Closing the Gap between Stability and Schedulability: A New Task Model for Cyber-Physical Systems

Hoon Sung Chwa and Kang G. Shin
Electrical Engineering and Computer Science
The University of Michigan, Ann Arbor, Michigan, U.S.A.
{hchwa,kgshin}@umich.edu

Jinkyu Lee
Department of Computer Science and Engineering
Sungkyunkwan University (SKKU), Republic of Korea
jinkyu.lee@skku.edu

***Abstract*—A cyber-physical system (CPS) usually contains multiple control loops, each responsible for controlling different physical subprocesses, that run simultaneously upon a shared platform. The foremost design goal for CPSes is to guarantee system stability and control quality with limited cyber resources. We show, via an in-depth case study, that two inter-related design parameters — *sampling period* and *consecutive control update misses* — play a key role in determining stability and control performance. However, most CPS designs, such as control–schedule co-design and fault-tolerant scheduling, focus on either sampling period or control update misses alone, but not both. To remedy this problem, we propose a new CPS task model that captures both system stability and control performance in terms of sampling period and maximum allowable number of consecutive control update misses. To demonstrate the utility and power of this model, we develop two new scheduling mechanisms, *offline parameter assignment* and *online state-aware scheduling*. The former determines the sampling period and the maximum allowable number of consecutive job deadline misses for each task while preserving system stability. The latter then generates a schedule by exploiting the state of each physical subprocess to manage job deadline misses so as to improve the overall system performance without compromising system stability. Our in-depth evaluation results demonstrate that the proposed task model and the corresponding scheduling algorithm not only enable the efficient use of computing resource, but also significantly improve control performance without compromising system stability.**

I. Introduction

Cyber-physical systems (CPSes) are engineered systems consisting of physical and cyber/computational components, where the operation of the former is controlled by the latter. The main design challenge for CPSes is to provide tight integration of physical and cyber components for efficient resource utilization while guaranteeing system stability.

Cyber or physical components are abstracted with well-known models, based on which numerous design and analysis techniques have been proposed. For example, based on the well-known periodic hard real-time task model by Liu and Layland [1], many real-time scheduling algorithms and schedulability analyses have been developed to make deadline guarantees with limited resources. Similarly, based on the well-known state-space model [2], numerous control designs and stability analyses have been proposed to make stability guarantees. However, there exists a significant gap between guaranteeing physical system stability and guaranteeing task deadlines. For example, some control update misses may not compromise the stability of a physical component, and hence the associated components need not be treated as hard real-time tasks. They will otherwise result in poor resource utilization and control performance during runtime; not all the resources that are needed to guarantee the execution of an unncessarily "pessimistic" task model are required by its physical counterpart, and a large fraction of the assigned cyber resources remain unused during runtime.

We propose to close this gap between the physical and cyber parts of CPSes by (i) identifying key parameters that link the control and scheduling domains, (ii) understanding the relationship between the key parameters and its impact on control and scheduling performance, and (iii) exploiting the relationship for system design. As an initial step to solve this problem, we find, from a case study, that *sampling period* and *consecutive control update misses* are tightly coupled and play key roles in system stability and control performance for a physical plant. Specifically, each physical plant has its own limit on the number of consecutive update misses before losing stability, and the limit varies with sampling period. In general, a shorter sampling period allows the plant to tolerate more consecutive update misses. Moreover, different combinations of these two parameters yield different quality of control. We, therefore, identify a trade-off between tuning sampling period and managing update misses in terms of the amount of cyber resources required for system stability and control performance. Thus, the overall control performance can be improved significantly while using a minimal amount of cyber resources for system stability by making such a trade-off in task parameter assignment and scheduling.

Although many researchers have considered either sampling period or control update misses, the relationship between the two has not yet been fully explored and exploited to improve control performance under resource constraints. A majority of existing studies [3–12] in control–schedule co-design have focused on how to determine task periods and deadlines so as to maximize control performance. However, they all assume a strict/hard guarantee of every deadline and overlook the inherent robustness of physical systems to control update misses, yielding pessimistic results or resource over-provisioning. By relaxing such hard real-time guarantees, a few researchers proposed new task models [13–17] that capture the tolerance of some deadline misses and the corresponding scheduling approaches [18–23]. However, they have not yet exploited the relationship between the task period and the tolerable number of consecutive deadline misses, and instead treated the task period as a fixed and unchangeable parameter although the periods of typical control tasks can be adjusted

without losing system stability.

To fill this gap, we investigate the benefits achieved by considering both task period and the number of consecutive control update misses together, and taking their interplay into account for management of cyber resources. Our goal is to optimize the overall control performance while guaranteeing system stability with limited resources. Specifically, we propose a new CPS task model that can capture possible combinations of sampling period and the number of consecutive control update misses without losing stability, and estimate the control performance for each configuration of the two parameters. Based on the new CPS task model, we develop new scheduling mechanisms: *optimal parameter assignment* and *online state-aware scheduling*. The former finds the optimal combination of the two parameters for each task in a given task set such that the worst-case control performance is maximized without losing stability. For a given parameter assignment, the online state-aware scheduling algorithm takes into account the current state of each physical plant and adaptively manages job deadline misses for each task in order to maximize the overall control performance without compromising stability. Our in-depth evaluation results show that the proposed approach outperforms the existing period assignment/adjustment significantly in terms of schedulability and control performance. Our approach is shown to make 39% and 54% more control task sets schedulable without losing stability and also improve control performance by up to 71% and 197%, respectively, over the existing static period assignment and dynamic period adjustment.

This paper makes the following main contributions:

- An in-depth case study that reveals the relationship between task period and the number of consecutive control update misses, and its connection to the stability and control performance of a physical component (Section III);
- Development of a new task model that captures the stability requirement and control performance in terms of sampling period and the maximum allowable number of consecutive control update misses (Section IV);
- Development of offline task parameter assignment that significantly improves the overall system performance without losing stability (Section V);
- Development of online state-aware scheduling that enables dynamic management of control update misses for each task by considering the actual states of the physical components (Section VI); and
- Demonstration of the effectiveness of the proposed scheduling approaches in terms of resource efficiency and control performance (Section VII).

II. Related Work

Control–schedule co-design. Considerable efforts have been made on control–scheduling co-design. The seminal paper by Seto *et al.* [3] formulated this co-design as an optimal task-period assignment problem so as to maximize control performance while guaranteeing tasks' schedulability. Subsequent studies extended the period-assignment problem by taking control delays [4, 5] and jitters [6] into account and by determining not only task periods but also deadlines [7]. Aminifar *et al.* [8] considered the effect of delay and jitter on the stability of physical plants and developed period assignment and scheduling schemes to ensure stability. Palopoli *et al.* [9] proposed the notion of stability radius and developed an algorithm to determine periods and feedback gains based on the stability radius. Khatib *et al.* [24] considered the problem of synthesizing a set of timing contracts that guarantee both stability and schedulability. In addition to these offline resource-allocation schemes, several researchers [10–12] presented online period-adjustment schemes that dynamically change the periods of control tasks based on their physical plant states. Although these control–schedule co-design schemes considered the effects of scheduling parameters on control performance and system stability, they all focused on the stringent requirement of meeting *all* deadlines by following the traditional notion of schedulability.

Fault-tolerant scheduling. A few researchers considered relaxation of the strict hard real-time guarantees. Palopoli *et al.* [25] showed via a case study that considering control tasks as soft real-time threads, as opposed to enforcing hard real-time constraints, can improve control performance. Others [18–21] proposed fault-tolerant controller designs that allow occasional control signal drops for distributed control systems. Majumdar *et al.* [22] derived bounds of the fraction of control signal drops to guarantee plant stability, and this work was later extended in [26] to account for network packet losses and develop a dynamic scheduler. Yoshimoto and Ushio [23] introduced a job skipping mechanism to minimize control performance while guaranteeing stability under overloaded situations. However, all these fault-tolerant scheduling schemes are based on the assumption that scheduling parameters including task periods are fixed or given.

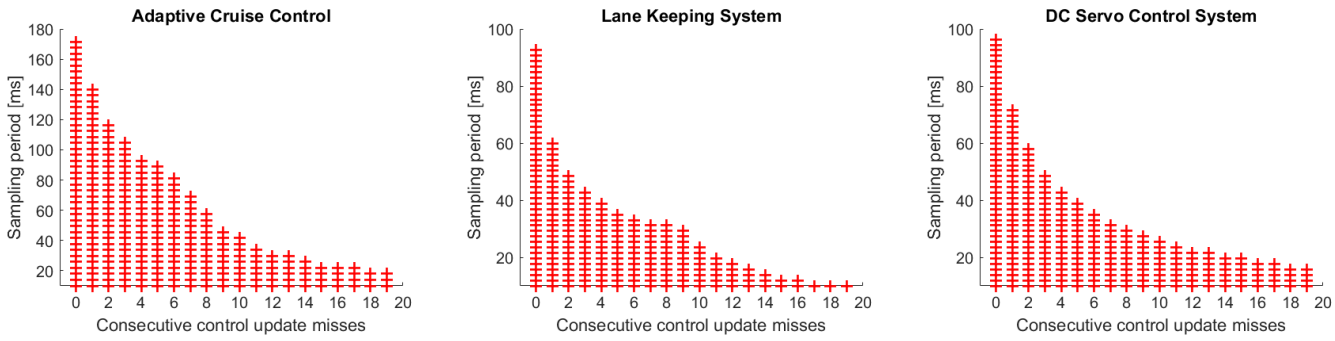
Our proposal. Although researchers have shown that either adjusting task periods or allowing some deadline misses can improve control performance without compromising stability, the *interaction* between task periods and deadline misses — despite its importance in CPSes — has not been considered before. To fill this gap, we first conduct an in-depth case study to examine the relationship between task periods and deadline misses in terms of stability and control performance. We then propose a new general task model capable of expressing the relation, and develop a new scheduling mechanism that (i) jointly determines task periods and manages deadline misses to improve control performance without losing stability and (ii) utilizes limited cyber resources efficiently.

III. Motivation and Goal

We first present a case study to demonstrate the relation between period and control update misses in terms of stability and control performance, and then provide our motivation and goal for our new task model and scheduling schemes thereof.

A. The System

We consider a typical CPS composed of multiple independent physical plants controlled by cyber components (or feedback controllers). Their sampling periods may be different from each other. Each controller is implemented as a periodic



(a) Adaptive cruise control system (b) Lane keeping system (c) DC-servo control system

Fig. 1. Stability region of each physical plant as a function of sampling period and the number of maximum consecutive control update misses.

control task whose period equals its sampling period and runs upon a shared uniprocessor platform. Basically, each control task samples the state of its plant at every sampling instant, computes a control input, and applies it to the plant; this process is repeated periodically. Scheduling control tasks on the platform will update the control input with some delays and jitters. If the control input is computed within a sampling period $[t_k,\ t_{k+1})$, the update is made at the next sampling instant t_{k+1}. On the other hand, if the computation is not completed within a sampling period, the uncompleted computation will be abandoned, thus leading to a *control update miss*. In case of a control update miss, the most recent past update will be used instead, as commonly done in existing control designs [18–23]. In case of one update miss, the control input is held constant between two consecutive sampling periods using a zero-order hold. We assume that the algorithm used in each control task is designed to account for the time delay between control updates. We also assume that the relative deadline of a task equals its period and all tasks are preemptive.

We consider an autonomous vehicle system — a prototypical CPS — consisting of three types of physical plants, including adaptive cruise control [27], lane keeping control [28], and DC-servo control [29] loops. The adaptive cruise control loop periodically adjusts the vehicle speed to maintain a safety distance from the vehicle in front. The lane keeping control loop affects lateral position and steering angle of the vehicle. The DC-servo control loop modifies the acceleration of a rotating servo motor. The details of these plants' dynamics and design parameters are provided in Appendix B of the supplement file [30].

B. Motivation

Via this case study, we investigate the effects of task period and control update misses on a physical plant. In particular, we gain the following physical intuitions that motivate the need for a new task model and its applications.

M1. The stability region of a physical plant is determined with respect to its sampling period and the number of *consecutive* control update misses, and there exists a relationship between these two parameters. In particular, as the sampling period becomes shorter, the plant can tolerate more consecutive control update misses. This relationship can be exploited to achieve stability with much less resources by assigning a short period and then allowing more deadlines misses.

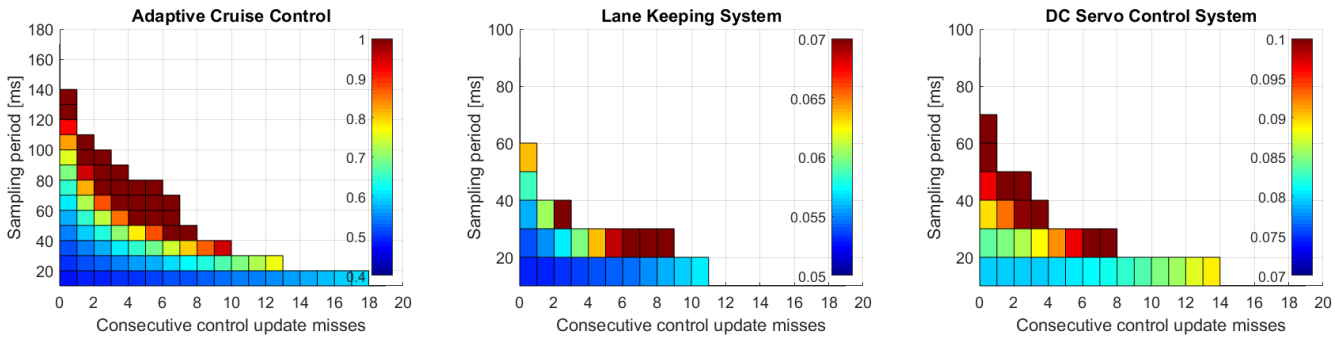
M2. Control performance also depends on both the sampling period and the number of consecutive control update misses. In some cases, we can improve control performance by decreasing the sampling period and allowing a few update misses, as long as the two parameters are within the stability region.

Next, we take a closer look at these two observations and provide the rationale behind our new task model and associated scheduling mechanisms.

M1. We derive the stability region of a physical plant as a function of sampling period T and the number of maximum consecutive control update misses m. The dynamics of a physical plant can be described by a general linear discrete-time model [2]. We then use a well-known stability analysis [2] to calculate the relationship between T and m.[1] Fig. 1 shows the stability region in (m, T)-plane for each physical plant. The stability region is plotted by incrementally increasing m and T and performing the stability analysis. If the plant is stable, a point is marked at the corresponding location of the stability region. One can see from the stability region that a shorter sampling period (i.e., a higher sampling rate) allows the plant to tolerate more consecutive update misses. On the other hand, as the sampling period gets longer, less consecutive update misses or even stricter control updates are required for stability. For example, in case of the adaptive cruise control, when $T = 100ms$, the plant can tolerate up to 3 consecutive control update misses, while it can tolerate up to 10 update misses when $T = 40ms$. Note that when $T > 140ms$, the plant even with one update miss becomes unstable.

It is important to observe that stability can be achieved with much less resources by allowing more deadline misses with a shorter period. We define an *effective control update period* as the time interval between two consecutive control updates in the presence of control update misses. If up to m control updates are missed for a given sampling period T, the effective control update period is $(m+1)\cdot T$. For example, in case of the adaptive cruise control as shown in Fig. 1(a), when $m = 0$, the maximum effective control update period without losing stability is $171ms$, while when $m = 1$, the maximum effective

[1] Due to space limitation, the details of the physical plant dynamics and stability analysis are provided in Appendix A of the supplement file [30].



(a) Adaptive cruise control system (b) Lane keeping system (c) DC-servo control system

Fig. 2. Control performance index calculated by Eq. (1) as a function of sampling period and the number of consecutive control update misses.

control update period is $280ms$ ($(1+1)\cdot 140 = 280$). When $m = 10$, it becomes even larger, i.e., $451ms$ ($(10+1)\cdot 41 = 451$). This indicates that the physical plant can tolerate a longer effective control update period by assigning a short period and then allowing more deadline misses as opposed to allowing no deadline miss. Therefore, stability can be achieved with much less resources by exploiting such a relationship for scheduling.

M2. The control performance of a physical plant can be captured by a standard quadratic performance index [31]:

$$J = \sum_{k=0}^{N} \frac{1}{T} \int_{kT}^{(k+1)T} y(t)^2 dt. \quad (1)$$

where N is the total number of sampling intervals each of which is T seconds, and $y(t)$ is the control output.[2] The performance index J represents an accumulated state error, i.e., a deviation from the desired state. A larger J indicates a larger deviation from the desired states, or worse control performance.

Fig. 2 (with only stable points plotted) illustrates the performance index as a function of the sampling period and the number of consecutive control update misses. The figure shows that the control performance degrades as both the sampling period and the number of consecutive update misses increase.

Note that allowing more update misses with a shorter period tends to yield better control performance than a longer period with less update misses. For example, in case of the adaptive cruise control as shown in Fig. 2(a), when $m = 0$ and $T = 120ms$, the performance index is 1.0, while it is 0.58 when $m = 5$ and $T = 20ms$. Note that the effective control update periods of both cases are the same as, and equal to $120ms$. To understand such a tendency, let us consider two example cases when (i) the sampling period is T and the number of consecutive update misses is 2 (see Fig. 7(a) in Appendix of the supplement file [30]), and (ii) the sampling period is extended by 3x and no update is missed (see Fig. 7(b)). As shown in Fig. 7, the effective control update periods of both (i) and (ii) would be the same as, and equal to $3 \cdot T$. However, the accuracy of each control update is different because cases (i) and (ii) use T and $3{\cdot}T$ time-units old states, respectively. For example, the first updates in Figs. 7(a) and 7(b) are made with States 3 and 1, respectively, while they are applied at the same time. This difference yields different effects of adjusting periods and controlling update misses on control performance. Likewise, the observation in M1 can be reasoned about. It is worth noting that M1 and M2 also hold for other types of physical plants, i.e., the lane keeping system and DC-servo control systems, although specific relations between periods and tolerable update misses slightly differ from each other.

[2]The detailed derivation of the performance index can be found in Appendix A of the supplement file [30].

C. Goal

Ideally, for each individual control task, a shorter period with no update miss guarantee provides better control performance as confirmed in the above case study. However, when multiple control tasks share and hence compete for limited computational resources as in typical CPSes, it may not possible to let each individual task run with the best configuration. Moreover, without a proper scheduling decision, it may cause significant performance degradation or even system instability since the execution of a task is affected by the execution of the other tasks. Thus, it is essential to capture the effect of task parameters on system stability and control performance and then schedule control tasks to achieve high control performance while guaranteeing stability.

Our case study has shown that both period and the number of maximum consecutive update misses are the primary task parameters affecting stability and control performance, and are also related to each other. Specifically, each physical plant has a limit on the number of consecutive update misses to preserve its stability, and the limit varies with the period of its corresponding control task. The control performance of each plant is affected by these two parameters. More importantly, there exists a trade-off between adjusting period and allowing consecutive update misses in terms of the amount of resources required for ensuring stability (M1) and control quality (M2). Thus, it is possible to improve the overall control performance while using resources efficiently to guarantee stability by making such a trade-off.

Our goal is, therefore, to achieve optimal control performance while guaranteeing system stability with limited resources by considering the relationship between period and consecutive update misses for the parameter assignment and scheduling of control tasks. To meet this goal, we propose a new task model that captures the relationship between sampling period and control update misses in terms of stability

and control performance, and then develop a new scheduling mechanism associated with the new task model.

IV. New CPS Task Model

We now introduce a new CPS task model and elaborate on its benefits in comparison with existing task models. Then, we present the overview of our approach to address the scheduling problems associated with the new task model.

A. The New CPS Task Model

We propose a new task model that captures the number of maximum tolerable consecutive control update misses and its relationship with the sampling period in terms of stability and control performance. Each control task τ_i is specified by control- and scheduling-related parts:

- The control-related part of τ_i is characterized by a set of *stable pairs* $\{(m_i, [T_i^{min}(m_i),\ T_i^{max}(m_i)])\}$, where each pair represents the maximum number of tolerable consecutive deadline misses (m_i), $m_i^{min} \leq m_i \leq m_i^{max}$, and its corresponding minimum and maximum periods ($T_i^{min}(m_i)$ and $T_i^{max}(m_i)$) as a function of m_i without losing stability. Each control task is also specified by the performance index function $J_i(m_i, T_i)$ in Eq. (1), where $T_i \in [T_i^{min}(m_i), T_i^{max}(m_i)]$; and
- The scheduling-related part of τ_i is characterized by (T_i, C_i, D_i, m_i), where T_i is the actual period, C_i is the worst-case execution time, and D_i is the relative deadline equal to T_i.

The set of stable pairs and the parameters in each pair for τ_i are known *a priori* via the stability region shown in Fig. 1, whereas T_i and m_i are the design parameters to be determined. C_i is also assumed known. Each task τ_i is assumed to generate a potentially infinite sequence of jobs every T_i time-units. Once each job is released and put into the ready queue, we consider it as *active*, and such an active job needs to complete C_i units of work within a relative deadline of D_i time-units. If an active job cannot finish its execution by its deadline, a *deadline miss*[3] occurs. Note that the CPS task model is more general than the classical real-time task model [1]. In fact, a task having $m_i = 0$ and $T_i^{min}(m_i) = T_i^{max}(m_i) = T_i$ is equivalent to a hard real-time task with a fixed period.

B. Why not existing task models?

Several existing task models allow some job deadline misses, such as (m, k)-firm deadline model [14], skip-over model [15], weakly-hard task model [16], dropout task model [32], and periodic CPS task model [17]. For example, in the (m, k)-firm deadline model, a task allows up to $k - m$ deadlines to be missed during any k consecutive job intervals. The periodic CPS task model specifies a number of tolerable consecutive job deadline misses and its corresponding cost of missed deadlines for each task. Although these task models are designed to capture the tolerance to missed deadlines, they assume a fixed task period and hence cannot capture the interplay between period and tolerable deadline misses with respect to stability and control performance.

Besides such fault-tolerant models, there are other task models proposed to accommodate more general task parameters, such as elastic task parameters [33], imprecise computation workload [34], period-deadline selection [35], control-driven execution rules [36]. Since these task models do not allow any deadline miss, they cannot capture the relationship between period and tolerable deadline misses in terms of stability and control performance.

Unlike the above existing models, our task model can capture all possible combinations of period and the number of tolerable consecutive deadline misses without losing stability and estimate the control performance for each configuration. This provides more flexibility in task scheduling and system-wide optimization in parameter assignment while considering individual control task requirements. Thus, our model facilitates more cost-effective system design that makes better use of the available resources to improve control performance while guaranteeing system stability, as demonstrated in Section VII.

Note that one may argue that a task with period T_i and consecutive deadline misses m_i is equivalent to a hard real-time task with period $(m_i + 1) \cdot T_i$ and relative deadline T_i, which is not true. The former samples the plant state once every T_i time-units and can try a control update at every T_i before missing m_i deadlines consecutively, potentially resulting in a higher control update frequency than the latter which can only update its control every $(m_i+1)\cdot T_i$. Therefore, our model represents control loops more accurately and offers a more general form of control updates.

C. Problem Statement and Approach Overview

We want to maximize the overall control performance without compromising stability for each control task with limited resources. Building upon our new CPS task model, we consider the following two problems:

P1. The optimal parameter assignment problem that determines the period and the number of maximum allowable consecutive deadline misses for each control task such that the worst-case control performance is maximized while guaranteeing stability; and

P2. The online state-aware scheduling problem that generates an actual schedule by considering the current state of each physical plant and dynamically controlling job deadline misses based on the plant state so as to maximize the overall system performance without compromising stability.

The major challenges in addressing P1 and P2 arise from two facts: i) the stability guarantee depends on both parameter assignment and schedule generation, and ii) control performance also varies with the current state of each physical plant. For ease of meeting these challenges, we separate the parameter assignment from schedule generation as follows. We define the notion of *critical* job of a control task that makes the most significant impact on the stability of its physical plant, i.e., if a critical job misses its deadline, the physical plant becomes unstable. To solve P1, we assume a *static minimal* job schedule where only critical jobs are scheduled excluding

[3]It refers to the same situation as a control update miss.

the execution of non-critical jobs. We derive an exact condition that all critical jobs of a task meet their deadlines under the static minimal job schedule for a given pair of period and the number of maximum consecutive deadline misses. Building upon the exact condition for guaranteeing the stability of each physical plant, we formulate the parameter assignment as an optimization problem such that the sum of control performance indices is minimized under the minimal job schedule. We also propose a 2-step approach to efficiently reduce the search space of the optimization problem. To solve P2, building upon the solution to P1, we relax the assumption of a minimal job schedule and develop an online scheduling algorithm that takes the current state of a physical plant into account and selectively chooses non-critical jobs to execute at runtime by utilizing slack resources so as to further optimize control performance without compromising the schedulability of critical jobs. We provide an efficient analysis scheme to determine the execution of a non-critical job by using runtime information without checking all possible future critical job release patterns.

V. Offline Parameter Assignment

To solve the offline parameter assignment problem, we need to answer the following two questions. For a given control task set τ running on a uniprocessor platform,

- Q1. How to find a *stable* parameter assignment of m_i and T_i for every $\tau_i \in \tau$ such that every corresponding physical plant π_i remains stable when its control task is scheduled with the assignment of $\{m_i\}$ and $\{T_i\}$?
- Q2. Among stable parameter assignments, how to find a *performance-optimal* parameter assignment of $\{m_i\}$ and $\{T_i\}$ such that the sum of control performance indices $\sum_i J_i(m_i, T_i)$ is minimized?

To answer Q1, we introduce necessary conditions for a task set to be stable from both control and scheduling perspectives. From an individual control plant perspective, we derived the stability region of a physical plant π_i in the (m_i, T_i)-plane by considering the dynamics of π_i as shown in Fig. 1. Therefore, a pair of (m_i, T_i) must be within the stability region of π_i in order for the physical plant to be stable. This requirement specifies the lower and upper bounds of m_i and T_i:

$$\text{C1: } m_i^{min} \le m_i \le m_i^{max}, \forall \tau_i$$
$$\text{C2: } T_i^{min}(m_i) \le T_i \le T_i^{max}(m_i), \forall \tau_i.$$

Note that the lower and upper bounds of T_i in C2 depend on the value of m_i.

From a control task scheduling perspective, we need to consider the meaning of the stability of a physical plant in terms of its corresponding control task. For a task τ_i with parameters m_i and T_i satisfying C1 and C2, a job of τ_i is said to be *critical* if it is released after missing m_i consecutive job deadlines of τ_i (see Fig. 3). Then, the following lemma describes how the schedule of τ_i relates to the stability of its corresponding physical plant π_i.

Lemma 1: A physical plant π_i is stable if for task τ_i with a given pair of m_i and T_i and its schedule, every critical job of τ_i meets its deadline.

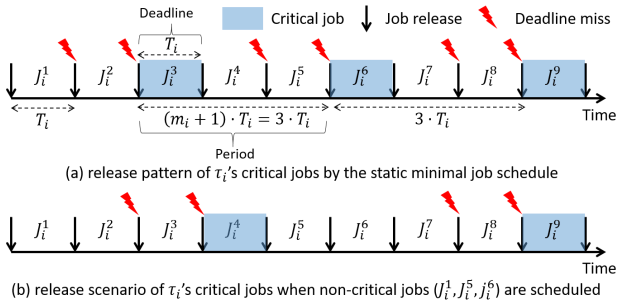


Fig. 3. The notion of a critical job of a task τ_i and release patterns of critical jobs when $m_i = 2$.

Proof: If every critical job of τ_i meets its deadline, by definition, τ_i misses no more than m_i consecutive job deadlines any time instant. This implies that its physical plant π_i is stable. ■

According to Lemma 1, in order to derive the necessary condition for a physical plant to be stable, we need to decide if all critical jobs of τ_i given by a schedule are schedulable. However, it is difficult to determine whether all critical jobs are schedulable because, by definition, a critical job changes dynamically depending on whether a non-critical job misses its deadline (see Fig. 3). Besides, the amount of interference on a critical job of τ_i may vary upon the execution of non-critical jobs of higher priority tasks. Therefore, it is difficult to find a critical job that receives the largest amount of interference among all critical jobs of τ_i. To meet this challenge effectively, we assume a *static minimal* job schedule where only critical jobs are scheduled. Under the static minimal job schedule, the release pattern of τ_i's critical jobs can be viewed as the one of a constrained-deadline periodic task with period $(m+1)\cdot T_i$, the worst-case execution time C_i, and deadline T_i as shown in Fig. 3(a). Such a view allows the direct use of a well-known exact schedulability analysis for the constrained-deadline periodic task model, such as response-time analysis and processor-demand analysis, to check if critical jobs of τ_i are schedulable. In this paper, we consider deadline-monotonic (DM) fixed-priority scheduling,[4] where critical jobs are scheduled in non-decreasing order of relative deadlines. Then, we use the exact response-time analysis [37] to check whether all critical jobs are schedulable under the static minimal job schedule. This analysis becomes a necessary condition for a physical plant to be stable:

$$\text{C3: } R_i^{x+1} = C_i + \sum_{k \in hp(i)} \left\lceil \frac{R_i^x}{(m_k+1)\cdot T_k} \right\rceil C_k \le D_i, \forall \tau_i$$

where $hp(i)$ is the set of higher-priority tasks than τ_i. The left-hand side of the inequality in C3 is an iterative formula starting with an initial value $R_i^0 = C_i$ and ending when $R_i^{x+1} \le R_i^x$ (for schedulable), or $R_i^{x+1} > R_i^x$ (for unschedulable).

It is worth noting that, depending on controller implementation, every job of τ_i, including non-critical jobs, may have the minimum required execution time regardless of the control update. In such a case, the minimum necessary computation demand for τ_i can be modeled as another separate task τ_i' with

[4]Note that DM is an optimal fixed-priority scheduling algorithm for constrained-deadline periodic tasks running on a uniprocessor.

period $T_i' = T_i$, the minimum execution time C_i', deadline $D_i' = T_i$, and $m_i' = 0$, and can be included in the static minimal job schedule. With this view, C3 can be directly extended to this case.

The following theorem presents the necessary conditions for system stability.

Theorem 1: Suppose every control task τ_i in a task set τ is assigned m_i and T_i to satisfy C1, C2, and C3, and scheduled by the static minimal job schedule with DM scheduling. Then, its corresponding physical plant π_i is stable.

Proof: If every task τ_i satisfies C1 and C2, a pair of m_i and T_i is within the stability region of π_i. If τ_i also satisfies C3, every critical job of τ_i meets its deadline. Then, by Lemma 1, its corresponding physical plant π_i is stable. ■

To address Q2, we formulate parameter assignment as an optimization problem subject to the necessary conditions (C1, C2, and C3) for stability derived in Q1. Thus, the optimization problem that minimizes the sum of control performance indices without losing stability is stated as:

$$\begin{aligned} &\text{Minimize } \sum_i J_i(m_i, T_i) \qquad (2)\\ &\text{s.t. C1: } m_i^{min} \le m_i \le m_i^{max}, \forall \tau_i\\ &\quad \text{C2: } T_i^{min}(m_i) \le T_i \le T_i^{max}(m_i), \forall \tau_i\\ &\quad \text{C3: } R_i^{x+1} = C_i + \sum_{k \in hp(i)} \left\lceil \frac{R_i^x}{(m_k+1)\cdot T_k} \right\rceil C_k \le D_i, \forall \tau_i. \end{aligned}$$

Note that $\{m_i\}$ and $\{T_i\}$ for every $\tau_i \in \tau$ are variables to be determined, where m_i is an integer variable. The optimization problem in Eq. (2) involves integers and complex functions, such as iteration and ceiling operators, making it difficult to solve. Hence, we use well-known optimization techniques, such as genetic algorithms [38] and simulated annealing [39], both of which have been shown to be effective in solving numerous optimization problems. However, a main difficulty in applying these techniques is that the time required to solve the problem increases rapidly with the number of variables and constraints. To reduce the search space efficiently, we use the following observation:

Observation 1: i) The upper bound of period ($T_i^{max}(m_i)$) is a monotonically decreasing function of the number of consecutive deadline misses m_i, meaning that $T_i^{max}(m_i^a) \ge T_i^{max}(m_i^b)$ for $m_i^{min} \le m_i^a < m_i^b \le m_i^{max}$. ii) Increasing period T_i for a given m_i only worsens control performance.

Based on this observation, we propose a two-step approach:

1) Limit the number of consecutive deadline misses to 0, i.e., $m_i = 0$, for all tasks (instead of C1) and find $\{T_i\}$ that minimizes the sum of performance indices while satisfying C2 and C3, using the optimization techniques.
2) Set the upper bound of T_i in C2 as derived in the previous step, relax the assumption $m_i = 0$ (consider C1), and find both $\{m_i\}$ and $\{T_i\}$ by setting the initial solution as derived in the previous step and applying the optimization techniques again.

VI. Online State-Aware Scheduling

So far, we have discussed how to determine period and the maximum number of consecutive deadline misses for each task τ_i when executed on a shared platform. We now consider how to schedule the tasks and control their job deadline misses online in order to enhance control performance while guaranteeing stability. Specifically, upon every task invocation, we need to decide whether or not to schedule its job for execution. If the current job of τ_i is not critical then we may or may not execute it, else we have no option but to insert it into the ready queue.

One simple way is to run only critical jobs as was done in the offline parameter assignment. Although such a deterministic scheduling algorithm guarantees stability and control performance, there still remains room for performance improvement by executing non-critical jobs as well, because executing non-critical jobs only decreases the state errors of their physical plants with more frequent control updates using the latest (accurate) state samples instead of old (inaccurate) ones. To this end, we develop an online state-aware scheduling algorithm that selectively chooses non-critical jobs to execute at runtime in order to improve control performance while meeting the deadlines of all critical jobs. In particular, the control performance improvement by executing non-critical jobs was found to depend on the current physical plant state. So, we take the runtime state of each physical plant into account and want to answer the following questions.

I1. How many and which non-critical jobs to execute so as to maximize control performance?

I2. How to ensure that the execution of non-critical jobs will not compromise the deadline guarantees of all critical jobs?

We present below a state-aware scheduling algorithm that addresses the above questions.

Algorithm 1 (State-aware scheduling algorithm): The scheduler is invoked upon (i) release of a new job (`JOB RELEASE`), or (ii) completion of a job (`JOB COMPLETION`). The scheduler maintains two types of queue; one for active jobs released and determined to be scheduled according to priority (ready queue), and the other for non-critical jobs released but not scheduled (wait queue). In case of a `JOB RELEASE` event, if the newly released job is critical, then it is directly placed into the ready queue, else it goes to the wait queue and waits there to be moved to the ready queue. Upon each invocation (either `JOB RELEASE` or `JOB COMPLETION`), our scheduling algorithm checks whether some jobs in the wait queue can be moved to the ready queue and scheduled by using available slack resources. In particular, if there exists a job in the wait queue satisfying Lemmas 2 and 3, we move the job to the ready queue. If there are multiple jobs in the wait queue, they are ordered according to the current state errors of their corresponding physical plants (i.e., deviations from the desired state). We choose a job in decreasing order of the amount of error, i.e., the job with the largest error first. Then, the jobs in the ready queue are scheduled under the DM scheduling policy.

In order to check the possibility of executing a non-critical job, we need to address I1 and I2. Let J_i^c denote a candidate

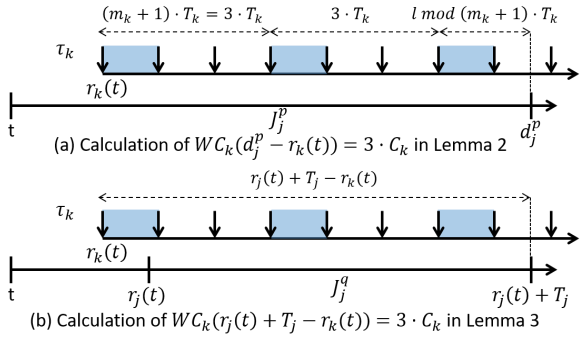


Fig. 4. Upper-bounds on the amount of execution of critical jobs of τ_k released after t.

non-critical job of τ_i to be moved to the ready queue Q_r at time t. Regarding I1, putting J_i^c into Q_r is beneficial only if its execution is completed before its deadline. Otherwise, the computed control input cannot be applied to its physical plant, thus wasting resources. Let $Q_r(t)$ be the set of active jobs in Q_r at time t. Regarding I2, all jobs in $Q_r(t)$ and all critical jobs not in $Q_r(t)$ but to be released in future must meet their deadlines even if J_i^c is added to $Q_r(t)$. We first derive a sufficient condition for any job J_j^p in $Q_r(t)$ to be schedulable. Suppose no non-critical job will be added to Q_r after time t. The execution of J_j^p will be blocked by either the remaining execution of higher priority jobs in $Q_r(t)$ or the execution of higher priority critical jobs to be released in the future. Let $C_k(t)$ be the remaining execution time of an active job of τ_k at time t, and $WC_k(l)$ denote an upper-bound on the amount of execution of critical jobs of τ_k in an interval of length l such that the interval starts at one of the release times of τ_k's critical jobs. Fig. 4 illustrates the scenario of $WC_k(l)$, where critical jobs of τ_k are released every $(m_k+1)\cdot T_k$ time-units starting at the beginning of the interval. By considering the jobs fully executing for C_k and the last job executing for at most C_k, we can calculate the $WC_k(l)$ as:

$$WC_k(l) = \left\lfloor \frac{l}{(m_k+1)\cdot T_k} \right\rfloor \cdot C_k + min(C_k, l \bmod (m_k+1)\cdot T_k). \tag{3}$$

Using $C_k(t)$ and $WC_k(l)$, the following lemma can guarantee the schedulability of every job in $Q_r(t) \cup \{J_i^c\}$ under fixed-priority scheduling.

Lemma 2: Suppose every job J_j^p in $Q_r(t) \cup \{J_i^c\}$ satisfies the following condition. Then, J_i^c and all jobs in $Q_r(t)$ are schedulable under fixed-priority scheduling unless any non-critical job is added to Q_r after t.

$$\sum_{\tau_k \in hp(j)} \Big(C_k(t) + WC_k(\max(d_j^p - r_k(t), 0)) \Big) + C_j(t) \le d_j^p - t, \tag{4}$$

where $hp(j)$ is a set of higher priority tasks than τ_j, d_j^p is the absolute deadline of J_j^p, and $r_k(t)$ is the earliest possible next release time of a critical job of τ_k at time t.[5]

[5]Note that $r_k(t)$ can be calculated as the time at which τ_k was last released plus $(m_k+1)\cdot T_k$.

Proof: We will prove that the execution of J_j^p will be completed by d_j^p if Inequality (4) holds. In order for J_j^p to finish its execution by d_j^p, the sum of its remaining execution time $C_j(t)$ and the interference imposed by higher priority jobs during $[t,\ d_j^p)$ must be less than or equal to $d_j^p - t$. We consider two sources of the interference: (a) one by the execution of higher-priority jobs in $Q_r(t)$ and (b) the other by the execution of higher-priority critical jobs released after t. In case (a), the execution of higher-priority jobs in $Q_r(t)$ in $[t, d_j^p)$ is upper-bounded by $\sum_{\tau_k \in hp(j)} C_k(t)$. In case (b), if $d_j^p < r_k(t)$, no critical job of τ_k will be executed in $[t, d_j^p)$, leading to $WC_k(\min(d_j^p - r_k(t), 0)) = 0$. Otherwise, by definition, $WC_k(d_j^p - r_k(t))$ is an upper-bound of the execution of critical jobs of τ_k in $[r_k(t),\ d_j^p)$, and no critical job of τ_k released after t is executed in $[t,\ r_k(t))$ as shown in Fig. 4(a). This implies that J_j^p will finish its execution by d_j^p if Inequality (4) holds, thus proving the lemma. ■

In addition to Lemma 2, we must guarantee the schedulability of all critical jobs released in future. However, there exist an infinite number of such critical jobs, and it is non-trivial to find a critical job that receives the largest amount of interference among all critical jobs released after time t. This is because the synchronous arrival sequence becomes no longer the critical instant due to the execution of non-critical jobs. Instead, we prove that at time t, it is sufficient to check only the schedulability of the critical job of τ_j that will be released earliest, denoted as J_j^q (see Lemma 4).

The following lemma guarantees the schedulability of J_j^q.

Lemma 3: Suppose J_j^q satisfies the following condition at time t. Then, J_j^q are schedulable under fixed-priority scheduling unless any non-critical job is added to Q_r after t.

$$\max\Big(\sum_{\tau_k \in hp(j)} C_k(t) - (r_j(t) - t), 0 \Big) + \sum_{\tau_k \in hp(j)} WC_k(\max(r_j(t) + T_j - r_k(t), 0)) + C_j \le T_j. \tag{5}$$

Proof: This lemma can be proved similarly as Lemma 2. See the details in Appendix C of the supplement file [30]. ■

The following lemma guarantees the schedulability of all other critical jobs of τ_j released after J_j^q.

Lemma 4: Suppose a task τ_j satisfies Lemma 3 at time t and C3 presented in Section V. Then, all critical jobs of τ_j that will be released after t are schedulable under fixed-priority scheduling unless any non-critical job is added to Q_r after t.

Proof: By definition, Lemma 3 guarantees the schedulability of the critical job J_j^q of τ_j that will be released the earliest after t. C3 presented in Section V calculates the worst-case response time of τ_j (denoted as R_j^*), where only critical jobs are scheduled. We will prove that for every critical job J_j^r of τ_j that will be released after J_j^q, its response time should be less than, or equal to R_j^*. Otherwise, J_j^q misses its deadline, which contradicts our assumption. We consider two scenarios where the response time of J_j^r is greater than R_j^*: (a) the execution of non-critical jobs in $Q_r(t) \cup \{J_i^c\}$ directly delays the execution of J_j^r (direct interference), and (b) the execution of non-critical jobs in $Q_r(t) \cup \{J_i^c\}$ delays

TABLE I. CONTROL TASK PARAMETERS

	The number of deadline misses (m_i)	Period (T_i) (ms)
Adaptive cruise	$\{1, 2, 4, 8, 12\}$	$\{20 : 20 : 160\}$
Lane keeping		$\{10 : 10 : 80\}$
DC-servo		$\{10 : 10 : 90\}$

the execution of critical jobs of higher priority than J_j^r, which in turn delays the execution of J_j^r (indirect interference). Case (a) is possible only if there exists any remaining execution of higher-priority non-critical jobs in $Q_r(t) \cup \{J_i^c\}$ at the release time of J_j^r, denoted as r_j^r. This implies that there is no idle time instant in $[t,\ r_j^r)$, and thus the job J_j^q released earlier than r_j^r could not finish its execution by its deadline. This contradicts our assumption. In case (b), if J_j^q meets its deadline by the assumption, then there exist at least C_j time-units where no higher-priority jobs will be executed during $[r_j(t),\ r_j(t)+T_j)$. This implies that the execution of every higher-priority critical job released after $r_j(t) + T_j$ cannot be deferred further by the execution of non-critical jobs in $Q_r(t) \cup \{J_i^c\}$, meaning that the response time of J_j^r cannot be greater than R_j^*. ■

Based on Lemmas 2, 3, and 4, the following theorem derives an important property of our online state-aware scheduling algorithm.

Theorem 2: Our online state-aware scheduling algorithm 1 guarantees to schedule not only all critical jobs of each task but also all non-critical jobs placed into the ready queue.

Proof: Trivially, the theorem holds at time 0. Suppose the theorem holds at time t. Upon next invocation of either `JOB RELEASE` or `JOB COMPLETION`, our scheduling algorithm checks Lemmas 2 and 3 to see if the wait queue is not empty. A non-critical job can put into the ready queue only if the following conditions are satisfied: (a) the job itself is schedulable (by Lemma 2), (b) all non-critical jobs already in the ready queue are schedulable (by Lemma 2), and (c) all critical jobs are schedulable (by Lemmas 3 and 4). Otherwise, it cannot put into the ready queue. ■

Time complexity. Let n_c denote the number of control tasks in a task set. At each invocation (either `JOB RELEASE` or `JOB COMPLETION`), Algorithm 1 checks Lemmas 2 and 3 for each job in the wait queue. In Lemma 2, it checks Eq. (4) for every job in the ready queue that requires O(n_c^2). The same holds for Lemma 3. There exist at most n_c jobs in the wait queue. Thus, the complexity of Algorithm 1 is O(n_c^3).

VII. EVALUATION

We now demonstrate the capability of the proposed task model and scheduling approaches making a significant improvement of control performance and resource-efficiency. We use two metrics: *schedulability ratio* and *control performance index*. The schedulability ratio is defined as the percentage of schedulable task sets of the total number of generated task sets by using an algorithm. The control performance index is defined in Eq. (1), which represents the difference between actual and desired states. The larger the performance index, the worse the control performance.

A. Simulation Setup

Task set generation. In our simulation, we consider the autonomous vehicle control system shown in our case study (Section III). There are three control tasks — adaptive cruise control (denoted by τ_1), lane keeping control (denoted by τ_2), and DC-servo control (denoted by τ_3) tasks — which run with hard real-time tasks on a uniprocessor platform. According to the stability region shown in Fig. 1, the set of possible values for the number of consecutive deadline misses (m_i) and period T_i are specified in Table I. For example, for the adaptive cruise control task τ_1, m_1 is chosen among 1, 2, 4, 8, and 12, and T_1 is chosen from $20ms$ to $160ms$ with increments of $20ms$. Note that for each value of m_i, the upper bound of T_i, i.e., $T_i^{max}(m_i)$ is set, corresponding to the stability region in Fig. 1. Also, note that all values shown in Table I are derived from the real vehicle dynamics and the setting shown in Appendix B of the supplement file [30]. We set the worst-case execution times to $C_1 = C_2 = C_3 = 15ms$.

In addition to the control tasks, we generate hard real-time tasks by using the UUniFast algorithm [40], which has been widely used for the generation of uniprocessor synthetic task sets. We generate 910 hard real-time task sets in total while varying their total utilization of tasks from 0.1 to 1.0 with an incremental step of 0.1. The details of hard real-time task set generation are provided in Appendix D of the supplement file [30]. For each generated hard real-time task set Γ_H, we add the above three control tasks Γ_C and complete the task set generation ($\Gamma = \Gamma_C \cup \Gamma_H$).

With the generated task sets, we compare the following four different scheduling approaches:

- Our-PA-SAS: offline parameter assignment in Eq. (2) and online state-aware scheduling in Algorithm 1;
- Our-PA: offline parameter assignment under static critical-job-only scheduling;
- DPA-EDF: dynamic period adjustment under earliest deadline first (EDF) [11][6]; and
- SPA-RM: static period assignment under RM.[7]

Note that other related techniques for dynamic period adjustment and static period assignment (mentioned in Section II) are not included in this comparison, since they focus on different directions for performance improvement. For example, some of them considered different types of scheduling algorithms or performance indices [10], and others considered the impact of jitter and delay on the control performance [4–6, 12]. In contrast, we aim to compare the performance improvement of our proposed approaches against period assignment/adjustment under hard deadline constraints without considering other factors that affect performance. Our proposed approaches can be extended to consider other scheduling algorithms, such as EDF, and jitter and delay effects, but it is left as our future work.

[6] DPA-EDF dynamically changes task periods at runtime without violating EDF constraints, i.e., the total utilization must be not greater than 1. In particular, we use the optimal policy shown in Eq. (17) in [11], where at any time t, the control task whose physical plant faces the largest state error is assigned its period as short as possible while assigning longest possible periods to the other control tasks.

[7] SPA-RM solves the optimization problem in Eq. (2) by fixing $m_i = 0$.

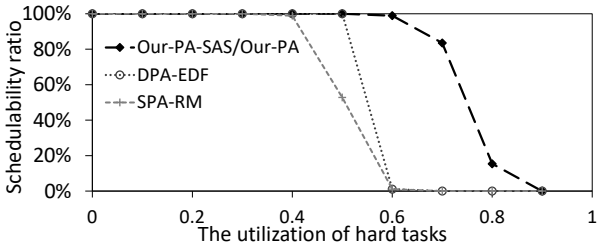


Fig. 5. Schedulability ratio with different utilizations of hard real-time tasks.

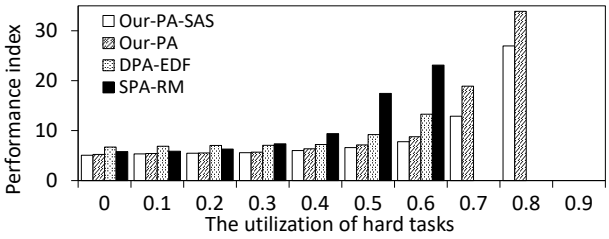


Fig. 6. Control performance with different utilizations of hard real-time tasks

B. Simulation Results

Our proposed approach is evaluated in terms of schedulability as well as control performance.

Schedulability. Fig. 5 compares the percentage of schedulable task sets by four scheduling approaches while varying the total utilization of hard real-time tasks U_{Γ_H} from 0 to 1. Both Our-PA-SAS and Our-PA find 39% and 54% more schedulable task sets than DPA-EDF and SPA-RM, respectively.[8] The performance gap between Our-PA-SAS/Our-PA and DPA-EDF/SPA-RM is shown to become larger as U_{Γ_H} increases. As U_{Γ_H} increases, less resources $(1-U_{\Gamma_H})$ become available for control tasks. Note that the minimum possible value of the total utilization for control tasks $U_{\Gamma_C}^{min}$ is $15/160+15/80+15/90 \approx 0.45$ when no deadline miss is allowed. Both DPA-EDF and SPA-RM cannot find any schedulable task set when $U_{\Gamma_C}^{min} + U_{\Gamma_H} > 1$ (i.e., $U_{\Gamma_H} \geq 0.6$), since they assume that no deadline miss is allowed when assigning control task periods. On the other hand, using Our-PA-SAS/Our-PA, 83% and 15% of the task sets are schedulable at $U_{\Gamma_H} = 0.7$ and 0.8, respectively. Such an improvement can be interpreted as the benefit of taking the effect of periods and deadline misses on system stability into account (by our new task model) when assigning a pair of period and the number of maximum consecutive deadline misses (by our offline parameter assignment). As a result, Our-PA-SAS and Our-PA can accommodate more control tasks without compromising system stability under limited computing resources.

Control performance. To evaluate control performance, for each task set, we simulate the response of each control plant for 30 seconds, and calculate the performance index shown in Eq. (1). To emulate set-point changes and external disturbances for each control plant, random disturbances are generated at 3-second intervals and added to the input of each control plant.[9] Fig. 6 compares the performance index of four scheduling schemes with different values of U_{Γ_H}. Our-PA-SAS is shown to outperform the existing schemes for all U_{Γ_H} values. Our-PA-SAS and Our-PA are shown to dominate SPA-RM, because our offline parameter assignment uses the solution of SPA-RM as an initial input to the optimization process according to the two-step approach presented in Section V. When there exist only control tasks in the task set, i.e., $U_{\Gamma_H} = 0$, the performance improvement by Our-PA-SAS is relatively marginal, since a sufficiently short period can be assigned to every control task by fully utilizing the total processing capacity. However, the performance gap between Our-PA-SAS and other schemes becomes larger as less resources become available for control tasks. For example, when $U_{\Gamma_H} = 0.6$, Our-PA, DPA-EDF, and SPA-RM show 13%, 71%, and 197% worse performance than Our-PA-SAS. Due to limited resources available for control tasks, SPA-RM assigns a longer period to each control task to meet all deadlines, and DPA-EDF also has a limited range of adjustable periods. On the other hand, Our-PA-SAS and Our-PA assign a relatively shorter period and hence allow a few deadline misses. So, a period assignment along with proper online management of deadline misses yields better control performance than a period assignment/adjustment under hard deadline constraints. The performance improvement of Our-PA-SAS over Our-PA is also observed to increase as U_{Γ_H} increases. As U_{Γ_H} increases, Our-PA determines the parameters of each control task to allow more deadline misses in order to guarantee stability, implying that more non-critical jobs wait to be scheduled by Our-PA-SAS. Meanwhile, it is more likely to have less slack resources available to non-critical jobs. Nevertheless, Our-PA-SAS selects proper non-critical jobs to schedule by considering their current plant states online, and thus yields up to 32% better performance than Our-PA. Note that our simulation assumes every job consumes its worst-case execution time. If we consider the case where each job finishes its execution earlier than expected, Our-PA-SAS can further improve control performance by utilizing more slack resources to schedule more non-critical jobs, whereas Our-PA cannot.

VIII. CONCLUSION

We have found that sampling period and the number of consecutive job deadline misses must be considered *together* to maximize control performance without compromising the stability of a CPS, and then presented three components that address this issue. First, we have developed a new task model that captures the relation between the two key parameters. Second, we have derived an optimal static parameter assignment policy that maximizes the worst-case control performance while guaranteeing stability. Third, we have developed an online scheduling algorithm that utilizes the current state of a CPS and dynamically controls job deadline misses, making control performance improvement without compromising stability. Our evaluation results demonstrated that online control update management along with appropriate parameter assignment by exploiting the relationship between period and deadline misses makes a significant improvement of both control performance and schedulability.

[8]Note that in terms of schedulability, Our-PA-SAS and Our-PA show the same performance since both are based on the offline parameter assignment in Eq. (2), but they show differences in control performance due to use of different scheduling algorithms as shown later.

[9]For real vehicle scenarios, it is possible to apply well-known traffic analysis tools, such as NGSIM [41] and CarSim [42], to obtain real-world vehicle trajectory data. We leave it as part of our future work.

ACKNOWLEDGMENT

The work reported in this paper was supported in part by the Office of Naval Research under Grants N00014-15-1-2163 and N00014-18-1-2141, and the National Science Foundation under Grant CNS-1329702. This work was also supported in part by the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (2017R1A2B2002458, 2017H1D8A2031628, 2017K2A9A1A01092689).

REFERENCES

[1] C. Liu and J. Layland, "Scheduling algorithms for multi-programming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[2] K. J. Astrom and B. Wittenmark, *Computer-controlled systems*. Prentice-Hall, Inc., 1997.

[3] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin, "On task schedulability in real-time control systems," in *RTSS*, 1996.

[4] E. Bini and A. Cervin, "Delay-aware period assignment in control systems," in *RTSS*, 2008.

[5] Y. Xu, K.-E. Arzen, A. Cervin, E. Bini, and B. Tanasa, "Exploiting job response-time information in the co-design of real-time control systems," in *RTCSA*, 2015.

[6] A. Cervin, B. Lincoln, J. Eker, K.-E. Arzen, and G. Buttazzo, "The jitter margin and its application in the design of real-time control systems," in *RTCSA*, 2004.

[7] Y. Wu, G. Buttazzo, E. Bini, and A. Cervin, "Parameter selection for real-time controllers in resource-constrained systems," *IEEE Transactions on Industrial Informatics*, vol. 6(4), pp. 610–620, 2010.

[8] A. Aminifar, S. Samii, P. Eles, Z. Peng, and A. Cervin, "Designing high-quality embedded control systems with guaranteed stability," in *RTSS*, 2012.

[9] L. Palopoli, C. Pinello, A. S. Vincentelli, L. Elghaoui, and A. Bicchi, "Synthesis of robust control systems under resource constraints," in *Proceedings of Hybrid Systems: computation and control*, 2002.

[10] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Arzen, "Feedback-feedforward scheduling of control tasks," *Real-Time Systems*, vol. 23, pp. 25–53, 2002.

[11] P. Marti, C. Lin, S. A. Brandt, M. Velasco, and J. M. Fuertes, "Optimal state feedback based resource allocation for resource-constrained control tasks," in *RTSS*, 2004.

[12] R. Castane, P. Marti, M. Velasco, A. Cervin, and D. Henriksson, "Resource management for control tasks based on the transient dynamics of closed-loop systems," in *ECRTS*, 2006.

[13] P. Ramanathan, "Overload management in real-time control application using (m, k)-firm guarantees," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 6, June 1999.

[14] M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m, k)-firm deadlines," *IEEE Transactions on Computers*, vol. 44, no. 12, pp. 1443–1451, 1995.

[15] G. Koren and D. Shasha, "Skip-over: Algorithms and complexity for overloaded systems that allow skips," in *RTSS*, 1995.

[16] G. Bernat, A. Burns, and A. Llamosi, "Weakly hard real-time systems," *IEEE Transactions on Computers*, vol. 50, no. 4, pp. 308–321, 2001.

[17] J. Lee and K. G. Shin, "Development and use of a new task model for cyber-physical systems: A real-time scheduling perspective," *Journal of Systems and Software*, vol. 126, pp. 45–56, 2017.

[18] M. S. Branicky, S. M. Phillips, and W. Zhang, "Scheduling and feedback co-design for networked control systems," in *Proceedings of the 41st IEEE Conference on Decision and Control*, 2002.

[19] M. Kauer, D. Soudbakhsh, D. Goswami, S. Chakraborty, and A. M. Annaswamy, "Fault-tolerant control synthesis and verification of distributed embedded systems," in *DATE*, 2014.

[20] D. Goswami, R. Schneider, and S. Chakraborty, "Relaxing signal delay constraints in distributed embedded controllers," *IEEE Transactions on Control Systems Technology*, vol. 22(6), pp. 2337–2345, 2014.

[21] D. Soudbakhsh, L. T. Phan, O. Sokolsky, I. Lee, and A. Annaswamy, "Co-design of control and platform with dropped signals," in *ICCPS*, 2013.

[22] R. Majumdar, I. Saha, and M. Zamani, "Performance-aware scheduler synthesis for control systems," in *EMSOFT*, 2011.

[23] T. Yoshimoto and T. Ushio, "Optimal arbitration of control tasks by job skipping in cyber-physical systems," in *ICCPS*, 2011.

[24] M. A. Khatib, A. Girard, and T. Dang, "Scheduling of embedded controllers under timing contracts," in *HSCC*, 2017.

[25] L. Palopoli, L. Abeni, F. Conticelli, M. D. Natale, and G. Buttazzo, "Real-time control system analysis: an integrated approach," in *RTSS*, 2000.

[26] I. Saha, S. Baruah, and R. Majumdar, "Dynamic scheduling for networked control systems," in *HSCC*, 2015.

[27] G. Guo and W. Yue, "Sampled-data cooperative adaptive cruise control of vehicles with sensor failures," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15(6), pp. 2404–2418, 2014.

[28] D. Soudbakhsh and A. Eskandarian, *Handbook of Intelligent Vehicles*. Springer London, 2012.

[29] A. Cervin and P. Alriksson, "Optimal on-line scheduling of multiple control tasks: A case study," in *ECRTS*, 2006.

[30] H. S. Chwa, K. G. Shin, and J. Lee, "Supplement of "closing the gap between stability and schedulability: A new task model for cyber-physical systems"," https://kabru.eecs.umich.edu/wordpress/wp-content/uploads/CSL18sub.pdf.

[31] A. Cervin and J. Eker, "Control-scheduling codesign of real-time systems: The control server approach," *Journal of Embedded Computing*, vol. 1(2), pp. 209–224, 2005.

[32] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, and V. Verdugo, "A scheduling model inspired by control theory," in *RTNS*, 2017.

[33] G. Buttazzo, G. Lipari, and L. Abeni, "Elastic task model for adaptive rate control," in *Proceedings of IEEE Real-Time Systems Symposium*, 1998.

[34] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Optimal reward-based scheduling for periodic real-time tasks," *IEEE Transactions on Computers*, vol. 50(2), pp. 111–130, 2001.

[35] T. Chantem, X. Wang, M. Lemmon, and X. S. Hu, "Period and deadline selection for schedulability in real-time systems," in *ECRTS*, 2008.

[36] M. Velasco, P. Marti, and E. Bini, "Control-driven tasks: Modeling and analysis," in *RTSS*, 2008.

[37] N. Audsley, A. Burns, M. Richardson, and A. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, vol. 8, no. 5, pp. 284–292, 1993.

[38] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., 1989.

[39] P. J. M. van Laarhoven and E. H. L. Aarts, *Simulated annealing: theory and applications*. Kluwer Academic Publishers, 1987.

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

[41] Next generation simulation (NGSIM) vehicle trajectories. [Online]. Available: https://catalog.data.gov/dataset/next-generation-simulation-ngsim-vehicle-trajectories

[42] CarSim mechanical simulation. [Online]. Available: https://www.carsim.com/