Pfair Scheduling: Beyond Periodic Task Systems*

James H. Anderson and Anand Srinivasan
Department of Computer Science, University of North Carolina, Chapel Hill, NC 27599-3175.

Abstract

In this paper, we consider variants of Pfair and ERfair scheduling in which subtasks may be released late, i.e., there may be separation between consecutive windows of the same task. We call such tasks *intra-sporadic* tasks. There are two main contributions of this paper. First, we show the existence of a Pfair (and hence ERfair) schedule for any intra-sporadic task system whose utilization is at most the number of available processors. Second, we give a polynomial-time algorithm that is optimal for scheduling intra-sporadic tasks in a Pfair or ERfair manner on systems of one or two processors.

1 Introduction

Pfair scheduling was proposed by Baruah et al. as a way of optimally and efficiently scheduling periodic tasks on a multiprocessor system [4, 5]. Pfair scheduling differs from more conventional real-time scheduling disciplines in that tasks are explicitly required to make progress at steady rates. In the classic periodic task model, each task T executes at an implicit rate given by $T.e/T.p$, where $T.e$ is the *execution cost* of each job of T, and $T.p$ is the *period* of T. However, this notion of a rate is a bit inexact: a job of T may be allocated $T.e$ time units at the beginning of its period, or at the end of its period, or its computation may be spread out more evenly. Under Pfair scheduling, this implicit notion of a rate is strengthened to require each task to be executed at a rate that is uniform across each job.

Pfair scheduling algorithms ensure uniform execution rates by breaking tasks into quantum-length "subtasks." Each subtask must execute within a "window" of time slots, the last of which is its deadline. These windows divide each period of a task into subintervals of approximately equal length. By breaking tasks into smaller executable units, Pfair scheduling algorithms circumvent many of the bin-packing-like problems that lie at the heart of intractability results involving multiple-resource real-time scheduling problems. Intuitively, it is easier to evenly distribute small, uniform items among the available bins than larger, non-uniform items.

In recent work, we considered a work-conserving variant of Pfair scheduling called "early-release" fair (ERfair) scheduling [2]. ERfair scheduling differs from Pfair scheduling in a rather simple way. Under Pfair scheduling, if some subtask of a task T executes "early" within its window, then T is ineligible for execution until the beginning of its next window. Under ERfair scheduling, if two subtasks are part of the same job, then the second subtask is eligible for execution as soon as the first completes. In other words, a subtask may be released "early," i.e., before the beginning of its Pfair window.

Contributions of this paper. One limitation of most prior work on Pfair and ERfair scheduling is that only synchronous, periodic task systems have been considered. In this paper, we consider variants of Pfair and ERfair scheduling in which subtasks may sometimes be released late, i.e., there may be separation between consecutive windows of the same task. We call such tasks *intra-sporadic* tasks. The notion of an intra-sporadic task generalizes that of a sporadic task (which in turn generalizes the notion of an asynchronous task). Why generalize the sporadic model in this way? Our primary motivation here is to identify the most flexible notion of a rate that can be optimally supported in a multiprocessor system. Indeed, if early releases are allowed, then the intra-sporadic task model is essentially a quantum-based *multiprocessor* variant of the *uniprocessor* rate-based execution model proposed recently by Jeffay and Goddard [6]. Being able to support a flexible notion of a rate simplifies the design of applications in which some processing steps may be highly jittered. A good example of such an application is a multimedia system in which packets may sometimes arrive early or late.

All prior proofs known to us involving Pfair and ERfair scheduling rely *crucially* on the fact that at any point of time, the exact alignment of future windows can be predicted. However, for intra-sporadic task sys-

*Work supported by NSF grants CCR 9732916, CCR 9972211, CCR 9988327, and ITR 0082866.

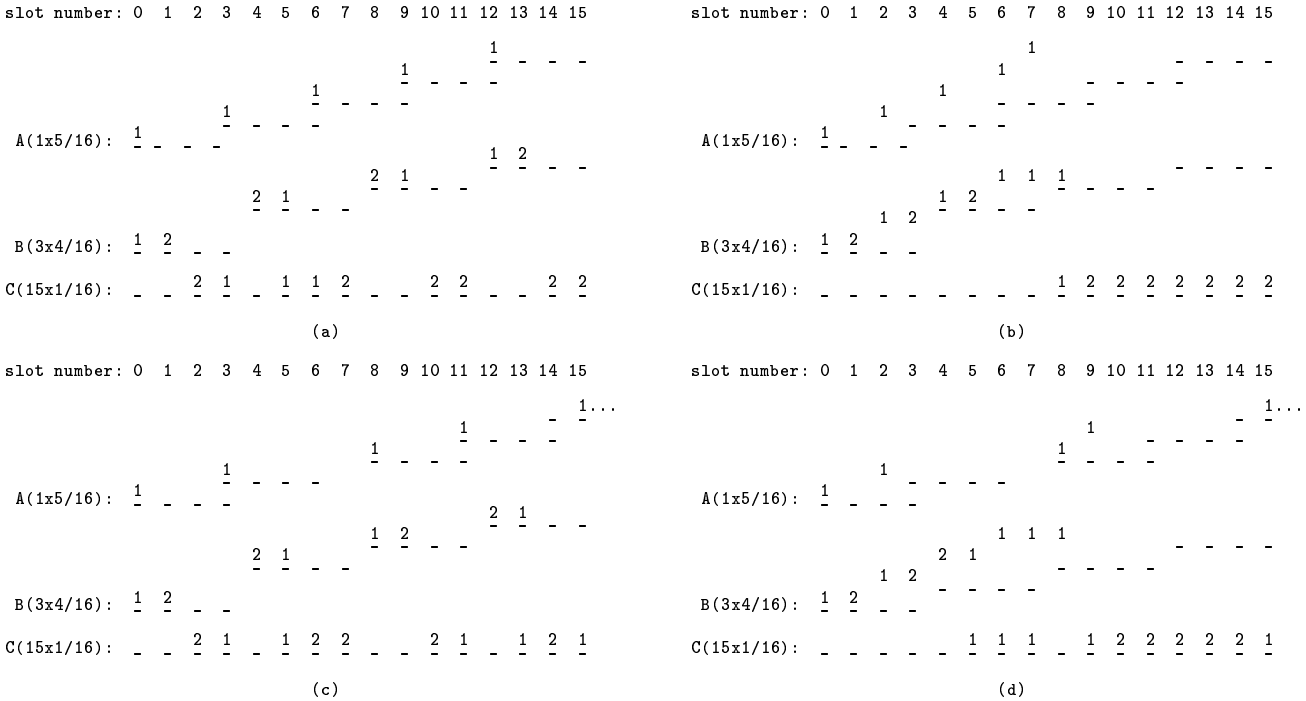

Figure 1: A schedule for a task set under **(a)** Pfair scheduling, **(b)** ERfair scheduling, **(c)** Pfair scheduling with an intra-sporadic task, and **(d)** ERfair scheduling with an intra-sporadic task. In this figure, tasks of a given weight are shown together. Each Pfair window is shown on a separate line and is depicted by showing the time slots it spans. Each column corresponds to a time slot. In the Pfair schedules, a slot t within a window is denoted by a dash. An integer value n means that n of the subtasks that must execute within that window are scheduled in slot t. No integer value means that no such subtask is scheduled in slot t. In the ERfair schedules, a subtask may be scheduled before its Pfair window.

tems such predictions cannot be made, because future subtasks can be released late. This is the main reason why earlier proofs do not directly extend to intra-sporadic task systems, or even to sporadic task systems.

Aside from introducing the intra-sporadic task model, there are two main contributions of this paper. First, we state a feasibility condition (which is proved in the full paper) for scheduling a system of intra-sporadic tasks on a system of M processors. Second, we give a simple, polynomial-time algorithm for optimally scheduling intra-sporadic tasks on systems of one or two processors. The development of an algorithm for efficiently scheduling intra-sporadic tasks on systems of three or more processors is left as an open problem.

Some example schedules. Before continuing, we consider some example schedules that highlight the differences among the task models considered in this paper. In the Pfair scheduling literature, the ratio of a task T's execution cost and period, $T.e/T.p$, is referred to as its *weight*. A task's weight determines the length and alignment of its Pfair windows. Figure 1 shows some schedules involving three sets of tasks: a set A of one task of weight 5/16, a set B of three tasks of weight 4/16, and a set C of 15 tasks of weight 1/16. Inset (a) shows the schedule for these tasks up to time slot 15 in a Pfair-scheduled system. Note that successive windows of a task are either disjoint or overlap by one slot. As we shall see, this is a general property of Pfair systems. Inset (b) shows the same task set under ERfair scheduling. In this case, each subtask is eligible as soon as its predecessor completes. Notice that all set-A and set-B jobs have finished by time slot 8, which is much sooner than in the Pfair schedule. Insets (c) and (d) show schedules similar to those in insets (a) and (b), respectively, except that the third window of the set-A task is released two time units late. In inset (d), we have assumed that the third subtask of the set-A task is not actually eligible until time slot 8 (for example, maybe this late release represents a packet that arrived late at time 8). The system would still be schedulable if we were to allow this subtask to be eligible as soon as the second completes. This would correspond to an artificial shifting of all future subtask deadlines of this task (although this is possible, we know of no practical reason for wanting to shift deadlines like this).

The rest of this paper is organized as follows. In Section 2, we define Pfair and ERfair scheduling. Then, in Section 3, we introduce the concept of an intra-sporadic task. In Section 4, we present a polynomial-time algorithm to optimally schedule intra-sporadic tasks on systems of one or two processors. Concluding remarks appear in Section 5.

2 Pfair and ERFair Scheduling

Consider a collection of synchronous, periodic real-time tasks to be executed on a system of multiple processors. (For the moment, we are only considering periodic tasks. The notion of an intra-sporadic task will be defined later.) We assume that processor time in such a system is allocated in discrete time units, or quanta; the time interval $[t, t+1)$, where t is a nonnegative integer, is called *slot* t. Associated with each task T is a *period* $T.p$ and an *execution cost* $T.e$. Every $T.p$ time units, a new invocation of T with a cost of $T.e$ time units is released into the system; we call such an invocation a *job* of T. Each job of a task must complete execution before the next job of that task begins. Thus, $T.e$ time units must be allocated to T in each interval $[k \cdot T.p, (k+1) \cdot T.p)$, where $k \geq 0$. T may be allocated time on different processors in such an interval, as long as it is not allocated time on different processors at the same time.

The sequence of allocation decisions over time defines a "schedule." Formally, a *schedule* S is a mapping $S : \tau \times \mathcal{N} \mapsto \{0, 1\}$, where τ is a set of periodic tasks and $\mathcal{N}$ is the set of natural numbers. If $S(T, t) = 1$, then we say that *T is scheduled at slot t*. S_t denotes the set of tasks scheduled in slot t. The statements $T \in S_t$ and $S(T, t) = 1$ are equivalent.

Lag constraints. As mentioned previously, we refer to the ratio $T.e/T.p$ as the *weight* of task T, denoted $wt(T)$. We assume each task's weight is strictly less than one — a task with weight one would require a dedicated processor, and thus is quite easily scheduled. A task's weight defines the rate at which it is to be scheduled. Because processor time is allocated in quanta, we cannot guarantee that a task T will execute for *exactly* $(T.e/T.p)t$ time during each interval of length t. Instead, in a Pfair-scheduled system, processor time is allocated to each task T in a manner that ensures that its rate of execution never deviates too much from that given by its weight $T.e/T.p$. More precisely, correctness is defined by focusing on the *lag* between the amount of time allocated to each task and the amount of time that would be allocated to that task in an ideal system with a quantum approaching zero. Formally, the *lag of task T at time t*, denoted $lag(T, t)$, is defined as follows:

$$lag(T, t) = (T.e/T.p)t - allocated(T, t), \tag{1}$$

where $allocated(T, t)$ is the amount of processor time allocated to T in $[0, t)$. A schedule is *Pfair* iff

$$(\forall T, t :: -1 < lag(T, t) < 1). \tag{2}$$

Informally, the allocation error associated with each task must always be less than one quantum.

Our notion of early-release scheduling is obtained by simply dropping the -1 lag constraint. Formally, a schedule is *early-release fair* (*ERfair*) iff

$$(\forall T, t :: lag(T, t) < 1). \tag{3}$$

Note that any Pfair schedule is ERfair, but not necessarily vice versa. It is straightforward to show that any ERfair schedule (and hence, any Pfair schedule) is periodic. In particular, in an ERfair schedule, $lag(T, t) = 0$ for $t = 0, T.p, 2T.p, 3T.p, \ldots$. This is because, for these values of t, $(T.e/T.p)t$ is an integer, and therefore by (1), $lag(T, t)$ is an integer as well. By (3), if $lag(T, t)$ is an integer, then it must be 0 or some negative integer. However, it cannot be a negative integer because this would imply that more processor time has been allocated to T than has been requested by jobs of T up to time t. Hence, $lag(T, t)$ is 0 for these values of t.

Feasibility. Baruah et al. [5] showed that a periodic task set τ has a Pfair schedule on M processors iff

$$\sum_{T \in \tau} \frac{T.e}{T.p} \leq M. \tag{4}$$

Because every Pfair schedule is also an ERfair schedule, (4) is a feasibility condition for ERfair systems as well.

Windows. The Pfair lag bounds given in (2) have the effect of breaking each task T into an infinite sequence of unit-time *subtasks*. We denote the ith subtask of task T as T_i, where $i \geq 1$. As in [4], we associate with each subtask T_i a *pseudo-release* $r(T_i)$ and a *pseudo-deadline* $d(T_i)$. If T_i is synchronous and periodic, then $r(T_i)$ and $d(T_i)$ are as follows.

$$r(T_i) = \left\lfloor \frac{i-1}{wt(T)} \right\rfloor \tag{5}$$

$$d(T_i) = \left\lceil \frac{i}{wt(T)} \right\rceil - 1 \tag{6}$$

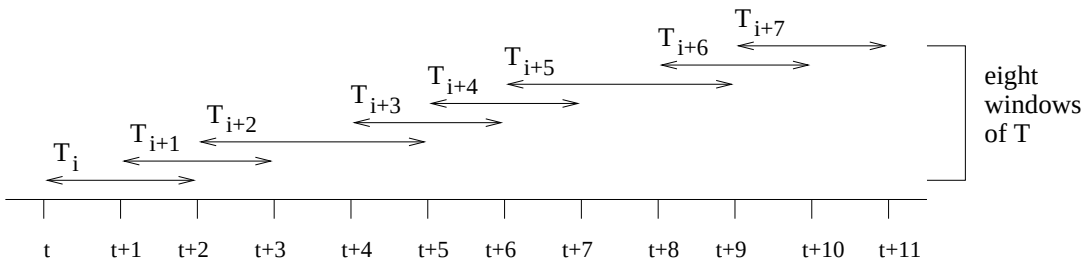

Figure 2: The eight "windows" of a task T with weight $wt(T) = 8/11$. Under Pfair scheduling, each of T's eight units of computation must be allocated processor time during its window, or a lag-bound violation will result.

(Derivations of these expressions can be found in [3].) In a Pfair-scheduled system, $r(T_i)$ is the first slot into which T_i could potentially be scheduled, and $d(T_i)$ is the last such slot. In an ERfair-scheduled system, if T_i and T_{i-1} are part of the same job, then T_i becomes eligible immediately after T_{i-1} is scheduled, so $r(T_i)$ is of less relevance. For brevity, we often refer to pseudo-deadlines and pseudo-releases as simply deadlines and releases, respectively. The interval $[r(T_i),d(T_i)]$ is called the *window* of subtask T_i and is denoted by $w(T_i)$. The *length* of window $w(T_i)$, denoted by $|w(T_i)|$, is defined as $d(T_i) - r(T_i) + 1$. As an example, consider a task T with weight $wt(T) = 8/11$. Each job of this task consists of eight windows, one for each of its unit-length subtasks. Using Equations (5) and (6), it is possible to show that the windows within each job of T are as depicted in Figure 2. Note that successive windows of T overlap by one slot and are of two different lengths. In general, consecutive windows of a task are either disjoint or overlap by one slot. In addition, either all windows of a task are of the same length, or they are of two different lengths (this property is proved in [3]).

Scheduling algorithms. For synchronous, periodic task systems, the most efficient Pfair scheduling algorithm proposed to date is an algorithm called PD [4, 5]. PD schedules subtasks by pseudo-deadline. In the original PD algorithm, four tie-break parameters were used to resolve ties among subtasks with the same deadline. In recent work, we proved that only two tie-break parameters suffice [3]. In other recent work, we have shown that an early-release version of PD, called ER-PD, can be used to optimally schedule synchronous, periodic tasks in an ERfair manner.

3 Intra-sporadic Tasks

Intra-sporadic tasks are a generalization of sporadic tasks. In a sporadic task system, there can be separation between consecutive jobs of a task. The intra-sporadic task model is obtained by carrying this a step further and allowing separation between consecutive subtasks of the same task.

Formal definition. An *intra-sporadic task system* is defined by the tuple (τ, e), where τ represents a set of tasks, and e is a function that indicates when each subtask first becomes eligible for execution. Each task may release either a finite or infinite number of subtasks. We assume that $e(T_i) \geq e(T_{i-1})$ for all $i \geq 2$, i.e., subtask T_i cannot become eligible before its predecessor T_{i-1}.

In the intra-sporadic model, each subtask has two windows of interest, namely its *Pfair window* (PF-window) and its *intra-sporadic window* (IS-window), respectively. A subtask's IS-window defines the span of time slots during which it may be potentially scheduled. Its IS-window must include its PF-window, the definition of which is similar to that given previously for synchronous, periodic task systems. Formally, T_i's IS-window is defined to be $[e(T_i), d(T_i)]$ and its PF-window is defined to be $[r(T_i), d(T_i)]$. As we shall see, the terms $r(T_i)$ and $d(T_i)$ have a similar interpretation to that given previously for synchronous, periodic task systems. We will use the term $w(T_i)$ to refer to the PF-window of subtask T_i, as before.

The terms $r(T_i)$ and $d(T_i)$ are defined inductively by examining the alignment of the PF-windows of T in a synchronous, periodic task system. In such a system, consecutive PF-windows of T are either disjoint or overlap by one slot. The bit $b(T_i)$, defined below, distinguishes between these two possibilities.

$$b(T_i) = \begin{cases} 0, & \text{if } \lceil \frac{i}{wt(T)} \rceil = \lfloor \frac{i}{wt(T)} \rfloor \\ 1, & \text{otherwise.} \end{cases}$$

By examining Equations (5) and (6) it should be clear that in a *synchronous*, *periodic* task system, $r(T_{i+1}) = d(T_i)$ (i.e., $w(T_i)$ and $w(T_{i+1})$ overlap by one slot) if

$b(T_i) = 1$, and $r(T_{i+1}) = d(T_i) + 1$ (i.e., $w(T_i)$ and $w(T_{i+1})$ do not overlap) if $b(T_i) = 1$. For intra-sporadic tasks, we define $b(T_i)$ exactly as above. Given this definition, we can define $r(T_i)$, which defines the beginning of the PF-window of T_i.

$$r(T_i) = \begin{cases} e(T_i), & \text{if } i = 1 \\ max(e(T_i),\ d(T_{i-1}) + 1 - b(T_{i-1})) & \text{if } i \geq 2 \end{cases} \quad (7)$$

According to this definition, the PF-window of T_1 begins when it first becomes eligible. For $i \geq 2$, $r(T_i)$ is defined to be the later of $e(T_i)$ and $d(T_{i-1})+1-b(T_{i-1})$. Thus, if T_i becomes eligible *during* T_{i-1}'s PF-window, then $r(T_i) = d(T_{i-1})+1-b(T_{i-1})$, and hence, the spacing between $r(T_{i-1})$ and $r(T_i)$ is exactly as in a synchronous, periodic task system. (Note that the notion of a job is not mentioned here. For systems in which subtasks are grouped into jobs that are released in sequence, the definition of e would preclude T_i from becoming eligible during T_{i-1}'s PF-window if T_i is the first subtask of its job.) On the other hand, if T_i becomes eligible *after* T_{i-1}'s PF-window, then T_i's PF-window begins when T_i becomes eligible. Note that the definition above implies that consecutive PF-windows of the same task are either disjoint or overlap by one slot, just as in a synchronous, periodic task system. Also, if T_i is scheduled within its PF-window, then it is also scheduled within its IS-window.

T_i's deadline $d(T_i)$ is defined to be $r(T_i)+|w(T_i)|-1$, where $|w(T_i)|$ denotes the length of T_i's PF-window. PF-window lengths are the same in intra-sporadic and synchronous, periodic systems. Thus, by Equations (5) and (6), we have

$$|w(T_i)| = \left\lceil \frac{i}{wt(T)} \right\rceil - \left\lfloor \frac{i-1}{wt(T)} \right\rfloor, \quad (8)$$

and hence,

$$d(T_i) = r(T_i) + \left\lceil \frac{i}{wt(T)} \right\rceil - \left\lfloor \frac{i-1}{wt(T)} \right\rfloor - 1. \quad (9)$$

From the definitions above, it should be clear that our notion of an intra-sporadic task system is obtained by allowing a task's PF-windows to be right-shifted from where they would appear if that task were synchronous and periodic. In addition, we allow a subtask to become eligible before the beginning of its PF-window. In the proofs given later, we sometimes find it convenient to refer to the quantity by which a PF-window $w(T_i)$ is right-shifted. This quantity is given by

$$\Delta(T_i) = r(T_i) - \left\lfloor \frac{i-1}{wt(T)} \right\rfloor. \quad (10)$$

Note that the term $\lfloor \frac{i-1}{wt(T)} \rfloor$ gives the beginning of $w(T_i)$ in a synchronous, periodic system. From Equations (9) and (10), we have

$$d(T_i) = \left\lceil \frac{i}{wt(T)} \right\rceil - 1 + \Delta(T_i). \quad (11)$$

Using the definitions given above, it can be shown that each subtask is right-shifted by an amount that is at least that of its predecessor, i.e.,

$$i \geq j \Rightarrow \Delta(T_i) \geq \Delta(T_j). \quad (12)$$

Some special cases. Using the definitions above, we can show that sporadic and asynchronous tasks are special cases of intra-sporadic tasks. In particular, a *sporadic task* T is an intra-sporadic task in which only the first subtask of each job may be released late, i.e., if T_i and T_{i+1} are part of the same job, then $\Delta(T_i) = \Delta(T_{i+1})$. An *asynchronous task* T is an intra-sporadic task such that only the very first subtask of each task may be released late, i.e., $\Delta(T_i) = \Delta(T_1)$ for all $i \geq 1$. Note that, by defining the function e appropriately, we can obtain eligibility intervals (i.e., IS-windows) like those in either a Pfair or ERfair system. In fact, we can define eligibility intervals that are longer than in a Pfair system but shorter than in an ERfair system.

Validity of schedules. A schedule for a system of intra-sporadic tasks on M processors is said to be *valid at time* t iff the following properties are satisfied.

(P1) Each subtask scheduled at time t is scheduled within its IS-window.

(P2) No two subtasks of the same task are scheduled at time t.

(P3) There are at most M tasks scheduled at time t.

A schedule is said to be *valid* iff it is valid at every time slot.

In the full paper [1], we prove that an intra-sporadic task system τ is schedulable on M processors iff

$$\sum_{T \in \tau} \frac{T.e}{T.p} \leq M. \quad (13)$$

The proof is based on a network flow construction that is similar to that given by Baruah et al. [5] to prove that Expression (4) is a feasibility condition for synchronous, periodic tasks. In fact, the proof shows that there exists a valid schedule in which each subtask is scheduled in its PF-window. In the next section, we

```
slot number: 0    1    2    3

                        1    2
   A(3x1/2): 3    _

                        _    1
                        2
   B(2x3/4): _    _
                  2
```

Figure 3: An example showing that the EPDF algorithm is not optimal on a system of three processors

show that the *earliest pseudo-deadline first* (EPDF) algorithm can be used to correctly schedule any feasible set of intra-sporadic tasks on two processors. (EPDF is essentially the same as PD, but without the tie-breaks.)

One may wonder how an intra-sporadic task T's lag changes with time. When using EPDF to schedule intra-sporadic tasks, the notion of lag is completely irrelevant. In particular, if we merely require the scheduler to maintain a count of the subtasks that have been released in the current job of T, then T's current priority can always be determined by using the formulae given above. These formulae also do not depend on T's current lag. Note that if we were to extend our model to allow tasks to join and leave the system dynamically, as considered in work on proportional-share scheduling [7], then the notion of lag *would* be relevant. Such systems will be considered in future work.

4 Two-Processor Systems

In this section, we prove that EPDF is optimal for scheduling intra-sporadic tasks on a system of two processors. The proof of optimality is complicated by the fact that successive windows of a task may overlap. Indeed, precisely because of this reason, we cannot directly extend the proof to work for systems of three or more processors. In fact, the following theorem shows that EPDF is not optimal for such systems.

Theorem 1 *The EPDF algorithm is not optimal for scheduling synchronous, periodic tasks on three or more processors.*

Proof: Consider a task set consisting of a set A of three tasks of weight 1/2 and a set B of two tasks of weight 3/4. The total utilization of this task set is 3 and therefore it is feasible on 3 processors. Figure 3 shows one of the possible schedules that can be obtained using EPDF. Note that only one of the set-B tasks is scheduled in slot 3 — the other misses its deadline. □

Corollary 1 *The EPDF algorithm is not optimal for scheduling intra-sporadic tasks on three or more processors.*

We now turn our attention to proving that the EPDF *is* optimal for two-processor systems. The proof is by contradiction. Throughout this section, we let τ denote a feasible task system that (by assumption) misses a deadline when scheduled on two processors using EPDF. Let t_d be the earliest time at which a deadline is missed in τ. A contradiction is reached by proving that EPDF correctly schedules τ over $[0, t_d]$. This is shown by considering certain schedules over $[0, t_d]$. To facilitate the description of these schedules, we find it convenient to totally order all subtasks in τ that have deadlines in $[0, t_d]$. Let T_i and U_j be two such subtasks. Then, T_i is ordered before U_j, denoted $T_i \trianglelefteq U_j$ iff $d(T_i) \leq d(U_j)$. Let $\triangleleft$ be an irreflexive total order that is consistent with $\trianglelefteq$, i.e., $\triangleleft$ is obtained by arbitrarily breaking any ties left by $\trianglelefteq$. We define the *rank* of subtask T_i to be its position in the total order $\triangleleft$. If $T_i \triangleleft U_j$, then we say that T_i has *higher rank* than U_j.

We define a schedule S to be *k-compliant* iff **(i)** each subtask that is not among the first k according to the relation $\triangleleft$ is scheduled within its PF-window, and **(ii)** the first k subtasks according to $\triangleleft$ are scheduled in accordance with EPDF. We will prove that a k-compliant schedule exists by induction on k. Note that a 0-compliant schedule is any valid schedule such that each subtask is scheduled in its PF-window and the existence of such a schedule follows from the feasibility proof given in the full paper [1]. Also, if N is the total number of subtasks with deadlines in $[0, t_d]$, then the existence of an N-compliant schedule contradicts the fact that there is a missed deadline at t_d.

We now state and prove several lemmas. In the first of these lemmas, an interval of time slots $[t, u]$ is considered, and a set of conditions is stated that is sufficient to conclude that some subtask scheduled in $[t, u]$ can be shifted left or right out of this interval. In the rest of this section, we call a slot *nonfull* if one or both processors are idle during that slot.

Lemma 1 *Let S be a two-processor schedule for $\{T_i \mid T \in \tau \ \wedge \ d(T_i) \leq t_d\}$ that is valid over $[t, u]$, where $t < u$. Suppose that there are no nonfull slots in $[t, u]$ and that there exists a subtask U_j not scheduled in $[t, u]$ such that $d(U_j) \geq t$. Then, there exists a subtask T_i scheduled in $[t, u]$ such that $r(T_i) < t$ or $d(T_i) > u$.*

Proof: Let A be the set of all subtasks scheduled by S in $[t, u]$. Suppose, to the contrary, that

$$(\forall T_i : T_i \in A :: r(T_i) \geq t \ \wedge \ d(T_i) \leq u).$$

We derive a contradiction by showing that total utilization exceeds two, which contradicts the fact that τ is feasible. Let V be a task with subtasks in A. Let $V.n$ denote the number of such subtasks in A, and let V_k (respectively, V_l) be the first (respectively, last) subtask of V scheduled in $[t, u]$. Then, $V.n = l - k + 1$. Because V is in A, $d(V_l) \leq u$ and $r(V_k) \geq t$. Thus,

$$d(V_l) - r(V_k) \leq u - t. \tag{14}$$

By Equations (10) and (11) in Section 3, we have $d(V_l) = \lceil \frac{l}{wt(V)} \rceil - 1 + \Delta(V_l)$ and $r(V_k) = \lfloor \frac{k-1}{wt(V)} \rfloor + \Delta(V_k)$. Substituting these expressions in Equation (14), we get $\lceil \frac{l}{wt(V)} \rceil - 1 + \Delta(V_l) - \lfloor \frac{k-1}{wt(V)} \rfloor - \Delta(V_k) \leq u - t$. By (12), this implies that $\lceil \frac{l}{wt(V)} \rceil - 1 - \lfloor \frac{k-1}{wt(V)} \rfloor \leq u - t$. Hence, $\lceil \frac{l}{wt(V)} \rceil - \lfloor \frac{k-1}{wt(V)} \rfloor \leq u - t + 1$. Because $\lceil \frac{l}{wt(V)} \rceil \geq \frac{l}{wt(V)}$ and $\lfloor \frac{(k-1)}{wt(V)} \rfloor \leq \frac{(k-1)}{wt(V)}$, we have $l \cdot \frac{V.p}{V.e} - (k-1) \cdot \frac{V.p}{V.e} \leq u - t + 1$. This implies that $\frac{V.p}{V.e} \leq \frac{u-t+1}{l-k+1}$, which implies that

$$\frac{V.e}{V.p} \geq \frac{V.n}{u - t + 1}. \tag{15}$$

We now show that this inequality can be strengthened for U, yielding

$$\frac{U.e}{U.p} > \frac{U.n}{u - t + 1}. \tag{16}$$

If $U.n = 0$, then the above expression clearly holds, so assume that $U.n \neq 0$. Then, by the statement of the lemma, U_{j+1} must be the first subtask of U scheduled in $[t, u]$. Let U_h be the last such subtask. Then,

$$d(U_h) - r(U_{j+1}) \leq u - t. \tag{17}$$

Because $d(U_j) \geq t$, we have $r(U_{j+1}) \geq t$. If $r(U_{j+1}) > t$, then $d(U_h) - r(U_{j+1}) < u - t$. By reasoning as above (refer to (14)), this implies that $\frac{U.e}{U.p} > \frac{U.n}{u-t+1}$.

On the other hand, if $r(U_{j+1}) = t$, then $d(U_j) = t$, which implies that $b(U_j) = 1$. From Equations (10) and (11), we have $d(U_h) = \lceil \frac{h}{wt(U)} \rceil - 1 + \Delta(U_h)$ and $r(U_{j+1}) = \lfloor \frac{(j+1)-1}{wt(U)} \rfloor + \Delta(U_{j+1})$. Substituting these expressions into (17), we get

$$\left\lceil \frac{h}{wt(U)} \right\rceil - 1 + \Delta(U_h) - \left\lfloor \frac{j}{wt(U)} \right\rfloor - \Delta(U_{j+1}) \leq u - t.$$

Because $b(U_j)$ is 1, $\lfloor \frac{j}{wt(U)} \rfloor < \frac{j}{wt(U)}$. Also, by (12), $\Delta(U_h) \geq \Delta(U_{j+1})$. It follows that

$$\frac{h}{wt(U)} - \frac{j}{wt(U)} < u - t + 1.$$

Therefore, by reasoning that is similar to that prior to Expression (15), we have $\frac{U.e}{U.p} > \frac{U.n}{u-t+1}$.

From (15) and (16), we conclude that

$$\sum_{V \in A \cup \{U\}} \frac{V.e}{V.p} > \sum_{V \in A} \frac{V.n}{u - t + 1}.$$

Because there are a total of $2(u - t + 1)$ slots in $[t, u]$ and two subtasks are scheduled in each slot, we have

$$\sum_{V \in A \cup \{U\}} V.n = 2(u - t + 1).$$

Therefore,

$$\sum_{V \in A \cup \{U\}} \frac{V.e}{V.p} > 2.$$

This contradicts the fact that the total utilization of τ is at most 2. Hence, there exists a subtask T_i scheduled in $[t, u]$ such that either $r(T_i) < t$ or $d(T_i) > u$. □

The lemma above actually holds for any number of processors. In contrast, Lemma 2, given next, is valid only for two-processor systems. This lemma gives a set of conditions under which a subtask U_j can be right-shifted to a later slot.

Lemma 2 *Let S be a two-processor schedule for $\{T_i \mid T \in \tau \ \wedge \ d(T_i) \leq t_d\}$. Let $t \leq t_d$. Assume the following:* (1) *subtask U_j is scheduled at slot t, and t is not the last slot of U_j's PF-window;* (2) *either no other subtask is scheduled at slot t or U_{j-1} is scheduled there (in which case S is not valid at t);* (3) *each subtask scheduled at or after t is scheduled in its PF-window and S is valid for all $v > t$. Then, there exists a schedule S' for $\{T_i \mid T \in \tau \ \wedge \ d(T_i) \leq t_d\}$ such that* (1) *U_j is scheduled at some slot after t;* (2) *$S_v = S'_v$ for all $v < t$;* (3) *each subtask scheduled at or after t is scheduled in its PF-window and S is valid for all $v \geq t$. Moreover, if t is nonfull in S, then it is also nonfull in S'.*

Proof: We prove the lemma by inducting over the rank of U_j. Because t is not the last slot of U_j's PF-window, the deadline of U_j is after t, i.e.,

$$d(U_j) \geq t + 1. \tag{18}$$

Base case: *U_j is the lowest-ranked subtask scheduled in S.* If no subtasks are scheduled after slot t in S, then by (18), we can clearly shift U_j to slot $t + 1$. In the rest of the proof for the base case, we assume that there exist subtasks that are scheduled after slot t.

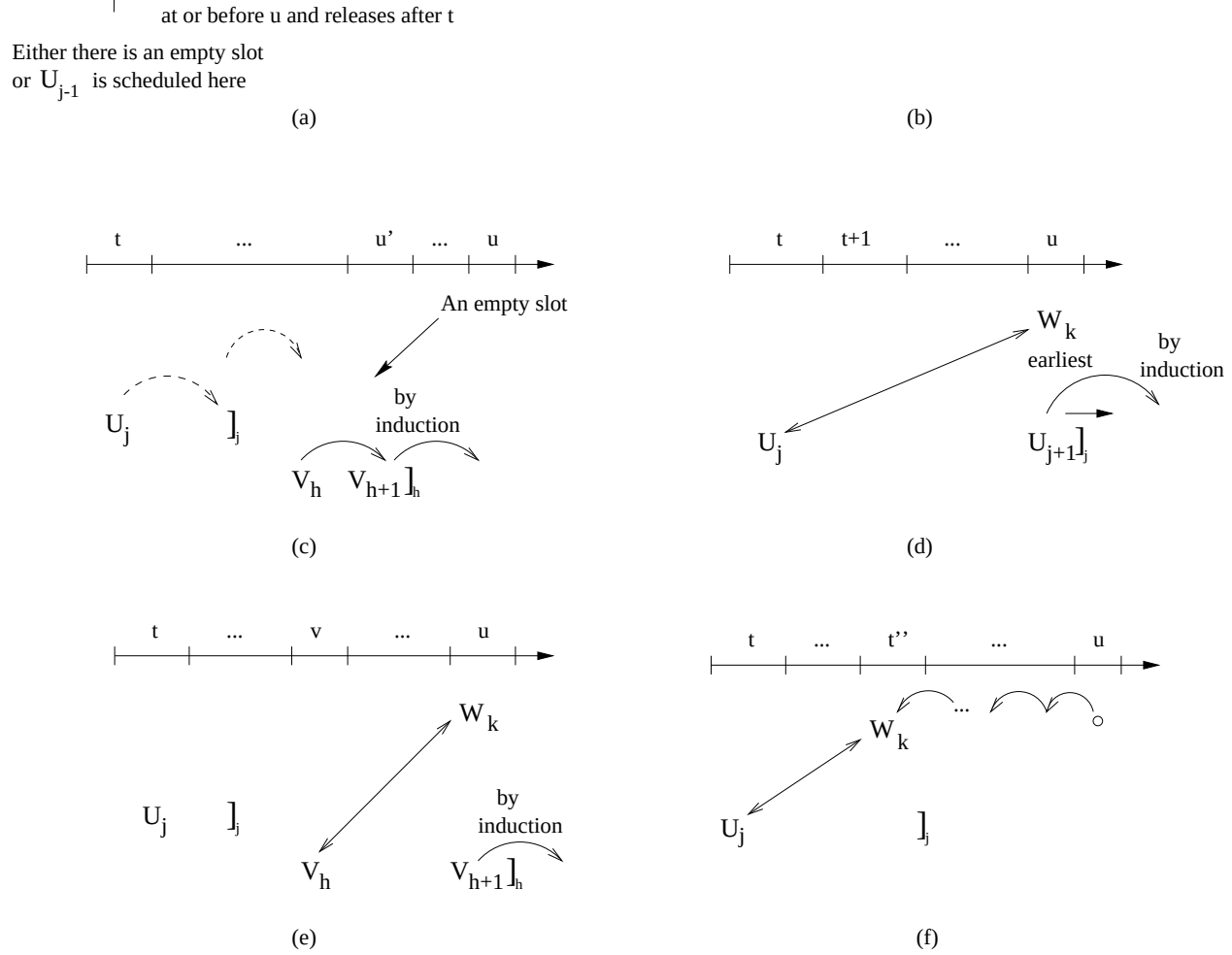

Figure 4: Cases of Lemma 2. We use the following notation in this and the next figure. "[" and "]" indicate the release and deadline of a subtask; subscripts indicate which subtask. Each task is shown on a separate line. An arrow from subtask T_i to subtask U_j indicates that T_i is now scheduled in place of U_j. An arrow over "[" (or "]") indicates that the actual position of "[" (or "]") can be anywhere in the direction of the arrow. Time is divided into unit-time slots that are numbered. If T_i is released at slot t, then "[" is aligned with the left side of slot t. If T_i has a deadline at slot t, then "]" is aligned with the right side of slot t. In each inset, $d(U_j) > t$. **(a)** Case 1: Smallest interval $[t+1, u-1]$ such that all subtasks have deadlines and releases inside the interval. **(b)** Subcase 1.A: $u' = t+1$. **(c)** Subcase 1.B: $u' > t+1$. **(d)** Subcase 2.A: $d(U_j) \geq u$, and W_k is the earliest subtask with $r(W_k) < t+1$. **(e)** Subcase 2.B: $d(U_j) < u$. **(f)** Repeated application of Subcase 2.B to finally apply Subcase 2.A.

Suppose that there exists a subtask W_k scheduled after t such that $r(W_k) \leq t$. By the statement of the lemma, W_{k-1} is not scheduled in slot t. Thus, we can swap W_k with U_j to get the desired schedule. (Note that U_{j+1} does not exist in S because U_j is of lowest rank. Thus, swapping W_k with U_j will not create a schedule in which two subtasks of U are scheduled in the same slot.)

The remaining possibility is that, for each subtask W_k scheduled after t, $r(W_k) > t$ and $d(W_k) \leq d(U_j)$ (the latter follows because U_j is of lowest rank). If there are no nonfull slots in $[t+1, d(U_j)]$, then we have a contradiction of Lemma 1. Therefore, there exists a nonfull slot in $[t+1, d(U_j)]$. This implies that we can schedule U_j in that slot. Thus, a valid schedule exists in which U_j is scheduled at a slot later than t.

Induction step: *U_j is not the lowest-ranked subtask scheduled in S.* Assume that the lemma holds for all subtasks with lower rank than U_j. We consider two cases.

Case 1: *For each subtask W_k scheduled after t, $r(W_k) > t$.* Let $[t+1, u]$ be the smallest interval such that for each subtask V_h scheduled in $[t+1, u]$, $r(V_h) \geq t+1$ and $d(V_h) \leq u$ (see Figure 4(a)). Note that such a u exists, because S includes only a finite collection of subtasks. If there are no nonfull slots in this interval, then we have a contradiction of Lemma 1. Therefore, there exists a slot u' in $[t+1, u]$ at which one or both processors are idle. Without loss of generality, let u' be the earliest such slot in $[t+1, u]$. Either $u' = t+1$ or $u' > t+1$.

Subcase 1.A: $u' = t+1$. By (18), U_j can be shifted to slot u'. Unfortunately, subtask U_{j+1} might be scheduled there, in which case the resulting schedule would be invalid (see Figure 4(b)). However, by the induction hypothesis, U_{j+1} can be shifted to a later slot. This implies that the desired schedule exists.

Subcase 1.B: $u' > t+1$. If for all subtasks V_h sched-

uled in $[t+1, u'-1]$, $d(V_h) \leq u'-1$, then we have a contradiction of the fact that $[t+1, u]$ was the smallest such interval. Therefore, there exists a subtask V_h such that $d(V_h) \geq u'$ (refer to Figure 4(c)). Note that V_h can be U_j itself. Observe that we can move V_h to slot u' to get a schedule in which the nonfull slot occurs earlier. Unfortunately, this movement of V_h is not valid if V_{h+1} is scheduled at u'. However, by the induction hypothesis, if V_{h+1} *is* scheduled at u', then there exists a schedule in which it is scheduled at a slot later than u' and slot u' is still nonfull.

By the reasoning above, if Subcase 1.B applies, then it is always possible to get a schedule in which either U_j is shifted to a later slot, or in which the first nonfull slot in $[t+1, u]$ occurs earlier. If we repeatedly apply Subcase 1.B without shifting U_j to a later slot, then Subcase 1.A will eventually apply, in which case U_j can be shifted as desired.

Case 2: *There exists a subtask W_k scheduled after t such that $r(W_k) \leq t$.* Without loss of generality, assume that W_k is the earliest such subtask. Let W_k be scheduled at slot u. If there are any nonfull slots in $[t+1, u]$, then W_k could be moved to the first such slot u', and the reasoning below applies with the smaller interval $[t+1, u']$. Thus, we can assume without loss of generality that there are no nonfull slots in $[t+1, u]$.

Subcase 2.A: $d(U_j) \geq u$. Refer to Figure 4(d). Because $d(U_j) \geq u$, we can swap U_j and W_k directly. Unfortunately, if U_{j+1} is scheduled at u, then this might result in a schedule in which U_j and U_{j+1} are scheduled in the same slot. However, as before, if U_{j+1} is scheduled in slot u, then we can apply the induction hypothesis to move it to a later slot.

Subcase 2.B: $d(U_j) < u$. In this subcase, we try to identify subtasks that can be used as an intermediate for swapping. Because W_k is the earliest scheduled subtask after t such that $r(W_k) \leq t$, for each subtask V_h scheduled in $[t+1, u]$, we have $r(V_h) \geq t+1$. Hence, because there are no nonfull slots in $[t+1, u]$, by Lemma 1, there exists a subtask V_h scheduled in $[t+1, u]$ for which $d(V_h) \geq u$. Let V_h be scheduled at time $v \in [t+1, u-1]$ (see Figure 4(e)). We can swap V_h and W_k to get a schedule in which W_k is scheduled earlier (i.e., nearer to slot t). This swapping will be valid only if W_{k-1} is not scheduled at v and V_{h+1} is not scheduled at u. Because $r(W_k) < t+1 \leq v$, W_{k-1} cannot be scheduled at v. On the other hand, V_{h+1} *can* be scheduled at time u. However, as before, we can apply the induction hypothesis to move V_{h+1} to a later slot.

By repeatedly applying Subcase 2.b, we will obtain either the required schedule or a schedule in which Subcase 2.A can be applied. This is illustrated in Figure 4(f). □

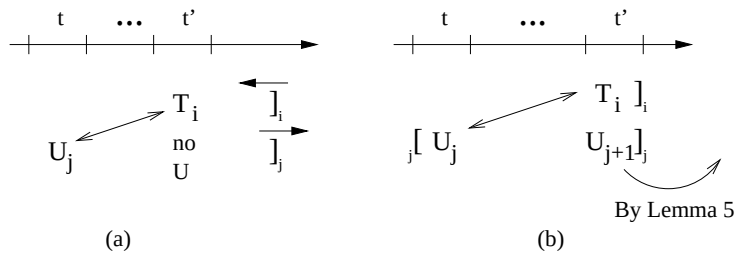

Figure 5: Cases of Lemma 3. **(a)** $t' < d(U_j)$ or U_{j+1} is not scheduled at t'. **(b)** $t' = d(U_j)$ and U_{j+1} is scheduled at t'.

The following lemma gives the inductive step of our proof.

Lemma 3 *Let S be a k-compliant two-processor schedule for $\{T_i \mid T \in \tau \ \wedge \ d(T_i) \leq t_d\}$. Then, there exists a $(k+1)$-compliant two-processor schedule S' for $\{T_i \mid T \in \tau \ \wedge \ d(T_i) \leq t_d\}$.*

Proof: Let T_i be the $(k+1)^{st}$ subtask according to $\lhd$. If T_i is scheduled in accordance with EPDF, then take S' to be S. Otherwise, we have the following: there exists a time slot t such that T_i is eligible at t, some subtask ranked lower than T_i according to $\lhd$ is scheduled at t, and T_i is scheduled at a slot later than t. Without loss of generality, let t be the earliest such slot and U_j be the lowest-ranked subtask scheduled at t. Let t' be the slot where T_i is scheduled.

Because T_i has higher rank than U_j, $d(T_i) \leq d(U_j)$. Because T_i is scheduled at t', this implies that $t' \leq d(T_i) \leq d(U_j)$. Because $t' \leq d(U_j)$, we have $t' \leq r(U_{j+1})$, and therefore U_{j+1} is scheduled at or after t'.

If U_{j+1} is not scheduled at t', then we can directly swap T_i and U_j to get the required schedule (see Figure 5(a)). On the other hand, if U_{j+1} *is* scheduled at t', then $r(U_{j+1}) = t'$. In this case, directly swapping T_i and U_j results in U_j and U_{j+1} being scheduled in the same slot. However, by Lemma 2, there exists a schedule in which U_{j+1} is scheduled after slot t'. The resulting schedule is valid. This is illustrated in Figure 5(b).

Thus, in all cases, we can show the existence of a schedule in which T_i is scheduled in accordance with EPDF. □

Theorem 2 *EPDF optimally schedules intra-sporadic tasks on systems of one or two processors.*

Proof: Establishing the optimality of EPDF for one-processor systems is straightforward, so we consider

only two-processor systems. Suppose to the contrary, that EPDF is not optimal for such systems. Then, there exists a task system τ that is feasible but not schedulable using EPDF. Let t_d be the earliest time at which τ misses a deadline under EPDF. Then, the set of subtasks $\{T_i \mid T \in \tau \wedge d(T_i) \leq t_d\}$ misses a deadline at t_d as well. Because τ is feasible, there exists a valid schedule for this set of subtasks such that each subtask is scheduled in its PF-window. Starting with this schedule, we can apply Lemma 3 inductively (as discussed earlier) to get a valid schedule for $\{T_i \mid T \in \tau \ \wedge \ d(T_i) \leq t_d\}$ under EPDF. Contradiction. □

5 Concluding Remarks

Prior work on Pfair and ERfair scheduling has been almost exclusively limited to synchronous, periodic task systems. In this paper, we have defined the notion of an intra-sporadic task. This notion generalizes that of a sporadic task, which in turn generalizes the notion of an asynchronous task. We have stated and proved a feasibility condition for scheduling intra-sporadic tasks. We have also given a polynomial-time algorithm that can be used to optimally schedule intra-sporadic tasks on systems of one or two processors.

The development of an algorithm for efficiently scheduling intra-sporadic tasks on systems of three or more processors remains as an open problem. It follows from Corollary 1 that EPDF is not optimal in this case. We conjecture that ER-PD (see Section 2) *is* optimal for such systems, and we are hopeful that some of our proof techniques may be helpful in showing this. The main challenge is to find an M-processor counterpart of Lemma 2. Note that many of the swappings in Figure 4 would be potentially invalid in a system of three or more processors. For example, consider Figure 4(d). According to the statement of Lemma 2, either slot t is nonfull or U_{j-1} is scheduled there. On a two-processor system, this implies that W_{k-1} is not scheduled in slot t. On a system of three or more processors, this conclusion cannot be reached.

In the intra-sporadic task model, deadlines are assigned to each subtask on arrival. These deadlines are assigned in a way that ensures that bursts of subtasks can be tolerated. Thus, the intra-sporadic task model can effectively deal with jittered inputs or packet arrivals. The intra-sporadic task model has many characteristics in common with the *uniprocessor* rate-based execution (RBE) model proposed recently by Jeffay and Goddard [6]. In the RBE model, tasks execute at well-defined average rates, but have no constraints on their instantaneous rate of invocation. Formally, an RBE task T is characterized by a four-tuple (x, y, d, c) where y is an interval of time, x is the maximum number of jobs of T expected in any interval of length y, d is a relative job deadline, and c is the maximum execution time per job, i.e., $c = T.e$. The pair (x, y) is called the *rate specification* of T. In the RBE model, a task's jobs may sometimes arrive "early" or "late." However, a task with a rate specification (x, y) expects to execute, on average, x jobs in every interval of length y.

One may wonder whether there are substantial differences between the intra-sporadic and RBE rates. One obvious difference is that in the intra-sporadic task model, the basic unit of execution is a quantum-length subtask. In essence, this is tantamount to requiring all RBE jobs to be of the same length. We are doubtful that this restriction can be removed on a multiprocessor system without sacrificing optimality. A second difference is that RBE deadlines are determined in a different manner from subtask deadlines in an intra-sporadic task system. In the RBE model, if the first $c = T.e$ jobs of T all arrive at time 0, then each of these jobs will have a deadline of d. In contrast, in the intra-sporadic task model, if several subtasks of the same task arrive almost simultaneously, then each will be given a different deadline (corresponding to the end of its PF-window).

References

[1] J. Anderson and A. Srinivasan. Pfair scheduling: Beyond periodic task systems (full version). Available at `http://www.cs.unc.edu/~anderson/papers.html`.

[2] J. Anderson and A. Srinivasan. Early-release fair scheduling. In *Proc. of the 12th Euromicro Conference on Real-Time Systems*, pages 35–43, June 2000.

[3] J. Anderson and A. Srinivasan. A new look at pfair priorities. Technical Report TR00-023, University of North Carolina at Chapel Hill, Sept. 2000. Available at `http://www.cs.unc.edu/~anderson/papers.html`.

[4] S. Baruah, N. Cohen, C.G. Plaxton, and D. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15:600–625, 1996.

[5] S. Baruah, J. Gehrke, and C. G. Plaxton. Fast scheduling of periodic tasks on multiple resources. In *Proc. of the 9th International Parallel Processing Symposium*, pages 280–288, Apr. 1995.

[6] K. Jeffay and S. Goddard. The rate-based execution model. In *Proc. of the 20th IEEE Real-Time Systems Symposium*, pages 304–314, Dec. 1999.

[7] Ion Stoica, Hussein Abdel-Wahab, Kevin Jeffay, Sanjoy Baruah, Johannes Gehrke, and C. Greg Plaxton. A Proportional Share Resource Allocation Algorithm for Real-Time, Time-Shared Systems. In *Proc. of the 17th IEEE Real-Time Systems Symposium*, pages 288–299, Dec. 1996.