Efficient Scheduling of Soft Real-time Applications on Multiprocessors

Anand Srinivasan, *Member, IEEE* and James H. Anderson, *Member, IEEE*

***Abstract*— In soft real-time applications, tasks are allowed to miss their deadlines. Thus, less-costly scheduling algorithms can be used at the price of occasional violations of timing constraints. This may be acceptable if reasonable tardiness bounds (*i.e.*, bounds on the extent to which deadlines may be missed) can be guaranteed.**

In this paper, we consider soft real-time applications implemented on multiprocessors. Pfair scheduling algorithms are the only known means of optimally scheduling *hard* real-time applications on multiprocessors. For this reason, we consider the use of such algorithms here. In the design of Pfair scheduling algorithms, devising schemes to correctly break ties when several tasks have the same deadline is a critical issue. Such tie-breaking schemes entail overhead that may be unacceptable or unnecessary in soft real-time applications. In this paper, we consider the earliest pseudo-deadline first (EPDF) Pfair algorithm, which avoids this overhead by using *no* tie-breaking information. Our main contributions are twofold. First, we establish a condition for ensuring a given tardiness under EPDF. The condition for ensuring a tardiness of one is very liberal and should often hold in practice. Second, we present simulation results involving randomly-generated task sets, including those that do not satisfy our condition. In these experiments, deadline misses rarely occurred, and *no* misses by more than one quantum ever occurred.

***Index Terms*— Fairness, real-time scheduling, multiprocessors, quality-of-service, soft real-time applications, tardiness.**

I. INTRODUCTION

REAL-TIME SYSTEMS are typically categorized as being either *hard* or *soft*. In a hard real-time system, task deadlines must be guaranteed, while in a soft real-time system, some task deadlines may be (occasionally) missed. Examples of hard real-time systems include fly-by-wire controllers for airplanes, monitoring systems for nuclear reactors, and automotive braking systems. Examples of soft real-time systems include multimedia and gaming systems.

While deadline misses are tolerated in soft real-time systems, they are obviously undesirable, and system quality and performance may be negatively impacted if tasks miss their deadlines either too often or by too much. One criterion used to measure system quality in the study of soft real-time scheduling algorithms is *tardiness*. If a job (*i.e.*, task invocation) with a deadline at time d completes at time t, then its tardiness is $max(0, t - d)$. That is, if a job misses its deadline, then its tardiness indicates by how much.

In this paper, we consider the problem of scheduling soft real-time applications implemented on tightly-coupled multiprocessors, where minimizing tardiness is the main concern. Our specific focus is fair scheduling algorithms based on Pfairness [1]. Such algorithms are the only known way of optimally scheduling recurrent *hard* real-time tasks on multiprocessors [1]–[4]. Our main contribution is to show that Pfair scheduling algorithms that are optimal for hard real-time systems can be substantially simplified if deadline misses can be tolerated.

While optimality is certainly desirable, fairness is interesting in its own right because it results in *temporal isolation*, *i.e.*, each task's processor share is guaranteed even if other tasks "misbehave" by attempting to execute for more than their prescribed shares. For this reason, fair (uniprocessor) scheduling mechanisms have been proposed in the networking literature as a means for supporting differentiated service classes in connection-oriented networks [5]–[9]. (Here, packet transmissions from various connections or flows are the entities to be scheduled.) In addition, by using fair algorithms to schedule operating system activities, problems such as *receive livelock* [10] can be ameliorated. In the following paragraphs, we consider in more detail three soft real-time applications in distributed systems in which simplified fair scheduling is useful.

a) Connection scheduling in web servers: A web server may need to service a large number of connections concurrently, some of which involve audio or video streaming that requires quality-of-service (QoS) guarantees. Such guarantees can be ensured by using fair scheduling disciplines. To handle a large number of connections, a multiprocessor platform may be necessary [11], [12]. Ensim Corp. has deployed fair multiprocessor scheduling algorithms in its product line for this very reason [13]. (The fair algorithms employed by them use heuristics for which tardiness bounds have not been derived.)

b) Packet scheduling on parallel links: Next-generation multimedia applications such as immersive reality systems will result in high bandwidth usage. One way to increase network bandwidth is to install multiple parallel links between pairs of connected routers. The problem of scheduling packets on outgoing links at a router then becomes a multiprocessor scheduling problem [14]. Fairness is desirable in this setting, just as it is in single-link scheduling. Similar scheduling issues also arise in optical networks where wavelength-division-multiplexing (WDM) techniques are used to transmit multiple "light packets" at different wavelengths simultaneously [15].

c) Packet processing in multiprocessor routers: As a final example, consider *multiprocessor* router platforms that process multiple packets simultaneously. The need for multiprocessor platforms in this context is necessitated by the growing disparity between link capacities and processor speeds [16], [17]. Routers built using programmable network processors

This work was supported by NSF grants CCR 9972211, CCR 9988327, ITR 0082866, and CCR 0204312.

are destined to implement fairly complex packet-processing functions in software, making *processing* capacity (as opposed to *link* capacity) a critical resource to be managed [18]. In this setting, fair scheduling is needed to ensure that QoS guarantees can be provided to different flows.

Note that in each of the above applications, an extreme degree of fairness that requires *all* deadlines to be met is not warranted. That is, these systems fall within the class of soft real-time applications. Having motivated the usefulness of efficient fair multiprocessor scheduling algorithms for soft real-time systems, we now describe the fair scheduling concepts used in this paper in greater detail. (Formal definitions are given in Sec. II.) After that, we present a more detailed overview of the contributions of this paper.

Pfair scheduling. Periodic task systems can be optimally scheduled on multiprocessors using *Pfair* scheduling algorithms [1]–[3]. Under Pfair scheduling, each task must execute at a uniform rate, while respecting a fixed-size allocation quantum. Uniform rates are ensured by requiring the allocation error for each task to be always less than one quantum, where "error" is determined by comparing to an ideal fluid system. Due to this requirement, each task T is effectively subdivided into quantum-length *subtasks* $T_1, T_2, \ldots$ that must execute within *windows* of approximately equal lengths: if a subtask of a task T executes outside of its window, then T's error bounds are exceeded. The length and alignment of a task's windows is determined by its *weight* or *utilization*, which is the ratio of its per-job execution cost and period. A task's weight determines the processor share it requires. Fig. 1(a) illustrates the subtasks and windows for the first job of a periodic task of weight $8/11$. The end of a subtask's window defines a *pseudo-deadline* for that task; for example, in Fig. 1(a), subtask T_2 has a pseudo-deadline at time 3. (Fig. 1(b) gives an example of an *intra-sporadic* task, a notion generalizing the concept of a periodic task that is defined later in Sec. III.)

At present, three optimal Pfair scheduling algorithms are known: PF [1], PD [2], and PD^2 [3]. These algorithms prioritize subtasks on an earliest-pseudo-deadline-first (EPDF) basis, but differ in the choice of tie-breaking rules. PD^2 is the most efficient of the three and uses two tie-break parameters. In hard real-time systems, the choice of tie-breaking rules is crucial. In particular, if either of the PD^2 tie-breaks is eliminated, then task sets exist in which deadlines are missed [3].

Contributions. In this paper, we consider the following question: Under the simpler EPDF algorithm, in which no tie-break parameters are used, by how much can deadlines be missed? If deadline misses are rare under EPDF, then it may be preferable to use EPDF in soft real-time systems than the more-costly ones noted above. Nonetheless, it is important to note that the two tie-break parameters of PD^2 can be calculated and maintained with only an $O(1)$ cost. Still, using EPDF may be a much better choice. Consider the following.

- For good throughput, a router must process each packet very quickly (typically within 10 ns. in today's technology). Moreover, the number of bits that can be allocated for storing priority information in a packet header may be very limited. (Such information can be sent in packet headers to avoid storing per-flow information in routers [19], [20].) Thus, incurring an additional constant overhead to store and evaluate tie-breaking information may be costly and unnecessary.
- In prior work on fairness in networks [5]–[8], much work was done on mechanisms for reallocating spare bandwidth as connections are created and destroyed. In Pfair terminology, this amounts to *reweighting* tasks so that all processing capacity is utilized as tasks dynamically join and leave the system. As explained later, it is possible to reweight a task so that its next pseudo-deadline is preserved. However, weight changes can cause tie-breaking information to change. In the worst case, this may necessitate a complete resorting of the scheduler's priority queue at a $\Theta(N \log N)$ cost, where N is the number of tasks (connections). In a networking application, this cost might be incurred *every time a connection is created or destroyed*. If no tie-breaking information is maintained, this extra overhead can be eliminated.

The main contributions of this paper are twofold. First, we establish a condition for ensuring a given tardiness in EPDF-scheduled systems. The condition corresponding to a tardiness of one is very liberal and should often hold in practice. For example, it imposes no restrictions on systems of four or fewer processors. We also consider larger tardiness thresholds and briefly discuss conditions under which EPDF ensures zero tardiness. The latter provides us scenarios in hard real-time systems where EPDF is preferable over PF, PD, or PD^2. Second, we present simulation results involving randomly-generated task sets, including those that do not satisfy our condition (for any tardiness threshold). In these experiments, deadline misses rarely occurred, and no misses by more than one quantum ever occurred.

The rest of the paper is organized as follows. In Sec. II, we give relevant definitions related to Pfair scheduling, and in Sec. III, we describe the different task models considered in this paper. In Sec. IV, we present some schedulability results involving EPDF-scheduled hard real-time systems. In Sec. V, the tardiness bounds mentioned above are derived. Simulation results are presented in Sec. VI. We conclude in Sec. VII.

II. Pfair Scheduling

In defining notions relevant to Pfair scheduling, we limit attention (for now) to periodic tasks.[1] A periodic task T with an integer *period* $T.p$ and an integer *execution cost* $T.e$ has a *weight* $wt(T) = T.e/T.p$, where $0 < wt(T) \leq 1$. A task is called *light* if its weight is less than $1/2$, and *heavy* otherwise.

Pfair algorithms allocate processor time in discrete quanta; the time interval $[t, t+1)$, where t is a nonnegative integer, is called *slot* t. (Hence, *time* t refers to the beginning of slot t.) A task may be allocated time on different processors, but not in the same slot (*i.e.*, interprocessor migration is allowed but parallelism is not permitted). The sequence of allocation decisions over time defines a *schedule* S. Formally, $S : \tau \times \mathcal{N} \mapsto \{0, 1\}$, where τ is a set of tasks and $\mathcal{N}$ is the set of nonnegative integers. $S(T, t) = 1$ iff T is scheduled in slot t.

[1] Unless specified otherwise, we assume that each periodic task begins execution at time 0.

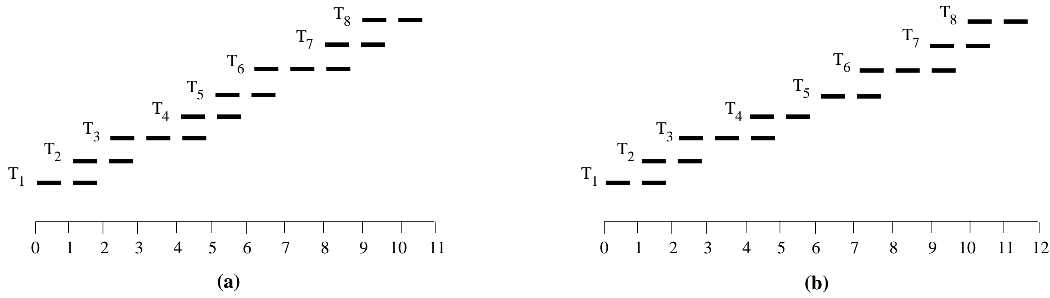

Fig. 1. **(a)** Windows of the first job of a periodic task T with weight $8/11$. This job consists of subtasks $T_1, \ldots, T_8$, each of which must be scheduled within its window, or else a lag-bound violation will result. (This pattern repeats for every job.) **(b)** The Pfair windows of an IS task. Subtask T_5 becomes eligible one time unit late.

In any schedule for M processors, $\sum_{T \in \tau} S(T,t) \leq M$ holds for all t.

Lags and subtasks. The notion of a Pfair schedule is defined by comparing such a schedule to an ideal fluid schedule, which allocates $wt(T)$ processor time to task T in each slot. Deviance from the fluid schedule is formally captured by the concept of *lag*. Formally, the *lag of task T at time t* is[2] $lag(T,t) = wt(T) \cdot t - \sum_{u=0}^{t-1} S(T,u)$. T is said to be *over-allocated* at time t if $lag(T,t) < 0$, and *under-allocated* if $lag(T,t) > 0$. A schedule is defined to be *Pfair* iff

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

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

These lag bounds have the effect of breaking each task T into an infinite sequence of *quantum-length subtasks*. We denote the ith subtask of task T as T_i, where $i \geq 1$. As in [1], we associate a *pseudo-release* $r(T_i)$ and *pseudo-deadline*[3] $d(T_i)$ with each subtask T_i, as follows. (For brevity, we often drop the prefix "pseudo-.")

$$r(T_i) = \left\lfloor \frac{i-1}{wt(T)} \right\rfloor \tag{2}$$

$$d(T_i) = \left\lceil \frac{i}{wt(T)} \right\rceil \tag{3}$$

To satisfy (1), T_i must be scheduled in the interval $w(T_i) = [r(T_i), d(T_i))$, termed its *window*. Note that $r(T_{i+1})$ is either $d(T_i) - 1$ or $d(T_i)$. Thus, consecutive windows either overlap by one slot or are disjoint. The *length* of T_i's window, denoted $|w(T_i)|$, is $d(T_i) - r(T_i)$. As an example, consider subtask T_1 in Fig. 1(a). Here, we have $r(T_1) = 0$, $d(T_1) = 2$ and $|w(T_1)| = 2$. Therefore, T_1 must be scheduled at either time 0 or time 1.

Pfair scheduling algorithms. In earlier work [21], we proved that the earliest-pseudo-deadline-first (EPDF) Pfair algorithm is optimal on one or two processors, but not on more than two processors. As its name suggests, EPDF gives higher priority to subtasks with earlier deadlines. A tie between subtasks with equal deadlines is broken arbitrarily. At present, three Pfair scheduling algorithms are known to be optimal on an arbitrary number of processors: PF [1], PD [2], and PD^2 [3]. These algorithms prioritize subtasks on an EPDF basis, but differ in the choice of tie-breaking rules. Selecting appropriate tie-breaks turns out to be *the* most important concern in designing optimal Pfair algorithms. PD^2 is the most efficient among the three Pfair algorithms cited above and it uses two tie-break parameters, which are described below.

The first tie-break parameter used by PD^2 is a bit, denoted by $b(T_i)$. By (2) and (3), $r(T_{i+1})$ is either $d(T_i)$ or $d(T_i) - 1$, *i.e.*, consecutive windows are either disjoint or overlap by one slot. $b(T_i)$ distinguishes between these two possibilities:

$$b(T_i) = \left\lceil \frac{i}{wt(T)} \right\rceil - \left\lfloor \frac{i}{wt(T)} \right\rfloor \tag{4}$$

For example, in Fig. 1(a), $b(T_i) = 1$ for $1 \leq i \leq 7$ and $b(T_8) = 0$. PD^2 favors a subtask with a b-bit of 1 over one with a b-bit of 0. Informally, it is better to execute T_i "early" if its window overlaps that of T_{i+1}, because this potentially leaves more slots available to T_{i+1}.

The second PD^2 tie-break, the "group deadline," is needed in systems with heavy tasks of weight less than one. It is easy to show that all windows of such a task are of length two or three (see Fig. 1(a)). Consider a sequence $T_i, \ldots, T_j$ of subtasks of such a task T such that $b(T_k) = 1$ and $|w(T_{k+1})| = 2$ for all $i \leq k < j$. Then, scheduling T_i in its last slot forces the other subtasks in this sequence to be scheduled in their last slots. For example, in Fig. 1(a), scheduling T_3 in slot 4 forces T_4 and T_5 to be scheduled in slots 5 and 6, respectively. A group deadline corresponds to a time by which any such "cascade" of scheduling decisions must end. Formally, it is a time t such that either $(t = d(T_i) \wedge b(T_i) = 0)$ or $(t + 1 = d(T_i) \ \wedge \ |w(T_i)| = 3)$ for some subtask T_i. For example, the task in Fig. 1(a) has group deadlines at times 4, 8, and 11.

The group deadline of subtask T_i is denoted $D(T_i)$. If T is a heavy task of weight less than one, then $D(T_i)$=(**min** u :: $u \geq d(T_i)$ and u is a group deadline of T). For example, in Fig. 1(a), $D(T_1) = 4$ and $D(T_6) = 11$. If T is light or of weight one, then $D(T_i) = 0$. In the event of a tie between heavy tasks, PD^2 favors the subtask with the larger group deadline because *not* scheduling it can lead to a longer cascade, which places more constraints on the future schedule.

[2]For conciseness, we leave the schedule implicit and use $lag(T,t)$ instead of $lag(T,t,S)$.

[3]In our earlier work [3], [4], [21], pseudo-deadlines were defined to refer to slots; here, they refer to time. Hence, the formula for $d(T_i)$ given here is slightly different.

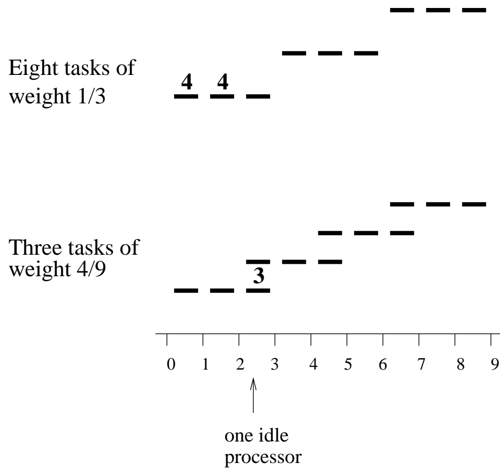

Fig. 2. A partial schedule on four processors is shown for a task set consisting of eight tasks of weight $1/3$ and four tasks of weight $4/9$ (tasks of a given weight are shown together). The Pfair window of each subtask is shown on a separate line. An integer value n in slot t means that n of the corresponding subtasks are scheduled in slot t. No integer value means that no such subtask is scheduled in slot t.

Having explained the notion of a group deadline, we can now state the PD^2 priority definition.

PD^2 Priority Definition: Subtask T_i's priority at slot t is defined to be $(d(T_i), b(T_i), D(T_i))$, if it is eligible at t. Priorities are ordered using the following relation.

$$(d, b, D) \succeq (d', b', D') \equiv [d < d'] \vee [(d = d') \wedge (b > b')] \\ \vee \; [(d = d') \wedge (b = b') \wedge (D \geq D')]$$

If T_i and U_j are both eligible at t, then priority of subtask T_i is at least that of U_j at t if $(d(T_i), b(T_i), D(T_i)) \succeq (d(U_j), b(U_j), D(U_j))$. ■

According to the definition above, T_i has higher priority than U_j if it has an earlier deadline. If T_i and U_j have equal deadlines, but $b(T_i) = 1$ and $b(U_j) = 0$, then the tie is broken in favor of T_i. If T_i and U_j have equal deadlines and b-bits, then the tie is broken in favor of the one with the later group deadline. Any ties not resolved by PD^2 can be broken arbitrarily.

In earlier work [3], we showed that both PD^2 tie-breaks are necessary for optimality. In particular, if either is eliminated, then tasks can miss their deadlines. To see that the b-bit is necessary, consider Fig. 2. In the example schedule, the tasks of weight 1/3 are favored over those of weight 4/9 at times 0 and 1 even though the former have a b-bit of 0. Note that $\frac{8}{3} + \frac{4}{3} = 4$. Thus, all four processors are fully utilized, which implies that no processor should ever be idle. However, in $[2, 3)$, only three tasks can be scheduled, implying that a deadline is missed in the future. (Note that this schedule would be allowed by EPDF.) Other examples exist that show that the group deadline is also necessary.

III. Task Models

In this paper, we mainly consider the *intra-sporadic* (IS) task model proposed by us in earlier work [4], [21] because it provides a general notion of recurrent execution that subsumes that found in the well-studied periodic and sporadic models. The IS model generalizes the sporadic model. The sporadic model generalizes the periodic model by allowing *jobs* to be released "late"; the IS model allows *subtasks* to be released late, as illustrated in Fig. 1(b). More specifically, the separation between $r(T_i)$ and $r(T_{i+1})$ is allowed to be more than $\lfloor i/wt(T) \rfloor - \lfloor (i-1)/wt(T) \rfloor$, which would be the separation if T were periodic. Thus, an IS task is obtained by allowing a task's windows to be right-shifted from where they would appear if the task were periodic. Fig. 1(b) illustrates this.

Let $\theta(T_i)$ denote the offset of subtask T_i, *i.e.*, the amount by which $w(T_i)$ has been right-shifted. Then, by (2) and (3), we have the following.

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

$$d(T_i) = \theta(T_i) + \left\lceil \frac{i}{wt(T)} \right\rceil \quad (6)$$

The offsets are constrained so that the separation between any pair of subtask releases is at least the separation between those releases if the task were periodic. Formally, the offsets satisfy the following property.

$$k \geq i \Rightarrow \theta(T_k) \geq \theta(T_i) \quad (7)$$

Each subtask T_i has an additional parameter $e(T_i)$ that specifies the first time slot in which it is eligible to be scheduled. It is assumed that $e(T_i) \leq r(T_i)$ and $e(T_i) \leq e(T_{i+1})$ for all $i \geq 1$. Allowing $e(T_i)$ to be less than $r(T_i)$ is equivalent to allowing "early" subtask releases as in ERfair scheduling [3]. The interval $[r(T_i), d(T_i))$ is called the *PF-window* of T_i and the interval $[e(T_i), d(T_i))$ is called the *IS-window* of T_i. (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 a subtask from becoming eligible before the beginning of its job.)

The IS model is more suitable than the periodic model for the networking examples described in Sec. I. Due to network congestion and other factors, packets may arrive late or in bursts. The IS model treats these possibilities as first-class concepts and handles them more seamlessly. In particular, a late packet arrival corresponds to an IS delay. On the other hand, if a packet arrives early (as part of a bursty sequence), then its eligibility time will be less than its Pfair release time. Note that its Pfair release time determines its deadline. Thus, in effect, an early packet arrival is handled by postponing its deadline to where it would have been had the packet arrived on time. This is very similar to the approach taken in the (uniprocessor) virtual-clock scheduling scheme [9].

Feasibility. In [21], we showed that an IS task system τ is feasible on M processors iff

$$\sum_{T \in \tau} wt(T) \leq M. \quad (8)$$

In [4], we proved the following theorem, which implies the optimality of PD^2 for scheduling IS tasks.

Theorem 1: PD^2 correctly schedules any feasible IS task system on multiprocessors.

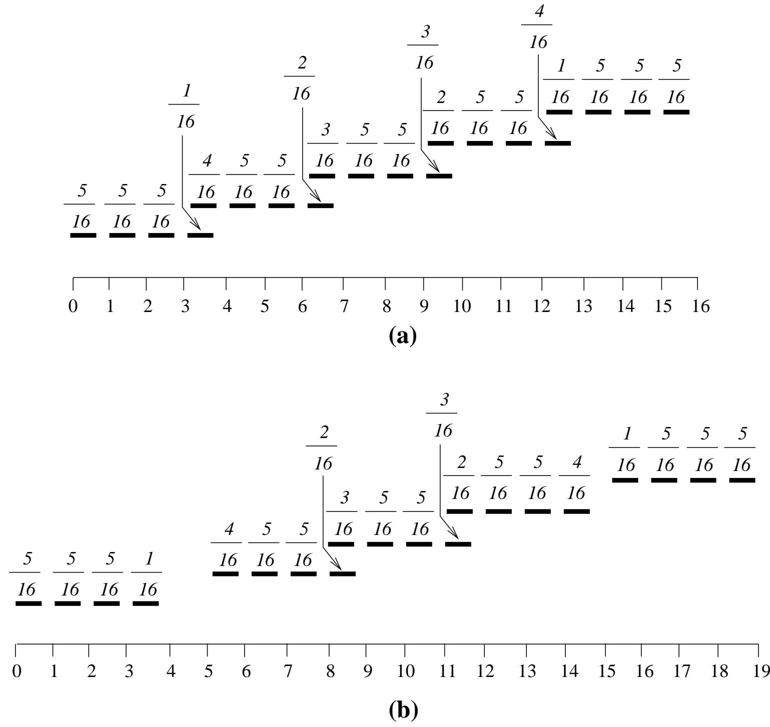

Fig. 3. Fluid schedule for the first five subtasks ($T_1, \ldots, T_5$) of a task T of weight $5/16$. The share of each subtask in each slot of its PF-window is shown. In **(a)**, no subtask is released late; in **(b)**, T_2 and T_5 are released late. Note that $share(T, 3)$ is either $5/16$ or $1/16$ depending on when subtask T_2 is released.

Shares and lags in IS task systems. The lag of T at time t is defined in the same way as it is defined for periodic tasks. Let $ideal(T, t)$ denote the share that T receives in an ideal fluid (processor-sharing) schedule in $[0, t)$. Then,

$$lag(T, t) = ideal(T, t) - \sum_{u=0}^{t-1} S(T, u). \quad (9)$$

Before defining $ideal(T, t)$, we define $share(T, u)$, which is the share assigned to task T in slot u. $share(T, u)$ is defined in terms of a function f that indicates the share assigned to each subtask in each slot.

$$f(T_i, u) = \begin{cases} (\left\lfloor \frac{i-1}{wt(T)} \right\rfloor + 1) \times wt(T) - (i-1), & u = r(T_i) \\ i - (\left\lceil \frac{i}{wt(T)} \right\rceil - 1) \times wt(T), & u = d(T_i) - 1 \\ wt(T), & r(T_i) < u < d(T_i) - 1 \\ 0, & \text{otherwise} \end{cases} \quad (10)$$

Fig. 3 shows the values of f for different subtasks of a task of weight $5/16$. $share(T, u)$ is simply defined as $share(T, u) = \sum_i f(T_i, u)$. Observe that $share(T, u)$ usually equals $wt(T)$, but in certain slots, it may be less than $wt(T)$, so that each subtask of T has a unit share. Using (10), it is easy to show the following.

(F1) For all time slots t, $share(T, t) \leq wt(T)$.

We can now define $ideal(T, t)$ as $\sum_{u=0}^{t-1} share(T, u)$. Hence, from (9), $lag(T, t+1) = \sum_{u=0}^{t}(share(T, u) - S(T, u)) = lag(T, t) + share(T, t) - S(T, t)$. Similarly, the total lag for a schedule S and task system τ at time $t + 1$, denoted by $LAG(\tau, t+1)$, is defined as follows. ($LAG(\tau, 0)$ is defined to be 0.)

$$LAG(\tau, t+1) = LAG(\tau, t) + \sum_{T \in \tau}(share(T, t) - S(T, t)). \quad (11)$$

A. Dynamic Tasks

In each of the examples considered in Sec. I, connections are not permanent. Thus, in addition to mechanisms for handling burstiness and delays, support for task dynamism is needed. When considering dynamic task systems, a key issue faced is that of devising conditions under which tasks may join and leave the system.

A condition for joining is an immediate consequence of the feasibility test in (8), *i.e.*, admit a task if the total utilization is at most M after its admission. The important question left is: *when should a task be allowed to leave the system*? (Here, we are referring to the time when the task's utilization can be reclaimed. The task may actually be allowed to leave the system earlier.) As shown in [23], [24], if an over-allocated task (a task with negative lag) is allowed to leave, then it can re-join immediately and effectively execute at a rate higher than its specified rate causing other tasks to miss their deadlines. It is easy to show that, in such a case, tardiness cannot be bounded by any constant value even on one processor.

Consider a task system consisting of one task of weight $\frac{1}{2}$ and $k \geq 2$ tasks of weight $\frac{1}{2k}$ to be EPDF-scheduled on one processor. At time 0, the task of weight $\frac{1}{2}$ is scheduled. Suppose that it leaves after completing its job and re-joins immediately. Again, at time 1, it will be assigned higher

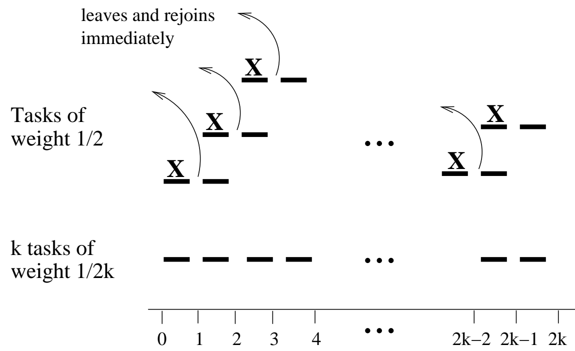

Fig. 4. An EPDF schedule for a dynamic task set. k tasks of weight $\frac{1}{2k}$ and one task of weight $\frac{1}{2}$ join at time 0. The PF-windows of all the subtasks are shown. The task of weight $\frac{1}{2}$ is scheduled in $[0, 1)$. (The time slot in which it is scheduled is denoted by an 'X'.) It leaves at time 1 and re-joins immediately. At time 1, it is scheduled again. This pattern repeats until time $2k - 2$ and leads to deadline misses for $k - 1$ of the tasks with weight $\frac{1}{2k}$.

priority and hence, will be scheduled again. Repeating this $2k - 2$ times gives us the schedule shown in Fig. 4. The task of weight $\frac{1}{2}$ effectively executes at a rate of $(2k-1)/2k$ over the interval $[0, 2k)$, and $k - 1$ of the tasks of weight $\frac{1}{2k}$ miss their deadlines at time $2k$. This implies that at least one subtask completes $k - 1$ time units after its deadline.

This example applies to PF, PD, and PD^2 as well. The above situation can be averted if all future task arrival times are known. However, the problem under consideration is inherently an online problem in which such information is not available. Therefore, as in [23], [24], we allow only non-over-allocated tasks (*i.e.*, tasks with non-negative lags) to leave the system.

(J) *Join condition*: A task T can join at time t iff the total utilization after joining is at most M. If T joins at time t, then $\theta(T_1)$ is set to t. (A task that re-joins after having left can be viewed as a new task.)

(L) *Leave condition*: A task T can leave at time t iff $t \geq d(T_i)$, where T_i is the last-released subtask of T.

The condition $t \geq d(T_i)$ implies that $lag(T, t) \geq 0$. To see why, note that since T_i is the last-released subtask of T, by time $d(T_i)$ it receives a share equal to i in the ideal schedule. In the EPDF schedule, it cannot have received more than i time units. Thus, $lag(T, t) \geq 0$.

Actually, assuming that tasks leave only with zero lag is consistent with Condition (L), so we assume this in our proofs (Sec. V). To see why this is reasonable, suppose that T leaves with positive lag; this implies that it has released more subtasks than have been scheduled. Let T_i be T's last-*released* subtask and T_k $(k < i)$ be its last-*scheduled* subtask. Since T leaves at time t, $T_{k+1}, \ldots, T_i$ do not need to be scheduled; thus, we can assume they were never released. In that case, the share in the ideal schedule for T equals k, and hence, $lag(T, t) = 0$.

Partitioning versus Pfair scheduling. One commonly-used approach in multiprocessor scheduling is partitioning, in which each task is statically assigned to a particular processor. Unfortunately, finding an optimal assignment of tasks to processors is NP-hard and non-optimal heuristics need to be used. Another problem with partitioning is that it is inherently sub-optimal: task sets with utilization at most M cannot always be partitioned. Nonetheless, partitioning has its advantages: migration overhead is zero and simpler and widely-studied uniprocessor scheduling algorithms can be used on each processor. However, partitioning is quite problematic if task dynamism is allowed. In particular, every new task that joins can potentially cause a re-partitioning of the entire system. The resulting overhead would clearly be unacceptable in many applications.

While the frequency of preemptions under Pfair scheduling may be a potential concern, a recent experimental comparison conducted by us and colleagues [25] showed that PD^2 has comparable performance (in terms of schedulability) to earliest-deadline-first (EDF) scheduling with partitioning. In this study, real system overheads such as context-switching costs were considered. Moreover, the study was biased *against* Pfair scheduling in that only static systems with independent[4] tasks of low utilization[5] were considered. In short, PD^2 and multiprocessor-EDF were comparable because the schedulability loss due to migration and preemption costs under PD^2 was comparable to the loss in schedulability due to partitioning under EDF.

B. Task Reweighting

As mentioned in Sec. I, task reweighting might be necessary to fully utilize all processors as tasks join and leave the system. A task can be reweighted by allowing it to leave with its old weight and re-join with its new weight. In EPDF-scheduled systems, this reweighting can be done in a simple manner using the following procedure, which ensures that the relative priorities of the current subtasks do not change. This procedure has the significant advantage that the scheduler's current priority queue of eligible subtasks does not have to be resorted when reweighting occurs. Suppose that task T needs to be reweighted at time t. Let T_i be the subtask of T that is eligible at t. Task T can be reweighted by simply replacing it by a new task U with the new weight and by aligning U_1's window so that $d(U_1) = d(T_i)$ and $e(U_1) \leq t$. (In reality, T's weight can simply be redefined, instead of creating a new task. Further, this new weight comes into effect only when the next subtask of that task is released, at which time the subtask's deadline is calculated using the new weight.) This procedure is problematic under PF, PD, or PD^2 because tie-break parameters also have to be matched to avoid changing task priorities, and this is not always possible (refer to Fig. 5).

[4]Considering only independent tasks is advantageous to EDF. While efficient synchronization techniques for Pfair-scheduled systems exist [26], to the best of our knowledge, no general synchronization protocols that are directly applicable to multiprocessor-EDF have been proposed in the literature. While the multiprocessor priority ceiling protocol [27] probably could be adapted for this purpose, its overhead in EDF-scheduled systems would likely be quite high.

[5]The performance of partitioning approaches is relatively good when individual task utilizations are low. As the mean task utilization increases, the schedulability loss due to partitioning increases. Note that tasks with high utilization may arise in systems that are hierarchically scheduled [28], [29].

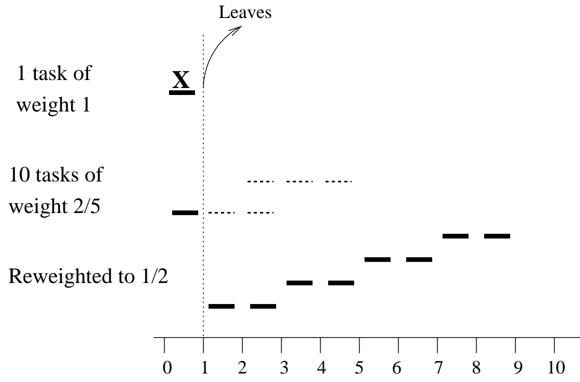

Fig. 5. A reweighting example on five processors. At time 0, there are ten tasks of weight $2/5$ and one task of weight 1. At time 1, the task of weight 1 leaves. The excess capacity of 1 is re-distributed among the remaining ten tasks. The new weight obtained is $1/2$. The dotted lines indicate the original PF-windows of those tasks. The new PF-windows can be aligned so that the new and old deadlines match. However, it is impossible to match the tie-break parameters used in PF, PD or PD^2, causing the priority of every subtask to change.

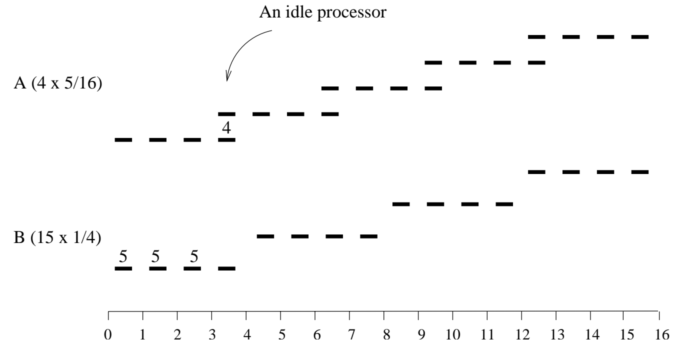

Fig. 6. Theorem 3. A partial EPDF schedule is depicted for four tasks of weight $5/16$ and 15 tasks of weight $1/4$ on 5 processors. (The notation used in this figure is the same as that in Fig. 2.) A processor is idle in slot 4; since the total utilization is 5, this implies a deadline miss in the future.

IV. Hard Real-time Systems

Though our focus in this paper is on soft real-time systems, we also present some results involving hard real-time systems scheduled using EPDF. It is sometimes useful to view a soft real-time system as a hard real-time system in order to simplify the specification of timing constraints. There are some cases in which EPDF is preferable to PD^2 for use in hard real-time systems because it provides the same guarantees as PD^2. In earlier work [22], we proved the following.

Theorem 2: EPDF correctly schedules any IS task system τ on M processors if $\sum_{T\in H} T.f < 1$, where $T.f = \frac{T.e - gcd(T.e, T.p)}{T.p}$ and H is any set of $M-1$ tasks in τ.

As a corollary of the above theorem, it follows that EPDF is optimal for scheduling IS tasks in one or two-processor systems, and optimal in a M-processor system if the weight of each task is at most $\frac{1}{M-1}$ [22]. We now show that this result is tight and later present different schedulability tests that do not place restrictions on individual task weights.

A. Tightness of Theorem 2

Let τ be a periodic task system consisting of the following $(M-1)^2$ tasks to be scheduled on $M(>2)$ processors: a set A consisting of $M-1$ tasks of weight $\frac{1}{M-1} + \frac{1}{(M-1)^2}$ $\left(= \frac{M}{(M-1)^2}\right)$ and a set B consisting of $M(M-2)$ tasks of weight $\frac{1}{M-1}$. (Fig. 6 illustrates the case for $M = 5$.) Note that τ fully utilizes the M processors because $(M-1) \times \frac{M}{(M-1)^2} + \frac{M(M-2)}{M-1} = \frac{M+M(M-2)}{M-1}$, which simplifies to $\frac{M^2-M}{M-1} = M$. Further note that for each task $T \in A$, $T.f = \frac{M-1}{(M-1)^2}$ because $gcd(M, (M-1)^2) = 1$. Thus, $\sum_{T\in A} T.f = (M-1) \times \left(\frac{1}{M-1}\right) = 1$.

We first show that for all tasks $T \in \tau$, $d(T_1) = M-1$. If $T \in B$, then by (3), we have $d(T_1) = \lceil M-1 \rceil = M-1$. If $T \in A$, then by (3), $d(T_1) = \left\lceil \frac{(M-1)^2}{M} \right\rceil$. Note that $\frac{(M-1)^2}{M} = \frac{M^2-2M+1}{M} = M-2+\frac{1}{M}$. Therefore, $d(T_1) = \lceil M-2+\frac{1}{M} \rceil = M-1$.

Thus, EPDF may assign higher priority to all the tasks in set B at time 0, thereby scheduling them for the first $M-2$ slots (refer to Fig. 6). The tasks in set A will be scheduled in slot $M-2$ (*i.e.*, the interval $[M-2, M-1)$). Because $r(T_2) = \lfloor M-1 \rfloor = M-1$ for all $T \in B$, no other tasks are eligible to be scheduled in slot $M-2$. Since A has only $M-1$ tasks, a processor will be idle in slot $M-2$. Because τ fully utilizes M processors, this implies that a deadline is missed in the future. Thus, we have the following theorem.

Theorem 3: There exists a feasible periodic (and hence, IS) task system τ that misses a deadline on M (>2) processors under EPDF if $\sum_{T\in H} T.f$ is allowed to be at least 1 for some set $H \subseteq \tau$ of at most $M-1$ tasks.

B. New Schedulability Results

We now present new schedulability tests that are based on the following result.

Theorem 4: EPDF correctly schedules any feasible IS task system if the weight of each task is a reciprocal of some integer.

Proof: Let τ be a feasible IS task system τ in which the weight of each task is a reciprocal of some integer, *i.e.*, for each task $T \in \tau$, $wt(T) = \frac{1}{k}$ for some integer k. We show that, for such a task system, the b-bit and the group deadline can be eliminated from the PD^2 priority definition without any change in its scheduling decisions. If $wt(T) = \frac{1}{k}$, then by (6), we have $d(T_i) = \theta(T_i) + ik$ for all i.

Also, by (4), $b(T_i) = \lceil ik \rceil - \lfloor ik \rfloor = 0$ for all i. Thus, the b-bit is zero for all subtasks of each task in τ. Further, by the definition of a group deadline, because $b(T_i) = 0$, it follows that $D(T_i) = d(T_i)$.

Thus, if $d(T_i) = d(U_j)$ for any two subtasks T_i and U_j, then $b(T_i) = b(U_j)$ and $D(T_i) = D(U_j)$. This implies that PD^2 breaks ties between equal subtask deadlines in an arbitrary manner, and hence behaves identically to EPDF. It follows from the optimality of PD^2 that EPDF correctly schedules τ. ■

Now, given any task system τ, we can transform it into a new task system τ' that satisfies the conditions of Theorem 4. We modify the parameters of every task in τ to obtain τ' as follows. For every task $T \in \tau$, we construct a task $T' \in \tau'$

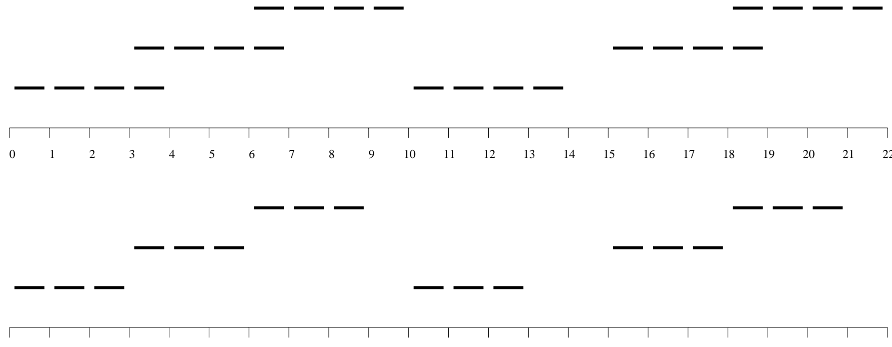

Fig. 7. Example illustrating transformation of a task T of weight $3/10$ into a task T' of weight $\frac{1}{\lfloor 10/3 \rfloor} = 1/3$. The subtask windows of T' are placed so that the release times of its subtasks coincide with those of T's subtasks.

with weight $\frac{1}{\lfloor 1/wt(T) \rfloor}$. The parameters of each subtask T'_i of T' is obtained from those of subtask T_i as follows.

(a) $e(T'_i) = e(T_i)$,

(b) $r(T'_i) = r(T_i)$,

(c) $\theta(T'_i) = r(T'_i) - \left\lfloor \frac{i-1}{wt(T')} \right\rfloor = r(T'_i) - \left\lfloor (i-1) \cdot \left\lfloor \frac{1}{wt(T)} \right\rfloor \right\rfloor = r(T'_i) - (i-1) \cdot \left\lfloor \frac{1}{wt(T)} \right\rfloor$, and

(d) $d(T'_i) = \theta(T'_i) + \left\lceil \frac{i}{wt(T')} \right\rceil = \theta(T'_i) + \left\lceil i \cdot \left\lfloor \frac{1}{wt(T)} \right\rfloor \right\rceil = \theta(T'_i) + i \cdot \left\lfloor \frac{1}{wt(T)} \right\rfloor$.

Thus, the window of T'_i is set to start at the same time as T_i's window. Fig. 7 illustrates this for a task of weight $3/10$. The scheduler schedules τ by effectively scheduling τ' using EPDF and selecting T_i for execution whenever EPDF selects T'_i.

By (c) and (d), $d(T'_i) = r(T'_i) + \left\lfloor \frac{1}{wt(T)} \right\rfloor$, and by (5) and (6), $d(T_i) = r(T_i) + \left\lceil \frac{i}{wt(T)} \right\rceil - \left\lfloor \frac{i-1}{wt(T)} \right\rfloor$. Thus, by (b), we have the following.

$$\begin{aligned} d(T_i) - d(T'_i) &= \left\lceil \frac{i}{wt(T)} \right\rceil - \left\lfloor \frac{i-1}{wt(T)} \right\rfloor - \left\lfloor \frac{1}{wt(T)} \right\rfloor \\ &\geq \left\lceil \frac{i}{wt(T)} \right\rceil - \left\lfloor \frac{i}{wt(T)} \right\rfloor \\ &\geq 0 \end{aligned}$$

(The second step above follows from the fact that $\lfloor x \rfloor + \lfloor y \rfloor \leq \lfloor x + y \rfloor$.) Thus, $d(T'_i) \leq d(T_i)$. Therefore, if T'_i meets its deadline, then T_i also meets its deadline. By Theorem 4, EPDF correctly schedules τ' on M processors if $\sum_{T' \in \tau'} wt(T') \leq M$. Thus, we have the following theorem.

Theorem 5: EPDF (with deadlines determined by (d)) correctly schedules τ on M processors if $\sum_{T \in \tau} \frac{1}{\lfloor 1/wt(T) \rfloor} \leq M$.

The following results follow as simple corollaries to Theorem 5.

Corollary 1: EPDF (with deadlines determined by (d)) correctly schedules τ on M processors if $\sum_{T \in \tau} \frac{wt(T)}{1-wt(T)} \leq M$.

Proof: Note that $\left\lfloor \frac{1}{wt(T)} \right\rfloor > \frac{1}{wt(T)} - 1$, *i.e.*, $\left\lfloor \frac{1}{wt(T)} \right\rfloor > \frac{1-wt(T)}{wt(T)}$. Therefore, $\frac{1}{\lfloor 1/wt(T) \rfloor} < \frac{wt(T)}{1-wt(T)}$. The required result follows from Theorem 5. ■

The following corollary states that EPDF correctly schedules any IS task system on M processors if its total utilization is at most $M/2$.

Corollary 2: EPDF (with deadlines determined by (d)) correctly schedules τ on M processors if $\sum_{T \in \tau} wt(T) \leq M/2$.

Proof: We show that $2 \cdot wt(T) \cdot \left\lfloor \frac{1}{wt(T)} \right\rfloor > 1$, which implies that $\frac{1}{\lfloor 1/wt(T) \rfloor} < 2 \cdot wt(T)$. The required result then follows from Theorem 5. Let $k \geq 1$ be such that $\frac{1}{k+1} < wt(T) \leq \frac{1}{k}$. Then, $\frac{1}{wt(T)} \geq k$. Since k is an integer, this implies that $\left\lfloor \frac{1}{wt(T)} \right\rfloor \geq k$. Because $wt(T) > \frac{1}{k+1}$, we obtain $2 \cdot wt(T) \cdot \left\lfloor \frac{1}{wt(T)} \right\rfloor > \frac{2k}{k+1}$. Note that $2k \geq k+1$ because $k \geq 1$. Thus, $2 \cdot wt(T) \cdot \left\lfloor \frac{1}{wt(T)} \right\rfloor > 1$. ■

The above-described schedulability tests are useful in systems in which some tasks have weights more than $\frac{1}{(M-1)}$, and the task system as a whole does not fully utilize all processors.

V. Tardiness Bounds for EPDF

In this section, we focus on soft real-time systems in which deadline misses are tolerated and the goal is to minimize the tardiness of all the jobs in the system.

The notion of tardiness for jobs was defined earlier in Sec. I. *Tardiness for subtasks* is defined similarly: if t is the time at which subtask T_i completes, then $max(0, t - d(T_i))$ is its tardiness. The *tardiness of a task system* is defined as the maximum tardiness among all of its subtasks in any schedule. In this section, we first present a condition on task weights that is sufficient for ensuring that EPDF maintains a tardiness of at most one quantum. Later, we generalize this condition for tardiness thresholds that exceed one.

Before continuing, we introduce some notation to refer to slots in which some processors are idle. In a schedule S, if k processors are idle in slot t, then we say that there are k *holes* in S in slot t. Note that holes may exist because of late subtask releases, even if total utilization is M. The lemma below concerning LAG values and holes is used in our proofs.

Lemma 1: If $LAG(\tau, t) < LAG(\tau, t+1)$, then there is a hole in slot t.

Proof: Suppose there is no hole in slot t, *i.e.*, M subtasks are scheduled in slot t. Then, the total share in the EPDF schedule increases by M from t to $t+1$. On the other hand, the share in the ideal schedule can increase by at most M, since the total weight of all tasks at any time is at most M (by (J)). Thus, LAG cannot increase from t to $t+1$. ∎

A. Sufficient Condition for Tardiness of at most One

We prove that any feasible task system that satisfies the following condition has a tardiness of at most one.

(M1) At all times, the sum of the $M-1$ largest task weights is at most $\frac{M+1}{2}$.

Assume to the contrary that there exists a feasible task system τ satisfying (M1) with tardiness greater than one. Let T_i be the subtask (in some given schedule) with the earliest deadline among all subtasks with tardiness greater than one, and let $t_d = d(T_i)$. Thus, all subtasks with deadlines less than t_d have tardiness at most one.

It is easy to see that the tardiness of T_i is not affected by subtasks with deadlines greater than t_d. Note that any such subtask is scheduled before t_d only if no subtask with a deadline at most t_d is eligible at that slot. In other words, the presence or absence of subtasks with deadlines beyond t_d does not change the scheduling of subtasks with deadlines at most t_d. Thus, we can assume that no task in τ releases any subtask with a deadline greater than t_d. In other words,

$$\text{for every subtask } U_j \in \tau, d(U_j) \leq t_d. \tag{12}$$

We first show that for T_i to have a tardiness of at least two, at least $M+1$ subtasks miss their deadlines at t_d. This, in turn, implies that the LAG of τ at time t_d is at least $M+1$.

Lemma 2: At least $M+1$ subtasks in τ miss their deadlines at t_d.

Proof: Consider any subtask U_j such that U_j misses its deadline at t_d. Because $d(U_j) = t_d$, by (6) and (7), the deadline of U_{j-1} is at or before $t_d - 1$. Therefore, by the definition of T_i and t_d, U_{j-1} has tardiness at most one and is scheduled in $[0, t_d)$. Hence, U_j is eligible to be scheduled at time t_d. If there are at most M subtasks like U_j (including T_i), then each of these subtasks will be scheduled in $[t_d, t_d+1)$. Therefore, subtask T_i cannot have tardiness greater than one. Contradiction. ∎

Lemma 3: $LAG(\tau, t_d) \geq M+1$.

Proof: By (11), we have

$$LAG(\tau, t_d) = \sum_{t=0}^{t_d-1} \sum_{T\in\tau} share(T,t) - \sum_{t=0}^{t_d-1} \sum_{T\in\tau} S(T,t).$$

The first term on the right-hand side of the above equation is the total share in the ideal schedule in $[0, t_d)$, which equals the total number of subtasks in τ. (This follows from (12) and because τ is feasible.) The second term corresponds to the number of subtasks scheduled by EPDF in $[0, t_d)$. Since at least $M+1$ subtasks miss their deadlines at t_d (by Lemma 2), the difference between these two terms is at least M+1. ∎

Because $LAG(\tau, 0) = 0$, it follows by Lemma 3 that there exists a time $t < t_d$ such that $LAG(\tau, t) < M+1$ and $LAG(\tau, t+1) \geq M+1$. We now prove some properties about task lags at time $t+1$; using these properties and (M1), we later derive a contradiction concerning the existence of t.

By Lemma 1, there is at least one hole in slot t (*i.e.*, in $[t, t+1)$). Therefore, if A denotes the set of tasks scheduled in slot t, then we have

$$|A| \leq M-1. \tag{13}$$

Let B denote the set of tasks not in A that are "active" at t. A task U is *active* at time t if it has a subtask U_j such that $e(U_j) \leq t < d(U_j)$. (A task may be inactive either because it has already left the system or because of a late subtask release.) Consider any task $U \in B$ and let U_j be such that $e(U_j) \leq t < d(U_j)$. Because slot t has a hole and no subtask of U is scheduled at t, and since $e(U_j) \leq t < d(U_j)$, U_j must be scheduled in $[0, t)$.

Let I denote the set of the remaining tasks that are not active at time t. Fig. 8(a) shows how the tasks in A, B, and I are scheduled. We now estimate the lag values for the tasks in each of A, B, and I at time $t+1$.

Lemma 4: For $W \in I$, $lag(W, t+1) = 0$.

Proof: Consider any subtask W_h of task W. If $e(W_h) \geq t+1$, then $r(W_h) \geq t+1$. Therefore, by (10), $f(W_h, u) = 0$ for all slots $u \leq t < r(W_h)$. Hence, the share of W_h in the ideal schedule in $[0, t+1)$ is zero. Also, in the EPDF schedule, W_h is scheduled at or after $t+1$. On the other hand, if $e(W_h) \leq t$, then by the definition of I, $d(W_h) \leq t < t_d$. Because the tardiness of such a subtask is at most one, W_h is scheduled in $[0, t+1)$. Hence, the share received by W_h in $[0, t+1)$ is one in both the ideal and EPDF schedules. (This is true even if W is reweighted at any instant.) Thus, for all subtasks of W, the share of that subtask in $[0, t+1)$ is the same in both the ideal and EPDF schedules. Furthermore, recall from Sec. III-A that any task that leaves the system does so with zero lag. Therefore, $lag(W, t+1) = 0$. ∎

Lemma 5: For $V \in B$, $lag(V, t+1) \leq 0$.

Proof: Consider any subtask V_k of task V. Again, as in the proof of Lemma 4, if $r(V_k) \geq t+1$, then by (10), the share of V_k in $[0, t+1)$ in the ideal schedule is zero. On the other hand, if $r(V_k) \leq t$, then, as discussed earlier, V_k is scheduled before t because of the hole in slot t. Thus, the share of V_k in $[0, t+1)$ is one in the EPDF schedule, and at most one in the ideal schedule. Note that $d(V_k)$ may be greater than $t+1$, in which case a portion of V_k's share in the ideal schedule is allocated after $t+1$. Thus, for any subtask of V, its share in $[0, t+1)$ in the EPDF schedule is at least its share in $[0, t+1)$ in the ideal schedule. (Note that this is true even if V is reweighted during V's window because the new weight takes effect only for subtasks after V_k.) Hence, $lag(V, t+1) \leq 0$. ∎

Lemma 6: For $U \in A$, $lag(U, t+1) < 2 \times wt(U)$.

Proof: Let U_j be the subtask of U scheduled at time t. If $d(U_j) \leq t$, then because $t < t_d$, $d(U_j) < t_d$. By the choice of T_i and t_d, it follows that the tardiness of U_j is at most one. Therefore, $d(U_j) = t$ in this case. Thus, in general, $d(U_j) \geq t$. We now consider four cases depending on the value of $d(U_j)$.

- **Case 1:** $\boldsymbol{d(U_j) > t+1}$**.** In this case, by (5)–(7), $r(U_{j+1}) \geq t+1$. Reasoning exactly as in the proof of

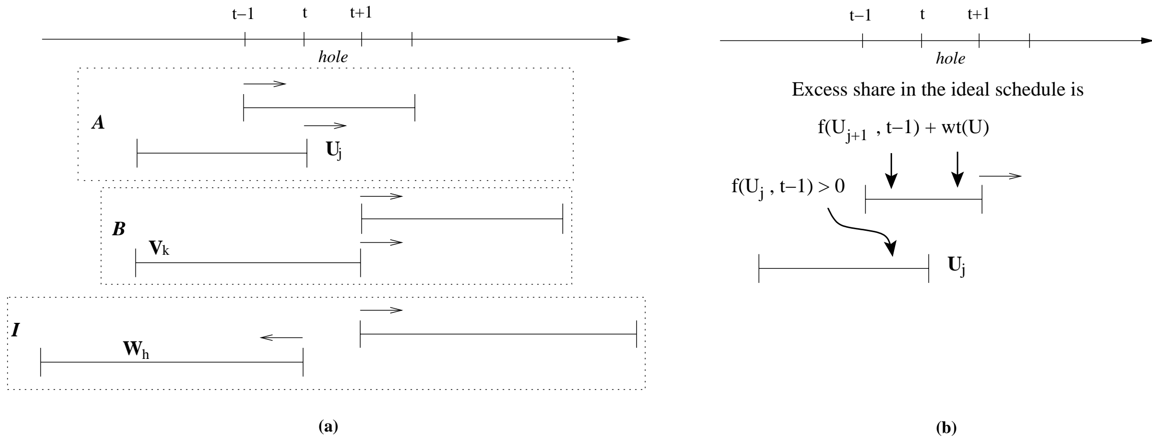

Fig. 8. In this figure, PF-windows are denoted by line segments. An arrow over a release (respectively, deadline) indicates that the release (respectively, deadline) could be anywhere in the direction of the arrow. There is a hole in slot t. **(a)** Sets A, B, and I. The PF-windows of a sample task of each set are shown. **(b)** Case 4 of Lemma 6. The excess share received by U in $[0, t+1)$ in the ideal schedule is equal to the shares received by subtasks after U_j in $[0, t+1)$. (Note that if U_{j+1} has a deadline at time $t+1$, then the PF-window of U_{j+2} may begin at time t, in which case part of the $wt(U)$ share in slot t is due to U_{j+2}.)

Lemma 5, we can show that $lag(U, t+1) \leq 0$.

- **Case 2: $d(U_j) = t+1 \wedge b(U_j) = 0$.** Again, by (5)–(7), $r(U_{j+1}) \geq t+1$. It follows that the share received by any subtask U_k $(k > j)$ in the ideal schedule in $[0, t+1)$ is zero. Thus, the share received by task U in $[0, t+1)$ is the same in both the ideal and EDPF schedules. Therefore, $lag(U, t+1) = 0$.
- **Case 3: $d(U_j) = t+1 \wedge b(U_j) = 1$.** By (5)–(7), we have
$$r(U_{j+1}) \geq d(U_j) - 1. \qquad (14)$$
Therefore, $r(U_{j+1}) \geq t$. Similarly, by (5)–(7), we obtain $r(U_{j+2}) \geq t+1$. Therefore, the share of any subtask U_k $(k > j+1)$ is zero in $[0, t+1)$ in the ideal schedule.
Now, the total share that U receives in $[0, t+1)$ in the EPDF schedule is j. However, in the ideal schedule, the share received by U in $[0, t+1)$ may be more because of the share that U_{j+1} (if it exists) receives in $[0, t+1)$. This share will be non-zero only if $r(U_{j+1}) \leq t$, *i.e.*, $r(U_{j+1}) \leq d(U_j) - 1$. In this case, by (14), we have $r(U_{j+1}) = d(U_j) - 1$. This implies that $\theta(U_{j+1}) = \theta(U_j)$. It also implies that the excess share in the ideal schedule in $[0, t+1)$ is at most $f(U_{j+1}, t)$. Because $r(U_{j+1}) = t$, by (10), we have $f(U_{j+1}, t) = (\lfloor j/wt(U) \rfloor + 1) \cdot wt(U) - j$.
Because $\theta(U_{j+1}) = \theta(U_j)$, by (5) and (6), $\lfloor j/wt(U) \rfloor = \lceil j/wt(U) \rceil - 1$. Hence, $\lfloor j/wt(U) \rfloor < j/wt(U)$. Therefore, $f(U_{j+1}, t) < (j/wt(U) + 1) \cdot wt(U) - j$, *i.e.*, $f(U_{j+1}, t) < wt(U)$. Thus, $lag(U, t+1) < wt(U)$.
- **Case 4: $d(U_j) = t$.** In this case, by (5)–(7), we have $r(U_{j+1}) \geq t-1$ and $r(U_{j+2}) \geq t$. Note that all subtasks of U up to and including U_j receive a share of one over $[0, t+1)$ in both the EPDF and ideal schedules (see Fig. 8(a)). By (10), no subtask receives a share in the ideal schedule before its Pfair release time. Therefore, the only subtask after U_j that may contribute to U's share in the ideal schedule in $[0, t)$ is U_{j+1}. Further, by (F1), the share of U in slot t (*i.e.*, in $[t, t+1)$) is at most its weight. Thus, it follows that the excess share that U can receive in $[0, t+1)$ in the ideal schedule is at most $f(U_{j+1}, t-1) + wt(U)$ (refer to Fig. 8(b)). The first term, $f(U_{j+1}, t-1)$, will be non-zero only if $r(U_{j+1}) = t-1$, *i.e.*, $r(U_{j+1}) = d(U_j) - 1$. Reasoning exactly as in Case 3, it follows that $f(U_{j+1}, t-1) < wt(U)$. Hence, $lag(U, t+1) < 2 \times wt(U)$.

Thus, in all cases, we have $lag(U, t+1) < 2 \times wt(U)$. All the above cases apply even if U is reweighted. Note that if U is reweighted in the interval $[r(U_j), t)$, then the new weight comes into effect only for U_{j+1} and later subtasks, and $wt(U)$ above refers to the new weight. ■

Because $LAG(\tau, t+1) = \sum_{U \in A \cup B \cup I} lag(U, t+1)$, by Lemmas 4–6, $LAG(\tau, t+1) < 2 \times \sum_{U \in A} wt(U)$. By (13), $|A| \leq M-1$. Therefore, by (M1), $LAG(\tau, t+1) < M+1$, contradicting our assumption about t. Thus, we have the following theorem.

Theorem 6: Every feasible IS task system satisfying (M1) has a tardiness of at most one under EPDF.

Discussion. One way to ensure (M1) is to restrict the weight of every task to at most $\frac{M+1}{2M-2}$. Note that $\frac{M+1}{2M-2} > \frac{1}{2}$. Thus, EPDF guarantees a tardiness of at most one for a task system consisting solely of light tasks.

It is possible to improve (M1) by more accurately bounding the lag values for tasks in set A. In particular, using the proof technique used in [4], we can show that there must be at least one task in A with a deadline at $t+1$. Thus, either Case 2 or 3 in the proof of Lemma 6 applies for such a task. Thus, the lag for that task at time $t+1$ is bounded by its weight. Using this information, we can obtain the following (slightly) improved condition.

(M1′) Let $w_1, w_2, \ldots, w_{M-1}$ denote the $M-1$ largest task weights in non-increasing order. Then, $w_{M-1} + 2 \times \sum_{i=1}^{M-2} w_i \leq M+1$.

This allows us to improve the individual weight restriction to $\frac{M+1}{2M-3}$. Note that, for $M \leq 4$, we have $2M-3 \leq M+1$, *i.e.*, $\frac{M+1}{2M-3} \geq 1$. Because the weight of any task is at most one,

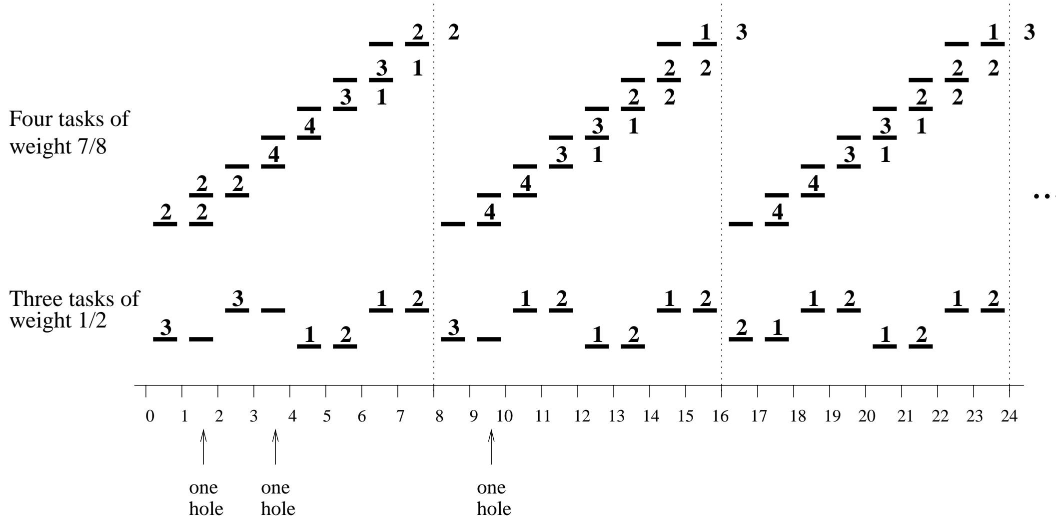

Fig. 9. The notation used in this figure is the same as that in Fig. 2; in addition, subtasks that miss their deadlines are shown scheduled after their windows. In the schedule shown, EPDF breaks ties in favor of the tasks of weight 1/2. Note that the schedule over $[16, 24]$ repeats after time 24. Thus, a maximum of three subtasks miss their deadlines simultaneously, and hence, the tardiness of each subtask is at most one.

the individual weight restriction is always satisfied if $M \leq 4$. That is, if the number of processors is at most 4, then EPDF guarantees a tardiness of at most one with no weight restriction.

Tightness. At present, we do not know whether the above condition is tight. Though we have not been able to find an M-processor schedule in which more than M subtasks miss their deadlines simultaneously, we do have examples in which up to $M - 2$ simultaneous misses occur.

Consider the following set of periodic tasks: three tasks of weight $\frac{1}{2}$ and $M-1$ tasks of weight $\frac{2M-3}{2M-2}$. Fig. 9 illustrates the case $M = 5$. In these examples, the number of simultaneous deadline misses increases up to some multiple of the least common multiple of the task periods, and then remains steady after that. Note that, in these examples, the percentage of jobs that miss their deadlines is quite high (approaches 25% for large M). However, as our simulation experiments (described in Sec. VI) indicate, such cases are very rare indeed.

B. Generalizing the Condition

The above approach can be easily extended to obtain conditions similar to (M1) for guaranteeing a tardiness of at most k. In particular, the following condition ensures that tardiness is never more than k.

(Mk) At all times, the sum of the $M - 1$ largest task weights is at most $\frac{kM+1}{k+1}$.

The proof for this result is very similar to the sufficiency proof for (M1). Suppose that t_d, T_i, and τ are defined in the same manner, except that the tardiness of T_i is $k + 1$. Similar to Lemma 2, we can show that for a subtask to have tardiness $k+1$, $kM + 1$ subtasks simultaneously miss their deadline at time t_d. Hence, $LAG(\tau, t_d) \geq kM+1$, implying the existence of time t as follows: $t < t_d$ and $LAG(\tau, t) < kM + 1$ and $LAG(\tau, t + 1) \geq kM + 1$.

The rest of the proof is the same except for Lemma 6, which is modified as follows.

Lemma 7: For $U \in A$, $lag(U, t + 1) < (k + 1) \times wt(U)$.

Proof: Let U_j be the subtask of U scheduled in slot t. Since $t < t_d$, U_j has a tardiness of at most k. Since U_j completes at time $t+1$, we have $d(U_j) \geq t+1-k$. It follows that $r(U_{j+1}) \geq t - k$. Since U_j is scheduled in slot t in S, it follows that the excess share in the ideal schedule over $[0, t+1)$ is at most $f(U_{j+1}, t-k)$ plus the share over $[t-k+1, t+1)$. By (F1), the second term is at most $k \times wt(U)$. Further, as in Lemma 6, we can show that $f(U_{j+1}, t - k) < wt(T)$. Therefore, $LAG(U, t + 1) < (k + 1) \times wt(U)$. ■

In other words, the lag of a task U in A may be up to $(k+1)$ times the weight of U. By Lemmas 4, 5, and 7, $LAG(\tau, t + 1) < \sum_{U \in A}(k+1) \times wt(U)$. By (Mk), the right-hand-side of this inequality is at most $M+1$. Thus, $LAG(\tau, t+1) < M+1$; a contradiction. Thus, we have the following result.

Theorem 7: Every feasible IS task system satisfying (Mk) has a tardiness of at most k under EPDF on M processors.

As before, we can improve (Mk) to (Mk$'$) as follows.

(Mk$'$) Let $w_1, w_2, \ldots, w_{M-1}$ denote the $M-1$ largest task weights in non-increasing order. Then, $w_{M-1} + (k+1) \times \sum_{i=1}^{M-2} w_i \leq kM + 1$.

Condition (Mk) gives us an individual weight restriction of $\frac{kM+1}{(k+1)(M-1)}$, while (Mk$'$) gives us a slightly better value of $\frac{kM+1}{(k+1)(M-2)+1}$. Note that both these values are at least $\frac{k}{k+1}$.

VI. Simulation Results

To determine how frequently deadlines are missed under EPDF, and by how much, we computed EPDF schedules for a number of randomly generated task sets. Only periodic (not IS) task sets were considered in these experiments. Intuitively, introducing IS delays should only reduce demand and hence lessen the likelihood of missed deadlines. In addition, each task set was defined to fully utilize all available processors to further increase the possibility of deadline misses.

The following procedure was followed in every run of the simulation. First, the number of processors was chosen as

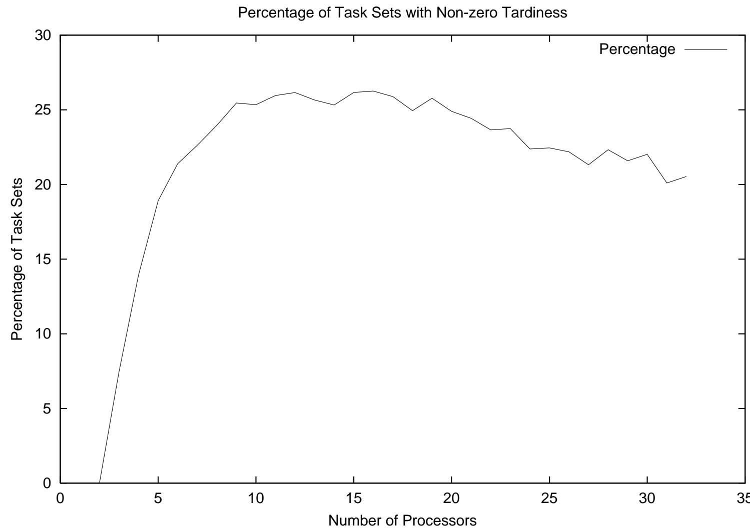

Fig. 10. Percentage of task sets with non-zero tardiness versus the number of processors.

a random number between 1 and 32. Then, task weights were generated randomly in the interval $(0, 1]$. (No weight restrictions were applied, *i.e.*, weights were allowed to be as large as one.) The weight of the last task was chosen so that the total weight equaled the number of processors. For each generated task set, we constructed an EPDF schedule over the time interval $[0, 10L)$, where L is the least common multiple of all task periods. We repeated this procedure approximately 195,000 times. Thus, the number of data points obtained per processor in each graph is approximately 6,000.

In all the runs, no subtask ever missed its deadline by more than one quantum. In other words, the tardiness for each generated task set was at most one. Though this is not a proof, it does strongly indicate that any task set for which tardiness is greater than one (if such a task set exists) is probably pathological and rare.

The graph in Fig. 10 plots the percentage of task sets with tardiness greater than zero versus the number of processors. (Recall that a task set has tardiness greater than zero even if just a *single* job misses its deadline.) For example, EPDF misses at least one deadline for approximately 19% of the generated task sets on five processors. (No task set misses a deadline for systems of one or two processors because of the optimality of EPDF for such systems [22].) The percentage of task sets with non-zero tardiness initially increases as the number of processors increases and then steadies. The maximum is around 25%. Though this value may seem high, as the remaining graphs show, the fraction of deadlines that were missed tended to be extremely low.

Fig. 11 shows the average percentage of missed deadlines versus the number of processors. Inset (a) shows the average over all generated task sets, while inset (b) shows the average over task sets that have at least one deadline miss. Both the percentage of job deadlines missed and the percentage of subtask deadlines missed are plotted. As can be seen from these graphs, deadline misses are quite rare.

Note that the percentage of job deadlines missed is more than the percentage of subtask deadlines missed. There are two reasons for this. First, the total number of subtask deadlines is much more than the total number of job deadlines. Second, subtask deadline misses within a job have a tendency to cascade, causing the later subtasks within a job (and hence the job itself) to have a higher likelihood of missing their deadlines.

Surprisingly, the largest average of job deadlines missed in Fig. 11 is obtained for three processors: 0.04% in inset (a) and 0.55% in inset (b). It is also surprising that the percentage of misses decreases as the number of processors increases. Recall from Sec. IV that EPDF is optimal on M processors if the weight of each task is at most $\frac{1}{M-1}$. (Also recall that this condition is fairly tight.) As M increases, $\frac{1}{M-1}$ decreases and hence, intuitively, the chance of a deadline miss should also increase.

However, as M increases, more tasks need to be generated to fully utilize the system. Because our samples are generated randomly, the probability of having low-weight tasks in the system also increases. This in turn drives the number of tasks (and hence, the total number of deadlines) up. Thus, though the number of missed deadlines may increase, the percentage of deadline misses decreases. Analogously, for fewer processors, deadline misses are more common when most tasks have a large weight. In this case, the number of tasks is small and therefore, the percentage of deadlines missed tends to be higher.

VII. Conclusions

We have considered the scheduling of soft real-time tasks on multiprocessors. We presented some new schedulability results for EPDF-scheduled hard real-time systems and established a sufficient condition for ensuring that the EPDF algorithm meets a given tardiness threshold in soft real-time systems. Our simulation results indicate that the percentage of deadlines missed is very low even for systems that do not satisfy our condition (for any threshold). These experiments also indicate

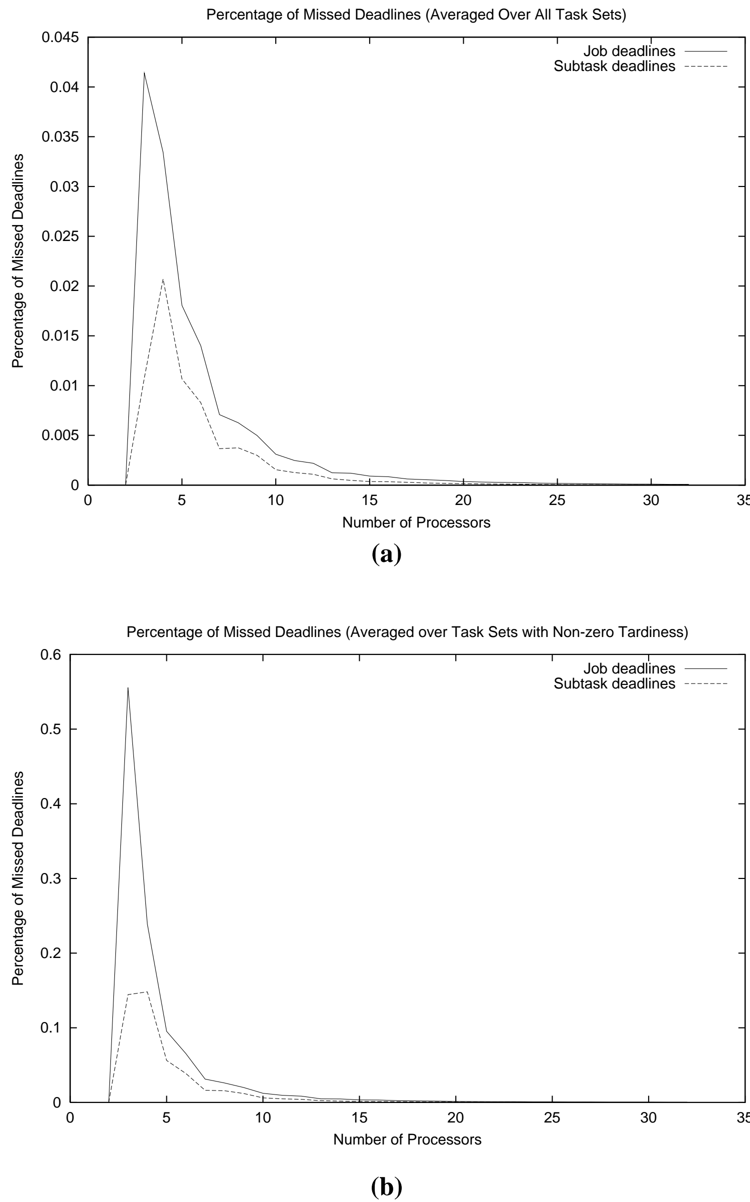

Fig. 11. Solid (dotted) lines denote the percentage of job (subtask) deadlines missed. **(a)** Percentage of deadlines missed averaged over all task sets. **(b)** Percentage of deadlines missed averaged over task sets with non-zero tardiness. (99% confidence intervals were computed for each graph but are not shown because the relative error associated with each point is very small — less than 0.2% of the reported value.)

that the performance (in terms of tardiness and percentage of missed deadlines) of EPDF scales well as the number of processors increase.

REFERENCES

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

[2] S. Baruah, J. Gehrke, and C. Plaxton, "Fast scheduling of periodic tasks on multiple resources," in *Proceedings of the 9th International Parallel Processing Symposium*, April 1995, pp. 280–288.

[3] J. Anderson and A. Srinivasan, "Mixed Pfair/ERfair scheduling of asynchronous periodic tasks," *Journal of Computer and System Sciences*, to appear.

[4] A. Srinivasan and J. Anderson, "Optimal rate-based scheduling on multiprocessors," in *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, May 2002, pp. 189–198.

[5] J. Bennett and H. Zhang, "WF^{2}Q: Worst-case fair queueing," in *Proceedings of IEEE INFOCOM*, March 1996, pp. 120–128.

[6] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in *Proceedings of the ACM Symposium on Communications Architectures and Protocols*, 1989, pp. 1–12.

[7] S. Golestani, "A self-clocked fair queueing scheme for broadband applications," in *Proceedings of IEEE INFOCOM*, April 1994, pp. 636–646.

[8] A. K. Parekh and R. G. Gallagher, "A generalized processor sharing approach to flow-control in integrated services networks: the single-node case," *IEEE/ACM Transactions on Networking*, vol. 1(3), pp. 344–357, 1993.

[9] L. Zhang, "Virtual clock: A new traffic control algorithm for packet-switched networks," *ACM Transactions on Computer Systems*, vol. 9, no. 2, pp. 101–124, May 1991.

[10] K. Jeffay, F. D. Smith, A. Moorthy, and J. Anderson, "Proportional share scheduling of operating system services for real-time applications," in *Proceedings of the IEEE Real-time Systems Symposium*, Dec. 1998, pp. 480–491.

[11] A. Chandra, M. Adler, P. Goyal, and P. Shenoy, "Surplus fair scheduling: A proportional-share CPU scheduling algorithm for symmetric multiprocessors," in *Proceedings of the 4th ACM Symposium on Operating System Design and Implementation*, Oct. 2000, pp. 45–58.

[12] A. Chandra, M. Adler, and P. Shenoy, "Deadline fair scheduling: Bridging the theory and practice of proportionate-fair scheduling in multiprocessor servers," in *Proceedings of the 7th IEEE Real-time Technology and Applications Symposium*, May 2001, pp. 3–14.

[13] S. Keshav, 2001, personal communication.

[14] J. Anderson, S. Baruah, and K. Jeffay, "Parallel switching in connection-oriented networks," in *Proceedings of the 20th IEEE Real-time Systems Symposium*, Dec. 1999, pp. 200–209.

[15] E. Bampis and G. N. Rouskas, "The scheduling and wavelength assignment problem in optical wdm networks," *IEEE/OSA Journal of Lightwave Technology*, vol. 20(5), pp. 782–789, 2002.

[16] V. Agarwal, M. Hrishikesh, S. Keckler, and D. Burger, "Clock rate versus IPC: The end of the road for conventional microarchitectures," in *Proceedings of the 27th International Symposium on Computer Architecture*, June 2000, pp. 248–259.

[17] K. Coffman and A. Odlyzko, "The size and growth rate of the internet," March 2001, available at http://www.firstmonday.dk/issues/issue3_10/coffman/.

[18] A. Srinivasan, P. Holman, J. Anderson, S. Baruah, and J. Kaur, "Multiprocessor scheduling in processor-based router platforms: Issues and ideas," in *Proceedings of the 2nd Workshop on Network Processors*, Feb. 2002, pp. 48–62.

[19] J. Kaur and H. Vin, "Core-stateless guaranteed rate scheduling algorithms," in *Proceedings of IEEE INFOCOM*, April 2001, pp. 1484–1492.

[20] I. Stoica and H. Zhang, "Providing guaranteed services without per flow management," in *Proceedings of ACM Sigcomm*, Sept. 1999, pp. 81–94.

[21] J. Anderson and A. Srinivasan, "Pfair scheduling: Beyond periodic task systems," in *Proceedings of the 7th International Conference on Real-time Computing Systems and Applications*, Dec. 2000, pp. 297–306.

[22] A. Srinivasan and J. Anderson, "Fair scheduling of dynamic task systems on multiprocessors," *Journal of Systems and Software*, to appear.

[23] S. Baruah, J. Gehrke, C. Plaxton, I. Stoica, H. Abdel-Wahab, and K. Jeffay, "Fair on-line scheduling of a dynamic set of tasks on a single resource," *Information Processing Letters*, vol. 26(1), pp. 43–51, Jan. 1998.

[24] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. Baruah, J. Gehrke, and C. Plaxton, "A proportional share resource allocation algorithm for real-time, time-shared systems," in *Proceedings of the 17th IEEE Real-time Systems Symposium*, Dec. 1996, pp. 288–299.

[25] A. Srinivasan, P. Holman, J. Anderson, and S. Baruah, "The case for fair multiprocessor scheduling," in *Proceedings of the 11th International Workshop on Parallel and Distributed Real-time Systems*, Apr. 2003.

[26] P. Holman and J. Anderson, "Locking in pfair-scheduled multiprocessor systems," in *Proceedings of the 23rd IEEE Real-time Systems Symposium*, Dec. 2002, pp. 149–158.

[27] R. Rajkumar, *Synchronization In Real-Time Systems – A Priority Inheritance Approach*. Kluwer Academic Publishers, 1991.

[28] P. Holman and J. Anderson, "Guaranteeing pfair supertasks by reweighting," in *Proceedings of the 22nd IEEE Real-time Systems Symposium*, December 2001, pp. 203–212.

[29] M. Moir and S. Ramamurthy, "Pfair scheduling of fixed and migrating periodic tasks on multiple resources," in *Proceedings of the 20th IEEE Real-time Systems Symposium*, Dec. 1999, pp. 294–303.

Anand Srinivasan Anand Srinivasan will receive his Ph.D. in computer science in December 2003 from the University of North Carolina at Chapel Hill. He received a B.Tech. in computer science and engineering from the Indian Institute of Technology Mumbai in 1994, and an M.S. in computer science from the University of North Carolina at Chapel Hill in 2000. His main research interests are in the areas of real-time scheduling and operating systems.

James H. Anderson James H. Anderson is a professor in the Department of Computer Science at the University of North Carolina at Chapel Hill. He received a B.S. in computer science from Michigan State University in 1982, an M.S. in computer science from Purdue University in 1983, and a Ph.D. in computer sciences from the University of Texas at Austin in 1990. Before joining UNC-Chapel Hill in 1993, he was with the Computer Science Department at the University of Maryland between 1990 and 1993. In 1995, Prof. Anderson received the U.S. Army Research Office Young Investigator Award, and in 1996, he was named an Alfred P. Sloan Research Fellow. In 2000, he served as program chair for the 19th Annual ACM Symposium on Principles of Distributed Computing. Prof. Anderson's main research interests are within the areas of concurrent and distributed computing, and real-time systems.