backup configuration.

The efficiency of Algorithm 3 can be further improved by reducing the number of calls to the $SkipDecision()$ function.

To avoid duplicated checks, the previously-explored infeasible assignments can be stored. For each deadline-miss task τ_{dm}, we store the corresponding higher priority task set $hp(\tau_{dm})$ in the collection $\mathcal{U}(\tau_{dm})$ of τ_{dm}, i.e., $\mathcal{U}(\tau_{dm}) = \mathcal{U}(\tau_{dm}) \bigcup \{hp(\tau_{dm})\}$. Such higher priority task sets can be represented as bitsets for compact storage and fast check. When demoting the priority of τ_j, if $hp(\tau_j)$ is a superset of any set in $\mathcal{U}(\tau_j)$, i.e., $\exists hp' \in \mathcal{U}(\tau_j), hp(\tau_j) \supseteq hp'$, such demotion will cause a deadline miss of τ_j and can be directly rejected without calling $SkipDecision()$. Note that the order to select a higher priority tasks τ_j for priority demotion (line 8) should start from the lowest priority task in $hp(\tau_{dm})$ to the highest, since it is usually more likely to keep a task schedulable after demoting its priority if the priority change is smaller.

4.4 Backup Configuration and Infinite-Time Guarantees

The adaptation solution proposed by the runtime adaptation method in Section 4.3 can ensure system safety and schedulability for time interval $[t, t+\Delta t]$. To guarantee schedulability and safety in infinite-time horizon, a backup configuration is designed to ensure that the system is safe and schedulable when 1) running under the backup configuration and 2) switching between a normal configuration[5] and the backup configuration. Below, we introduce more of the backup configuration and prove the infinite-time schedulability and safety of our proposed adaptation framework.

First, a backup configuration includes: 1) the priority assignment for all tasks, 2) the backup period T^{bk} for each SA task, and 3) the safe invariant set X_I^{bk} for each SA task. A backup configuration is feasible if the following two conditions are satisfied.

- The backup configuration is schedulable under static hard real-time system scheduling for arbitrary task offsets. This means that the worst-case response time of each task under the worst-case (i.e., critical instant) analysis must be less than or equal to its deadline.
- For each SA task, the safe invariant set $X_I^{bk} \subseteq X$ is non-empty. For any other target period T, the corresponding safe invariant set X_I^T needs to be a subset of X_I^{bk}, i.e., $X_I^T \subseteq X_I^{bk}$.

When switching from a normal (adapted) configuration to the backup configuration at the end of an adaptation interval, there is no remaining workload. When switching from the backup configuration to a normal configuration, we also require that there is no remaining workload on the core. Moreover, for each SA task, as introduced in Section 4.1, the verification of the allowed number of skipped jobs needs to use the (m,K) safe sets that is derived from $X_I^T \bigcap \mathcal{H}^{bk}(X_I^{bk})$, where $\mathcal{H}^{bk}(X_I^{bk})$ is the one-step hit set of the backup period.

Based on these requirements for the backup configuration and rules for switching, our adaptation framework can ensure infinite-time schedulability and safety, as shown below.

[5]Normal here simply means any non-backup configuration, which in our framework could correspond to any configuration generated under an adaptation goal.

4.4.1 Schedulability. Based on the finite-time schedulability guarantee of each adaptation interval (according to Theorem 4.3) and the requirements for the backup configuration, we can derive the infinite-time schedulability as follows.

THEOREM 4.5. *Given a backup configuration that is schedulable under hard real-time constraints, the system is always schedulable in the proposed adaptation framework.*

PROOF. In the proposed adaptation framework, switching from a backup configuration to a normal configuration happens only when there is no remaining workload on the core, which ensures that no non-skipped job will miss its deadline in the upcoming adaptation interval, based on Theorem 4.3. Moreover, Theorem 4.3 ensures that there is no remaining workload at the end of the adaptation interval, i.e., the end of the busy window. Therefore, if it is followed by another adaptation interval, no job in the upcoming adaptation will miss deadline either. Similarly, when switching back to the backup configuration, no jobs will miss its deadline as there is no remaining workload from the previous interval and the backup configuration is schedulable under arbitrary task release offsets. In summary, no remaining job between configuration switching, schedulability of the backup configuration, and finite-time schedulability of normal configurations (Theorem 4.3) together ensure the infinite-time schedulability of our adaptation framework. □

4.4.2 SU task Safety.

THEOREM 4.6. *The SU tasks are always safe in the proposed adaptation framework according to Definition 3.2.*

PROOF. For each SU task, as the system is schedulable (Theorem 4.5), all non-skipped jobs can meet their deadline. During the backup configuration, there is no job skipping. During a normal configuration, the feasible job skipping decision (made in Algorithm 2) of the SU task ensures at most m skipped jobs in any K consecutive jobs. Since the skipping feasibility also considers $K-1$ jobs before the adaptation interval, the SU tasks will always satisfies their (m,K) constraints. □

4.4.3 SA task Safety. For an SA task, as introduced in Section 4.1, the safe invariant set X_I^T at the target period T is derived by treating X_I^{bk} as its safe set, which ensures $X_I^T \subseteq X_I^{bk}$. When switching from the backup period to the target period, the (m,K) safe set $X^{(m,K)}$ is derived from $X_I^T \bigcap \mathcal{H}^{bk}(X_I^{bk})$. When the task remains at the target period, $X^{(m,K)}$ is derived from X_I^T.

THEOREM 4.7. *For each SA task, if its initial state is in the safe invariant set of the backup period, i.e., $x_0 \in X_I^{bk}$, it will always be safe in the proposed adaptation framework according to Definition 3.3.*

PROOF. As the system starts from the backup period with no skipped jobs, and the initial state is in the invariant set X_I^{bk}, the state x will always remain within X_I^{bk} until switching to the target period. For the first adaptation interval switched from the backup period, where m jobs are skipped among K consecutive jobs, the system state of the last job before switching must be within $X^{(m,K)}$ derived from $X_I^T \bigcap \mathcal{H}^{bk}(X_I^{bk})$. Thus, the system state in the entire adaptation interval is within X_I^T. The skipping decisions for the following adaptation intervals are according to $X^{(m,K)}$ derived from X_I^T. As long as the task is under the target period, its state $x \in X_I^T$. Since $X_I^T \subseteq X_I^{bk}$, anytime when it is switched back to backup period, we have $x \in X_I^{bk}$. In summary, when the task is at its backup period, its state is always in X_I^{bk}; when switched to the target period, the state is in X_I^T; when it is at the target period, the state is always in X_I^T; and when switched back to the backup period, the state is still in X_I^{bk}. Therefore, the SA task state is always in $X_I^{bk} \subseteq X$, i.e., it is always safe. □

5 EXPERIMENTAL RESULTS

We evaluate our adaptation framework with a case study of a robot car. The robot car has a robotic arm and a rotatable camera, and runs in an environment with other robots and accomplishes the tasks distributed by a centralized server. It has three SA tasks, including an adaptive cruise control (ACC) task τ_{ACC} to avoid collision with other robots, an arm control task τ_{ARM} to keep it at certain position under external disturbance, and a camera control task τ_{CAM} to target the camera to a desired direction. It also has several SU tasks such as power management, error monitoring, trajectory planning, sensing data processing, message transmission, etc.

In the experiments, we evaluate the percentage of time when the system can meet the adaptation goal, the performance of the control functions, and the robustness of the adapter to maintain safety when the real external disturbance exceeds the bound assumed in verification. The results are compared with three baseline approaches: a no-skipping adaptation approach that does not consider job skipping, which is inspired by [15], and two *static* scheduling approaches that allow deadline misses [2, 44] or job skippings [39]. We first conduct experiments in a Python-based simulation environment we developed, where the system dynamics is precisely captured in our analysis model. We then conduct further experiments in the open-source robotic simulator Webots [49], where the non-linear system dynamics is approximated and bounded with errors in our linear analysis model, to demonstrate how our approach can be applied to more practical and challenging scenarios. Several ablation studies are also conducted to further demonstrate the benefits from considering state-aware job skipping, and to evaluate the computation overhead of our adapter.

5.1 Robot Car Model

5.1.1 System Configuration. There are 3 SA tasks and 7 SU tasks sharing a single-core processor on the robot car. The sampling periods of SU tasks span between 65ms to 1150ms. 4 out of the 7 SU tasks are weakly-hard tasks with pre-defined (m, K) constraints as $(2, 10)$, $(1, 20)$, $(3, 20)$, $(1, 5)$. The total utilization of all 7 SU tasks is 72.7%. For the SA tasks τ_{ACC}, τ_{ARM} and τ_{CAM}, the adaptive sampling period candidates are $T_{ACC} \in \{100, 150, 250, 400, 600, 1000, 1500\}$ms, $T_{ARM} \in \{100, 150, 200, 250\}$ms, and $T_{CAM} \in \{300, 400, 500, 600, 800, 1000\}$ms. Each SA task is assumed to have the same WCET for different sampling periods. For τ_{ACC}, the utilization is in the range of $[1.73\%, 26\%]$ without considering job skipping. For τ_{ARM} and τ_{CAM}, the utilization ranges are $[15.2\%, 38\%]$ and $[2.4\%, 8\%]$, respectively. In the derived backup configuration, all SA tasks are under their longest periods, the system utilization is 92.0%, and the priority assignment is based on the deadline monotonic policy.

5.1.2 Functional Models. We consider all three SA tasks as LTI systems with linear feedback control and bounded external disturbance, which can be captured by the continuous-time dynamics:

$$\dot{x} = A_c x + B_c u + E_c w. \tag{10}$$

The control task is assumed to update the control input at each deadline, which is equal to its sampling period, and use zero-order hold for the next period. Thus, the control input is applied with one period delay and the continuous-time dynamics can be discretized as $x[t+1] = A_d x[t] + B_d u[t-1] + E_d w[t]$, where $A_d = e^{A_c T}$, $B_d = (\int_0^T e^{A_c t} dt) B_c$, and $E_d = (\int_0^T e^{A_c t} dt) E_c$ for sampling period T. Then, it can be converted into the standard form discrete LTI as in Equation (1) by state augmentation: $x_a[t+1] = A x_a[t] + B u[t] + E w[t]$, where

$$x_a[t] = \begin{bmatrix} x[t] \\ u[t-1] \end{bmatrix}, \quad A = \begin{bmatrix} A_d & B_d \\ 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad E = \begin{bmatrix} E_d \\ 0 \end{bmatrix}.$$

In this case study, the linear feedback control law $u[t] = Kx_a[t]$ is derived by the discrete linear–quadratic regulator (LQR) for each sampling period. More details of the three SA tasks are explained below.

ACC Control. We consider that the ego robot car with position y_e and velocity v_e follows a reference car with position y_r and velocity v_r. The distance between two vehicles is $\Delta y = y_r - y_e$, which satisfies the ODE $\Delta\dot{y} = -v_e + v_r$. The velocity of the ego car satisfies the ODE $\dot{v}_e = a_e - k_e v_e$, where a_e is the acceleration of throttle and the second term is the resistance that is proportional to the speed. The goal of the ACC is to control the distance and the ego car velocity to the equilibrium $\Delta\bar{y} = 80$, $\bar{v} = 30$ and maintain them within a safe range: $\Delta y \in [40, 120]$, $v_e \in [20, 40]$, while the velocity of the reference car is bounded as $v_r \in [22, 38]$. The ACC system states can be formed as $x^{acc} = [\Delta y - \Delta\bar{y}, v_e - \bar{v}]^\top$, along with the control input $u^{acc} = a_e - k_e\bar{v}$, and the disturbance $w^{acc} = v_r - \bar{v}$.

Robot Arm Control. For the robot arm control, we leverage a 2 degree-of-freedom (DOF) robot arm. A linear system dynamics is linearized from the nonlinear model introduced in [21]. The ODE can be written as $M_0\ddot{\theta} + G(\theta_0) + \nabla_\theta G(\theta_0)(\theta - \theta_0) = \tau + w$, where $\theta = [\theta_1, \theta_2]^\top$ are the angles of two arms and $\theta_0 = [-\pi/6, 2\pi/3]$ is the equilibrium. $\tau = [\tau_1, \tau_2]^\top$ and $w = [w_1, w_2]^\top$ are respectively the torques of two motors and the disturbance on two arms. $M_0 = M(\theta_0)$ is the inertia matrix at the equilibrium. $G(\theta_0)$ and $\nabla_\theta G(\theta_0)$ are the gravity torques of two arms and their gradients. To form a LTI system, the system states is choosen as $x^{arm} = [\theta - \theta_0, \dot{\theta}]^\top$, and the control input as $u^{arm} = \tau - G(\theta_0)$.

Rotatable Camera Control. For the camera control task, we use the image tracking example described in [6] to model the dynamics of the control system. The camera has 1 DOF and the goal is to rotate towards the target direction. The system state includes the relative position and camera speed $x^{cam} = [y, v_c]^\top$, while the disturbance includes the target movement speed and a disturbance force caused by the moving of the robot: $w^{cam} = [v_t, f]^\top$. The control input u^{cam} is the torque controlling the camera.

The ODEs of these three controllers can all be in the form of the LTI continuous system dynamics as in Equation (10), with the transaction matrices:

$$A_c^{acc} = \begin{bmatrix} 0 & -1 \\ 0 & -k_e \end{bmatrix}, B_c^{acc} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, E_c^{acc} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad A_c^{arm} = \begin{bmatrix} 0 & I \\ -M_0^{-1}\nabla_\theta G(\theta_0) & 0 \end{bmatrix}, B_c^{arm} = E_c^{arm} = \begin{bmatrix} 0 \\ M_0^{-1} \end{bmatrix},$$

$$A_c^{cam} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, B_c^{cam} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, E_c^{cam} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}.$$

The safety verification of these control systems, including the computation of safe invariant set and (m, K) safe sets, follows the procedure introduced in Section 4.1.[6]

5.2 Comparison with a No-Skipping Adaptation Approach

To evaluate our adaptation framework, we first build a simulation environment with a combination of Matlab and Python code, and simulate the dynamics of the aforementioned control systems. We randomly generate the external disturbance for controllers and evaluate the system performance for different adaptation goals. For comparison, we also implement a No-Skipping adaptation approach that is similar to the cross-layer runtime adaptation approach in [15], which adapts the sampling

[6]Note that each control task has its own (m, K) safe sets. The choice of m and K values depends on the specific controller. In our experiments, we choose all non-empty (m, K) safe sets for $m \leq 20$ and $m \leq K \leq 40$. As the polyhedron safe set is stored as linear constraints, the storage requirement is at MB-level (14MB in total) and can be further reduced with simpler inner approximation for the polyhedron.

period of a control task and the priority assignment for all tasks during runtime. Unlike [15], which considers only one SA task in the system and focuses on its stability, we extend the approach to support multiple SA tasks in the no-skipping adaptation approach and ensure their reachability-based safety under external disturbance. This no-skipping adaptation approach provides a more fair comparison to our proposed adaptation framework, as the main difference between the two is whether job skipping is allowed.

5.2.1 Control Performance Improvement. We randomly generated 50 cases. In each case, the randomly generated external disturbance is always under the disturbance bound W, and the initial states are in the safe invariant set X_I. Each case is simulated for 1000 seconds. For each case, the adaptation goals of all possible combinations of the periods of SA tasks are evaluated. All the following results are the average among all 50 cases.

Fig. 3a shows the percentage of the time when the system is meeting adaptation goals (the system is in the backup configuration for the rest of the time). The X and Y axis are the periods set for ACC and ARM tasks in the adaptation goal, while the adaptation goal for the CAM task is fixed to 300ms. From the figure, we can see that our adaptation framework significantly outperforms the no-skipping adaptation approach in meeting the adaptation goals. In fact, the adaptation success rate is close to 100% in our framework when the desired ACC period is no less than 400ms.

The objective of choosing a smaller sampling period for a control task (motivation of the adaptation goal) is usually to improve the performance of the controller. For the controllers in this experiment, we consider the control performance in the form of $\int x(t)^\top Q x(t) dt$, which is the quadratic cost of their system states. As there are job skipping and the adaptation may not always succeed, the control performance variation for different adaptation goals is not monotonic with the sampling period. The average control performance among all three controllers are shown in Fig. 3b. Compared with the no-skipping adaptation approach, our adaptation framework can significantly improve the average control performance for most adaptation goals.

5.2.2 Safety Improvement under Stronger Disturbances. In previous experiments, we assume that the external disturbance is within the bound $w[t] \in W$ that is used for deriving the safe set in Section 4.1. In such case, both allowing skipping (our approach) and not allowing skipping (no-skipping adaptation) can always ensure the system safety, and the difference is only in their capabilities in meeting adaptation goals and improving control performance.

In practice there may be unexpected scenarios where the disturbance exceeds our assumption. Thus, we evaluate the safety of control tasks under such stronger disturbances to assess the robustness of our approach. In this experiment, we set the range of disturbance as 1.5 times of the assumed bound. Other settings are the same as in the previous experiments. Fig. 3c shows the percentage of time when the system is in unsafe state. A system is in unsafe state when at least one controller is unsafe, i.e., its state $x(t)$ is out of the safe set X.

From Fig. 3c, we have following observations:

- In the no-skipping adaptation, if the adaptation goal is chosen properly, the unsafe rate can be reduced from 31.6% (backup configuration) to 11.4% (with adaptation goal $T_{ACC} = 250$, $T_{ARM} = 250$, and $T_{CAM} = 1000$).
- In our approach, by allowing job skipping, the unsafe rate can be further reduced to 10.6% (with adaptation goal $T_{ACC} = 250$, $T_{ARM} = 200$, and $T_{CAM} = 500$).
- For most adaptation goals, by allowing skipping, the unsafe rate can be significantly reduced. For instance, when the adaptation goal is $T_{ACC} = 250$, $T_{ARM} = 200$, and $T_{CAM} = 300$, the unsafe rate can be reduced from 28.3% to 10.7% by allowing skipping.

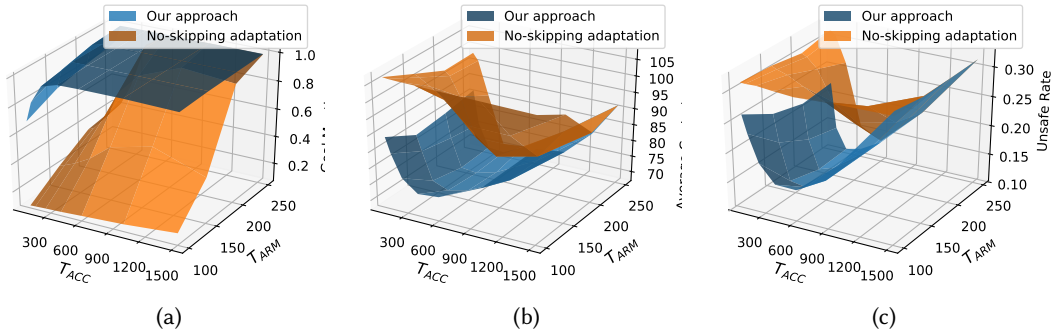


Fig. 3. Comparison between our approach and the no-skipping adaptation approach: (a) The percentage of the time the system can meet the adaptation goals using the two approaches. The adaptation goals are set as desired periods (in milliseconds in the figure) for SA tasks, which are changing for the ACC and ARM tasks, and fixed to 300ms for the CAM task. (b) The average control performance among all three controllers under each adaptation goal (the lower the better). (c) The percentage of the time when at least one controller is unsafe. The disturbance range in this experiment is 1.5x of the assumed range in deriving the safe set.

In summary, the comparison with the no-skipping adaptation approach demonstrates that by enabling safety-assured job skipping, our adaptation framework can *significantly improve the success rate to reach adaptation goals, the control performance, and the system robustness with respect to strong environment disturbances.*

5.3 Comparison with Static Scheduling Approaches that Allow Deadline Misses

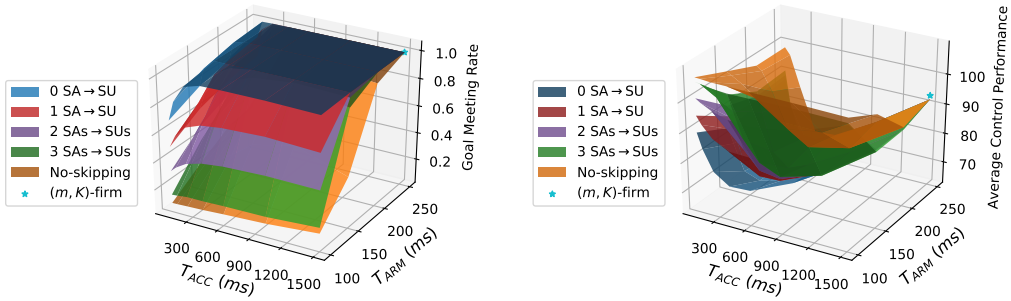
In this section, we compare our runtime adaptation framework with two static scheduling methods that allow deadline misses, the weakly-hard scheduling in [2, 44] and the (m, K)-firm scheduling in [39], to demonstrate the importance of developing runtime adaptation approaches. Weakly-hard scheduling allows tasks to miss their deadlines under the (m, K) constraints, but the missed job will continue its execution until finished. (m, K)-firm scheduling will evenly pick m fixed jobs in every K consecutive jobs to skip, while all other jobs need to meet deadline.

For both methods, there is no online verification of task safety. Thus, we leverage the state-of-the-art weakly-hard control system verification tool SAW [24] to evaluate the (m, K) constraints of all three SA tasks under their targets periods. Through the verification, both the ACC controller and ARM controller are only safe when no jobs are skipped. For CAM controller, when $T_{CAM} = 300$, the controller is safe under $(1, 4)$ constraints, and for other targets periods, no jobs skipping is allowed. As there is no runtime adaptation, SA tasks are assumed to execute under their target periods, and the priorities are assigned based on the rate-monotonic policy. Combining the weakly-hard constraints of SU tasks with the SAW-verified weakly-hard constraints of SA tasks, the schedulability of every adaptation goal is evaluated. For weakly-hard scheduling, the schedulability is evaluated based on the hyper-period analysis from [2]. For (m, K)-firm scheduling, it is evaluated based on the critical instance analysis from [39].

As presented in Table 1, schedulability analyses show that only several adaptation goals out of the 168 goals in total ($7 \times 4 \times 6$) are schedulable for either static approach. In contrast, both online adaptation approaches, ours and the no-skipping adaptation, have 100% goal meeting rates for the adaptation goals listed in Table 1. This demonstrates the importance of developing runtime adaptation approaches. As for control performance, the best performance of weakly-hard scheduling, (m, K)-firm scheduling, and our approach are respectively 82.9, 93.0, and 67.9, which occurs at

T_{ACC}/ms	600	1000	1000	1000	1000	1500	1500	1500	1500	1500	1500
T_{ARM}/ms	250	250	250	250	250	250	250	250	250	250	250
T_{CAM}/ms	1000	500	600	800	1000	300	400	500	600	800	1000
Weakly-hard [2, 44]	✓	✓	✓	✓	✓	-	-	✓	✓	✓	✓
(m,K)-firm [39]	-	-	-	-	✓	✓	✓	✓	✓	✓	✓

Table 1. Schedulable configurations under weakly-hard scheduling and under (m,K)-firm scheduling. The other 157 adaptation goals (not listed in this table) are not schedulable for either static scheduling approaches.



(a) (b)

Fig. 4. Results of our approach when 0, 1, 2 or 3 SA tasks are treated as SU tasks. (a) The percentage of the time the system can meet the adaptation goals. The desired periods are changing for the ACC and ARM tasks, and fixed to 300ms for the CAM task. (b) The average control performance among all three controllers under each adaptation goal (the lower the better). As reference, the feasible results for the baseline approaches are also shown. Note that (m, K) firm scheduling is only feasible at the period configuration $(1500, 250, 300)$, and no adaptation goals in the plot are feasible under weakly-hard scheduling.

configuration $(1000, 250, 500)$, $(1500, 250, 400)$ and $(400, 200, 300)$, respectively. We can see that by allowing runtime adaptation, our approach can significantly improve control performance (lower is better) by adapting to the period configuration that is not schedulable for the two static methods.

5.4 Further Analysis of the Benefit of Our State-Aware Approach

Methods that do not consider the runtime state of tasks address worst-case scenarios in their safety verification (e.g., evaluating (m, K) constraints with offline verification tools such as SAW [24]). In contrast, our adaptation approach explores the state-aware job skipping, where SA tasks can ensure their safety based on the analysis of runtime system state. Such consideration of runtime state usually provides more skipping choices and leads to a larger feasible adaptation space – while we cannot theoretically guarantee this is always the case (given that the system state is affected by the skipping choices), we conduct empirical analysis in the following experiments that further demonstrate the benefit of considering system state in safety-assured adaptation.

Specifically, our adaptation framework is evaluated in the same setting as previous experiments, except that some of the aforementioned SA control tasks are treated as SU tasks. That is, we compare the performance of our approach with the cases where zero, one, two, and three SA tasks are treated as SU tasks (the zero case is the same as our original approach). Note that when an SA task is treated as an SU task, we still try to adapt its period to the one in the adaptation goal. However, its job skipping is based on the offline (m, K) constraint verification, which is the same as the one derived for static weakly-hard scheduling as mentioned in Section 5.3.

Table 2. Webots simulation results for adaptation goal $T_{ACC} = 150ms$ and $T_{CAM} = 50ms$.

T_{ARM}		20ms	30ms	40ms	50ms	70ms	90ms
Utilization		153%	138%	130%	126%	121%	118%
Adaptation goal meeting rate	Our approach	8.06%	59.5%	90.3%	96.7%	98.9%	99.6%
	No-skipping adaptation	1.45%	2.22%	2.83%	5.58%	13.6%	15.5%
Average control performance	Our approach	65.2	62.8	59	58.1	58.8	61.6
	No-skipping adaptation	65.0	67.3	65.8	65.6	63.0	67.4

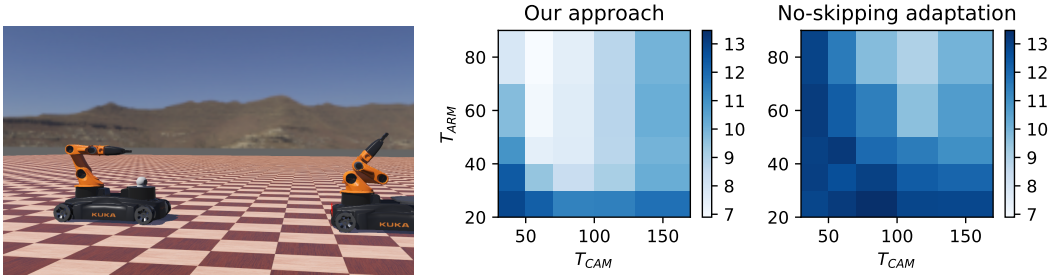
The comparison results are shown in Fig. 4. In Fig. 4a, we can observe that the adaptation goal meeting rates are notably reduced when one or more SA tasks are treated as SU tasks, i.e., without considering their runtime state. Such drop is more significant when the target periods are smaller (i.e., harder to achieve). This clearly shows the benefit of our state-aware adaptation approach. We also compare the average control performance in Fig. 4b, which also demonstrates that considering more SA tasks leads to better performance. The baseline methods are also plotted as reference, which behave significantly worse than our approach.

5.5 Application to More Practical Scenarios in Webots

We also carry out experiments in the Webots simulator to demonstrate the applicability of our approach in more practical scenarios. The robot is youBot developed by KUKA. The simulation scenario is shown in Fig 5a. The left youBot car (ego car) is adaptively controlled by three controllers: the wheel controller controls the wheels to follow the right youBot car; the arm controller controls the robotic arm to keep the first arm with 30 degree and keep the second arm horizontal; the camera controller controls the camera to the direction that has an interest object in its view. The ego car is required to maintain a distance with the front car (the car at right side hand in this case) between 0.25m to 1.75m, with the speed of the front car in the range between 0.25m/s to 0.75m/s. The camera direction is required to be within 30 degree of the target direction. And the two robot arms are required to be within 15 degree and 10 degree of the set points. The sampling periods of the wheel control, camera control and arm control are in the range of $[100, 600]$ milliseconds, $[30, 170]$ milliseconds, and $[20, 90]$ milliseconds, respectively. These controllers are assumed to run with seven other tasks on a resource-limited processor. The periods of these seven tasks are in the range of $[30, 620]$ milliseconds, where 4 of them allow bounded job skipping captured by their weakly-hard constraints.

The overall setting is similar to the previous experiments, except that the model parameters such as robot arm length and mess, the operation points, and the safety ranges are updated based on the Webots models. In addition, as Webots simulates the robots with its own physical engine, the robot system dynamics is much close to reality, and the inaccuracy of the LTI system models used for controllers is not negligible anymore. The inaccuracy includes inaccurate system state perception (i.e., the angular velocities of the robot arm and the rotatable camera are not accurate), and the abstraction of system model (for example, by linearizing the robot arm dynamics, the Coriolis and centrifugal forces are ignored). To ensure system safety, we model the inaccuracy as internal disturbance. The external disturbance for the wheel controller is from the front (right) car. The external disturbance for the camera controller is from the movement of its target. The external disturbance for the robot arm is from the acceleration /deceleration of the robot car. By combining both internal and external disturbance into the disturbance term in the system dynamics, the safety verification will provide a correct safety guarantee.

We compare our adaptation framework with the no-skipping adaptation approach in Webots, and the results similarly demonstrate the significant advantages of our approach. During the simulation,



(a) (b)

Fig. 5. (a) Webot simulation scenario: the ego youBot car (left) is following the other youBot car (right) while keeping its robot arm in a fixed position. A rotatable camera is mounted on the ego youBot and targets at interest object(s). (b) The camera control performance in the Webots simulation for different adaptation goals of T_{CAM} and T_{ARM}. T_{ACC} is set to $150ms$.

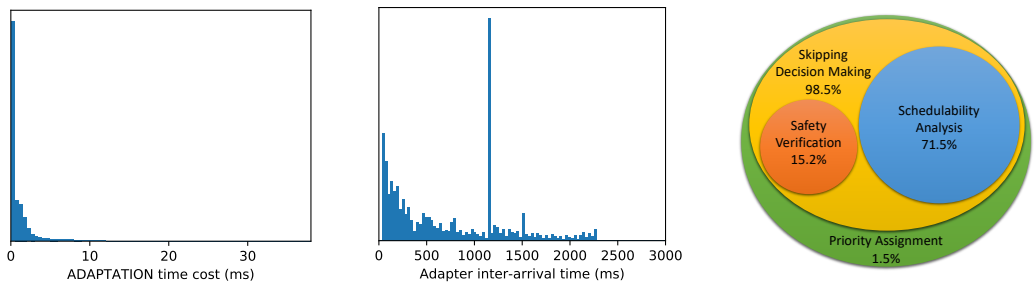
with our proposed adaption framework, all controllers always satisfy their safety requirements. The simulation results for some adaptation goals are summarized in Table 2. Specifically, the target period of the ACC and CAM are fixed to $T_{ACC} = 150ms$ and $T_{CAM} = 50ms$. The utilization is the processor utilization when the system successfully meets the adaptation goal. As shown in Table 2, our adaptation framework provides a much higher goal meeting rate and better average control performance (lower cost) than the no-skipping adaptation approach. In Fig. 5b, we further demonstrate the camera control performance for different adaptation goals. We can observe that our adaptation framework provides better control performance than the no-skipping adaptation approach for most of those adaptation goals. These comparisons in Webots again demonstrate the effectiveness of our approach in meeting adaptation goals via safety-assured job skipping, and show the applicability of our approach in more practical scenarios.

5.6 Overhead Analysis

The computational complexity of our adaptation approach depends on a number of system parameters. For instance, the complexity of schedulability analysis and skipping decision making are both related to the number of jobs in a busy window. The complexity of online job skipping safety verification is related to the control system state dimension, the number of (m, K)-safe sets, and the complexity of each (m, K)-safe set. In the worst case, the priority decision making may enumerate all possible priority assignments, which is $O(n!)$ with respect to the number of tasks. By recording the non-schedulable higher priority tasksets of each task, as introduced in Section 4.3.2, the worst-case complexity of priority decision making can still be $O(2^n)$. However, because our framework can handle cases where there is no feasible adaptation solution, the complexity of the priority exploration can be simply reduced by setting a timeout threshold.

Quantitatively, We evaluate the overhead of our approach for the experiments introduced before, which were run on a 3.6GHz Intel Xeon processor. The average execution time of the adapter task was 1.1 ms, while the average activation interval between two runs of the adapter was 820 ms, resulting in an average overhead of 0.14%. The execution time was 40 ms at the longest, under 16 ms for 99.9% of the cases, under 11 ms for 99%, and under 5.5 ms for 95%. The entire distribution of the execution time is shown in Fig 6a. The inter-arrival time between two consecutive adapter executions is in the range of $[30, 2970]$ ms, of which the entire distribution is shown in Fig. 6b.

As for the execution time of each component in our framework, one run of skipping decision making (Algorithm 2) takes 0.87 ms in average; one run of the schedulability analysis (Algorithm 1)



(a) (b) (c)

Fig. 6. Overload analysis of our approach: (a) The distribution of the execution time of each run of the adapter. (b) The distribution of the inter-arrival time between two runs of the adapter, i.e., the busy window size. (c) The percentage of the total execution time for each component of the adapter.

takes 0.18 ms in average; and one run of the online safety verification (i.e., line 6 in Algorithm 2) takes 0.13 ms in average. Note that the schedulability analysis and safety verification are part of the skipping decision making, and will be called different numbers of times during execution. Fig. 6c shows the percentage of the total execution time for each component of the adapter.

For priority assignment, even though its worst-case complexity is $O(2^n)$, in experiments the initial solution is feasible for 91.0% of the cases, and in the rest 9.0%, 43.5% of them can eventually find a feasible solution. This shows that our initial solution design is usually good and consequently the priority assignment is efficient in experiments (taking 1.5% of the total execution time).

To evaluate the average timing complexity with respect to the the number of tasks in the task set, besides the original 10-task system we considered (7 SUs and 3 SAs), we also evaluate our approach on three additional tasksets, which include 5 tasks (3 SUs and 2 SAs), 15 tasks (11 SUs and 4 SAs), and 20 tasks (15 SUs and 5 SAs), respectively. The average execution time of the adapter is 0.36 ms, 2.3 ms, and 6.7 ms, respectively. These results show that the average computation time $\bar{C}_{adp}$ is exponential to the number of tasks n, where $\bar{C}_{adp} \sim 1.2^n$.

6 CONCLUSION

In this work, we present a cross-layer runtime adaptation framework for real-time safety-critical systems. The framework explores proactive skipping of control task jobs and task priority assignment to meet adaptation goals, while guaranteeing reachability-based system safety and schedulability (when the external disturbance is within the assumed bounds). The framework includes three novel elements: a formal state-aware safety verification method that is based on computing invariant sets and backward reachable sets, an event-based schedulability analysis method that considers job skipping, and an efficient runtime adaptation algorithm that explores the adaptation space. Experimental results demonstrate that our approach can significantly outperform a no-skipping adaptation approach and two static weakly-hard scheduling approaches, in terms of meeting adaptation goals, improving control performance, and being robust under unexpected disturbances that exceed the assumed bounds.

ACKNOWLEDGMENTS

We gratefully acknowledge the support from NSF grants 1834701, 1834324, 1839511, 1724341, 1646497, 1943265 and ONR grant N00014-19-1-2496.

REFERENCES

[1] L. Ahrendts, S. Quinton, T. Boroske, and R. Ernst. 2018. Verifying Weakly-Hard Real-Time Properties of Traffic Streams in Switched Networks. In *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, Vol. 106. Dagstuhl, Germany.

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

[3] A. Biondi, G. Buttazzo, and S. Simoncelli. 2015. Feasibility Analysis of Engine Control Tasks under EDF Scheduling. In *2015 27th Euromicro Conference on Real-Time Systems*. 139–148. https://doi.org/10.1109/ECRTS.2015.20

[4] A. Biondi, M. Di Natale, and G. Buttazzo. 2016. Performance-Driven Design of Engine Control Tasks. In *2016 ACM/IEEE 7th International Conference on Cyber-Physical Systems (ICCPS)*. 1–10. https://doi.org/10.1109/ICCPS.2016.7479111

[5] A. Biondi, A. Melani, M. Marinoni, M. Di Natale, and G. Buttazzo. 2014. Exact Interference of Adaptive Variable-Rate Tasks under Fixed-Priority Scheduling. In *2014 26th Euromicro Conference on Real-Time Systems*. 165–174.

[6] F. Blanchini and S. Miani. 2015. Dynamic programming. In *Set-Theoretic Methods in Control*. Springer, Cham, 193–234.

[7] G.C. Buttazzo, G. Lipari, and L. Abeni. 1998. Elastic task model for adaptive rate control. In *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No.98CB36279)*. 286–295. https://doi.org/10.1109/REAL.1998.739754

[8] G.C. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni. 2002. Elastic scheduling for flexible workload management. *IEEE Trans. Comput.* 51, 3 (2002), 289–302. https://doi.org/10.1109/12.990127

[9] R. Castane, P. Marti, M. Velasco, A. Cervin, and D. Henriksson. 2006. Resource management for control tasks based on the transient dynamics of closed-loop systems. In *18th Euromicro Conference on Real-Time Systems (ECRTS'06)*.

[10] A. Cervin, J. Eker, B. Bernhardsson, and K.E. Årzén. 2002. Feedback–feedforward scheduling of control tasks. *Real-Time Systems* 23, 1 (2002), 25–53.

[11] A. Cervin, M. Velasco, P. Marti, and A. Camacho. 2011. Optimal Online Sampling Period Assignment: Theory and Experiments. *IEEE Transactions on Control Systems Technology* 19, 4 (2011), 902–910.

[12] T. Chantem, X.S. Hu, and M.D. Lemmon. 2006. Generalized Elastic Scheduling. In *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*. 236–245. https://doi.org/10.1109/RTSS.2006.24

[13] H. Choi, H. Kim, and Q. Zhu. 2019. Job-Class-Level Fixed Priority Scheduling of Weakly-Hard Real-Time Systems. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 241–253.

[14] H. Choi, H. Kim, and Q. Zhu. 2021. Toward Practical Weakly Hard Real-Time Systems: A Job-Class-Level Scheduling Approach. *IEEE Internet of Things Journal* 8, 8 (2021), 6692–6708. https://doi.org/10.1109/JIOT.2021.3058215

[15] X. Dai, W. Chang, S. Zhao, and A. Burns. 2019. A Dual-Mode Strategy for Performance-Maximisation and Resource-Efficient CPS Design. *ACM Trans. Embed. Comput. Syst.* 18, 5s, Article 85 (Oct. 2019), 20 pages.

[16] P. S. Duggirala and M. Viswanathan. 2015. Analyzing real time linear control systems using software verification. In *RTSS*. IEEE, 216–226.

[17] J. Eker, P. Hagander, and K.E. Årzén. 2000. A feedback scheduler for real-time controller tasks. *Control Engineering Practice* 8, 12 (2000), 1369–1378. https://doi.org/10.1016/S0967-0661(00)00086-1

[18] G. Frehse, A. Hamann, S. Quinton, and M. Woehrle. 2014. Formal Analysis of Timing Effects on Closed-Loop Properties of Control Software. In *2014 IEEE Real-Time Systems Symposium*. 53–62. https://doi.org/10.1109/RTSS.2014.28

[19] M.B. Gaid, D. Simon, and O. Sename. 2008. A Design Methodology for Weakly-Hard Real-Time Control. *IFAC Proceedings Volumes* 41, 2 (2008), 10258 – 10264. https://doi.org/10.3182/20080706-5-KR-1001.01736

[20] D. Goswami, R. Schneider, and S. Chakraborty. 2014. Relaxing Signal Delay Constraints in Distributed Embedded Controllers. *IEEE Transactions on Control Systems Technology* 22, 6 (2014), 2337–2345.

[21] E. Guechi, S. Bouzoualegh, L. Messikh, and S. Blažic. 2018. Model predictive control of a two-link robot arm. In *2018 International Conference on Advanced Systems and Electric Technologies (IC_ASET)*. 409–414.

[22] Z. A. H. Hammadeh, R. Ernst, S. Quinton, R. Henia, and L. Rioux. 2017. Bounding deadline misses in weakly-hard real-time systems with task dependencies. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*.

[23] Z. A. H. Hammadeh, S. Quinton, and R. Ernst. 2014. Extending Typical Worst-case Analysis Using Response-time Dependencies to Bound Deadline Misses. In *ACM International Conference on Embedded Software (EMSOFT)*.

[24] C. Huang, K.C. Chang, C.W. Lin, and Q. Zhu. 2020. SAW: A Tool for Safety Analysis of Weakly-Hard Systems. In *Computer Aided Verification*, S. K. Lahiri and C. Wang (Eds.). Springer International Publishing, Cham, 543–555.

[25] C. Huang, J. Fan, W. Li, X. Chen, and Q. Zhu. 2019. ReachNN: Reachability Analysis of Neural-Network Controlled Systems. *ACM Trans. Embed. Comput. Syst.* 18, 5s, Article 106 (Oct. 2019), 22 pages. https://doi.org/10.1145/3358228

[26] C. Huang, W. Li, and Q. Zhu. 2019. Formal Verification of Weakly-Hard Systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control* (Montreal, Quebec, Canada) *(HSCC '19)*. 197–207.

[27] C. Huang, S. Xu, Z. Wang, S. Lan, W. Li, and Q. Zhu. 2020. Opportunistic Intermittent Control with Safety Guarantees for Autonomous Systems. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 1–6.

[28] J. Jiang and X. Yu. 2012. Fault-tolerant control systems: A comparative study between active and passive approaches. *Annual Reviews in Control* 36, 1 (2012).

[29] J. Kim, K. Lakshmanan, and R. Rajkumar. 2012. Rhythmic Tasks: A New Task Model with Continually Varying Periods for Cyber-Physical Systems. In *International Conference on Cyber-Physical Systems*.

[30] J. Lan and R. J. Patton. 2016. A new strategy for integration of fault estimation within fault-tolerant control. *Automatica* 69 (2016).
[31] J. Li, Y. Song, and F. Simonot-Lion. 2006. Providing Real-Time Applications With Graceful Degradation of QoS and Fault Tolerance According to (m, k)-Firm Model. *IEEE Trans. on Industrial Informatics* 2, 2 (2006), 112–119.
[32] H. Liang, Z. Wang, R. Jiao, and Q. Zhu. 2020. Leveraging Weakly-hard Constraints for Improving System Fault Tolerance with Functional and Timing Guarantees. In *International Conference On Computer Aided Design (ICCAD)*.
[33] H. Liang, Z. Wang, D. Roy, S. Dey, S. Chakraborty, and Q. Zhu. 2019. Security-Driven Codesign with Weakly-Hard Constraints for Real-Time Embedded Systems. In *IEEE International Conference on Computer Design (ICCD)*. 217–226.
[34] M. Maggio, A. Hamann, E. Mayer-John, and D. Ziegenbein. 2020. Control-System Stability Under Consecutive Deadline Misses Constraints. In *Euromicro Conference on Real-Time Systems (ECRTS)*.
[35] P. Marti, A. Camacho, M. Velasco, and M. E. M. Ben Gaid. 2010. Runtime Allocation of Optional Control Jobs to a Set of CAN-Based Networked Control Systems. *IEEE Transactions on Industrial Informatics* 6, 4 (Nov 2010), 503–520.
[36] P. Pazzaglia, A. Hamann, D. Ziegenbein, and M. Maggio. 2021. Adaptive design of real-time control systems subject to sporadic overruns. In *Design, Automation & Test in Europe Conference Exhibition (DATE)*, Vol. 2021.
[37] P. Pazzaglia, L. Pannocchi, A. Biondi, and M. Di Natale. 2018. Beyond the Weakly Hard Model: Measuring the Performance Cost of Deadline Misses. In *Euromicro Conference on Real-Time Systems (ECRTS)*.
[38] S. Quinton, M. Hanke, and R. Ernst. 2012. Formal Analysis of Sporadic Overload in Real-time Systems. In *Design, Automation and Test in Europe (DATE)*.
[39] P. Ramanathan. 1999. Overload management in real-time control applications using (m, k)-firm guarantee. *IEEE Transactions on Parallel and Distributed Systems* 10, 6 (Jun 1999), 549–559. https://doi.org/10.1109/71.774906
[40] D. Roy, W. Chang, S. K. Mitter, and S. Chakraborty. 2019. Tighter Dimensioning of Heterogeneous Multi-Resource Autonomous CPS with Control Performance Guarantees. In *ACM/IEEE Design Automation Conference (DAC)*. 1–6.
[41] D. Roy, S. Ghosh, Q. Zhu, M. Caccamo, and S. Chakraborty. 2020. GoodSpread: Criticality-Aware Static Scheduling of CPS with Multi-QoS Resources. In *2020 IEEE Real-Time Systems Symposium (RTSS)*. 178–190.
[42] D. Seto, B. Krogh, L. Sha, and A. Chutinan. 1998. The Simplex architecture for safe online control system upgrades. In *American Control Conference (ACC)*. 3504–3508 vol.6.
[43] Y. Song, Y. Wang, and C. Wen. 2016. Adaptive fault-tolerant PI tracking control with guaranteed transient and steady-state performance. *IEEE Transactions on automatic control* 62, 1 (2016).
[44] Y. Sun and M. Di Natale. 2017. Weakly Hard Schedulability Analysis for Fixed Priority Scheduling of Periodic Real-Time Tasks. *ACM Trans. Embed. Comput. Syst.* 16, 5s, Article 171 (Sept. 2017), 19 pages.
[45] P. Tabuada. 2007. Event-Triggered Real-Time Scheduling of Stabilizing Control Tasks. *IEEE Trans. Automat. Control* 52, 9 (2007), 1680–1685. https://doi.org/10.1109/TAC.2007.904277
[46] Y. Wang, C. Huang, Z. Wang, S. Xu, Z. Wang, and Q. Zhu. 2021. Cocktail: Learn a Better Neural Network Controller from Multiple Experts via Adaptive Mixing and Robust Distillation. *Design Automation Conference (DAC'21)* (2021).
[47] Y. Wang, C. Huang, and Q. Zhu. 2020. Energy-Efficient Control Adaptation with Safety Guarantees for Learning-Enabled Cyber-Physical Systems. In *International Conference on Computer-Aided Design (ICCAD '20)*. Article 22, 9 pages.
[48] Z. Wang, H. Liang, C. Huang, and Q. Zhu. 2020. Cross-Layer Design of Automotive Systems. *IEEE Design & Test* (2020).
[49] Webots. [n.d.]. Commercial Mobile Robot Simulation Software. http://www.cyberbotics.com
[50] W. Xu, Z. A. H. Hammadeh, A. Kröller, R. Ernst, and S. Quinton. 2015. Improved Deadline Miss Models for Real-Time Systems Using Typical Worst-Case Analysis. In *2015 27th Euromicro Conference on Real-Time Systems*. 247–256.
[51] Q. Zhu, C. Huang, R. Jiao, et al. 2021. Safety-Assured Design and Adaptation of Learning-Enabled Autonomous Systems. In *Asia and South Pacific Design Automation Conference (ASPDAC '21)*. 753–760. https://doi.org/10.1145/3394885.3431623
[52] Q. Zhu, W. Li, H. Kim, et al. 2020. Know the Unknowns: Addressing Disturbances and Uncertainties in Autonomous Systems. In *International Conference on Computer-Aided Design (ICCAD '20)*. https://doi.org/10.1145/3400302.3415768