Overload Management in Real-Time Control Applications Using (m, k)-Firm Guarantee

Parameswaran Ramanathan, *Member*, *IEEE*

Abstract—Tasks in a real-time control application are usually periodic and they have deadline constraints by which each instance of a task is expected to complete its computation, even in the adverse circumstances caused by component failures. Techniques to recover from processor failures often involve a reconfiguration in which all tasks are assigned to fault-free processors. This reconfiguration may result in processor overload where it is no longer possible to meet the deadlines of all tasks. In this paper, we discuss an overload management technique which discards selected task instances in such a way that the performance of the control loops in the system remain satisfactory even after a failure. The technique is based on the rationale that real-time control applications can tolerate occasional misses of the control law updates, especially if the control law is modified to account for these missed updates. The paper devises a scheduling policy which deterministically guarantees when and where the misses will occur. The paper also proposes a methodology for modifying the control law to minimize the deterioration in the control system behavior as a result of these missed control law updates.

Index Terms—Real-time systems, fault-tolerant controllers, real-time scheduling, overload management, optimal feedback control.

1 Introduction

A real-time control application is often modeled as a set of interacting tasks where each task is responsible for carrying out part of the control law computations. Examples of such applications include flight-control systems, vehicle-control systems, process-control systems, and life-support systems. Since control law computations are usually done at regular intervals, the tasks in a real-time control application are usually periodic in nature. Furthermore, each instance of a task also has a deadline constraint by which it is expected to complete its computation. In addition, tasks in a real-time control application have dependability constraints which require delivery of satisfactory performance even under adverse circumstances caused by component failures.

Many different scheduling techniques have been proposed in literature to deterministically guarantee the deadlines of all task instances in a given application when no faults are present. The techniques often differ in the models of the tasks and the system they can deal with. For instance, in [1], [2], [3], [4], [5], solutions are proposed for scheduling preemptive tasks in uniprocessor systems. Solutions for nonpreemptive tasks in uniprocessor systems are discussed in [6]. For multiprocessor systems, it has been shown that the problem of scheduling nonpreemptive tasks with deadline constraints is NP-hard [7]. Therefore, various heuristics have been proposed for solving this problem [8], [9]. Some of these heuristics consider resource constraints [9], while others mainly concentrate on precedence and communication requirements between tasks [8].

Most of these solutions can be made resilient to faults by combining them with techniques to recover from component failures. Broadly speaking, these techniques rely on one of the following two approaches [10]. One approach is to have adequate spare capacity in the system so that the tasks can be reassigned or reexecuted on fault-free processors upon detection of a failure without violating the deadline constraints of any task [11]. The main drawback of this approach is that the system resources are often underutilized when no faults are present. The other approach is to invoke an overload management technique upon detection of a failure. For example, one can prioritize tasks based on their importance to the application and discard tasks which do not adversely affect the performance delivered by the application [12]. The solution discussed in this paper for dealing with component failures is based on this latter approach.

During overload, the proposed solution invokes a scheduling policy which carefully discards task instances in order to reduce the effective utilization of the system. Since the discarded instances will not be executing the control law, this tends to degrade the performance of the control loops in the application. To minimize the amount of degradation, our solution modifies the control law implemented by the tasks in the application. A methodology for modifying the control law and a technique for selecting the instances to be discarded are discussed in this paper. The effectiveness of this solution is evaluated for an example system. The evaluation shows that a considerable reduction in the effective utilization of the system can be achieved without much degradation in the step response of the control loops in the application.

More specifically, the solution in this paper is based on the (m, k)-firm guarantee model proposed in [13]. In this model, a periodic task is said to have an (m, k)-firm guarantee requirement if it is adequate to meet the deadlines of m out of any k consecutive instances of the task where m and k are two positive integers with $m \leq k$. The

• *The author is with the Department of Electrical and Computer Engineering, University of Wisconsin-Madison, Madison, WI 53706-1691. E-mail: parmesh@ece.wisc.edu.*

For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 109047.

main advantage of this guarantee model is that one can represent a wide range of tolerance to deadline misses by properly choosing the values of m and k. In particular, the traditional hard deadline requirement can be represented as $(1,1)$-firm guarantee requirement and a soft deadline requirement of a bound on the fraction of deadline misses can be approximated by picking a large value for k and choosing m such that m/k equals the desired fraction. However, for most values of m and k, $m < k$, the (m,k)-firm guarantee requirement is less stringent than the hard deadline requirement, but more stringent than the soft deadline requirement.

Given the m and the k values for each task in the application, we first devise a scheduling policy to deterministically provide an (m,k)-firm guarantee to each task in the application. We then show that the control law implemented by a periodic task can be modified to deal with the (m,k)-firm guarantee without much loss in effectiveness. By combining the two solutions, we can design a computer controller for a real-time application with much reduced cost.

The rest of this paper is organized as follows: The motivation for the problem addressed in this paper is discussed in Section 2 and a formal description of the problem is presented in Section 3. A scheduling policy for providing deterministic (m,k)-firm guarantee is discussed in Section 4. The derivation of the optimal control law for a task with (m,k)-firm guarantee is described in Section 5. A numerical example to illustrate the benefit of the proposed approach is presented in Section 6. The paper concludes with Section 7.

2 Motivation

The problem considered in this paper is best motivated by a simple example. Consider an automobile control application with four subsystems: cruise-control, traction-control, brake-control, and engine control. Suppose that the control laws for these subsystems are implemented by their respective periodic tasks. Also suppose that these four periodic tasks have been assigned to a two processor system, as shown in Fig. 1. Further, suppose that as result of a reconfiguration after a processor failure, these four tasks must execute on the same processor and it is not possible to guarantee the deadlines of all the four tasks after this reconfiguration. The question then is how should the system deal with this overload so that it can continue to provide satisfactory level of service to all four subsystems?

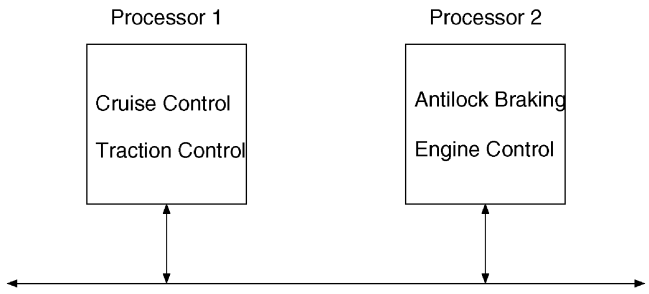

Fig. 1. An example automobile-control system.

Our answer to this question is based on the observation that most control systems can tolerate a few deadline misses in their control law computation, especially if the deadline misses are adequately spaced. For example, let us suppose that the cruise-control subsystem in the above example is a time invariant system shown in Fig. 2. In this subsystem, assume that $D(z)$ is an optimal LQR servo controller [14] and the control law is derived using the results later in this paper. Fig. 3 compares the impact of missed deadlines on the response of the system to a step change in the desired speed from 45 mph to 55 mph. The solid line shows the observed response when all instances of the cruise-control task meet their deadlines. The dashed line, on the other hand, shows the observed response when one out of every three consecutive instances of the task miss their deadlines. On comparing the solid and the dashed lines, we note that the two responses are very similar. This means that one can afford not to service one out of every three instances of this task without a significant degradation in the performance of the cruise-control subsystem. Since skipping one out of every three instances results in a 33 percent reduction in utilization of the task, it can be used to alleviate the overload problem. This is basic idea of the approach pursued in this paper.

An alternate approach to reduce the utilization of a periodic task is to increase its period. Changing the sampling period of a control system alters its dynamics. Also, a change in the period of one task may necessitate a change in the periods of related tasks because efficient exchange of information between interacting tasks is often accomplished through a careful selection of relative periods (see discussion on Assumption A1 in Section 3). Also, in most real-time control applications, the system must be designed to deliver satisfactory performance in the worst-case. Consequently, a processor is considered overloaded if

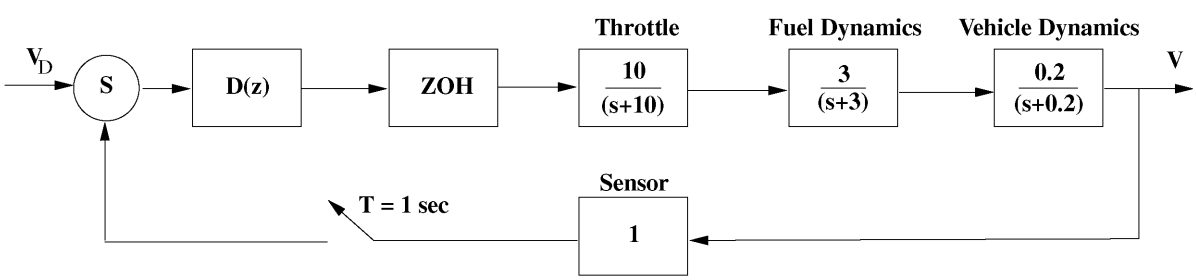

Fig. 2. An example automobile cruise-control system.

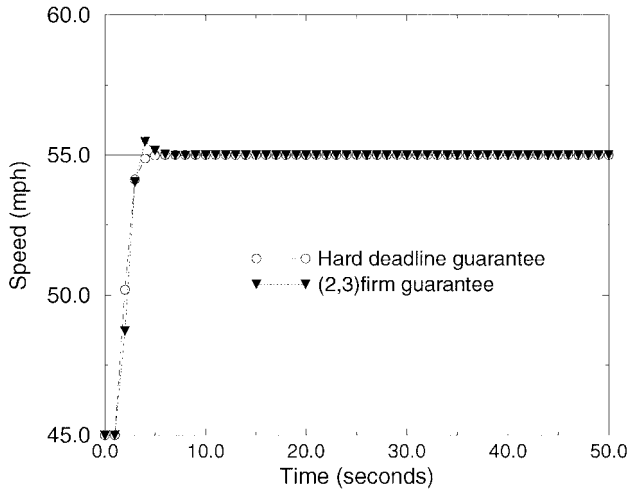

Fig. 3. Impact of (m,k)-firm guarantee model on the cruise-control system.

it cannot complete all the necessary computations in the worst-case. The (m,k)-firm guarantee based approach can be adapted to exploit the fact that the average case utilization is much less than the worst-case utilization. A comparison of the two approaches, namely the (m,k)-firm based approach and the reduced sampling rate approach, on an example system is presented in Section 6.

3 Problem Statement

The problem addressed in this paper can be formally stated as follows. We consider a real-time control application comprised of N periodic tasks, $\tau_1, \tau_2, \ldots, \tau_N$. Each task is assumed to be responsible for performing the control law computations for one subsystem in the application. In addition, we make the following assumptions about these tasks:

A1. The tasks are preemptive and independent.

A2. All tasks are scheduled to be executed on one processor and it is not possible to guarantee the deadlines of all instances of the tasks.

A3. Each task τ_i is characterized by two integers, m_i and k_i, $m_i \leq k_i$ such that it is adequate to meet the deadlines of m_i out of any k_i consecutive instances of the task.

A4. The subsystem controlled by each task τ_i, $1 \leq i \leq N$ is linear and time-invariant.

The rationale for Assumption A1 is as follows: First, the control law computations are usually arithmetic operations which can be preempted without much difficulty. Second, due to the repetitive nature of the control law computations, the dependence between tasks in a real-time control application is usually in the form producer-consumer relationship in which the consumer task utilizes the most recent output(s) from the producer task in performing its computations. Task-pairs with such dependencies can be treated as independent if their relative periods are carefully chosen and the tasks exchange information through a shared double buffer. In some cases, access to shared buffers may have to be regulated using semaphores. Extension for our scheme to deal with tasks interacting through semaphores is discussed in Section 4.2. Assumption A2 is made because we are only interested in the overloaded case. If the processor is not overloaded, there are several schemes in literature to effectively deal with the deadline constraints. The overloaded processors can be treated independently if the tasks are independent (see Assumption A1) and access to shared memory from each processor is not very expensive. The justification for the Assumption A3 comes from the example discussed in the previous section. Assumption A4 is often used in control theory because most nonlinear systems can be analyzed by linearizing them around their region of operation.

In the following two sections, we discuss our two-prong approach to deal with the overload problem. In Section 4, we describe a scheduling policy assuming that we need to guarantee only m_i out of any k_i consecutive instances of each task τ_i. Then, in Section 5, we focus on a typical task τ_i and derive the optimal control law to be implemented by τ_i given that it is scheduled along with other tasks using the policy in Section 4.

4 Providing Deterministic (m,k)-Firm Guarantee

Recall that we have N periodic tasks to be scheduled on a single processor. Each task τ_i is characterized by its maximum computation time C_i and its period T_i. We assume that the relative deadline of each instance of τ_i is equal to T_i and the scheduling policy must provide an (m_i,k_i)-firm guarantee to τ_i. We use Z_+ to denote the set of nonnegative integers.

Our solution is a combination of ideas from the imprecise computation approach [15], [16] and the Rate Monotonic Scheduling policy [3]. In the imprecise computation approach, the computation time of each instance of a periodic task is divided into a mandatory and an optional part. The mandatory part of every instance must complete within its deadline and it is better to complete as much of the optional part as possible. In contrast, in our approach, we classify instances of τ_i as either mandatory or optional such that if all the mandatory instances meet their respective deadlines, then τ_i's (m_i,k_i)-firm guarantee requirement is satisfied. The scheduling of the mandatory instances of all the tasks in the application are done using the Rate Monotonic Policy [3]. That is, the mandatory instances of τ_i are assigned a higher priority than the mandatory instances of τ_j if and only if $T_i < T_j$. The optional instances of all tasks are assigned the lowest priority.[1]

More formally, our approach for providing deterministic (m_i,k_i)-firm guarantee to each task τ_i can be described using the two concurrent processes, **Service_Process** and **Priority_Assign_Process**. The **Service_Process**, is a scheduler which services the task instances in a *Wait* queue. It implements the traditional fixed priority preemptive

1. Instead, we can use a server-based approach to service more optional instances without violating the deadlines of the mandatory instances. We do not consider a server-based approach in this paper because it will detract us from the main theme of the paper.

scheduling policy [3]. Such a policy is based on two simple rules. First, the server never idles if an instance is awaiting service. Second, the server always executes the highest priority instance that is waiting for service. To meet the second requirement, the server preempts the service of a lower priority instance if a higher priority instance is placed in the wait queue while the lower priority instance is being serviced.

The **Priority_Assign_Process**, on the other hand, assigns priorities to the activated instances and places them in a *Wait* queue. The novelty of the scheduling policy is in this process. As shown in Fig. 4, the basic idea is to selectively classify the instances of a task as either *mandatory* or *optional*. The mandatory instances are assigned a priority in such a way that their deadlines are guaranteed. The optional instances are assigned the lowest priority and are not guaranteed to meet their respective deadlines. The classification of instances of τ_i as mandatory or optional is based on the values m_i and k_i. More specifically, instances of τ_i are activated at times aT_i, for $a = 0, 1, 2, \ldots$. An instance activated at time aT_i is classified as mandatory if

$$a = \left\lfloor \left\lceil \frac{am_i}{k_i} \right\rceil \cdot \frac{k_i}{m_i} \right\rfloor$$

and as optional, otherwise. We show later in this section that in this approach at least m_i out of any k_i consecutive instances are classified as mandatory. Thus, if all the mandatory instances of τ_i meet their deadlines, then (m_i, k_i)-firm guarantee requirement of τ_i is met even if none of the optional instances meet their respective deadlines.

Note that this classification of instances is not necessarily optimal in terms of meeting the (m_i, k_i)-firm guarantees of each task τ_i in the application. For a dynamic-priority scheduling algorithm, the problem of optimal classification of instances is NP-hard [17]. Since the proposed approach relies on a static-priority scheduling algorithm, this NP-hardness result does not directly apply. The problem of optimally classifying the instances in the context of static-priority scheduling is still open. Also note that other solutions for meeting the (m, k)-firm guarantee requirements have been proposed in literature [18], [19]. These solutions dynamically decide which instance of a task will miss its deadline to better schedule aperiodic tasks. Although such a dynamic determination is better from scheduling point of view, especially for aperiodic tasks, it is not suitable for providing guarantees on the behavior of the control loop. Therefore, in this paper, we assume that the instances have been statically classified as mandatory or optional as described above.

To guarantee the deadlines of mandatory instances of τ_i, the priority assignment is based on the rate monotonic policy [3]. That is, since $T_1 \leq T_2 \leq \cdots \leq T_N$, the mandatory instances of τ_i are assigned the ith highest priority level. In the description of the **Priority_Assign_Process**, the priority levels are numbered $1, 2, \ldots, N+1$, with 1 as highest priority, followed by 2, 3, and so on until $N+1$. Priority level $N+1$ is used for the optional instances.

Algorithm Sched_mkfirm

/* Assume $T_1 \leq T_2 \leq \cdots \leq T_N$ */
Concurrently execute **Service_Process** and **Priority_Assign_Process**

Service_Process

Always execute the highest priority task instance in the Wait queue;

Priority_Assign_Process

Repeat
When a^{th} instance of τ_i is activated at time aT_i, $1 \leq i \leq N$, $a \in Z_+$ **do**
  if $a = \left\lfloor \left\lceil \frac{am_i}{k_i} \right\rceil \cdot \frac{k_i}{m_i} \right\rfloor$
    classify the instance as mandatory;
    assign priority i;
  else
    classify the instance as optional;
    assign priority $N+1$;
  endif
  place instance in *Wait* queue;
endwhen

Fig. 4. Servicing tasks with (m, k)-firm guarantee requirements.

Example. Consider a real-time control application with three periodic tasks τ_1, τ_2, and τ_3 with

$$\begin{array}{llll} C_1 = 1 & T_1 = 3 & m_1 = 1 & k_1 = 1, \\ C_2 = 2 & T_2 = 4 & m_2 = 2 & k_2 = 3, \quad \text{and} \\ C_3 = 3 & T_3 = 12 & m_3 = 3 & k_3 = 5. \end{array}$$

Note that, since the total utilization $\frac{1}{3} + \frac{2}{4} + \frac{3}{12} > 1$, no scheduling policy can guarantee the deadlines of all instances. However, by using Algorithm **Sched_mkfirm** one can satisfy the 11, 23, and 35 guarantee requirements of τ_1, τ_2, and τ_3, respectively. **Priority_Assign_Process** classifies all instances of τ_1 as mandatory because it has an 11 guarantee requirement which is equivalent to a hard deadline requirement. For τ_2, one out of every three instances are classified as optional, starting with the instance activated at time 8. For τ_3, instances with activation times $0, 12, 36, 60, 72, 96, \ldots$ are classified as mandatory, whereas those with activation times $24, 48, 84, 108, \ldots$ are classified as optionals. For example, consider the instances activated at times 24 and 36. Since $24 = 2 \times 12$ and $2 \neq \left\lfloor \left\lceil \frac{2\times 3}{5} \right\rceil \frac{5}{3} \right\rfloor$, the instance activated at time 24 is classified as optional, whereas the instance activated at time 36 is classified as mandatory because $36 = 3 \times 12$ and $3 = \left\lfloor \left\lceil \frac{3\times 3}{5} \right\rceil \frac{5}{3} \right\rfloor$. The resulting schedule of mandatory instances is a repetition of the subschedule shown in Fig. 5. In particular, observe that among the first five instances from τ_3, the ones activated at 0, 12, and 36 complete their execution prior to their respective deadlines in the schedule shown in Fig. 5. The instances

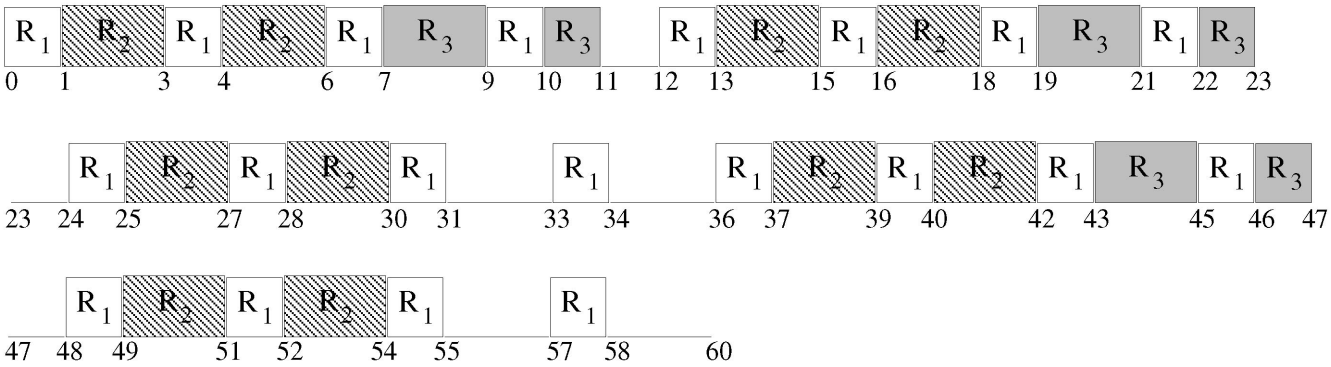

Fig. 5. Schedule of mandatory instances of the tasks in Example 1.

activated at 24 and 48 are not guaranteed to receive service. They may receive service at the idle times in the schedule of Fig. 5.

4.1 Schedulability Analysis

In this subsection, we formally prove that if all mandatory instances of τ_i meet their deadlines, then the (m,k)-firm guarantee requirement of τ_i is satisfied. We also derive sufficient conditions on the sets of periodic tasks whose respective (m,k)-firm guarantee requirement can be satisfied by Algorithm **Sched_mkfirm**. These two results are respectively proven in Theorems 1 and 3. Lemmas 1-3 are needed in the proof of Theorem 1, whereas Lemma 4 and Theorem 2 are used in the proof of Theorem 3.

Lemma 1. *For each task τ_i, the instance activated at time aT_i, $a \in Z_+$, is classified as mandatory if and only if there exists a nonnegative integer l such that $a = \left\lfloor l \cdot \frac{k_i}{m_i} \right\rfloor$.*

Proof. (*If part*) Suppose the instance of τ_i activated at time aT_i is classified as mandatory. Then, from Algorithm **Sched_mkfirm**,

$$a = \left\lfloor \left\lceil \frac{am_i}{k_i} \right\rceil \cdot \frac{k_i}{m_i} \right\rfloor.$$

Therefore, there exists a nonnegative integer $l = \left\lceil \frac{a \cdot m_i}{k_i} \right\rceil$ such that $a = \left\lfloor l \cdot \frac{k_i}{m_i} \right\rfloor$.

(*Only if part*) Consider an instance of τ_i activated at time aT_i and suppose that there exists a nonnegative integer l such that

$$a = \left\lfloor l \cdot \frac{k_i}{m_i} \right\rfloor.$$

Therefore, $\frac{l \cdot k_i}{m_i} - 1 < a \leq \frac{l \cdot k_i}{m_i}$. Since $m_i \leq k_i$, by rearranging the terms we get, $\frac{a \cdot m_i}{k_i} \leq l < \frac{a \cdot m_i}{k_i} + 1$. Since l is an integer, the above relation implies that

$$l = \left\lceil \frac{a \cdot m_i}{k_i} \right\rceil.$$

That is,

$$a = \left\lfloor \left\lceil \frac{am_i}{k_i} \right\rceil \cdot \frac{k_i}{m_i} \right\rfloor.$$

□

Lemma 2. *For each $1 \leq i \leq N$, the **Priority_Assign_Process** classifies at least m_i out of the first k_i instances of τ_i as mandatory.*

Proof. From Lemma 1, instances of τ_i are classified as mandatory if and only if their activation times are of the form

$$\left\lfloor l \cdot \frac{k_i}{m_i} \right\rfloor T_i$$

for some $l \in Z_+$. Among the first k_i instances of τ_i, at least the instances whose activation times are in the set

$$\left\{ \left\lfloor l \cdot \frac{k_i}{m_i} \right\rfloor T_i : 0 \leq l \leq m_i - 1 \right\}$$

are classified as mandatory. Since $k_i \geq m_i$, there are exactly m_i elements in this set. Hence, the lemma. □

Lemma 3. *For each $1 \leq i \leq N$, the **Priority_Assign_Process** classifies the instance of τ_i activated at time $(aT_i + k_iT_i)$, $a \in Z_+$, as mandatory if and only if the instance of τ_i activated at time aT_i is also classified as mandatory.*

Proof. (*If case*) If the instance activated at time $(aT_i + k_iT_i)$ is classified as mandatory, then, from Lemma 1, there exists an $l \in Z_+$ such that

$$\left\lfloor l \cdot \frac{k_i}{m_i} \right\rfloor = a + k_i.$$

Since $a + k_i \geq k_i$, we know that $l \geq m_i$. Also,

$$\left\lfloor \frac{(l - m_i)k_i}{m_i} \right\rfloor = \left\lfloor l \cdot \frac{k_i}{m_i} \right\rfloor - k_i = a.$$

That is, there exists a nonnegative integer $l' = l - m_i$ such that

$$\left\lfloor \frac{l'k_i}{m_i} \right\rfloor = a.$$

Therefore, the instance activated at time aT_i is also classified as mandatory.

(*Only if case*) If the instance activated at time aT_i is classified as mandatory, then there exists a $l \in Z_+$ such that

$$\left\lfloor l \cdot \frac{k_i}{m_i} \right\rfloor = a.$$

Thus,

$$\left\lfloor \frac{(l+m_i)k_i}{m_i} \right\rfloor = \left\lfloor l \cdot \frac{k_i}{m_i} \right\rfloor + k_i = a + k_i.$$

That is, there exists a nonnegative integer $l' = l + m_i$ such that $\left\lfloor \frac{l'k_i}{m_i} \right\rfloor = a + k_i$. Therefore, the instance activated at time $aT_i + k_iT_i$ is also classified as mandatory. Hence, the lemma. □

Theorem 1. *If all mandatory instances from τ_i meet their deadlines, then the (m_i, k_i)-firm guarantee requirement of τ_i is satisfied.*

Proof. From Lemma 2, the theorem holds for the first k_i instances. From Lemma 3, it follows that the classification of mandatory instances is periodic with period k_i. Thus, in any window of k_i consecutive instances of τ_i, at least m_i instances are classified as mandatory. If all these mandatory instances meet their respective deadlines, then at least m_i instances among any k_i consecutive instances of τ_i will meet their deadlines. Hence, the theorem. □

In the rest of this section, we derive sufficient conditions under which all mandatory instances from τ_i are guaranteed to meet their deadlines.

Definition 1. *The* response time *of a task instance is its service completion time minus its activation time.*

Definition 2. *A* critical instant *for a task is defined to be an instant at which a mandatory instance of the task will have the largest response time.*

Note that this definition is slightly different from that in [3] because we are only interested in deterministic guarantee of the deadlines of the mandatory instances.

Lemma 4. *In Algorithm* ***Sched_mkfirm****, the number of mandatory instances of τ_i in the interval $[aT_i, aT_i + b)$, $a \in Z_+$, $b \geq 0$, is maximum for $a = 0$ for any given b.*

Proof. In the interval $[aT_i, aT_i + b)$, $a \in Z_+$, $b \geq 0$, **Priority_Assign_Process** classifies the instances with activation times

$$\left\{ \left\lfloor l \cdot \frac{k_i}{m_i} \right\rfloor : l \in Z_+, aT_i \leq \left\lfloor l \cdot \frac{k_i}{m_i} \right\rfloor T_i < aT_i + b \right\}$$

as mandatory (see Lemma 1). Therefore, the number of mandatory instances in the interval $[0, b)$ for any given b is

$$\left| \left\{ \left\lfloor l \cdot \frac{k_i}{m_i} \right\rfloor : l \in Z_+, 0 \leq \left\lfloor l \cdot \frac{k_i}{m_i} \right\rfloor T_i < b \right\} \right|.$$

Hence, the lemma follows if

$$\left| \left\{ \left\lfloor l \cdot \frac{k_i}{m_i} \right\rfloor : l \in Z_+, 0 \leq \left\lfloor l \cdot \frac{k_i}{m_i} \right\rfloor T_i < b \right\} \right| \geq \left| \left\{ \left\lfloor l \cdot \frac{k_i}{m_i} \right\rfloor : l \in Z_+, aT_i \leq \left\lfloor l \cdot \frac{k_i}{m_i} \right\rfloor T_i < aT_i + b \right\} \right|,$$

where $|\cdot|$ denotes the cardinality of the corresponding set.

The lemma follows because

$$\begin{aligned}
&\left| \left\{ \left\lfloor l \cdot \frac{k_i}{m_i} \right\rfloor : l \in Z_+, aT_i \leq \left\lfloor l \cdot \frac{k_i}{m_i} \right\rfloor T_i < aT_i + b \right\} \right| \\
&= \left| \left\{ \left\lfloor \frac{lk_i}{m_i} \right\rfloor : l \in Z_+, \left\lceil \frac{m_i a}{k_i} \right\rceil \leq l \leq \lfloor (a+b) m_i k_i \rfloor + 1 \right\} \right| \\
&= \left\lfloor \frac{(a+b)m_i}{k_i} \right\rfloor - \left\lceil \frac{m_i a}{k_i} \right\rceil + 2 \\
&= \left\lfloor \frac{m_i a}{k_i} + \frac{bm_i}{k_i} \right\rfloor - \left\lceil \frac{m_i a}{k_i} \right\rceil + 2 \\
&\leq \left\lfloor \frac{bm_i}{k_i} \right\rfloor + \left\lceil \frac{m_i a}{k_i} \right\rceil - \left\lceil \frac{m_i a}{k_i} \right\rceil + 2 \\
&= \left\lfloor \frac{bm_i}{k_i} \right\rfloor + 2 \\
&= \left| \left\{ \left\lfloor l \cdot \frac{k_i}{m_i} \right\rfloor : l \in Z_+, 0 \leq \left\lfloor l \cdot \frac{k_i}{m_i} \right\rfloor < b \right\} \right|.
\end{aligned}$$

□

Theorem 2. *In Algorithm* ***Sched_mkfirm****, time 0 is a critical instant for task τ_i, $1 \leq i \leq N$.*

Proof At time 0, an instance of every task in the application is activated. Furthermore, all these instances are classified as mandatory by **Priority_Assign_Process**. Among these instances, the one from τ_i is assigned priority i and will therefore be serviced before all instances from τ_j, $j > i$. The response time of this instance from τ_i is therefore not determined by the instances from the tasks τ_j, $j > i$.

Consider a task τ_j, $j < i$. From Lemma 4, we know that among intervals of the form $[aT_j, aT_j + T_i]$, $a \in Z_+$, the maximum number of mandatory instances of τ_j occur in the interval $[0, T_i]$. Since this observation is true for any $j < i$, the maximum number of mandatory instances from other tasks which have to be serviced in any interval of length T_i will occur in the interval $[0, T_i]$. Therefore, among all instances of τ_i, the one activated at time 0 will have the largest response time. In other words, time 0 is a critical instant for task τ_i. □

Theorem 3. *Given $\tau_1, \ldots, \tau_N$ such that $T_1 < T_2 < \cdots < T_N$. Let*

$$R_{ij} = \left\{ \left\lfloor l \cdot \frac{k_i}{m_i} \right\rfloor T_j : \left\lfloor l \cdot \frac{k_i}{m_i} \right\rfloor T_j < T_i, l \in Z_+ \right\}$$

$$R_i = \bigcup_{j=1}^{i-1} R_{ij}$$

$$n_j(t) = \left\lceil \frac{m_j}{k_j} \left\lceil \frac{t}{T_j} \right\rceil \right\rceil$$

$$W_i(t) = C_i + \sum_{j=1}^{i-1} n_j(t) \cdot C_j.$$

*If $\min_{t \in R_i} W_i(t)/t \leq 1$ for all $1 \leq i \leq N$, then Algorithm **Sched_mkfirm** meets the (m_i, k_i)-firm guarantee requirement of each task τ_i.*

Proof. Consider the first instance of a typical task τ_i. It has a deadline of T_i. Set R_{ij} contains the activation times of τ_j's mandatory instances which are less than T_i. In Algorithm **Sched_mkfirm**, if $T_j < T_i$, then mandatory instances of τ_j have higher priority than mandatory instances of τ_i. Therefore, set R_i contains activation times, less than T_i, of mandatory instances with priority higher than the first instance of τ_i. From Lemma 1 and some algebra, the term $n_j(t)$ is the number of mandatory instances of τ_j whose activation times are less than t. $W_i(t)$ is the sum of the computation time of the first instance of τ_i and the computation times of all mandatory instances with higher priority than τ_i whose activation time is less than t. Therefore, if $W_i(t)/t < 1$, then the first instance of τ_i will be completed by time t.

Hence, if $\min_{t \in R_i} W_i(t)/t \leq 1$, then the first instance of τ_i will complete prior to its deadline. Since the first instance was activated at time 0 and 0 is a critical instant for τ_i, the (m_i, k_i)-firm guarantee requirement of τ_i is satisfied by Algorithm **Sched_mkfirm**. The theorem follows if this result holds for all i. □

Note that the proof of the above theorem is very similar to that of Theorem 2 in [2], where Lehoczky et al. derive the exact characterization of the Rate Monotonic Scheduling algorithm.

4.2 Extensions

We first relax Assumption A1 and consider tasks which interact with each other through shared resources regulated by semaphores. In this case, one can use techniques such as the Priority Ceiling Protocol [20] to bound the amount of priority inversion experienced by a mandatory instance. Let B_i denote the maximum amount of time a mandatory instance from τ_i can be blocked by an instance from a lower priority task τ_j. To account for this blocking time, $W_i(t)$ in Theorem 3 can be defined as $W_i(t) = C_i + B_i + \sum_{j=1}^{i-1} n_j(t) \cdot C_j$ and the (m_i, k_i)-firm guarantee requirement of each task τ_i is satisfied if $\min_{t \in R_i} W_i(t)/t \leq 1$ for all $1 \leq i \leq N$.

If the application contains aperiodic tasks in addition to periodic tasks, then one can extend the proposed approach using various solutions proposed in literature. For example, one can use a sporadic server to service the aperiodic tasks while treating the periodic tasks as described in this paper. By treating the sporadic server as a periodic task with (1, 1)-firm guarantee, the schedulability test of Theorem 3 can still be used to verify that all mandatory instances of periodic tasks meet their deadlines.

5 Optimal Control Law under (m,k)-Firm Guarantee

Recall that each task is responsible for performing the control law computations of one subsystem in the real-time control application. Also, from Assumption A4, we know the subsystem is linear and time-invariant. The customary assumption in control systems design is that the control law is updated at regular intervals. This assumption, however, is not true in this case because the optional instances of the periodic task are not guaranteed to complete in Algorithm **Sched_mkfirm**. When an instance does not complete, the control input to the subsystem retains its previous value, which may not necessarily be optimal. As a result, there is a deterioration in the performance of the subsystem. To minimize the amount of deterioration, we modify the control law implemented by the periodic task in order to compensate for the missed updates by the optional instances.

To describe the methodology for modifying the control law, we focus on a typical periodic task τ_i in the application. We assume that the subsystem controlled by τ_i can be modeled as

$$\begin{aligned} x(t+1) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) \qquad \text{for all } t \in Z_+ \end{aligned} \tag{5.1}$$

where $x(t)$, $y(t)$, and $u(t)$ are the state, output, and the control input to the subsystem at time $t \cdot T_i$, and A, B, and C are constant matrices of appropriate dimensions. We further assume that the objective of the control law is to minimize the error between subsystem output $y(t)$ and a desired output r. As in the previous section, we assume that τ_i has been provided with an (m_i, k_i)-firm guarantee using Algorithm **Sched_mkfirm**. Since the discussion below focuses on only one task, we drop the subscript i in the rest of this section.

Furthermore, for ease of presentation, we assume that $m \geq k/2$. If we do not make this assumption, some of the expressions derived in this section will be more complicated, which in turn makes it difficult to understand the basic idea of the proposed approach. Moreover, this assumption is likely to hold in most situations, because it is unlikely that a control system will be able to deliver satisfactory performance after more than 50 percent of the control law updates have been deleted.[2] In the special case when $m \geq k/2$, we can prove the following useful lemma about Algorithm **Sched_mkfirm**.

Lemma 5. *If $m \geq k/2$, then no two consecutive instances of τ will be classified as optionals by Algorithm **Sched_mkfirm**.*

Proof. Follows easily from Lemma 1. □

Lemma 5 means that there are no two consecutive misses of control law updates by task τ.

2. Unless the control system was initially excessively over designed.

In addition to missed control law updates due to optional instances, we must also account for one sample controller delay. The delay occurs because a mandatory instance may execute anywhere within its corresponding period. Specifically, the worst-case one sample controller delay occurs when a mandatory instance samples the inputs as soon as it is activated, but completes only just prior to its deadline. Derivation of optimal control law with controller delays was addressed by Mita in [21]. Mita, however, did not account for missed control law updates due to optionals. In this section, we adapt the results in [21] to deal with missed control law updates.

First, define M_τ to be the set of all mandatory instances of τ. From Lemma 1, $M_\tau = \left\{ \left\lfloor \frac{l \cdot k}{m} \right\rfloor : l \in Z_+ \right\}$. Since control law updates are guaranteed to be performed only by the mandatory instances, we assume based on the results in [21] that the optimal control law has the form

$$\begin{aligned} u(t+1) &= \begin{cases} -H_t x(t) - M_t u(t) + z(t) & \text{if } t \in M_\tau \\ u(t) & \text{otherwise} \end{cases} \\ z(t+1) &= \begin{cases} z(t) + K_t(r - y(t)) & \text{if } t \in M_\tau \\ z(t) & \text{otherwise,} \end{cases} \end{aligned} \tag{5.2}$$

where H_t, M_t, and K_t are matrices of appropriate dimensions. Observe that the control input $u(t+1)$ depends only on the samples from time t, thereby accounting for the controller delay. Also note that $z(t)$ is an integrator of the error between the desired output r and system output y. This integrator is included to deal with a possible nonzero value of r. Furthermore, note that the integrator also operates on a one sample delay because it is also implemented by the periodic task. In effect, this means that an error between r and y manifests in u with at least two sample delay.

Let $\widehat{x}(t) = [\, x(t) \quad u(t) \quad v(t) \,]'$. Incorporating (5.2) in (5.1), the subsystem behavior can be modeled as

$$\widehat{x}(t+1) = \begin{cases} \begin{bmatrix} A & B & 0 \\ 0 & 0 & I \\ W_{1,t} & W_{2,t} & W_{3,t} \end{bmatrix} \widehat{x}(t) \\ \qquad + [\, 0 \quad 0 \quad K_t \,]' r & \text{if } t \in M_\tau \\ \begin{bmatrix} A & B & 0 \\ 0 & 0 & I \\ 0 & I & 0 \end{bmatrix} \widehat{x}(t) & \text{otherwise,} \end{cases} \tag{5.3}$$

where

$$\begin{aligned} W_{1,t} &= -H_t(A - I) - K_t C, \\ W_{2,t} &= -H_t B + M_t, \\ W_{3,t} &= -M_t + I, \end{aligned}$$

and

$$v(t) = u(t+1)$$

is an additional state variable representing the next value of control input estimated from the current inputs. In the above equation and in the rest of this paper, I stands for the identity matrix. The dimension of the matrix is not explicitly specified so as to not complicate the notation. We assume that the dimensions are clear from the context.

If the system governed by (5.3) is stable,[3] then in steady-state,

$$\begin{bmatrix} \bar{x} \\ \bar{u} \\ \bar{v} \end{bmatrix} = \begin{bmatrix} A & B & 0 \\ 0 & 0 & I \\ W_{1,t} & W_{2,t} & W_{3,t} \end{bmatrix} \cdot \begin{bmatrix} \bar{x} \\ \bar{u} \\ \bar{v} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ K_t \end{bmatrix} r. \tag{5.4}$$

Introduce variables $dx(t) = x(t) - \bar{x}$, $du(t) = u(t) - \bar{u}$, $dv(t) = v(t) - \bar{v}$, and $\widehat{dx}(t) = [\, dx(t) \quad du(t) \quad dv(t) \,]'$. From (5.3) and (5.4) we get

$$\widehat{dx}(t) = \begin{bmatrix} A & B & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{bmatrix} \widehat{dx}(t) + \begin{bmatrix} 0 \\ 0 \\ I \end{bmatrix} w(t), \tag{5.5}$$

where

$$w(t) = \begin{cases} [\, W_{1,t} \quad W_{2,t} \quad W_{3,t} \,] \widehat{dx}(t) & \text{if } t \in M_\tau \\ [\, 0 \quad I \quad 0 \,] \widehat{dx}(t) & \text{otherwise.} \end{cases} \tag{5.6}$$

Derivation of optimal control law involves finding $w(t)$ for all $t \in Z_+$. Since $w(t) = du(t-1) = w(t-1)$ when $t \in M_\tau$, we need to determine $w(t)$ for only $t \in M_\tau$. We therefore focus on the behavior of the system only on $t \in M_\tau$. Let $t \oplus 1$ be the smallest element larger than t in M_τ, i.e., let

$$t \oplus 1 = \begin{cases} t+1 & t+1 \in M_\tau \\ t+2 & \text{otherwise.} \end{cases}$$

Also define

$$\mathcal{A}_t = \begin{cases} \Phi & \text{if } t+1 \in M_\tau \\ \Phi^2 & \text{otherwise and} \end{cases}$$

$$\mathcal{B}_t = \begin{cases} \Psi & \text{if } t+1 \in M_\tau \\ \Phi\Psi + \Psi & \text{otherwise and} \end{cases}$$

where

$$\Phi = \begin{bmatrix} A & B & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{bmatrix}$$

and

$$\Psi = \begin{bmatrix} 0 \\ 0 \\ I \end{bmatrix}.$$

Then,

$$\widehat{dx}(t \oplus 1) = \mathcal{A}_t \cdot \widehat{dx}(t) + \mathcal{B}_t \cdot w(t) \quad \text{for all } t \in M_\tau. \tag{5.7}$$

Equation (5.7) corresponds to a linear time-varying system whose optimal control law can be determined if the performance index is a traditional LQR objective

$$J = \sum_{t=1}^{h} \widehat{dx}'(t) \cdot Q \cdot \widehat{dx}(t) + w'(t) \cdot R \cdot w(t), \tag{5.8}$$

3. One can formally state the conditions for stability based on the matrices A, B, and C, and the values of m and k. Due to restrictions on the length of this paper, we assume here that the system is stable.

where h is the time horizon of the optimization and Q and R are constant matrices of appropriate dimensions. In particular, from [14], we know that the optimal control law has the form

$$w(t) = -F_t \cdot \widehat{dx}(t) \quad \text{for all } t \in M_\tau, \tag{5.9}$$

where F_t is the solution of the Riccati equation

$$F_t = (\mathcal{B}_t' * S_t * \mathcal{B}_t + R)^{-1}\mathcal{B}_t' S_t \mathcal{A}_t, \tag{5.10}$$

and where S_t is obtained from the following recursive equation.

$$\begin{aligned} S_h &= Q \\ S_{t-1} &= \mathcal{A}_t'(S_t - S_t\mathcal{B}_t(\mathcal{B}_t' * S_t * \mathcal{B}_t + R)^{-1}\mathcal{B}_t' S_t)\mathcal{A}_t \\ &\quad + Q,\ 1 \le t < h. \end{aligned}$$

Since the system in (5.7) is time varying, the solution of the Riccati equation does not converge to a unique value. However, note that from **Priority_Assign_Process**, $t \in M_\tau$ implies $t + k \in M_\tau$. Hence, $\mathcal{A}_t = \mathcal{A}_{t+k}$ and $\mathcal{B}_t = \mathcal{B}_{t+k}$ and the system in (5.7) is periodic with period k. As a result, the solution of the Riccati is also periodic with period k [22]. That is, if the time horizon h is large

$$F_t = F_{t+k} \quad \text{for all } t \in M_\tau$$

Therefore, from the following m values of F_t, $t \in \left\{\left\lfloor \frac{l \cdot k}{m} \right\rfloor : 0 \le l < m\right\}$, we can easily compute F_t for any $t \in M_\tau$.

The solution in (5.10) must match (5.9) for $t \in M_\tau$. Matching the two and with some algebra

$$\begin{aligned} M_t &= F_t * \mathcal{B}_t + I \\ [\,H_t \quad K_t\,] &= \left[\,F_t\mathcal{A}_t^2 \quad F_t\mathcal{A}_t\mathcal{B}_t + F_t\mathcal{B}_t + I\,\right]E_t, \end{aligned} \tag{5.11}$$

where

$$E_t = \begin{bmatrix} \mathcal{A}_t - I & \mathcal{B}_t \\ C & 0 \end{bmatrix}^{-1}.$$

In summary, the optimal control law which minimizes the performance index in (5.8) is given by (5.2), where H_t, M_t, and K_t are as shown above in (5.11). Note that in the special case, when all the instances of the periodic task are guaranteed to meet their deadlines, this result degenerates to the optimal control law in [21].

6 Numerical Example

In this section, we present a numerical example to illustrate the derivation of the optimal control law under (m,k)-firm guarantee. Consider the cruise-control system in Fig. 2. The discretized state-variable representation of the combined system including throttle, fuel dynamics, and vehicle dynamics at sampling rate of 1 second is

$$\begin{aligned} x(t+1) = &\begin{bmatrix} 0.869 & -0.041 & 0.000 \\ 1.000 & 0.000 & 0.000 \\ 0.000 & 1.000 & 0.000 \end{bmatrix} \cdot x(t) + \begin{bmatrix} 1.0 \\ 0.0 \\ 0.0 \end{bmatrix} \cdot u(t) \\ y(t+1) = &\,[\,0.110 \quad 0.062 \quad 0.0004\,] \cdot x(t). \end{aligned}$$

If all instances of the periodic task controlling this subsystem are guaranteed to meet their deadlines, then the optimal control law which minimizes the performance index in (5.8) is

$$\begin{aligned} u(t+1) = &-[\,-2.226 \quad -0.226 \quad -0.002\,] \cdot x(t) \\ &-[\,-1.673\,]u(t) + z(t) \\ z(t+1) = &\, z(t) + 4.472(r - y(t)). \end{aligned}$$

This solution was obtained by solving the Riccati equation in (5.10) with $m = k = 1$ and computing H, M, and K using (5.11). Note that, in this case, the Riccati equation will converge to a unique F if the time horizon is reasonably large. Also, the optimal control law in this case corresponds exactly to optimal control law based directly on the results in [21]. The cruise-control task can use this control law when the system is not overloaded. With this control law, the response of the cruise-control system to a step change in the desired speed from 45 mph to 55 mph is shown as a solid line with $\circ$ in Fig. 6. Observe that the system converges to steady-state fairly rapidly.

Now suppose that the cruise-control task is scheduled by Algorithm **Sched_mkfirm** with 35 guarantee. In this case, $M_\tau = \{0, 1, 3, 5, 6, 8, \ldots\}$ and the solution of the Riccati equation in (5.10) gives

$$\begin{aligned} F_0 &= [\,-0.288 \quad 0.014 \quad 0.000 \quad -0.352 \quad -0.429\,] \\ F_1 &= [\,-0.237 \quad 0.012 \quad 0.000 \quad -0.290 \quad -0.786\,] \\ F_3 &= [\,-0.288 \quad 0.014 \quad 0.000 \quad -0.352 \quad -0.952\,]. \end{aligned}$$

Substituting the above values for F_0, F_1, and F_3 in (5.11) we get

$$\begin{aligned} &H_0 = [\,-1.781 \quad -0.152 \quad -0.001\,], \\ &H_1 = [\,-2.076 \quad -0.138 \quad -0.001\,], \\ &H_3 = [\,-2.304 \quad -0.163 \quad -0.0014\,], \\ &M_0 = [\,-1.429\,], M_1 = [\,-1.786\,], M_3 = [\,-1.952\,], \\ &K_0 = [\,3.371\,], K_1 = [\,3.385\,], \text{ and } K_3 = [\,3.891\,]. \end{aligned}$$

Therefore, from (5.2), the optimal control law under $(3,5)$-firm guarantee is

$$u(t+1) = \begin{cases} -H_0 x(t) - M_0 u(t) + z(t) & \text{if } t \bmod 5 = 0 \\ -H_1 x(t) - M_1 u(t) + z(t) & \text{if } t \bmod 5 = 1 \\ u(t) & \text{if } t \bmod 5 = 2 \\ -H_3 x(t) - M_3 u(t) + z(t) & \text{if } t \bmod 5 = 3 \\ u(t) & \text{if } t \bmod 5 = 4, \end{cases}$$

where

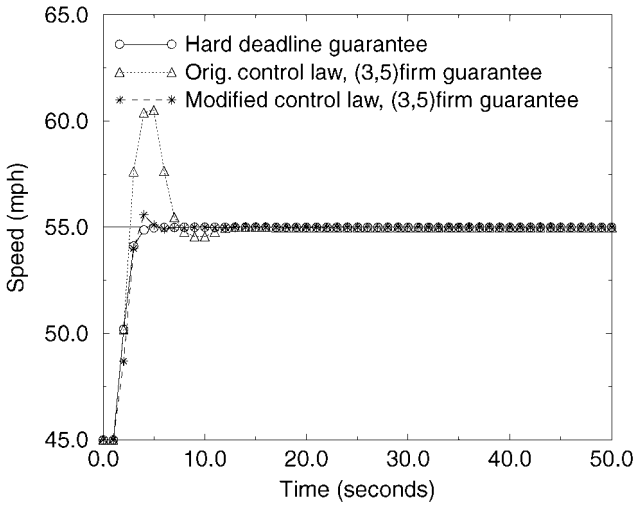

Fig. 6. Evaluation of the effectiveness of the proposed approach.

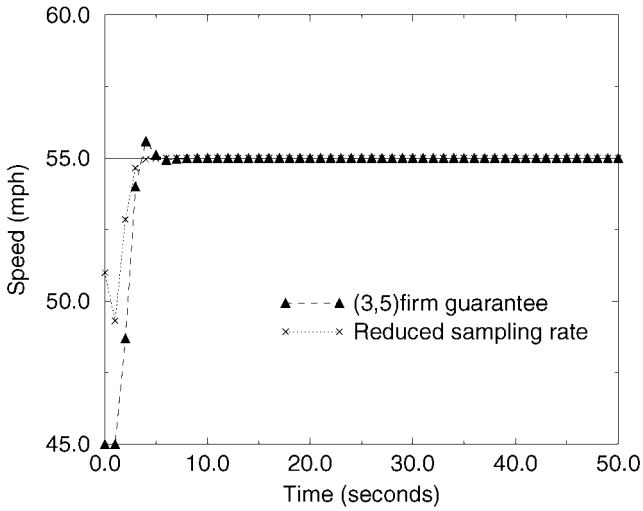

Fig. 7. Comparison for proposed approach and reducing sampling rate approach.

$$z(t+1) = \begin{cases} z(t) + K_0(r - y(t)) & \text{if } t \bmod 5 = 0 \\ z(t) + K_1(r - y(t)) & \text{if } t \bmod 5 = 1 \\ z(t) & \text{if } t \bmod 5 = 2 \\ z(t) + K_3(r - y(t)) & \text{if } t \bmod 5 = 3 \\ z(t) & \text{if } t \bmod 5 = 4. \end{cases}$$

With the above control law, the response of the cruise-control system to a step change in the desired speed from 45 mph to 55 mph is shown as a dashed line with $*$ in Fig. 6. Observe that the system behavior is very close to the behavior obtained when all deadlines are guaranteed, even though the effective utilization has been reduced to 70 percent. This reduction in effective utilization helps alleviate the overload problem.

To evaluate the benefit of modifying the control law, the figure also contains the response of the cruise-control system to the step change in the desired speed from 45 mph to 55 mph when Algorithm **Sched_mkfirm** provides $(3, 5)$-firm guarantee to the task, but the control law is kept the same as for the hard deadline guarantee. Note that the response is substantially worse in this case. This demonstrates the need for modifying the control law when the system provides only a (m, k)-firm guarantee.

6.1 Comparison to an Approach with Increasing Sampling Period

As stated earlier in 3, another way of reducing the effective utilization of a periodic task during overload is to increase its period so that the system can guarantee the deadlines of all its instances. For example, to reduce the utilization of the cruise-control task to 70 percent, we can change sampling period to $5/3$ seconds instead of the 1 second used above. This also tends to degrade the performance of the system. However, one can modify the optimal control law to account for the reduced sampling rate with 11 guarantee.

Fig. 7 compares the step response of the cruise-control system for our approach with $(3, 5)$-firm guarantee and the approach with equivalently reduced sampling rate. Note that the response with $(3, 5)$-firm guarantee is slightly better. This is not always true, but the responses are comparable. However, as noted earlier, using (m, k)-firm guarantee method is better because: 1) changing the sampling rate alters the dynamics of the subsystem, 2) changing the sampling rate in one subsystem may necessitate a change in the sampling rate of other related subsystems in the application, and 3) since optional instances may get service in the (m, k)-firm guarantee approach, the system has a much better response on the average.

7 Conclusions

In this paper, we proposed a technique for management of processor overload in real-time control applications. The technique selectively discards task instances to reduce the effective utilization of each task. If no further action is taken, the missed control updates due to the discarded instances will cause substantial degradation in the performance of the control system. To minimize the degradation, the paper also proposes a methodology for modifying the control law implemented by the task to account for the updates scheduled to be performed by the discarded instances. The combined scheme alleviates the overload problem without any significant degradation in the performance of the system.

In this paper, we essentially ignored the optional instances of the periodic tasks. Better scheduling policies can be used to service more of the optional instances without violating the deadlines of mandatory instances. We can also utilize the optional instances to improve the performance of the system in the average case. If some optional instances are serviced, we must account for them in deriving the optimal control law for the mandatory and the completed optional instances. Research work on addressing these issues is in progress.

Acknowledgments

The author would like to thank Profs. R. Barmish and C. DeMarco for their insightful criticisms on the ideas in this paper. This work is supported in part by the U.S. National

Science Foundation under Grant MIP-9526761. A shorter version of this paper appeared in the *Proceedings of the Fault-Tolerant Computing Symposium*, pp. 132-141, 1997.

REFERENCES

[1] H. Chetto and M. Chetto, "Some Results of the Earliest Deadline Scheduling Algorithm," *IEEE Trans. Software Eng.,* vol. 15, no. 10, pp. 1,261–1,269, Oct. 1989.
[2] J. Lehoczky, L. Sha, and Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," *Proc. Real-Time Systems Symp.,* pp. 166–171, Dec. 1989.
[3] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multi-Programming in a Hard Real-Time Environment," *J. ACM,* vol. 20, pp. 46–61, Jan. 1973.
[4] B. Sprunt, L. Sha, and J. Lehoczky, "Aperiodic Task Scheduling for Hard Real-Time Systems," *Real-Time Systems,* vol. 1, pp. 27–60, June 1989.
[5] J. Xu and D.L. Parnas, "Scheduling Processes with Release Times, Deadlines, Precedence and Exclusion Relations," *IEEE Trans. Software Eng.,* vol. 16, no. 3, pp. 360–369, Mar. 1990.
[6] K. Jeffay, D.F. Stanat, and C.U. Martel, "On Non-Preemptive Scheduling of Periodic and Sporadic Tasks," *Proc. Real-Time Systems Symp.,* pp. 129–139, Dec. 1991.
[7] M.R. Garey and D.S. Johnson, *Computer and Intractability: A Guide to the Theory of NP-Completeness.* San Francisco: Freeman, 1979.
[8] K. Ramamritham, "Allocation and Scheduling of Complex Periodic Tasks," *Proc. Distributed Computing Systems,* pp. 108–115, May 1990.
[9] W. Zhao, K. Ramamritham, and J.A. Stankovic, "Scheduling Tasks with Resource Requirements in Hard Real-Time Systems," *IEEE Trans. Software Eng.,* vol. 13, no. 5, pp. 564–577, May 1987.
[10] *Fault-Tolerant Computer System Design,* D.K. Pradhan, ed. Prentice Hall, 1996.
[11] R.M. Kieckhafer, C.J. Walter, A.M. Finn, and P.M. Thambidurai, "The MAFT Architecture for Distributed Fault Tolerance," *IEEE Trans. Computers,* vol. 37, no. 4, pp. 398–405, Apr. 1988.
[12] A.L. Hopkins, T.B. Smith, and J.H. Lala, "FTMP—A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft," *Proc. IEEE,* vol. 66, Oct. 1978.
[13] M. Hamdaoui and P. Ramanathan, "A Dynamic Priority Assignment Technique for Streams with (m, k)-Firm Deadlines," *IEEE Trans. Computers,* vol. 44, no. 12, pp. 1,443–1,451, Dec. 1995.
[14] G.F. Franklin, J.D. Powell, and M.L. Workman, *Digital Control of Dynamic Systems,* second ed. Addison-Wesley, 1990.
[15] J.Y. Chung, J.W.S. Liu, and K.-J. Lin, "Scheduling Periodic Jobs that Allow Imprecise Results," *IEEE Trans. Computers,* vol. 39, no. 9, pp. 1,156–1,174, Sept. 1990.
[16] J.W.S. Liu, K.-J. Lin, W.-K. Shih, A.C. Yu, J.-Y. Chung, and W. Zhao, "Algorithms for Scheduling Imprecise Computations," *Computer,* vol. 24, no. 5, pp. 58–68, May 1991.
[17] G. Koren and D. Shasha, "Skip-Over: Algorithms and Complexity for Overloaded Systems that Allow Skips," *Proc. Real-Time Systems Symp.,* pp. 110–117, Dec. 1995.
[18] G. Bernat and A. Burns, "Combining (n, m)-Hard Deadlines and Dual Priority Scheduling," *Proc. Real-Time Systems Symp.,* pp. 46–57, Dec. 1997.
[19] M. Caccamo and G. Buttazzo, "Exploiting Skips in Periodic Tasks for Enhancing Aperiodic Responsiveness," *Proc. Real-Time Systems Symp.,* pp. 330–339, Dec. 1997.
[20] L. Sha, R. Rajkumar, and J.P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Trans. Computers,* vol. 39, no. 9, pp. 1,175–1,185, Sept. 1990.
[21] T. Mita, "Optimal Digital Feedback Control Systems Counting Computation Time of Control Laws," *IEEE Trans. Automatic Control,* vol. 30, pp. 542–548, June 1985.
[22] S. Bittanti, P. Colaneri, and D. DeNicolao, "The Periodic Riccati Equation," *The Riccati Equation,* S. Bittanti, A.J. Laub, and J.C. Willems, eds., chapter 6, pp. 127–162. Springer-Verlag, 1991.

Parameswaran Ramanathan (M'89) received the BTech degree from the Indian Institute of Technology, Bombay, India, in 1984, and the MSE and PhD degrees from the University of Michigan, Ann Arbor, in 1986 and 1989, respectively. He is an associate professor in the Department of Electrical and Computer Engineering and in the Department of Computer Sciences at the University of Wisconsin, Madison. From 1984 to 1989, he was a research assistant in the Department of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor. He was an assistant professor in the Department of Electrical and Computer Engineering at the University of Wisconsin, Madison, from 1989 to 1995. From 1997-1998, he spent his sabbatical leave from the university at AT&T Labs, Whippany, New Jersey, and at Bellcore, Morristown, New Jersey. His research interests include the areas of real-time systems, high-speed networks, fault-tolerant computing, distributed systems, and parallel algorithms.