A scheduling model inspired by control theory

Sanjoy Baruah
Washington University in St. Louis
USA

Vincenzo Bonifaci
IASI-CNR
Italy

Alberto Marchetti-Spaccamela
Sapienza University of Rome
Italy

Victor Verdugo
ENS Paris, France
Universidad de Chile

ABSTRACT

Certain control computations may be modeled as periodic tasks with the correctness requirement that for each task, the fraction of jobs of the task that complete execution by their respective deadlines be no smaller than a specified value. This appears to be a correctness requirement that has not previously been studied in the real-time scheduling theory community; this paper formulates the problem and proposes some solution strategies.

CCS CONCEPTS

• **Computer systems organization** → **Real-time systems**; • **Software and its engineering** → **Scheduling**; • **Theory of computation** → *Scheduling algorithms*;

KEYWORDS

Control tasks, periodic preemptive scheduling, scheduling with dropout

ACM Reference format:
Sanjoy Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Victor Verdugo. 2017. A scheduling model inspired by control theory. In *Proceedings of RTNS '17, Grenoble, France, October 4–6, 2017,* 10 pages.
https://doi.org/10.1145/3139258.3139272

1 INTRODUCTION AND MOTIVATION

In feedback (or closed loop) control, the state of a plant is monitored, the *error* — deviation of the monitored value from a desired value – is determined, and a *control signal* that should decrease the error is computed and applied to the plant; this entire process is repeated periodically. The computation of the control signal in such control loops is often modeled as a periodic task, with each iteration of the loop represented by a job of the task. In many application systems there are multiple control loops, each responsible for controlling a different aspect of the system's behavior, that run simultaneously upon a shared platform; determining an appropriate strategy for scheduling these control computations is thus a periodic scheduling problem.

The property of bounded input bounded output (BIBO) stability is desired of certain control systems. (Loosely speaking, a control system is BIBO stable if every bounded input to the system results in a bounded output.) Branicky, Phillips, and Zhang [2] explored the effect upon stability of skipping the computation of the control signal during some iterations and instead reusing the value that was used during the previous iteration. They obtained strategies for determining a minimal fraction of control-signal computations that must be completed in order to ensure stability; in follow-up work, Majumdar, Saha, and Zamani [9] derived techniques for determining the minimal fraction of such computations that must be completed in order to additionally achieve optimal disturbance rejection performance. The work in [2, 9] suggests the following task model for the scheduling of control computations.

Task model. Each control loop is modeled as a control task τ_i that is characterized by three parameters: $\tau_i = (C_i, T_i, r_i) \in \mathbb{N} \times \mathbb{N} \times \mathbb{Q}_+$, where C_i is the WCET, T_i the period, and r_i a positive rational number ≤ 1 denoting the desired asymptotic completion rate for task τ_i ($i = 1, \ldots, n$). Such a task generates *jobs*; the k'th job generated by τ_i has a release time $(k-1) \times T_i$, an execution requirement no greater than C_i, and a deadline $k \times T_i$, for all integer $k \geq 1$.

It is *not* required that all the jobs generated by τ_i execute. Instead, let k_1, k_2 denote positive integers with $k_1 \leq k_2$, and let $\textsc{comp}_i(k_1, k_2)$ denote the number of jobs out of the k_1'th, (k_1+1)'th, (k_1+2)'th, $\cdots$, (k_2-1)'th, k_2'th jobs generated by τ_i that do complete execution by their deadlines in some schedule. Below, we define two alternative requirements for the schedule to be considered correct. The first definition, which we term the *weak* requirement (Definition 1), is a formalization of the notion considered in [2, 9]; it requires that for each task τ_i the ratio of completed jobs to the number of released jobs is asymptotically at least r_i. Formally, we require

Definition 1 (Weak requirement). *for each τ_i,*

$$\liminf_{k\to\infty} \frac{\textsc{comp}_i(1,k)}{k} \geq r_i. \tag{1}$$

Note that this is an *asymptotic* notion of correctness, which does not restrict what may happen within any finite sequence of successive jobs. In particular, it suffers from the limitation that the completed jobs of a task may "cluster" around contiguous periods. Consider, for example, a task τ_i with $r_i = 1/2$; as per the definition of the weak correctness requirement above, a schedule that skips

ACM ISBN 978-1-4503-5286-4/17/10…$15.00
https://doi.org/10.1145/3139258.3139272

every alternate job is as correct as one that skips the first million jobs and then schedules the next million.But this is likely not what the control engineer would believe; examples such as this indicate that something was likely lost in translation in [2, 9] from control theory to periodic scheduling. For this reason, in this paper we propose and also study a strenghtened notion of correctness, which we call the *strong* requirement (Definition 2) that requires that jobs of task τ_i are scheduled, roughly, every $1/r_i$ periods over all long enough intervals. Formally, we require

Definition 2 (Strong requirement). *there is a constant k^* such that, for each τ_i, for each $k \geq k^*$ and $k_0 \geq 1$,*

$$\text{comp}_i(k_0, k_0 + k - 1) \geq \lfloor kr_i \rfloor \,. \tag{2}$$

Moreover, the constant k^* should be polynomially related to the input parameters (i.e., n, the T_i's, and the r_i^{-1}'s).

Differently from the weak requirement, in the strong requirement the existence of the constant k^* guarantees an a priori bound on the number of concurrent (or neighboring) jobs that can be skipped. (For example, for the task mentioned above with $r_i = 1/2$, a schedule that skips the first million jobs and then schedules the next million would only be acceptable if $k^* = 2 \cdot 10^6$ was polynomially related to the input parameters.) Ideally we would like to find a schedule that verifies the above definition with a small value of the constant k^*; we point out here that the results we derive in Sections 4 and sec:general-period-version-3 verify the above definiton with $k^* = 1$.

This research. In this paper we address the following problem, which we call the scheduling with dropout problem: *Given a collection $\mathcal{T}$ of n control tasks of the kind described above, determine whether the collection can be scheduled correctly upon a preemptive uniprocessor.* We will consider both notions of correctness – the weak requirement of Definition 1 and the strong requirement of Definition 2.

Some obvious extensions of this open problem are also of interest, both individually and in combinations:

(1) Determine appropriate *algorithms* for scheduling collections of control tasks, and design efficient *schedulability conditions* for these scheduling algorithms.
(2) Consider the *multiprocessor* version of this problem.
(3) Consider more general task models. This could include periodic tasks with *offsets*; *sporadic* (rather than periodic) arrivals; models in which tasks are characterized by a *relative deadline* parameter in addition to WCET and period; and further generalizations.

In Section 6 we briefly discuss the extension of some of our results to multiprocessor implementations of control tasks; we postpone consideration of the remaining extensions to future work.

Relationship to prior work. The arbitration of control tasks has been studied in previous papers from the perspective of control performance [3, 10]. In the real-time systems literature, several models have previously been proposed that allow for the specification of the fact that some jobs of a task may be skipped during execution. Such models include the (m, k)-firm model [5], models that allow for the specification of a skip factor [8], and the weakly-hard task model [1]. However, it should be evident that the control tasks model that we are considering differs widely from these prior models, all of which require allowed job-drops within explicitly-specified intervals of a certain number of successive jobs. In contrast, the control tasks model only requires that the fraction of correctly-scheduled jobs be no smaller than a specified ratio asymptotically for the weak requirement, and within a polynomially-bounded, but not explicitly specified, duration for the strong requirement. This characteristic of a real-time requirement that is only required to hold over large time intervals appears to be novel and previously completely unexplored.

Organization. The remainder of this paper is organized as follows. We start out in Section 2 with some relatively straightforward results, and characterize the complexity of the problems considered. In Section 3 we discuss a special case of the problem, in which all tasks have equal periods, under the weak requirement; although this case is of limited interest (both because of the equal-periods restriction and the fact that the weak requirement appears to be a gross over-simplification of the actual requirements of control systems), this section introduces many of the ideas that are further exploited in order to develop solutions for the general (i.e., not equal-period) problem under the strong requirement — this we do in Sections 4 and 5. In Section 6 we extend some of our results to multiprocessor platforms.

2 PRELIMINARIES

A trivial sufficient schedulability condition is obtained by simply not exploiting the ability to skip jobs. In that case, it is obvious that

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \tag{3}$$

is clearly a sufficient condition for the system $\{\tau_1, \tau_2, \ldots, \tau_n\}$ to be schedulable upon a unit-speed processor. However, this condition is not necessary: it is obtained by completely ignoring the fact that for each i, a fraction $(1 - r_i)$ of the jobs of the i'th control task may be dropped without penalty.

We now discuss a necessary but not sufficient schedulability condition based on the observation that the quantity $\left(r_i \times \frac{C_i}{T_i}\right)$, which we will refer to as the *weighted utilization* of the i'th control task, denotes a tight upper bound upon the fraction of the computing capacity of a shared unit-speed processor that may need to be devoted to executing the i'th control task τ_i; hence,

$$\sum_{i=1}^{n} r_i \frac{C_i}{T_i} \leq 1 \tag{4}$$

is clearly a necessary condition for the system $\{\tau_1, \tau_2, \ldots, \tau_n\}$ to be schedulable upon a unit-speed processor. However, this condition is not sufficient: consider the system $\{\tau_1, \tau_2, \tau_3\}$ with $\tau_1 = \tau_2 = \tau_3 = (6, 10, 1/2)$. The summation evaluates to 0.9, but this instance is not schedulable since only one job can "fit" into each period while each of the three tasks desires to be scheduled during half the periods.

We observe that this example serves as a counter-example to [9, (Proposition 1)], which had claimed that Condition 4 is an exact test. Indeed, it is easily shown that there cannot be a non-trivial sufficient schedulability condition that depends only upon the sum of the weighted utilizations of the tasks.

Theorem 1. *For any constant $c > 0$, there are infeasible instances of the scheduling with dropout problem satisfying the property*

$$\sum_{i=1}^{n} r_i \frac{C_i}{T_i} \le c. \tag{5}$$

Proof: Consider the instance with the two tasks

$$\tau_1 = (T, T, 1/T), \tau_2 = (1, T, 1).$$

The weighted utilizations of these two tasks sum to $2/T$, which can be made smaller than c by choosing $T > 2/c$. However, this instance is clearly unschedulable since τ_2 needs to execute in every period, while τ_1 needs to execute in some period, and both tasks' jobs cannot be accommodated within any single period. □

We complete this section by proving that the decision problem of determining whether the scheduling with droput problem is hard even in the equi-period case when all rates are equal.

Theorem 2. *The scheduling with dropout problem assuming the strong requirement is NP-hard in the strong sense, even when $T_i = T$ for all $i = 1, 2, \dots, n$ and all rates are equal.*

Proof: We reduce from the NP-complete problem 3-Partition [4, SP15]: given a set A of $3m$ elements, a bound $T \in \mathbb{Z}_+$, and a size $s_i \in \mathbb{Z}_+$ for each $i \in A$ such that $T/4 < s_i < T/2$ and $\sum_{i\in A} s_i = mT$, decide whether A can be partitioned into m disjoint sets $A_1, A_2, \dots, A_m$ such that, for $1 \le k \le m$, $\sum_{i\in A_k} s_i = T$.

For each element $i \in A$, we create a control task $\tau_i = (C_i, T_i, r_i)$, where $C_i = s_i$, $T_i = T$, $r_i = 1/m$.

Suppose the 3-Partition instance is feasible. Then consider the following periodic schedule with period mT: during $[(k-1)T, kT]$, execute the control tasks corresponding to the elements of A_k. Over $[0, mT]$, the completion rate of a task τ_i is $1/m$, so the schedule satisfies the strong requirement.

Conversely, assume that a schedule exists such that $\text{COMP}_i(1, k) \ge \lfloor kr_i \rfloor$ for all $k \ge k^*$ and for all $i = 1, 2, \dots, n$. Let $K \ge k^*$ be a multiple of m. We make the following observations:

- Each task is completed at least K/m times during $[0, KT]$;
- Similarly, each task is completed at least $K/m+1$ times during $[0, (K+m)T]$;
- But since $\sum_{i=1}^{n}(K/m)s_i = KT$, each task is completed *exactly* K/m times during $[0, KT]$;
- Therefore, each task is completed at least once during $[KT, (K+m)T]$.

Based on the last property, we construct a solution to the 3-Partition instance: define A_k to be the set of tasks scheduled during $[(K+k-1)T, (K+k)T]$, for $k = 1, 2, \dots, m$. It is straightforward to check that this solution defines a feasible partition. □

3 SCHEDULING UNDER THE WEAK REQUIREMENT

In this section, we discuss the relationship between the scheduling with dropout problem assuming the weak requirement, and the bin-packing problem. As a warm-up, we introduce our approach by considering a simple case. Namely, we consider the case of equi-period tasks, and we propose an algorithm for generating a schedule satisfying the weak requirement for any instance $\mathcal{T} = (\tau_1, \tau_2, \dots, \tau_n)$ of the scheduling with dropout problem for which it holds that

$$\max_{i=1}^{n} \frac{C_i}{T} + \sum_{i=1}^{n} r_i \frac{C_i}{T} \le 1. \tag{6}$$

Even though the above result is of limited interest due to the above discussed limitations of the weak requirement, we elaborate upon it here since it provides a simple introduction to the techniques that are strengthened and further elaborated when we consider the strong requirement.

The *bin-packing* problem [6, 7], requires to partition a given collection of items, each of a specified size, among a given number of equi-sized bins of a specified capacity, in such a manner that the sum of the sizes of the items placed in each bin does not exceed the capacity of the bin. Bin-packing is known to be NP-hard in the strong sense; several heuristics have been proposed for solving it approximately. We focus here upon the *Worst-Fit Increasing* (WFI) heuristic, which may be defined as follows:

Definition 3 (WFI bin-packing). *Consider the items in non-decreasing order of size (i.e., smallest item first). In considering an item, place it in the bin in which it fits the "worst;" i.e., in which there is the maximum spare capacity left after the item has been added*[1]*. For notational convenience, we will assume that ties are broken in favor of the lower-indexed bin.*

The following theorem characterizes the kinds of partitioning obtained under WFI bin-packings.

Theorem 3. *Let $s_0, s_1, \dots, s_{N-1}$ denote a collection of N items sorted by non-decreasing size, that is to be bin-packed among m bins indexed $0, 1, \dots, (m-1)$. Under the WFI bin-packing heuristic, the items are placed in the bins in round-robin order starting with bin 0; i.e., for each i, $0 \le i < N$, item s_i is placed in the bin that is indexed $(i \bmod m)$.*

Proof: At any iteration t of the algorithm, let T_i^t be the load of bin i at the end of iteration t. We denote by i^t the bin used at iteration t and by p^t the size of the item considered.

To prove the theorem, it is sufficient to show that for every iteration t, $i^{t+1} = i^t + 1 \pmod m$. This is equivalent to showing $T_{i^t+1}^t \le T_{i^t+2}^t \le \cdots \le T_0^t \le \cdots \le T_{i^t}^t$, and we prove this invariant by induction on t. For $t = 1$, it is clearly true. At iteration $t+1$, we process the item on bin $i^{t+1} = i^t + 1$, because the invariant at iteration t guarantees that this is the least loaded bin. Suppose there exists a bin $g \ne i^t + 1$ with $T_{i^t+1}^{t+1} < T_g^{t+1}$, and let $h \le t$ the iteration when the last item on g was processed. At the beginning of iteration h, the load of bin g is $T_g^{h-1} \le T_{i^t+1}^{h-1}$ and then we have that

$$T_{i^{t+1}}^{h-1} + p^{t+1} \le T_{i^{t+1}}^{t+1} < T_g^{t+1} = T_g^h = T_g^{h-1} + p^h \le T_{i^{t+1}}^{h-1} + p^h,$$

which is a contradiction, because $p^h \le p^{t+1}$. Hence, $T_{i^t+1}^{t+1} \ge T_\ell^{t+1}$ for every $\ell \ne i^t + 1$. At iteration $t+1$ the only bin which changes

[1]It is easy to see that this corresponds to placing the item in the bin that has currently been filled the *least*.

its load is $i^t + 1$, and then we have that

$$T^{t+1}_{i^t+2} \le \cdots \le T^{t+1}_{m} \le \cdots \le T^{t+1}_{i^t} \le T^{t+1}_{i^t+1},$$

which proves the invariant at $t + 1$, and thus the theorem. □

We continue with an illustrative example to show how the WFI heuristic can be used to approach our problem.

Example. Consider the following instance of the scheduling with dropout problem comprising three equi-period tasks:

τ_i	C_i	T_i	$r_i = a_i/b_i$
τ_1	4	8	2/3
τ_2	3	8	1/3
τ_3	3	8	1/3

We will construct a schedule for this instance over three successive periods of the tasks, in which τ_1's jobs will be executed correctly in two periods and τ_2's and τ_3's jobs will be executed correctly in one period each; a complete schedule solving the scheduling with dropout problem is then obtained by repeatedly executing this 3-period schedule. To construct this 3-period schedule, we first map the scheduling with dropout problem to a bin-packing problem in which there are three bins of size 8 each, each bin representing an interval of duration equal to the common period of all the tasks„ and four items – two of size 4 each representing jobs of τ_1, one of size 3 representing a job of τ_2, and another of size 3 representing a job of τ_3. Note that packings that solve this bin-packing instance must satisfy an additional property in order to constitute acceptable solutions to the original scheduling with dropout problem: the two items representing jobs of τ_1 must be placed in *different* bins. And while bin-packing algorithms do not generally allow for the specification of such additional constraints (indeed, several popular bin-packing heuristics, such as First-Fit Decreasing (FFD) or Best-Fit Decreasing (BFD), would assign both jobs of τ_1 to the same bin), this is ensured by using WFI packing which considers the items in non-decreasing order of size[2]. The items would therefore by considered in the order $\tau_2, \tau_3, \tau_1, \tau_1$; by Theorem 3, the two items corresponding to jobs of τ_1 are placed in different bins.

A schedule-generation algorithm for the weak requirement in the equi-period case. We now describe our algorithm used in the example above in detail; it is also represented in high-level pseudo-code form in Figure 1. Let T denote the common period of all the tasks in the equi-period instance

$$\mathcal{T} = \left\{ \tau_i = \left(C_i, T, \frac{a_i}{b_i} \right) \right\}_{i=1}^{n}$$

of the scheduling with dropout problem for which we seek to construct a schedule. Let M denote the least common multiple of the b_i's:

$$M \overset{\text{def}}{=} \operatorname{lcm}(b_1, b_2, \ldots, b_n).$$

We now describe how to construct a schedule over the interval $[0, M \times T)$; this schedule may be executed repeatedly to obtain a schedule over the entire time-line.

[2]Although this is not an issue in this example, if multiple tasks have their C_i parameters equal then the items must be ordered such that all the items representing each task's jobs occur consecutively.

In order to construct this schedule, we first transform the task instance τ into an instance of the bin-packing problem as follows:

- Corresponding to each of the M time-intervals $[j \times T, (j + 1) \times T)$ for $0 \le j < M$, we define a bin, indexed j, of size equal to T, that represents the interval.
- Corresponding to each task $\tau_i, 1 \le i \le n$, we define $\alpha_i \leftarrow (a_i/b_i) \times M$ items each of size C_i. Each such item represents one job of τ_i.

We now pack the items amongst the bins using the WFI bin-packing heuristic. That is,

- We order the items according to non-decreasing size; in case of equal size we order according to non-decreasing index of the corresponding task, thereby ensuring that all items corresponding to each task are considered successively even if multiple tasks have the same C_i values.
- We consider the items in this order, assigning each to the bin that has been filled the least thus far – by Theorem 3, this is equivalent to assigning the items to the bins in round-robin order.

If the bin-packing fails (i.e., all the items cannot be assigned to bins), we report failure – we are unable to construct a schedule. Otherwise, we construct the schedule as follows: For each $j, 0 \le j < M$, we execute a job of each task of which a corresponding item is placed in j'th bin, over the interval $[j \times T, (j + 1) \times T)$.

Theorem 4. *Given an instance of the equi-period scheduling with dropout problem that satisfies Condition* (6)*, the algorithm of Figure 1 does not report failure, and it constructs a schedule satisfying the weak requirement.*

Proof: Note that $\alpha_i \le M$ for each τ_i, since $a_i/b_i = r_i \le 1$. Thus, Theorem 3 together with the correspondence between the WFI bin-packing heuristic and the scheduling algorithm of Figure 1 implies that there are no 2 jobs of the same task that are scheduled in the same time interval.

We show that if the input instance satisfies Condition (6), then for all j the total execution times of jobs to be scheduled in time-interval $[j \times T, (j + 1) \times T)$ is at most T, and so the algorithm will not report failure. Assume by contradiction that the claim is not correct and let us consider the smallest instance that violates the claim. Namely, the claim is not verified when the algorithm is considering a job of task τ_n; let $[\hat{j} \times T, (\hat{j} + 1) \times T)$ be the time-interval according to the WFI bin-packing heuristic selected by the algorithm to schedule such a job and $\hat{W}$ be the load of this time-interval before the assignment the last job of τ_n. Since the algorithm fails when we assign such a job we have that the total processing time assigned to the interval exceeds T that is $\hat{W} \le T < \hat{W} + C_n$.

Since the WFI assigns a job to the bin in which it fits the worst, it follows that before assigning the last job of τ_n every time-interval has a total load that is at least $\hat{W}$; therefore we have

$$M\hat{W} \le \sum_{i=1}^{n-1} \alpha_i C_i + (\alpha_n - 1)C_n.$$

Input. An equi-period instance $\mathcal{T} = \{\tau_i = (C_i, T, a_i/b_i)\}_{i=1}^n$ of the scheduling with dropout problem.

- Let M denote the least common multiple of the b_i's: $M \overset{\text{def}}{=} \text{lcm}\{b_i\}_{i=1}^n$.
- Construct a bin-packing instance as follows:
  - There are M bins, each of size T
  - Corresponding to each task $\tau_i, 1 \le i \le n$, there are $\alpha_i \overset{\text{def}}{=} (a_i/b_i) \times M$ items, each of size C_i.
- Pack the items amongst the bins using the WFI bin-packing heuristic. In case of equal-size items order according to non-decreasing index of the task to which the item corresponds.
- **If** some item does not fit into the bins, **then** report failure; **else** for each $j, 0 \le j < M$, execute a job of each task of which a corresponding item is placed in j'th bin, over the interval $[j \times T, (j+1) \times T)$.

Figure 1: Algorithm for equi-period instances

The total processing time of jobs assigned to interval $[\hat{j}\times T, (\hat{j}+1)\times T)$ is $\hat{W} + C_n$, and using the previous bounds it follows that

$$\begin{aligned}\hat{W} + C_n &\le \frac{1}{M}\sum_{i=1}^{n-1}\alpha_i C_i + \left(\frac{\alpha_n - 1}{M} + 1\right) C_n \\ &= \frac{1}{M}\sum_{i=1}^{n}\alpha_i C_i + \left(1 - \frac{1}{M}\right) C_n \\ &\le T.\end{aligned}$$

where the last inequality is obtained by observing that $\alpha_i/M = r_i$ and applying Condition (6). This contradicts the assumption that the assignment of the last job of τ_n gives an unfeasible schedule of time-interval $[\hat{j} \times T, (\hat{j}+1) \times T)$.

By the above construction, the number of completed jobs of task τ_i during the first k time-intervals is always at least $\lfloor k/M \rfloor r_i M$. Thus, as $k \to \infty$, the ratio of completed to released jobs approaches r_i, satisfying Condition (1). □

Note that since $\max_i C_i/T \le 1$ and $\sum_i r_i C_i/T \le 1$ are both necessary conditions for schedulability, the sufficient bound of Condition (6) is within a factor of 2 from an optimum bound.

4 GUARANTEEING THE STRONG REQUIREMENT: THE EQUI-PERIOD CASE

The algorithm derived in Section 3 above generates schedules that are correct under the weak requirement of Definition 1. As stated earlier, it is our opinion that the strong requirement of Definition 2 represents a more realistic formalization of the scheduling requirements of control loops than the weak requirement does. We therefore now seek to modify our algorithm in order that the schedules it generates satisfy the strong requirement (Definition 2), rather than merely the weak one. In this section, we will retain the equi-period restriction on the task instance; the ideas and results we develop here will be used in the next section to develop an algorithm for the correct scheduling, under the strong requirement, of task instances in which all periods need not be equal.

We first observe that the algorithm of Figure 1 possesses the property that the jobs of each task are scheduled in contiguous time-intervals. For example, if $r_i = 0.5$ then the WFI-based algorithm schedules jobs of task τ_i for $M/2$ contiguous time-intervals and it does not schedule jobs of τ_i for the subsequent $M/2$ time-intervals. If M is large this might not be acceptable. In the following we propose a modification of the WFI scheduling policy such that the jobs of task τ_i are scheduled – roughly – every $1/r_i$ time-intervals; we assume that all tasks have the same period T, and by scaling down the WCETs we can assume that $T = 1$.

We first transform the rates of the tasks by defining $r'_i = 2^{\lceil \log_2 r_i \rceil}$. Note that $r_i \le r'_i < 2r_i$. If $\sum_i r'_i C_i > 1$ or if $C_{\max} > 1$, we reject the instance, otherwise we continue.

We consider tasks in increasing order of their modified rates r'_i. We proceed as follows: first we consider tasks with rate 1; for each of this set of tasks, a job is scheduled in each bin (period).

Then we consider the items associated to jobs of task τ_i with rate 0.5 having maximum processing time; we assign jobs of this task to bins $0, 2, 4, 6 \ldots$: this increases the load of bins $0, 2, 4, 6 \ldots$ by C_i. Then we consider the items associated to the second task $\tau_{i'}$ with rate 0.5 and we assign items of this task to bins $1, 3, 5, \ldots$. This increases the load of bins $1, 3, 5, 7, \ldots$ by $C_{i'}$. Note that after this step, the load of all odd (even) bins is the same, and that the difference between the most loaded and the least loaded bin is bounded by $C_{\max}$. Subsequently we assign jobs of remaining tasks with rate 0.5; such tasks are assigned either to bins $0, 2, 4, 6 \ldots$ if the load of an even bin is lower than the load of an odd bin, or to bins $1, 3, 5, 7, \ldots$ otherwise. Note that in this way we maintain the invariant that the load of all even (odd) bins is the same and that the difference between the most loaded and the least loaded bin is bounded by $C_{\max}$.

The algorithm proceeds in a similar pattern. To assign jobs of a task with rate 2^{-h} we first find j, the bin with the least load among bins $1, 2, \ldots, 2^h - 1$; if there are ties, let j be the smallest indexed bin with least load. We assign the jobs of the considered task to bins $j + k2^h$, $k = 0, 1, \ldots$, thus increasing the load associated to such bins. Note that we never need to consider bins with index larger than $M \overset{\text{def}}{=} 1/\min\{r'_1, \ldots, r'_n\}$. The algorithm is summarized in Figure 2.

Lemma 1. *At each step, the algorithm of Figure 2 assigns the jobs of a task to bins having minimum load during that step.*

Proof: Recall that when the algorithm assigns jobs of a task τ, with rate $r' = 2^{-h}$ it assigns the jobs of the considered task to bins $j + k2^h$, $k = 0, 1, \ldots$, where j is the bin with the lowest load among bins $0, 1, 2, \ldots, 2^h - 1$. Therefore to prove the lemma it is sufficient to show that before the algorithm assigns jobs of task τ with rate

```
Input. An equi-period instance $\tau = \{\tau_i = (C_i, T, a_i/b_i)\}_{i=1}^n$ of the scheduling with dropout problem.
– For each task let $k(i)$ be the smallest value $2^{-k(i)}$ s.t. $2^{-k(i)} \geq r_i$;
  let $r'_i = 2^{-k(i)}$ be the rounded rate of $\tau_i$;
  let $M \overset{\text{def}}{=} 1/\min\{r'_1, \ldots, r'_n\}$
– For $h = 1, 2, \ldots, \log_2 M$ assign jobs of task of rate $2^{-h}$ as follows: for each task $\tau_i$ s.t. $r'_i = 2^{-h}$
  – let $j$ be the bin with minimum load among bins $1, 2, \ldots, 2^{h-1}$
  – assign jobs of $\tau_i$ to bins $j + k2^h$, $k = 0, 1, \ldots$
- If some item does not fit into the bins, then report failure; else for each $j, 0 \leq j < M$, execute a job of each task of which a
  corresponding item is placed in $j$'th bin, over the interval $[j \times T, (j+1) \times T)$.
```

Figure 2: A second algorithm for equi-period instances

$r' = 2^{-h}$ for all j, $j = 1, 2, \ldots, 2^h - 1$ the load of bins $j + k2^h$, $k = 0, 1, \ldots$, is the same.

We prove this by induction. Clearly the statement is obviously true when all tasks of rate equal to 1 (i.e. $h = 0$) have been considered. It is easy to see that it is also true when all tasks with rate $r' = 0.5$ have been considered. In fact at each iteration we either assign jobs of a task with rate $r' = 0.5$ either to time-intervals $0, 2, 4, 6 \ldots$ or to time-intervals $1, 3, 5, 7, \ldots$. Hence the load of time-intervals $0, 2, 4, 6 \ldots$ $(1, 3, 5, 7, \ldots)$ is the same.

Assume the claim is true before we consider the first task, say τ_i, of rate $r'_i = 2^{-h}$ and observe by induction that for all j, $j = 1, 2, \ldots, 2^{h-1}$, the load of bins $j + s2^{h-1}$, for $s = 1, 2, \ldots$ is the same. Therefore it is also true that for all j, $j = 1, 2, \ldots, 2^h$, the load of bins $j + s2^h$, for $s = 1, 2, \ldots$ is the same. □

We now derive a sufficient schedulability condition for this scheduling algorithm:

Theorem 5. *Given an instance of the equi-period scheduling with dropout problem that satisfies*

$$\max_{i=1}^{n} \frac{C_i}{T} + 2\sum_{i=1}^{n} r_i \frac{C_i}{T} \leq 1, \tag{7}$$

the algorithm of Figure 2 does not report failure, and it constructs a schedule satisfying the strong requirement.

Proof: As in the proof of Theorem 4, we will show that if the input instance satisfies Condition (7) then for all j the total execution times of jobs to be scheduled in time-interval $[j \times T, (j+1) \times T)$ is at most T. Assume by contradiction that the claim is not correct and let us consider the smallest instance that violates the claim. Namely, assume the claim is not satisfied when the algorithm is considering task τ_n; let $\hat{j}$ be the index of the bin selected by the algorithm for the first job of task τ_n and let $\hat{W}$ be the load of this bin before the assignment of the jobs of τ_n. Since the algorithm fails when we assign these jobs, we have that the total processing time assigned to bin $\hat{j}$ exceeds T, that is, $\hat{W} \leq T < \hat{W} + C_n$.

Since the algorithm chooses $\hat{j}$ as the least loaded bin in $0, 1, \ldots, {r'_n}^{-1} - 1$, we have

$$M\hat{W} \leq M\sum_{i=1}^{n} r'_i C_i \leq 2M\sum_{i=1}^{n} r_i C_i.$$

The total processing time of jobs assigned to interval $[\hat{j}\times T, (\hat{j}+1)\times T)$ is $\hat{W} + C_n$, and using the previous bound and Condition (7) it follows that

$$\hat{W} + C_n \leq 2\sum_{i=1}^{n} r_i C_i + C_n \leq T.$$

This contradicts the assumption that the assignment of the jobs of τ_n results in an infeasible schedule. □

Characterizing this algorithm. Note that since $\max_i C_i/T \leq 1$ and $\sum_i r_i C_i/T \leq 1$ are both necessary conditions for schedulability, the sufficient bound of Condition (7) is within a factor of 3 from an optimum bound.

Computational complexity. Observe that in order to determine the slot for processing the first job of task τ_i we need to consider $O(1/r_{min})$ periods, where $r_{min} = \min_i r'_i$; therefore, the time complexity is $O(n/r_{min})$. Moreover, if we store the bin index selected by the algorithm for the first job of each task, then at run-time, at the beginning of each period, we can decide in constant time for each task τ_i whether the job released by task τ_i should be executed or dropped.

5 GUARANTEEING THE STRONG REQUIREMENT: THE GENERAL CASE

We now move on to the case where the tasks may not share a common period. One approach could be to convert this case to the equi-period case; namely, let π denote the greatest common divisor (gcd) of the periods of all the tasks in the instance under consideration. One could replace each individual task $\tau_i = (C_i, T_i, r_i = a_i/b_i)$ by a task τ'_i that has period π and WCET $(\frac{C_i}{T_i}\pi)$. Observe that the instance obtained by replacing each task in this manner is an equi-period one, in which each task has a period equal to π. One could then invoke the algorithm of Figure 1; unfortunately, this approach is not correct, as the following example shows.

Example 1. *Consider the following instance of the scheduling with dropout problem, comprising three tasks.*

τ_i	C_i	T_i	$r_i = a_i/b_i$
τ_1	5	10	1/2
τ_2	2	5	1/4
τ_3	3	5	1/4

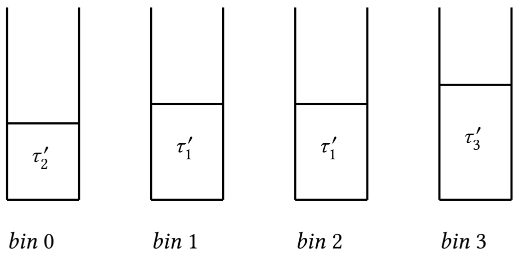
The greatest common divisor (gcd) of the periods of the tasks in the instance under consideration is $\pi = 5$. Thus, each individual task $\tau_i = (C_i, T_i, r_i = a_i/b_i)$ is replaced by a task τ_i' that has period π and WCET ($\frac{C_i}{T_i}\pi$). The obtained instance is an equi-period one, in which each task has a period equal to π.

After applying this step, our example task system is replaced by the following one:

τ_i'	C_i'	T_i'	r_i'
τ_1'	$2\frac{1}{2}$	5	1/2
τ_2'	2	5	1/4
τ_3'	3	5	1/4

If we use the algorithm of Figure 1 to construct a schedule for the equi-period instance obtained above, then the value of M is lcm(2, 4, 4) = 4*; while* $\alpha_1 = (2/4)\times 4 = 2$*;* $\alpha_2 = (4/4)\times 4 = 1$*; and* $\alpha_3 = (4/4)\times 4 = 1$.

Therefore the WFI bin-packing would look like this:



Unfortunately, this does not correspond to a correct schedule for our scheduling with dropout problem, since the two executions of τ_1' fall into bins that correspond to different *periods of the original task τ_1. That is, the period of τ_1 corresponding to bins 0 and 1 sees τ_1 receive 2.5 units of execution, as does the period corresponding to the bins 2 and 3; τ_1's job therefore does not receive the five units of execution needed to execute correctly in either period.* □

In the sequel we consider a different approach, based on applying the WFI heuristic to the tasks according to a special order, which can be seen as an extension of the approach of Section 4. We start out in Section 5.1 considering the special case when all periods are integer powers of two; subsequently in Section 5 we remove this restriction.

5.1 Special case: periods are powers of two

We will extend the approach of Section 4 to the general case. We first consider the case when periods are powers of two; the algorithm for this case is presented in Figure 3.

Initially, the algorithm rounds up the rates to powers of 2; namely, for every task τ_i we round its rate to $r_i' = 2^{\lceil \log_2 r_i \rceil}$, so that $r_i \leq r_i' < 2r_i$.

The algorithm processes the tasks in non-increasing order of the ratios r_i'/T_i; at stage h the algorithm considers all tasks s.t. $r_i'/T_i = 1/2^h$ and assigns the first job of each of these tasks to the time interval $[0, 2^h]$. Other jobs of the task are assigned by replicating the first assignment in subsequent intervals $[(j-1)2^h, j2^h]$, $j = 1, 2, \ldots$.

At each stage h, the tasks s.t. $r_i'/T_i = 1/2^h$ are considered in order of non-decreasing periods, breaking ties arbitrarily; since $r_i'/T_i = 1/2^h$, then, in the considered time interval $[0, 2^h]$ we must assign exactly one job for each considered task.

In order to explain how the first job of each task is assigned we need some additional notation; let $L(t)$ be the total processing time (load) assigned to interval $[t, t+1]$. When considering task τ_i s.t. $r_i'/T_i = 1/2^h$, with period, say, 2^k, the time interval $[0, 2^h]$ is divided into 2^{h-k} subintervals $I_f = [f2^k, (f+1)2^k]$, $f = 0, 1, \ldots, 2^{h-k} - 1$ of length 2^k; W_f denotes the total load assigned to I_f (i.e. $W_f = \sum_{t \in I_f} L(t)$).

The first job of τ_i is scheduled within the interval $I_{\hat{f}} = [\hat{f}2^k, (\hat{f}+1)2^k]$, s.t. $W_{\hat{f}}$ is minimum (breaking ties arbitrarily), and it is scheduled in such a way to minimize the maximum load $L(t)$ for $t \in I_{\hat{f}}$. This can be achieved by repeatedly assigning a small fraction of the job to the time interval with minimum load until the job is fully scheduled. The goal is to spread the load assigned to time interval $I_{\hat{f}}$ as evenly as possible.

Subsequent jobs of task τ_i are similarly scheduled at time intervals $[\hat{f}2^k + j2^h, (\hat{f}+1)2^k j2^h]$, $j = 1, 2, \ldots$. Observe that after all tasks τ_i s.t. $r_i'/T_i = 2^{-h}$ have been considered the load assigned to time t, $t > 2^h$, is equal to $L(t) = L(t \mod 2^h)$.

Theorem 6. *Given an instance of the scheduling with dropout problem in which the periods are powers of two, and that satisfies*

$$\max_{i=1}^{n} \frac{C_i}{T_i} + 2\sum_{i=1}^{n} r_i \frac{C_i}{T_i} \leq 1,$$

the algorithm of Figure 3 does not report failure, and it constructs a schedule satisfying the strong requirement.

Proof: We first show that if a schedule can be constructed by the algorithm, then it satisfies the rate requirement. Indeed, the algorithm schedules a job of task τ_i every T_i/r_i' time units; since $r_i' \geq r_i$, the claim follows.

We now show that the algorithm does not report failure. Without loss of generality we can assume that the smallest period of a task is 1 and we will show that if the input instance satisfies the conditions of the theorem, then for each t the total execution times of jobs to be scheduled in time-interval $[t, (t+1)]$ is at most 1.

Assume by contradiction that the claim is not correct, and let us consider the smallest instance that violates the claim. Then, by renaming the tasks in the order they are processed by the algorithm, we can assume that the algorithm reports failure when considering task τ_n.

Let $I_{\hat{f}}$ be the time interval selected by the algorithm for executing the first job of τ_n and assume that τ_n is processed in stage h, i.e., $r_n'/T_n = 2^h$.

Let $L(t)$ ($L'(t)$) be the load at time t before (after) jobs of task τ_n have been scheduled; similarly let $L_{\max}$ ($L'_{\max}$) be the maximum load before (after) jobs of task τ_n have been assigned to time intervals. Since after assigning task τ_n the solution is not feasible, it follows that from some t_c, $L_{\max} \leq 1 < L'_{\max} = L'(t_c)$; then $L(t) = L(t \mod 2^h)$ implies that there should be a t_0, $t_0 \in I_{\hat{f}}$, s.t. $L'(t_0) = L'_{\max} > L_{\max}$.

```
**Input.** An instance $\tau = \{\tau_i = (C_i, T_i, r_i)\}_{i=1}^n$ of the scheduling problem with dropout, such that each $T_i$ is a power of 2
– For each task let $k(i)$ be the smallest value $2^{-k(i)}$ s.t. $2^{-k(i)} \geq r_i$;
  let $r'_i = 2^{-k(i)}$ be the rounded rate of $\tau_i$;
– For $h = 0, 1, 2, \ldots$ assign the jobs of tasks $\tau_i$ s.t. $T'_i/r'_i = 2^h$, as follows:
  – let $\hat{f}$ be an index s.t. $W_f$, $f = 0, 1, \ldots, 2^{h-k} - 1$, is minimum
  – schedule the first job of task $\tau_i$ within $I_{\hat{f}}$, in such a way to maximize the minimum load in $[(f2^k - 1), f2^{k+1}]$, $f = 1, 2, \ldots$
  – **if** the load of any time-slot exceeds 1, report failure
  – assign the $j$- job of $\tau_i$ by replicating the previous schedule in time interval $[j \cdot 2^h, (j+1) \cdot 2^h]$, $j = 1, 2, \ldots$
```

Figure 3: Algorithm for the case where the periods are powers of 2

The crucial observation is that if $L'(t_0) = L'_{\max} > L_{\max}$, then the way the processing demand of the first job of task τ_n is scheduled implies that [3]

$$L'(t) = L'_{\max} \text{ for all } t \in I_{\hat{f}}.$$

Assume that the algorithm assigns the first job of τ_n to $I_{\hat{f}}$ and let $W_{\hat{f}}$ and $W'_{\hat{f}}$ be the total load of $I_{\hat{f}}$ before and after the assignment respectively. Clearly we have

$$W'_{\hat{f}} = W_{\hat{f}} + C_n.$$

Note that the way the job is scheduled in $I_{\hat{f}}$ implies that the load over the whole interval after the assignment is the same for all time instants belonging to the interval, i.e.,

$$L'_{\max} = W'_{\hat{f}}/2^k.$$

Since the algorithm chooses $I_{\hat{f}}$ as the interval with minimum total load, then we have that the average load in it is no greater than the average load in $[0, 2^h]$, that is

$$\frac{W_{\hat{f}}}{2^k} \leq \sum_{i=1}^{n-1} r'_i \frac{C_i}{T_i}.$$

Putting together the above inequalities we have

$$\begin{aligned} L'_{\max} &= W'_{\hat{f}}/2^k = (W_{\hat{f}} + C_n)/2^k \\ &\leq \sum_{i=1}^{n-1} r'_i C_i/T_i + C_n/2^k \\ &\leq 2\sum_{i=1}^{n} r_i C_i/T_i + C_n/T_n \\ &\leq 1. \end{aligned}$$

This contradicts the assumption that the assignment of jobs of τ_n results in the algorithm reporting failure. □

Characterization. Note, again, that the bound in Theorem 6 is within a factor of 3 of an optimum bound and that the strong requirement is satisfied with $k^* = 1$.

[3]We observe that in general it is not true that after assigning the first job of a task all intervals that are in the considered time interval have the same load; however, this is true if the assignment of a task increases the maximum load.

5.2 General case

We now show how to extend the algorithm of Figure 3 to the case of general periods. We proceed in two steps; in the first one we round the periods to powers of 2 and we apply the algorithm of Figure 3 to construct a schedule. The obtained solution, however, is not feasible because it incurs the same problem that was highlighted in the example at the beginning of this section (i.e., a job of task τ_i with period T_i may be scheduled in an interval $[t_1, t_2]$ of length T_i that overlaps with two intervals $[kT_i, (k+1)T_i]$ and $[(k+1)T_i, (k+2)T_i]$ for some k. In the second step we adapt this solution by assigning jobs of such tasks τ_i into properly aligned intervals.

Namely, for each task τ_i we define a modified task $\tau'_i = (C_i, T'_i, r'_i)$ with the same WCET and periods and rates that are powers of two. We first modify the periods of each task; namely if the period of τ_i is a power of two then the modified period T'_i is equal to T_i; for each task τ_i whose period is not a power of two we consider the rounded period of τ_i to be $T'_i = 2^b$, so that $T_i/2 < T'_i \leq T_i$ and b is integer. Then we compute the rounded rate $r'_i = 1/2^a$ so that $r_i/T_i \leq r'_i/T'_i < 2r_i/T_i$ and a is integer.

We now apply the algorithm of Figure 3 to the above modified instance of the problem, obtaining a solution S'. Namely, for each task τ'_i, we obtain an interval $J_i = [t_i, t_i + T'_i]$ where one job of τ'_i among those released in $[0, T'_i/r'_i]$ will be scheduled.

In order to obtain a feasible schedule from S' we now find an assignment for the first job of τ_i, $i = 1, 2, \ldots, n$; we observe that there exists an integer $g_{i,1}$ such that the interval $I_i = [g_{i,1}T_i, (g_{i,1} + 1)T_i]$ overlaps with J_i and such that in $I_i \cap J_i$ at least half of the first job of τ'_i is scheduled in S'. The first scheduled job of τ_i is executed in I_i; note that this can be accomplished by doubling (at most) the processor share assigned to τ'_i in S.

To schedule subsequent jobs of τ_i we proceed similarly. Namely, the j-th job of τ_i that is released at jT_i is scheduled if and only if there exists an integer $g_{i,j}$ such that the interval $J_i(h) = [(t_i + g_{i,j}T'_i/r'_i), (t_i + g_{i,j}T'_i/r'_i) + T'_i]$ overlaps with $[jT_i, (j+1)T_i]$ and at least half of h-th job of τ'_i is processed in $J_i(h)\hat{[}jT_i, (j+1)T_i]$.

Theorem 7. *Given an instance of the scheduling with dropout problem that satisfies*

$$2\max_{i=1}^{n} \frac{C_i}{T_i} + 8\sum_{i=1}^{n} r_i \frac{C_i}{T_i} \leq 1,$$

then the algorithm does not report failure, and it constructs a schedule satisfying the strong requirement.

Proof: Let S be the solution found by the algorithm. We first show that the algorithm is feasible. Namely, we show that for each time instant t the total processing time assigned by the algorithm to $[t, t+1]$ is bounded by 1.

Consider the solution S' that schedules the modified tasks obtained by rounding periods and rates. Let $L(t)$ and $L'(t)$ be the total load assigned to time interval $[t, t+1]$ in solution S and S', respectively. By reasoning similarly to the proof of Theorem 6, we have that for each t, $L(t)$, the load assigned to time t in solution S' verifies

$$L'(t) \leq \max_i (C_i/T'_i) + 2\sum_{i=1}^{n} r'_i C_i/T'_i.$$

Since the algorithm schedules jobs of task τ_i to the time interval that schedules at least half of the rounded task τ'_i it follows that $L(t) \leq 2L'(t)$, for all t. Since $T_i \geq T'_i$ and $r'_i/T'_i \leq 2r_i/T_i$ we have

$$\begin{aligned} L(t) \quad &\leq \quad 2L'(t) \leq 2\max_i(C_i/T'_i) + 4\sum_{i=1}^{n} r'_i C_i/T'_i \\ &\leq \quad 2\max_i(C_i/T_i) + 8\sum_{i=1}^{n} r_i C_i/T_i \\ &\leq 1 \end{aligned}$$

We then show the solution found by the algorithm verifies the rate requirement. In fact the algorithm starts the schedule of a job of task τ_i every $T'_i/r'_i + T'_i$ time units at most; since $T'_i r'_i \geq T_i r_i$, the claim follows. □

6 EXTENSION TO MULTIPROCESSOR PLATFORMS

We finally show how to extend our results to a platform of m identical processors. The following theorem generalizes the scaled utilization bound of Theorem 1.

Theorem 8. *For any $\epsilon > 0$, there are instances of the scheduling with dropout problem that are infeasible for m identical processors (assuming global scheduling) and yet satisfy*

$$\sum_{i=1}^{n} r_i \frac{C_i}{T_i} \leq (m-1) + 2\epsilon. \tag{8}$$

Proof: Let $T = \lceil 1/\epsilon \rceil$ and consider an instance with $(m+1)$ tasks; the first two tasks are defined as follows

$$\tau_1 = (T, T, 1/T), \ \tau_2 = (1, T, 1);$$

the remaining tasks $\tau_3, \ldots, \tau_{(m+1)}$ have parameters $(T, T, 1)$.

The weighted utilizations of the tasks sum to $(m-1) + 2/T$. However, this instance is clearly unschedulable according to either the weak or the strong requirement, even if global scheduling is assumed. □

We finally observe that our bin-packing approach to the problem can be extended to provide sufficient schedulability conditions at least in the equi-period case. In the sequel we extend the result of Theorem 5 to multiprocessors.

To find a feasible schedule on m identical processors we use the following algorithm:

(1) given a task set $\mathcal{T} = \{\tau_i = (C_i, T, a_i/b_i)\}_{i=1}^{n}$ consider a new task set $\mathcal{T}' = \{\tau'_i = (C_i, mT, a_i/b_i)\}_{i=1}^{n}$;
(2) apply algorithm of Figure 2 to $\mathcal{T}'$, obtaining a uniprocessor schedule S';
(3) each time slot of length mT in S' is subdivided into a schedule of a time slot of length T on the m machines.

Theorem 9. i) *Given an equi-period task set $\mathcal{T}$ that satisfies*

$$\max_{i=1}^{n} \frac{C_i}{T} + 2\sum_{i=1}^{n} r_i \frac{C_i}{T} \leq m$$

there exists a global scheduling algorithm that constructs a correct solution to $\mathcal{T}$ with respect to the strong invariant.

ii) *Given an equi-period task set $\mathcal{T}$ that satisfies*

$$\max_{i=1} \frac{C_i}{T} + 2\sum_{i=1}^{n} r_i \frac{C_i}{T} \leq \frac{m}{2}$$

there exists a partitioned scheduling algorithm that constructs a correct solution to $\mathcal{T}$ with respect to the strong invariant.

Proof: i) We proceed as described above and we define a modified task set where each task has period mT. Theorem 5 implies that the algorithm of Figure 2 applied to $\mathcal{T}'$ finds a feasible solution.

Recall that all tasks have the same period, and let $A(k)$ be the set of tasks scheduled by the algorithm in $[kmT, (k+1)mT]$. This set of tasks corresponds to a set of tasks with processing time at most mT. Since the processing time of each task is less than T, this set of tasks can be scheduled using global scheduling on m processors.

ii) In this case the modified task set $\mathcal{T}'$ is defined as follows: $\mathcal{T}' = \{\tau'_i = (C_i, mT/2, a_i/b_i)\}_{i=1}^{n}$. Let $A(k)$ be the set of tasks of $\mathcal{T}'$ scheduled by the algorithm in $[kmT, (k+1)mT]$. This set of tasks corresponds to a set of tasks with total processing time at most $mT/2$ and each processing time is at most $T/2$. By partitioning the set of tasks in $A(k)$ using heuristics such as Best Fit (BF) or First Fit Decreasing (FFD), one obtains a partition of the jobs $A(k)$ into m bins of size T. □

We expect that a similar approach can be used to extend the results of Section 5 as well; thorough investigation of this is left as future work.

7 SUMMARY AND CONCLUSIONS

The run-time behaviors of various sensing and actuation aspects of many complex cyber-physical systems are governed by control loops. As the functionalities performed by these cyber-physical systems become increasingly more complex, such control loops may become very computation-intensive; there is an increasing need to carefully schedule such control loops upon the limited computational platforms that may be available.

In this paper, we have considered a particular problem that has previously been considered in the control literature (in, e.g., [2, 9]) – how does one co-schedule a collection of independent control loops, each of which is allowed to skip occasional iterations, upon a shared computational platform? We have proposed two scheduling-theoretic formalizations of the notion of correctness that has previously been used in the control theory literature. The definition of correctness used in earlier work [2, 9] is formalized here as the weak correctness requirement (Definition 1); we believe that this definition does not do full justice to the actual expectations of control engineers. We accordingly propose a stronger notion of correctness, which we formalize as a strong correctness requirement (Definition 2). We characterize the computational complexity of determining whether a given collection of control loops, each represented as an implicit-deadline periodic task with an expected rate of successful executions, can be scheduled correctly upon a preemptive uniprocessor platform. We derive, prove correct, and characterize approximation algorithms for doing such scheduling.

While we believe we have made significant progress in initiating a formal scheduling-theoretic study of the problem of scheduling control loops, there is a lot of work that remains to be done — this includes efforts at more accurate characterization (and meaningful formalization) of the problems, and efforts at obtaining more efficient solutions. We expect that satisfactory progress in solving this problem would require close cooperation amongst domain specialists in control and scheduling theory.

ACKNOWLEDGMENTS

This research is supported by NSF grants CNS 1409175 and CPS 1446631, AFOSR grant FA9550-14-1-0161, and ARO grant W911NF-14-1-0499.

REFERENCES

[1] G. Bernat, A. Burns, and A. Liamosi, "Weakly hard real-time systems," *IEEE Transactions on Computers*, vol. 50, no. 4, pp. 308–321, 2001.
[2] M. Branicky, S. Phillips, and W. Zhang, "Scheduling and feedback co-design for networked control systems," in *Proceedings of the 41st IEEE Conference on Decision and Control*, vol. 2, Dec 2002, pp. 1211–1217.
[3] A. Cervin, "Analysis of overrun strategies in periodic control tasks," in *IFAC Proceedings*, vol. 38, no. 1, pp. 219–224. Elsevier, 2005.
[4] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.
[5] M. Hamadaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m, k)-firm deadlines," in *Proceedings of the 23rd Annual International Symposium on Fault-Tolerant Computing (FTCS '94)*. IEEE Computer Society Press, Jun. 1994, pp. 196–205.
[6] D. S. Johnson, "Near-optimal bin packing algorithms," Ph.D. dissertation, Department of Mathematics, Massachusetts Institute of Technology, 1973.
[7] D. S. Johnson, "Fast algorithms for bin packing," *Journal of Computer and Systems Science*, vol. 8, no. 3, pp. 272–314, 1974.
[8] G. Koren and D. Shasha, "D^{over}: An optimal on-line scheduling algorithm for overloaded real-time systems," Computer Science Department, New York University, Tech. Rep. TR 594, 1992.
[9] R. Majumdar, I. Saha, and M. Zamani, "Performance-aware scheduler synthesis for control systems," in *Proceedings of the Ninth ACM International Conference on Embedded Software*, ser. EMSOFT '11. ACM, 2011, pp. 299–308.
[10] T. Yoshimoto and T. Ushio, "Optimal arbitration of control tasks by job skipping in cyber-physical systems," in *Proceedings IEEE/ACM Second International Conference on Cyber-Physical Systems*. IEEE, 2011, pp. 55–64.