Fair Scheduling of Dynamic Task Systems on Multiprocessors*

Anand Srinivasan and James H. Anderson
Department of Computer Science, University of North Carolina
Chapel Hill, NC 27599-3175
Phone: (919) 962-1757
Fax: (919) 962-1799
E-mail: {anands,anderson}@cs.unc.edu

May 2002

Abstract

In dynamic task systems, tasks are allowed to join and leave the system. In previous work, Stoica *et al.* and Baruah *et al.* presented conditions under which such joins and leaves may occur in fair-scheduled uniprocessor systems. In this paper, we extend their work by considering fair-scheduled multiprocessors. We show that the uniprocessor conditions presented previously are sufficient when using the earliest-pseudo-deadline-first algorithm on M processors if the utilization of any subset of $M-1$ tasks is at most one. Further, for the general case in which task weights are not restricted in this way, we derive different sufficient conditions for joins and leaves when tasks are scheduled using the PD^2 Pfair algorithm. We show that, in general, these conditions cannot be improved upon without violating fairness bounds and deadlines.

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

1 Introduction

In many real-time systems, the set of runnable tasks may change dynamically. For example, in an embedded system, different modes of operation may need to be supported. A mode change may require adding new tasks and deleting existing tasks. Another example is a desktop system that supports real-time applications such as multimedia and collaborative-support systems, which may be initiated at arbitrary times.

The distinguishing characteristic of *dynamic* task systems such as these is that tasks are allowed to *join* and *leave* the system. If joins and leaves in such a system are allowed without any restrictions, then the system may become overloaded, and deadlines can be missed. Thus, joins and leaves must be performed only under conditions that ensure that system schedulability is not compromised. A suitable join condition usually can be obtained from the schedulability test associated with the scheduling algorithm being used. A leave condition is somewhat trickier. In particular, if an over-allocated task (*e.g.*, a task that completes before its deadline) is allowed to leave, then it can re-join immediately, and thus effectively execute at a higher-than-proscribed rate.

In this paper, we consider the problem of scheduling dynamic task systems on multiprocessors. This problem has been studied earlier in the context of uniprocessor systems in which tasks are scheduled using static-priority algorithms [11, 14] and fair allocation schemes [6, 13]. Our focus is fair scheduling as well, because it is the only known way of optimally scheduling recurrent real-time tasks on multiprocessors [3, 4, 5, 12].

In fair scheduling disciplines, tasks are required to make progress at steady rates. Steady allocation rates are ensured by scheduling in a manner that closely tracks an ideal, fluid allocation. The *lag* of a task measures the difference between its ideal and actual allocations. In fair scheduling schemes, lags are required to remain bounded. (Such a bound in turn implies a bound on the timeliness of real-time tasks.) If a task's lag is positive, then it has been under-allocated; if negative, then it has been over-allocated. In the uniprocessor join/leave conditions presented previously [6, 13], a task is allowed to leave iff it is not over-allocated, and join iff the total utilization after it joins is at most one.

The prior work on uniprocessors mentioned above is not straightforward to extend to multiprocessors; in fact, Baruah *et al.* explicitly noted the multiprocessor case as an open problem [6]. In recent work, dynamic multiprocessor systems *were* considered by Chandra *et al.* [7, 8]. However, their work was entirely experimental in nature, with no formal analysis of the algorithms considered. In this paper, we present multiprocessor join/leave conditions for which such analysis *is* provided.

We now describe the fair scheduling concepts and some related task models used in this paper. After that, we present a more detailed overview of the contributions of this paper.

Pfair scheduling. The *periodic* task model provides the simplest notion of a recurrent real-time task [9]. In this model, successive job releases by the same task are spaced apart by a fixed interval, called the task's *period*. Periodic tasks can be optimally scheduled on multiprocessors using *Pfair* scheduling algorithms [1, 3, 4, 5]. Pfairness requires the lag of each task to be bounded between -1 and 1, which is a stronger requirement than periodicity. As we shall see, these lag bounds have the effect of breaking each task into quantum-length *subtasks*

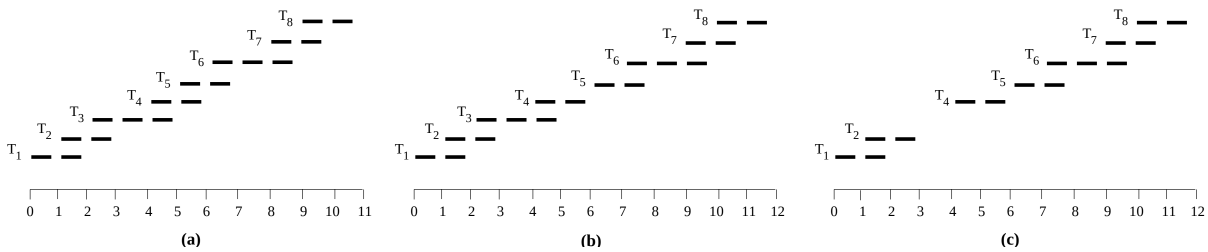

Figure 1: **(a)** Windows of the first job of a periodic task T with weight $8/11$. This job consists of subtasks $T_1, \ldots, T_8$, each of which must be scheduled during its window, or else a lag-bound violation will result. (This pattern repeats for every job.) **(b)** The Pfair windows of an IS task. Subtask T_5 becomes eligible one unit late. **(c)** The Pfair windows of a GIS task. Subtask T_3 is absent and T_5 is one unit late. (Because T_3 is absent, this is not an IS task.)

that must be executed within *windows* of approximately equal lengths. The length and alignment of a task's windows is determined by its *weight*, which is the ratio of its per-job execution cost and period. Intuitively, a task's weight denotes the processor share it requires. Fig. 1(a) illustrates the windows for a task of weight $8/11$.

In the *sporadic* model [10], the periodic notion of recurrence is relaxed by specifying a minimum (rather than exact) spacing between consecutive job releases of the same task. In recent work [2, 12], the sporadic model was extended to obtain the *intra-sporadic* (IS) and *generalized intra-sporadic* (GIS) models. The sporadic model allows *jobs* to be released "late"; the IS model allows *subtasks* to be released late, as illustrated in Fig. 1(b). The GIS model is obtained from the IS model by allowing subtasks to be absent. Fig. 1(c) shows an example. In [12], we showed that the PD^2 Pfair algorithm [1] optimally schedules static GIS task systems on multiprocessors. In [2], we proved that the (simpler) earliest-pseudo-deadline-first (EPDF) algorithm is optimal for scheduling static IS task systems on two processors.

Contributions. In this paper, we extend the result in [2] to *dynamic* GIS task systems by showing that the previously-presented uniprocessor join/leave conditions [6, 13] are sufficient when using EPDF on M processors if the total weight of any subset of $M-1$ tasks is at most one. From this result, it follows that EPDF correctly schedules any feasible *static* GIS task system if the weight of each task is at most $1/(M-1)$, where $M > 1$.

For the general case in which task weights are not restricted, we derive sufficient conditions for joins and leaves when tasks are scheduled using PD^2. Though these conditions are more restrictive than the previous uniprocessor conditions, we show that they are tight. In particular, we prove that if they are relaxed in any way, then task systems exist in which deadlines are missed. This result applies to a class of algorithms that includes all known Pfair scheduling algorithms.

Organization. The rest of the paper is organized as follows. In Sec. 2, needed definitions are given. In Sec. 3, we state the join and leave conditions that we consider. The results pertaining to EPDF- and PD^2-scheduled systems summarized above are then presented in Secs. 4 and 5, respectively. We conclude in Sec. 6.

2 Preliminaries

In the following subsections, relevant concepts and terms are defined. We begin with Pfair scheduling.

2.1 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 with weight less than (at least) $1/2$ is called a *light* (*heavy*) task.

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.*, parallelism is not permitted but interprocessor migration is). The sequence of allocation decisions over time defines a *schedule* S. Formally, $S : \tau \times \mathcal{I} \mapsto \{0, 1\}$, where τ is a set of tasks and $\mathcal{I}$ is the set of nonnegative integers. If T is scheduled in slot t, then $S(T, t) = 1$; otherwise, $S(T, t) = 0$.

The notion of a Pfair schedule is defined by comparing to a 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] $lag(T, t) = wt(T) \cdot t - \sum_{u=0}^{t-1} S(T, u)$. A schedule is *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.

The lag bounds above have the effect of breaking each task T into an infinite sequence of unit-time *subtasks*. We denote the i^{th} subtask of task T as T_i, where $i \geq 1$. As in [4], 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*. The *length* of T_i's window, denoted $|w(T_i)|$, is defined as $d(T_i) - r(T_i)$. As an example, consider subtask T_1 in Fig. 1(a). Here, we have $r(T_1) = 0$, $d(T_1) = 2$ and $|w(T_i)| = 2$. Therefore, T_1 must be scheduled either at time 0 or 1.

2.2 Scheduling Algorithms

In earlier work [2], we proved that the earliest-pseudo-deadline-first (EPDF) Pfair algorithm is optimal on at most 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 [4], PD [5], and PD^2 [1]. These algorithms prioritize subtasks on an EPDF basis, but differ in the choice of

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

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

[3] In our earlier work [1, 2, 3, 12], 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 from that in our earlier papers.

tie-breaking rules. PD^2 is the most efficient of the three. In this paper, we only present detailed results about EPDF and PD^2. Therefore, we only describe the tie-break parameters of PD^2 below.

PD^2 uses two tie-break parameters, the first of which 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:

$$b(T_i) = d(T_i) - r(T_{i+1}). \tag{4}$$

For example, in Fig. 1(a), $b(T_i) = 1$ for $1 \le i \le 7$ and $b(T_8) = 0$. PD^2 favors a subtask deadline 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 parameter is the "group deadline," which is needed in systems with heavy tasks. It is easy to show that all windows of a heavy task are of length two or three, as depicted in Fig. 1(a). Consider a sequence $T_i, \ldots, T_j$ of subtasks of a heavy task T such that, for all $i \le k < j$, $b(T_k) = 1$ and $w(T_{k+1}) = 2$. 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 "cascade" of scheduling decisions, caused by a subtask being scheduled in its last slot, 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 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 heavy, then $D(T_i)$=($\mathbf{min}\ u :: \ u \ge d(T_i)$ 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, 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.

We can now describe the PD^2 priority definition. If subtasks T_i and U_j are both eligible at time t, then PD^2 prioritizes T_i over U_j at t if $(d(T_i) < d(U_j))$, or $(d(T_i) = d(U_j) \ \wedge \ b(T_i) = 1 \ \wedge \ b(U_j) = 0)$, or $(d(T_i) = d(U_j) \ \wedge \ b(T_i) = b(U_j) \ \wedge \ D(T_i) \ge D(U_j))$. (Refer to [1] for a more detailed explanation pf PD^2.)

2.3 Generalized Intra-sporadic Tasks

Having described the concept of Pfair scheduling, we now describe the intra-sporadic (IS) and the generalized intra-sporadic (GIS) task models.

The IS model generalizes the sporadic model by allowing separation between consecutive subtasks of a task. More specifically, the separation between $r(T_i)$ and $r(T_{i+1})$ is allowed to be more than $\left\lfloor \frac{i}{wt(T)} \right\rfloor - \left\lfloor \frac{i-1}{wt(T)} \right\rfloor$, which is 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) depicts the windows of a sample IS task.

Each subtask of an IS task has an *offset* that gives the amount by which its window has been right-shifted. Let $\theta(T_i)$ denote the offset of subtask T_i. Thus, by (2) and (3), we have the following.

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

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

Each subtask T_i has an additional parameter $e(T_i)$ that corresponds to the first time slot in which T_i 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 releases" as in ERfair scheduling [1]. 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.

The GIS model is a generalization of the IS model, in which subtasks are allowed to be absent. Thus, the subtasks of a GIS task are a subset of the subtasks of an IS task. Fig. 1(c) shows an example. The formulae for subtask release times and deadlines of a GIS task are the same as for an IS task. The b-bit and group deadline for a subtask are defined assuming that future subtask releases are as early as possible. Thus, $b(T_i)$ is the same as for a periodic task and $D(T_i) = \theta(T_i) + D_p(T_i)$, where $D_p(T_i)$ is the group deadline if T were periodic.

Offsets for subtasks 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.

$$k \geq i \Rightarrow \theta(T_k) \geq \theta(T_i) \tag{7}$$

Because the GIS model generalizes the other task models above, it is the notion of recurrence considered hereafter. We now present some definitions and properties about GIS task systems.

Terminology. An *instance* of a task system represents a unique assignment of release times (and eligibility times) to each subtask, subject to (7). Note that the deadline of a subtask is automatically determined once its release time is fixed (refer to (5) and (6)). If a task T, after executing subtask T_i, releases subtask T_k, then T_k is called the *successor* of T_i and T_i is called the *predecessor* of T_k (*e.g.*, T_4 is the successor of T_2 in Fig. 1(c)).

Feasibility. In [2, 12], we showed that a GIS task system τ is feasible on M processors iff

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

The proof of this, in fact, shows that a schedule exists in which each subtask is scheduled in its PF-window. In [12], we also proved that PD^2 correctly schedules any GIS task system for which (8) holds.

Displacements. By definition, the removal of a subtask from one instance of a GIS task system results in another valid instance. Let $X^{(i)}$ denote a subtask of any task in a GIS task system τ. Let S denote any schedule of τ obtained by an EPDF-based algorithm (*e.g.*, any of the known Pfair algorithms). Assume that removing $X^{(1)}$ scheduled at slot t_1 in S causes $X^{(2)}$ to shift from slot t_2 to t_1, where $t_1 \neq t_2$, which in turn may cause other shifts. We call this shift a *displacement* and represent it by a four-tuple $\langle X^{(1)}, t_1, X^{(2)}, t_2 \rangle$. A displacement $\langle X^{(1)}, t_1, X^{(2)}, t_2 \rangle$ is *valid* iff $e(X^{(2)}) \leq t_1$. Because there can be a cascade of shifts, we may have a *chain* of displacements, as illustrated in Fig. 2.

Removing a subtask may also lead to slots in which some processors are idle. In a schedule S, if k processors

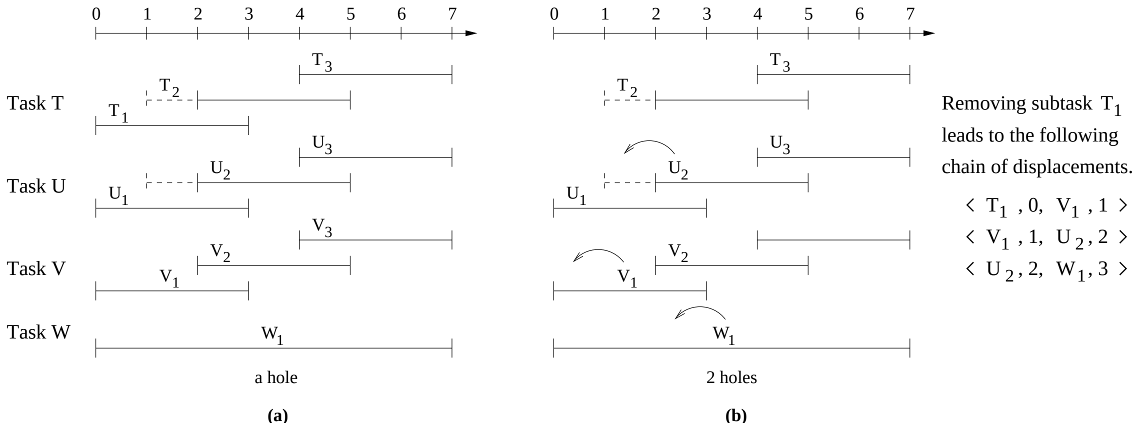

Figure 2: A schedule for three tasks of weight $3/7$ and one task of weight $1/7$ on two processors. The solid lines depict PF-windows; the dashed lines depict IS-windows. Here, only subtasks T_2 and U_2 are eligible before their PF-windows. Inset **(b)** illustrates the displacements caused by the removal of subtask T_i from the schedule shown in inset **(a)**.

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 lemmas below concern displacements and holes. The first two were proved earlier [12] in the context of PD^2; however, the proofs apply to all algorithms that prioritize subtasks on an EPDF basis. Lemma 1 states that a subtask removal can only cause left-shifts, as in Fig. 2(b). Lemma 2 indicates when a left-shift into a slot with a hole can occur. Lemma 3 shows that shifts across a hole cannot occur. In these lemmas, τ refers to an instance of any GIS task system and S denotes its schedule obtained by an EPDF-based algorithm. Throughout this paper, we assume that ties among subtasks are resolved consistently. Thus, if τ' is obtained by removing a subtask from τ, then the relative priorities of two subtasks in τ' are the same as in τ.

Lemma 1 *Let $X^{(1)}$ be a subtask that is removed from τ, and let the resulting chain of displacements in S be $C = \Delta_1, \Delta_2, \ldots, \Delta_k$, where $\Delta_i = \langle X^{(i)}, t_i, X^{(i+1)}, t_{i+1} \rangle$. Then $t_{i+1} > t_i$ for all $i \in \{1, \ldots, k\}$.*

Lemma 2 *Let $\Delta = \langle X^{(1)}, t_1, X^{(2)}, t_2 \rangle$ be a valid displacement in S. If $t_1 < t_2$ and there is a hole in slot t_1 in that schedule, then $X^{(2)}$ is the successor of $X^{(1)}$ in τ.*

Lemma 3 *Let $\Delta = \langle X^{(1)}, t_1, X^{(2)}, t_2 \rangle$ be a valid displacement in S. If $t_1 < t_2$ and there is a hole in slot t' such that $t_1 \le t' < t_2$ in that schedule, then $t' = t_1$ and $X^{(2)}$ is the successor of $X^{(1)}$ in τ.*

Proof: Because Δ is valid, $e(X^{(2)}) \le t_1$. If $t_1 < t'$, then $e(X^{(2)}) < t'$, which implies that $X^{(2)}$ is not scheduled in slot $t_2 > t'$, as assumed, since there is a hole in t'. Thus, $t_1 = t'$; by Lemma 1, $X^{(2)}$ is $X^{(1)}$'s successor. □

Flows and lags in GIS task systems. Let v be the time at which a GIS task T joins the system. The lag of T at time t is defined in the same way as it is defined for periodic tasks. Let $ideal(T, v, t)$ denote the processor time that T receives in a fluid schedule in $[v, t)$. Then,

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

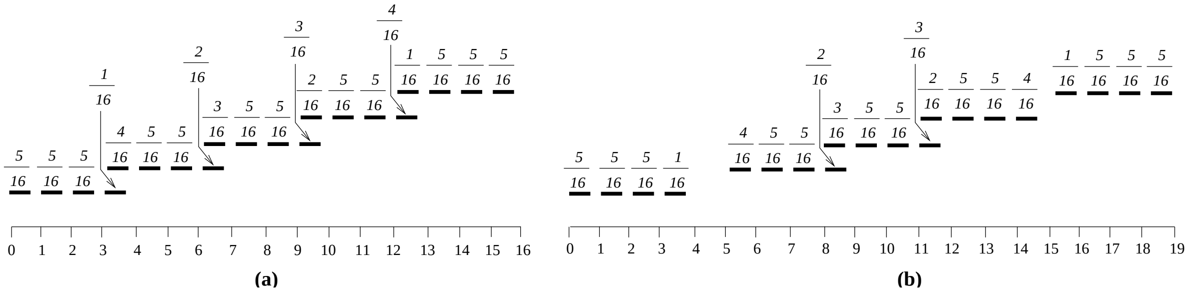

Figure 3: Fluid schedule for a task T of weight $5/16$. The share of each subtask in the slots of its window is shown. In **(a)**, no subtask is released late; in **(b)**, T_2 and T_5 are released late. Note that $flow(T, 3)$ is either $5/16$ or $1/16$ depending on when subtask T_2 is released.

$ideal(T, v, t)$ is defined as follows, where $\mathit{flow}(T, u)$ is the share assigned to task T in slot u.

$$ideal(T, v, t) = \sum_{u=v}^{t-1} \mathit{flow}(T, u). \tag{10}$$

As seen below, $\mathit{flow}(T, u)$ usually equals $wt(T)$. However, in certain slots, $\mathit{flow}(T, u)$ may be less than $wt(T)$, so that T's total share up to and including that slot is an integer. $\mathit{flow}(T, u)$ is defined in terms of a function f that indicates the share assigned to each subtask in each slot u.

$$f(T_i, u) = \begin{cases} \left(\left\lfloor \frac{i-1}{wt(T)} \right\rfloor + 1\right) \times wt(T) - (i-1), & \text{if } u = r(T_i) \\ i - \left(\left\lceil \frac{i}{wt(T)} \right\rceil - 1\right) \times wt(T), & \text{if } u = d(T_i) - 1 \\ wt(T), & \text{if } r(T_i) < u < d(T_i) - 1 \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

Fig. 3 shows the values of f for different subtasks of a task of weight $5/16$. $\mathit{flow}(T, u)$ is simply defined as $\mathit{flow}(T, u) = \sum_i f(T_i, u)$. In [12], we proved the following properties about flows.

(F1) For all time slots t, $\mathit{flow}(T, t) \leq wt(T)$.

(F2) Let T_i be a subtask of a GIS task and let T_k be its successor. If $b(T_i) = 1$ and $r(T_k) \geq d(T_i)$, then $\mathit{flow}(T, d(T_i) - 1) + \mathit{flow}(T, d(T_i)) \leq wt(T)$.

For example, in Fig. 3(b), $\mathit{flow}(T, 3) + \mathit{flow}(T, 4) = 1/16 < 5/16$ and $\mathit{flow}(T, 14) + \mathit{flow}(T, 15) = 5/16$.

From (9) and (10), we get $lag(T, t+1) = \sum_{u=0}^{t}(\mathit{flow}(T, u) - S(T, u)) = lag(T, t) + \mathit{flow}(T, t) - S(T, t)$. 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.

$$LAG(\tau, t+1) = LAG(\tau, t) + \sum_{T \in \tau} (\mathit{flow}(T, t) - S(T, t)) \tag{12}$$

($LAG(\tau, 0)$ is defined to be 0.) The lemma below is used in the following sections. It is proved in [12].

Lemma 4 *If $LAG(\tau, t) < LAG(\tau, t + 1)$, then there is a hole in slot t.*

3 Scheduling Dynamic Task Systems

As mentioned earlier, a static GIS task system is feasible iff its utilization is at most M. In particular, a valid schedule for such a task system can be shown to exist by constructing a flow network with a real-valued flow based on the *flow* values defined in Sec. 2 and illustrated in Fig. 3. A corresponding integral flow exists because all edge-capacities in the network are integers. This gives us a Pfair schedule. It is easy to extend this argument to show that any dynamic task system has a Pfair schedule if the total utilization of all tasks present at any time is at most M. This proof produces an offline schedule[4] in which *each subtask is scheduled in its PF-window.*

A condition for allowing tasks to join the system is an immediate consequence of this feasibility test, *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*? As shown in [6, 13], if an over-allocated task is allowed to leave and re-join immediately, then that task can effectively execute at a rate higher than its specified rate causing other tasks to miss their deadlines. Hence, we only consider allowing non-over-allocated tasks (*i.e.*, tasks with non-negative lags) to leave the system, as stated in (C1) below.

(C1) *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.[5] *Leave condition*: A task T can leave at time t iff $t \geq d(T_i)$, where T_i is the last subtask of T released before T leaves.

The condition $t \geq d(T_i)$ implies that $lag(T, t) = 0$. To see why, note that since T_i is the last-released subtask of T, T is neither under-allocated nor over-allocated at time $d(T_i)$. Thus, only tasks with zero lag are allowed to leave the system. It is easy to extend (C1) to also allow a task with positive lag to leave the system. This is because such a task is under-allocated, and hence its last-released subtask has not yet been scheduled. Intuitively, not scheduling a subtask is equivalent to removing it, and by Lemma 1, the removal of a subtask cannot lead to a missed deadline. Therefore, we can allow task T to leave the system if (C1) is satisfied by T_{i-1}, *i.e.*, the last-*scheduled* subtask of T. However, for simplicity, we assume (C1) as stated above in our proof.

In Sec. 4, we show that EPDF can be used to correctly schedule a restricted class of dynamic task systems provided (C1) is satisfied. In Sec. 5, we show that, in general, (C1) is not sufficient to guarantee all deadlines on multiprocessors when any of the known Pfair scheduling algorithms is used. On the other hand, we show that the following conditions are sufficient when PD^2 is used.

(C2) *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. *Leave condition*: Let T_i denote the last-released subtask of T. If T is light, then T can leave at time t iff either $t = d(T_i) \wedge b(T_i) = 0$ or $t > d(T_i)$ holds. If T is heavy, then T can leave at time t iff $t \geq D(T_i)$.

---

[4]The schedule produced is offline because the release time of each subtask needs to be known beforehand. An online algorithm that does not respect a fixed allocation quantum can also be easily obtained from the *flow* values.

[5]We consider a task that re-joins after having left the system to be a new task.

As before with (C1), when a task T leaves the system at time t, (C2) ensures that $lag(T,t) = 0$ holds. We can also allow T to leave with positive lag, provided its last-*scheduled* subtask satisfies the leave condition in (C2).

In the special case where a GIS task is periodic or sporadic and its final job always executes for its worst-case execution requirement, (C2) reduces to (C1). In other words, a periodic or a sporadic task can be allowed to leave at the deadline of its last job. To see why, note that if T_i is the last subtask of the final job, then $b(T_i) = 0$ (and hence $D(T_i) = d(T_i)$, if T is heavy). Thus, by (C2), T can leave at time $d(T_i)$, as in (C1).

4 Sufficiency of (C1) for EPDF

In [2], we proved that EPDF optimally schedules static IS task systems on two processors. In this section, we generalize this result by showing that *dynamic* task systems can be correctly scheduled using EPDF on M processors provided (C1) and (M1) below hold.

(M1) At any time, the sum of the weights of the $M-1$ heaviest tasks is at most 1.

We use the phrase "C1M1 task system" to refer to task systems satisfying both (C1) and (M1). We prove that such task systems can be correctly scheduled using EPDF by showing that an assumption to the contrary leads to a contradiction. Such an assumption implies the existence of a time t_d and an instance of a C1M1 task system τ as given in Definitions 1 and 2 below.

Definition 1 t_d is the earliest time at which some instance of a C1M1 task system misses a deadline under EPDF. □

Definition 2 τ is an instance of a C1M1 task system with the following properties.

(T1) τ misses a deadline under EPDF at t_d.

(T2) No instance of any C1M1 task system satisfying (T1) releases fewer subtasks in $[0, t_d]$ than τ.

(T3) No instance of any C1M1 task system satisfying (T1) and (T2) has a larger rank than τ, where the *rank* of an instance is the sum of the eligibility times of all subtasks with deadlines at most t_d.[6] □

By (T1), (T2), and Definition 1, exactly one subtask in τ misses its deadline: if several subtasks miss their deadlines, all but one can be removed and the remaining subtask will still miss its deadline, contradicting (T2).

We now prove several properties about the EPDF schedule for τ. All of these properties apply even when PD^2 is used, because PD^2 is a special case of EPDF in which ties are not broken arbitrarily. In Sec. 5, we use this lemma directly without re-stating it for PD^2.

Lemma 5 *The following properties hold for τ and S, the EPDF schedule for τ, where T_i is any subtask in S.*

[6] Note that these conditions are being applied *in sequence*. We are not, for example, claiming that τ is of maximal rank; rather, it has the maximal rank *among those C1M1 task systems satisfying* (T1) *and* (T2).

(a) *Let t be the time at which T_i is scheduled. Then, $e(T_i) \geq min(r(T_i), t)$.*

(b) *Let t be as in* (a). *If $d(T_i) > t+1$, then the successor of T_i is not eligible before $t+1$.*

(c) *For all T_i, $d(T_i) \leq t_d$.*

(d) *There are no holes in slot $t_d - 1$.*

(e) $LAG(\tau, t_d) = 1$.

(f) $LAG(\tau, t_d - 1) \geq 1$.

(g) *There exists a time t such that $0 \leq t < t_d - 1$, $LAG(\tau, t) < 1$, and $LAG(\tau, t+1) \geq 1$.*

Proof of (a): Suppose that $e(T_i) < min(r(T_i), t)$. Consider the C1M1 task system τ' obtained from τ by changing the eligibility time of T_i to $min(r(T_i), t)$. τ' is feasible because the feasibility proof produces a schedule in which each subtask is scheduled in its PF-window (refer to Sec. 3). Note that τ' has a larger rank than τ. It is easy to see that the relative priorities of the subtasks do not change for any slot $0 \leq u \leq t_d - 1$, and hence, the EPDF schedules for τ' and τ are the same. Thus, τ' misses a deadline at t_d, contradicting (T3).

Proof of (b): Because T_i is scheduled at t, the successor of T_i is scheduled at or after $t+1$. Also, if $d(T_i) > t+1$, then by (5), (6), and (7), we have $r(T_{i+1}) \geq t+1$. Hence, $r(T_k) \geq t+1$ for all $k > i$. In particular, if T_k is the successor of T_i, then $r(T_k) \geq t+1$. Therefore, by (a), $e(T_k) \geq t+1$.

Proof of (c): Suppose τ contains a subtask U_j with a deadline greater than t_d. U_j can be removed without affecting the scheduling of higher-priority subtasks with earlier deadlines. Thus, if U_j is removed, then a deadline is still missed at t_d. This contradicts (T2).

Proof of (d): If there were a hole in slot $t_d - 1$, then the subtask that misses its deadline at t_d would have been scheduled there by EPDF. Contradiction. (Note that its predecessor could not have been scheduled in slot $t_d - 1$, because its predecessor meets its deadline at or before time $t_d - 1$.)

Proof of (e): By (12), we have

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

The first term on the left-hand side of the above equation is the total flow in $[0, t_d]$, which is equal to the total number of subtasks in τ. (This is true even if tasks leave the system before t_d because, by (C1), leaving tasks have zero lag.) The second term corresponds to the number of subtasks scheduled by EPDF in $[0, t_d]$. Since exactly one subtask misses its deadline, the difference between these two terms is 1, *i.e.*, $LAG(\tau, t_d) = 1$.

Proof of (f): By (d), there are no holes in slot $t_d - 1$. Hence, by Lemma 4, $LAG(\tau, t_d - 1) \geq LAG(\tau, t_d)$. Therefore, by (e), $LAG(\tau, t_d - 1) \geq 1$.

Proof of (g): This follows from the fact that $LAG(\tau, 0) = 0$ and $LAG(\tau, t_d - 1) \geq 1$ (from (f)). $\square$

We now show that time t as defined below cannot exist, contradicting part (g) of Lemma 5.

$$0 \leq t < t_d - 1 \ \wedge \ LAG(\tau, t) < 1 \ \wedge \ LAG(\tau, t+1) \geq 1. \tag{13}$$

By (13), $LAG(\tau, t) < LAG(\tau, t+1)$. Hence, by Lemma 4, *there is at least one hole in slot t.*

Lemma 6 *Let U_j be the subtask of U with the largest index such that $e(U_j) \leq t < d(U_j)$. If no subtask of U is scheduled at time t, then U_j is scheduled before t and $d(U_j) = t + 1$.*

Proof: Because there is a hole in slot t, if U_j is not scheduled at or before t, then EPDF must have scheduled a previous subtask of U in slot t. Because no subtask of U is scheduled in slot t, U_j is thus scheduled before t.

We now show that $d(U_j) = t + 1$. Assume to the contrary that the following holds.

$$d(U_j) > t + 1 \tag{14}$$

Let τ' be the instance of the task system obtained by removing U_j from τ. Let S' be the EPDF schedule for τ'. Let the chain of displacements caused by removing U_j be $\Delta_1, \Delta_2, \ldots, \Delta_k$, where $\Delta_i = \langle X^{(i)}, t_i, \ X^{(i+1)}, t_{i+1} \rangle$, $X^{(1)} = U_j$ and t_1 is the slot in which U_j is scheduled. Recall that U_j is scheduled before t. Therefore, $t_1 < t$.

By Lemma 1, $t_{i+1} > t_i$ for all $1 \leq i \leq k$. Note that at slot t_i, subtask $X^{(i)}$ has higher priority than $X^{(i+1)}$, because $X^{(i)}$ was chosen over $X^{(i+1)}$ in S. Thus, because $X^{(1)} = U_j$, we have $d(X^{(i)}) \geq d(U_j)$ for each $X^{(i)}, 1 \leq i \leq k+1$. Hence, by (14), $d(X^{(i)}) > t+1$ and by part (b) of Lemma 5, we have the following property.

(E) The eligibility time of the successor of $X^{(i)}$ (if it exists in τ) is at least $t + 1$ for all $i \in [1, k+1]$.

We now show that the displacements do not extend beyond slot t. Assume, to the contrary, that $t_{k+1} > t$. Consider $h \in \{2, \ldots, k+1\}$ such that $t_h > t$ and $t_{h-1} \leq t$, as depicted in Fig. 4(b). Such an h exists because $t_1 < t < t_{k+1}$. Because there is a hole in slot t and $t_{h-1} \leq t < t_h$, by Lemma 3, $t_{h-1} = t$ and $X^{(h)}$ must be the successor of $X^{(h-1)}$. Therefore, by (E), $e(X^{(h)}) \geq t + 1$. This implies that Δ_{h-1} is not a valid displacement.

Thus, the displacements do not extend beyond slot t, implying that no subtask scheduled after t is left-shifted. Hence, a deadline is still missed at t_d in S', contradicting (T2). Thus, $d(U_j) = t + 1$. $\square$

Lemma 7 *Let U_j be a subtask scheduled in slot t. Then, $d(U_j) = t + 1$.*

Proof: By (13), $t < t_d - 1$. Thus, by Definition 1, no deadline is missed in $[0, t+1)$. Because U_j is scheduled in slot t, *i.e.*, $[t, t+1)$, we have $d(U_j) \geq t + 1$. Suppose that $d(U_j) > t + 1$. Then, by part (b) of Lemma 5, the successor of U_j (if it exists) is not eligible before $t + 1$. Hence, by Lemma 2, we can remove U_j and no displacements will result, *i.e.*, a deadline is still missed at t_d, contradicting (T2). Therefore, $d(U_j) = t + 1$. $\square$

We partition the tasks in τ into three different sets: A, B and I. The set A consists of tasks that are scheduled <u>at</u> time t. The set B includes each task T that is not in A and is "active" at time t. A task T is *active*

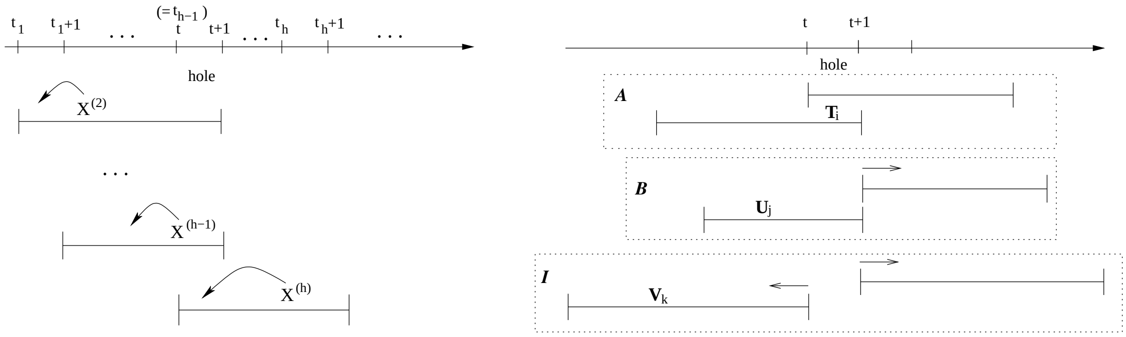

Figure 4: IS-windows are denoted by line segments. **(a)** Lemma 7. There is a hole in slot t. $X^{(h)}$ must be the successor of $X^{(h-1)}$ in τ. **(b)** Sets A, B, and I. The IS-windows of a sample task of each set are shown. An arrow over a window indicates that the window could be anywhere in the direction of the arrow (or the window may be absent).

at time t if there exists a subtask T_i such that $e(T_i) \le t < d(T_i)$. (A task may be inactive either because it has already left the system or because of a late subtask release.) By Lemma 6, all such subtasks are scheduled before t. The set I consists of the remaining tasks, which are inactive at time t. Note that $A \cup B \cup I = \tau$. Fig. 4(b) illustrates how the subtasks of A, B, and I are scheduled. We now prove some properties about these sets of tasks.

Lemma 8 *For each task $T \in A$, $lag(T, t+1) < wt(T)$.*

Proof: Let T_i be the subtask of T scheduled at time t. By Lemma 7, $d(T_i) = t+1$. Since $t < t_d - 1$ (by (13)), no deadline is missed in $[0, t+1)$. Thus, the total allocation to T in $[0, t+1)$ in the EPDF schedule equals the number of subtasks of T that have a deadline of at most $t+1$. Recall that in the fluid schedule (refer to (11)), each subtask is assigned a share only in its PF-window and the total allocation to each subtask is 1. Hence, the total allocation to task T in $[0, t+1)$ in the fluid schedule equals the number of subtasks with a deadline at most $t+1$ plus the flow in $[0, t+1)$ due to later subtasks.

Because $d(T_i) = t+1$ and because two consecutive PF-windows overlap by at most one slot, the difference between the allocations in the fluid and the EPDF schedules in $[0, t+1)$ is either 0 (if T_i has no successor) or $f(T_k, t)$, where T_k is the successor of T_i. In other words, $lag(T, t+1) \le f(T_k, t)$.

If $r(T_k) > t$, then by (11), $f(T_k, t) = 0$, and the required result follows. On the other hand, if $r(T_k) = t$, then we have $r(T_k) = d(T_i) - 1$. Therefore, by (5)–(7), we have $k = i+1$ and $\theta(T_{i+1}) = \theta(T_i)$. In other words, $r(T_{i+1}) = d(T_i) - 1$, which by (5) and (6) implies that $\lfloor i/wt(T) \rfloor = \lceil i/wt(T) \rceil - 1$. Hence, $\lfloor i/wt(T) \rfloor < i/wt(T)$.

Because $r(T_{i+1}) = t$, by (11), we have $f(T_{i+1}, t) = (\lfloor i/wt(T) \rfloor + 1) \cdot wt(T) - i$. Because $\lfloor i/wt(T) \rfloor < i/wt(T)$, $f(T_{i+1}, t) < (i/wt(T) + 1) \cdot wt(T) - i$. Hence, $f(T_{i+1}, t) < wt(T)$. Thus, in all cases, $lag(T, t+1) < wt(T)$. □

Lemma 9 *For each task $U \in B \cup I$, $lag(U, t+1) = 0$.*

Proof: We first show that the windows of the tasks in B and I are as shown in Fig. 4(b). For tasks in I, this easily follows from the definition of I. For tasks in B, we reason as follows. Let $U \in B$ and let U_j be the subtask of U with the largest index such that $e(U_j) \le t < d(U_j)$. Then, by Lemma 6, $d(U_j) = t + 1$. Furthermore, if U_k is the successor of U_j, then $e(U_k) > t$ (by the choice of U_j), which implies that $r(U_k) > t$, as shown in Fig. 4(b).

Because no deadlines are missed at or before $t + 1$ ($< t_d$), at the end of slot t, the total number of quanta received by $U \in B \cup I$ equals the number of subtasks of U released in $[0, t+1)$. (Also, recall that any task that leaves the system does so with zero lag.) Thus, $lag(U, t+1) = 0$. □

Because $A \cup B \cup I = \tau$, we have $LAG(\tau, t+1) = \sum_{U \in A \cup B \cup I} lag(U, t+1)$. Hence, by Lemmas 8 and 9, we have $LAG(\tau, t+1) < \sum_{T \in A} wt(T)$. Because there is a hole in slot t, the number of subtasks scheduled in slot t is at most $M - 1$, *i.e.*, $|A| \le M - 1$. Therefore, by (M1), $\sum_{T \in A} wt(T) \le 1$. This implies that $LAG(\tau, t+1) < 1$, contradicting our starting assumption about t in (13). Hence, such a t cannot exist, implying that t_d and τ as defined cannot exist. Thus, we have the following theorem.

Theorem 1 *EPDF correctly schedules any* C1M1 *task system on M processors.*

Note that, at every join request, ensuring the validity of (M1) involves identifying the $M - 1$ heaviest tasks and summing their weights. A more efficient (and more restrictive) way to enforce (M1) is to require each individual task weight to be at most $1/(M-1)$. Thus, we have the following.

Corollary 1 *EPDF correctly schedules any dynamic GIS task system satisfying* (C1) *on M (> 1) processors if the weight of each task is at most $1/(M-1)$.*

Because any feasible static task system satisfies (C1) by default, we have the following corollary. This generalizes an earlier result that EPDF is optimal for scheduling IS tasks on two processors [2].

Corollary 2 *EPDF correctly schedules any feasible static GIS task system on M (> 1) processors if the weight of each task is at most $1/(M-1)$.*

By Corollary 2, EPDF is optimal for scheduling light GIS tasks on three processors.

Improving (M1). It is possible to obtain conditions less restrictive than (M1) by more accurately bounding $lag(T, t+1)$ for $T \in A$. Let $T.f = \frac{T.e - gcd(T.e, T.p)}{T.p}$. Using (11), it is possible to show that if $d(T_i) = r(T_{i+1}) + 1$, then $T.f$ is the maximum share that subtask T_{i+1} can get in slot $r(T_{i+1})$. (We omit the proof here due to space limitations.) Thus, we can improve Lemma 8 to show that $lag(T, t+1) \le T.f$ for $T \in A$. Performing the same analysis as above, we obtain a contradiction if $\sum_{T \in A} T.f < 1$. Thus, EPDF produces a correct schedule if, at all times, $\sum_{T \in H} T.f < 1$ for all sets H consisting of at most $M - 1$ tasks.

5 Sufficiency of (C2) for PD^2

We now show that (C1) by itself is not sufficient in the general case when task weights are not restricted, even if PF, PD or PD^2 is used for scheduling. Later, we show that (C2) is sufficient when used with PD^2.

5.1 Insufficiency of (C1)

Our proof that (C1) is not sufficient applies to any "weight-consistent" Pfair scheduling algorithm. An algorithm is *weight-consistent* if the following holds: given two tasks T and U of equal weight and that have eligible subtasks T_i and U_j, respectively, such that $i = j$ and $r(T_i) = r(U_j)$ (and hence, $d(T_i) = d(U_j)$), T_i is favored over a third subtask V_k iff U_j is favored over V_k. This condition holds for any algorithm that determines the relative priorities of two tasks independently of other tasks. All known Pfair scheduling algorithms are weight-consistent.

Theorem 2 *No weight-consistent scheduler can guarantee all deadlines on multiprocessors if* (C1) *is used.*

Proof: Consider a class of task systems consisting of two sets of tasks X and Y of weights $w_1 = 2/5$ and $w_2 = 3/8$, respectively. Let X_f (Y_f) denote the set of first subtasks of tasks in X (Y). We construct a task system depending on the task weight favored by the scheduler. We say that X_f *is favored* (and analogously for Y_f) if, whenever subtasks in X_f and Y_f are released at the same time, those in X_f are favored over those in Y_f.

Case 1: X_f is favored. Consider a dynamic task system consisting of the following types of tasks to be scheduled on 15 processors. (In each counterexample that follows, no subtask is eligible before its PF-window.)

Type A: 8 tasks of weight w_2 that join at time 0.

Type B: 30 tasks of weight w_1 that join at time 0 and leave at time 3.

Type C: 30 tasks of weight w_1 that join at time 3.

Because $30w_1 + 8w_2 = 15$, this task system is feasible, and the join condition for type-C tasks in (C1) is satisfied. Note that $d(T_1) \leq 3$ for every type-B task T; hence, the leave condition in (C1) is also satisfied.

Since subtasks in X_f are favored, type-B tasks are favored over type-A tasks at times 0 and 1. Hence, the schedule for $[0, 3)$ will be as shown in Fig. 5(a). Consider the interval $[3, 8)$. Each type-A task has two subtasks remaining for execution, which implies that the type-A tasks need 16 quanta. Similarly, each type-C task also has two subtasks remaining, which implies that the type-C tasks need 60 quanta. However, the total number of quanta available in $[3, 8)$ is $15 \cdot (8 - 3) = 75$. Thus, one subtask will miss its deadline at or before time 8.

Case 2: Y_f is favored. Consider a dynamic task system consisting of the following types of tasks to be scheduled on 8 processors.

Type A: 5 tasks of weight w_1 that join at time 0.

Type B: 16 tasks of weight w_2 that join at time 0 and leave at time 3.

Type C: 16 tasks of weight w_2 that join at time 3.

Because $5w_1 + 16w_2 = 8$, this task system is feasible, and the join condition for type-C tasks in (C1) is satisfied. Note that $d(T_1) \leq 3$ for every type-B task T; hence, the leave condition in (C1) is also satisfied.

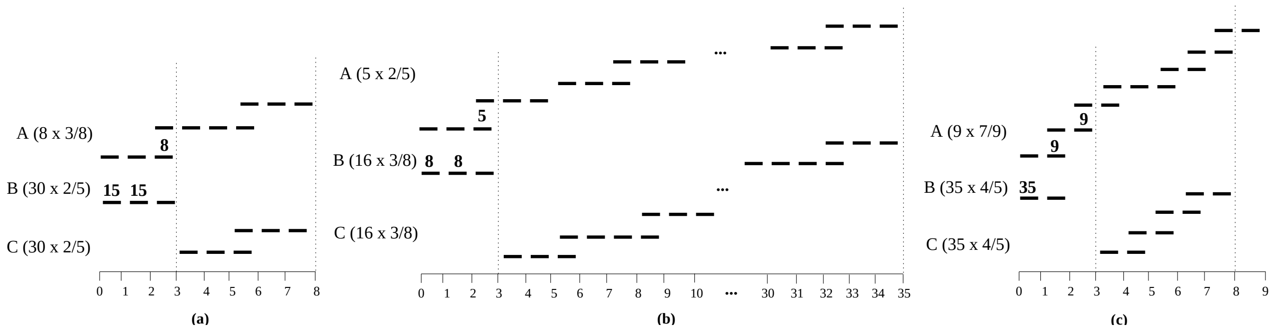

Figure 5: Counterexamples demonstrating insufficiency of (C1) and tightness of (C2). An integer value n in slot t of some window means that n of the subtasks that must execute within that window are scheduled in slot t. No integer value means that no such subtask is scheduled in slot t. The vertical lines depict intervals with excess demand. **(a)** Theorem 2. Case 1: Tasks of weight $2/5$ are favored at times 0 and 1. **(b)** Theorem 2. Case 2: Tasks of weight $3/8$ are favored at times 0 and 1. **(c)** Theorem 3. The tasks of weight $4/5$ are allowed to leave at time 3 and re-join immediately.

Since subtasks in Y_f are favored, type-B tasks are favored over type-A tasks at times 0 and 1. Hence, the schedule for $[0, 3)$ will be as shown in Fig. 5(b). Consider the interval $[3, 35)$. The number of subtasks of each type-A task that need to be executed in $[3, 35)$ is $1 + (35 - 5) \cdot 2/5 = 13$. Similarly, the number of subtasks of each type-C task is $(35 - 3) \cdot 3/8 = 12$. The total is $5 \cdot 13 + 16 \cdot 12 = 257$, whereas the number of quanta available in $[3, 35)$ is $(35 - 3) \cdot 8 = 256$. Thus, one subtask will miss its deadline at or before time 35. □

Theorem 2 can be "circumvented" if it can be known at the time a subtask is *released* whether it is the final subtask of its task. For example, in Fig. 5(a), if we knew that the first subtask T_1 of each type-B task is its last, then we could have given T_1 an effective b-bit of zero. Hence, PD^2 would have scheduled it with a lower priority than any type-A task. However, in general, such knowledge may not be available to the scheduler.

The examples given in insets (a) and (b) of Fig. 5 show that allowing a light task T to leave at $d(T_i)$ when $b(T_i) = 1$ can lead to missed deadlines. Thus, for light tasks, we cannot improve upon (C2) when using PD^2 (or any known Pfair algorithm). We now show the same for heavy tasks.

Theorem 3 *If a heavy task is allowed to leave before $D(T_i)$, where T_i is the last subtask of T that is released before it leaves the system, then there exists task systems that are not correctly scheduled using PD^2.*

Proof: Consider the following dynamic task system to be scheduled on 35 processors, where $2 \le t \le 4$.

Type A: 9 tasks of weight $7/9$ that join at time 0.

Type B: 35 tasks of weight $4/5$ that join at time 0 and leave at time t. Each task releases only one subtask.

Type C: 35 tasks of weight $4/5$ that join at time t.

According to PD^2, the priorities of the type-A and type-B tasks are the same at time 0, because each has a deadline at time 2, a b-bit of 1 and a group deadline at time 5. Hence, the type-B tasks may be given higher priority.[7] Assuming this scenario, Fig. 5(c) depicts the schedule for the case of $t = 3$.

[7]This counterexample applies to PF and PD as well, because each also favors the type-B tasks at time 0.

Consider the interval $[t, t+5)$. Each type-A and type-C task has four subtasks with deadlines in this interval (see Fig. 5(c)). Therefore, the total number of subtasks that need to be executed in $[t, t+5)$ is $9 \cdot 4 + 35 \cdot 4 = 35 \cdot 5 + 1$. Since $35 \cdot 5$ quanta are available over $[t, t+5)$, one subtask will miss its deadline. $\square$

5.2 Sufficiency of (C2)

We now prove that PD^2 correctly schedules any dynamic task system for which (C2) holds. Define task system τ and time t_d in the same manner as in Sec. 4, except substitute PD^2 for EPDF, and consider task systems satisfying (C2) instead of (C1) and (M1). Let S denote the PD^2 schedule for τ. We show that a contradiction results from these definitions, as before. As mentioned earlier, Lemma 5 applies to PD^2 schedules as well. We now strengthen part (b) of Lemma 5 as follows.

Lemma 10 *Let t be the slot in which a subtask T_i is scheduled. If either $d(T_i) > t+1$ or $d(T_i) = t+1 \wedge b(T_i) = 0$, then the successor of T_i is not eligible before $t+1$.*

Proof: If $d(T_i) > t+1$, then the result follows from part (b) of Lemma 5. If $d(T_i) = t+1 \ \wedge \ b(T_i) = 0$, then by (4), $r(T_{i+1}) = d(T_i)$. Therefore, by (7), if T_k is the successor of T_i, then $r(T_k) \geq t+1$. Because T_i is scheduled in slot t, T_k is scheduled at or after $t+1$. Hence, by part (a) of Lemma 5, $e(T_k) \geq t+1$. $\square$

As in Sec. 4, there exists a time t satisfying (13), *i.e.*, $0 \leq t < t_d - 1$, $LAG(\tau, t) < 1$, and $LAG(\tau, t+1) \geq 1$. Without loss of generality, let t be the latest such slot, *i.e.*, for all u such that $t < u \leq t_d - 1$, $LAG(\tau, u) \geq 1$. We now show that t cannot exist, thus contradicting our starting assumption that t_d and τ exist. We define sets A, B, and I in the same way as in Sec. 4 (refer to Fig. 4(b)). We begin by proving certain properties about B.

Lemma 11 *B is non-empty.*

Proof: By Lemma 4, there is at least one hole in slot t. Let the number of these holes be h. Then, the number of allocations in slot t in S is $M - h$, *i.e.*, $\sum_{T \in \tau} S(T, t) = M - h$. By (12), $LAG(\tau, t+1) = LAG(\tau, t) + \sum_{T \in \tau}(\mathit{flow}(T, t) - S(T, t))$. Thus, because $LAG(\tau, t) < LAG(\tau, t+1)$, we have $\sum_{T \in \tau} \mathit{flow}(T, t) > M - h$. Because for every $V \in I$, either $(d(V_k) < t)$ or $(e(V_k) > t)$, by (11), $\mathit{flow}(V, t) = 0$. It follows that $\sum_{T \in A \cup B} \mathit{flow}(T, t) > M - h$. Therefore, by (F1), $\sum_{T \in A \cup B} wt(T) > M - h$. Because the number of tasks scheduled in slot t is $M - h$, $|A| = M - h$. Because $wt(T) \leq 1$ for any task T, $\sum_{T \in A} wt(T) \leq M - h$. Thus, $\sum_{T \in B} wt(T) > 0$. Hence, B is not empty. $\square$

Lemma 12 *Let U be any task in B. Let U_j be the subtask with the largest index such that $e(U_j) \leq t < d(U_j)$. Then, $d(U_j) = t+1 \ \wedge \ b(U_j) = 1$.*

Proof: By Lemma 6, $d(U_j) = t+1$. Suppose that the following holds.

$$d(U_j) = t+1 \ \wedge \ b(U_j) = 0 \tag{15}$$

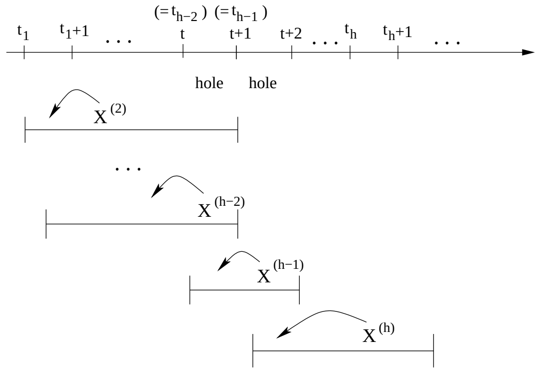

Figure 6: Lemma 13. IS-windows are denoted by line segments. If there is a hole in both slots t and $t+1$, then $X^{(h-2)}$ and $X^{(h-1)}$ must be scheduled at t and $t+1$ in S, respectively. Also, in τ, $X^{(h)}$ must be the successor of $X^{(h-1)}$, which in turn, must be the successor of $X^{(h-2)}$.

As in the proof of Lemma 6, we show that U_j can be removed and a deadline will still be missed at t_d, contradicting (T2). Let the chain of displacements caused by removing U_j be $\Delta_1, \Delta_2, \ldots, \Delta_k$, where $\Delta_i = \langle X^{(i)}, t_i, X^{(i+1)}, t_{i+1} \rangle$ and $X^{(1)} = U_j$. By Lemma 1, $t_{i+1} > t_i$ for $1 \le i \le k$.

Note that at slot t_i, the priority of subtask $X^{(i)}$ is at least that of $X^{(i+1)}$, because $X^{(i)}$ was chosen over $X^{(i+1)}$ in S. Thus, because $X^{(1)} = U_j$, by (15), for each subtask $X^{(i)}, 1 \le i \le k+1$, either $d(X^{(i)}) > t+1$ or $d(X^{(i)}) = t+1 \ \wedge \ b(X^{(i)}) = 0$. Therefore, by Lemma 10, property (E) as stated in Lemma 6 holds. By reasoning exactly as in the proof of Lemma 6, a contradiction can be reached, which shows that (15) does not hold. Hence, $d(U_j) = t+1 \ \wedge \ b(U_j) = 1$. $\square$

The following property is used in the proof of the next lemma. It follows from the fact that light tasks have PF-windows of length at least three and consecutive PF-windows overlap by at most one slot [1].

(L) For a light task T, if T_k is the successor of T_i, then $d(T_k) \ge d(T_i) + 2$.

Lemma 13 *Suppose there is at least one light task in B. Then, there is no hole in slot $t+1$.*

Proof: By (13), $t < t_d - 1$, and therefore, $t+1 \le t_d - 1$. Suppose that there is a hole in slot $t+1$. By part (d) of Lemma 5, $t+1 < t_d - 1$, *i.e.*,

$$t + 2 \le t_d - 1. \tag{16}$$

Let U be a light task in B and let U_j be the subtask of U with the largest index such that $e(U_j) \le t < d(U_j)$. Our approach is the same as in the proof of Lemma 12. Let the chain of displacements caused by removing U_j be $\Delta_1, \Delta_2, \ldots, \Delta_k$, where $\Delta_i = \langle X^{(i)}, t_i, X^{(i+1)}, t_{i+1} \rangle$ and $X^{(1)} = U_j$. By Lemma 1, we have $t_{i+1} > t_i$ for all $i \in [1, k]$. Also, the priority of $X^{(i)}$ is at least that of $X^{(i+1)}$ at t_i, because $X^{(i)}$ was chosen over $X^{(i+1)}$ in S. Because U is light and $d(U_j) = t+1 \ \wedge \ b(U_j) = 1$ (by Lemma 12), this implies the following.

(P) For all $i \in [1, k+1]$, either **(i)** $d(X^{(i)}) > t+1$ or **(ii)** $d(X^{(i)}) = t+1$ and $X^{(i)}$ is the subtask of a light task.

Suppose the chain of displacements extends beyond $t+1$, *i.e.*, $t_{k+1} > t+1$. Consider $h \in \{1, \ldots, k+1\}$ such that that $t_h > t+1$ and $t_{h-1} \le t+1$. Because there is a hole in slot $t+1$ and $t_{h-1} \le t+1 < t_h$, by

Lemma 3, $t_{h-1} = t+1$ and $X^{(h)}$ is the successor of $X^{(h-1)}$. Similarly, because there is a hole in slot t, $t_{h-2} = t$ and $X^{(h-1)}$ is the successor of $X^{(h-2)}$. This is illustrated in Fig. 6.

By (P), either $d(X^{(h-2)}) > t+1$ or $d(X^{(h-2)}) = t+1$ and $X^{(h-2)}$ is the subtask of a light task. In either case, $d(X^{(h-1)}) > t+2$. To see why, note that if $d(X^{(h-2)}) > t+1$, then because $X^{(h-1)}$ is the successor of $X^{(h-2)}$, by (6), $d(X^{(h-1)}) > t+2$. On the other hand, if $d(X^{(h-2)}) = t+1$ and $X^{(h-2)}$ is the subtask of a light task, then, by (L), $d(X^{(h-1)}) > t+2$.

Now, because $X^{(h-1)}$ is scheduled at $t+1$, by Lemma 10, the successor of $X^{(h-1)}$ is not eligible before $t+2$, *i.e.*, $e(X^{(h)}) \geq t+2$. This implies that the displacement Δ_{h-1} is not valid. Thus, the chain of displacements cannot extend beyond time $t+2$. Hence, because $t+2 \leq t_d - 1$ (by (16)), removing U_j cannot cause a missed deadline at t_d to be met. This contradicts (T2). Therefore, there is no hole in slot $t+1$. □

The following lemma is the counterpart of Lemma 13 for heavy tasks.

Lemma 14 *Let U be a heavy task in B and let U_j be the subtask of U with the largest index such that $e(U_j) \leq t < d(U_j)$. Then, there exists a slot $u \in \{d(U_j), \ldots, \min(D(U_j), t_d) - 1\}$ such that there is no hole in u.*

Proof sketch: We do not give a complete proof here due to space limitations. The approach is similar to that in the proof of Lemma 13, except that the interval under consideration is longer. If $\min(D(U_j), t_d) = t_d$, then by part (d) of Lemma 5, $u = t_d - 1$ satisfies the stated requirement. In the case where $D(U_j) < t_d$, if there is a hole in every slot in $\{d(U_j), \ldots, D(U_j) - 1\}$, then we show that U_j can be removed without changing the schedule beyond time $D(U_j)$. Thus, a deadline is still missed at t_d, contradicting (T2). □

Lemma 15 *Suppose that B contains at least one light task. Then, $LAG(\tau, t+2) < 1$.*

Proof: By (13), $LAG(\tau, t) < 1$ and $LAG(\tau, t+1) \geq 1$. Therefore, by Lemma 4, there is at least one hole in slot t. Let the number of such holes be h. We now derive properties about the *flow* values in slots t and $t+1$.

By the definition of I, only tasks in $A \cup B$ are active at time t. Thus, $\sum_{T \in \tau} flow(T,t) = \sum_{T \in A \cup B} flow(T,t)$. Because $wt(T) \leq 1$ for any task T, $\sum_{T \in A} wt(T) \leq |A|$. Hence, by (F1), $\sum_{T \in A} flow(T,t) \leq |A|$. Now, because there are h holes in slot t, $M - h$ tasks are scheduled at t, *i.e.*, $|A| = M - h$. Thus, $\sum_{T \in A} flow(T,t) \leq M - h$. Therefore,

$$\sum_{T \in \tau} flow(T,t) \leq M - h + \sum_{T \in B} flow(T,t). \tag{17}$$

Consider any task $U \in B$. Let U_j be the subtask of U with the largest index such that $e(U_j) \leq t < d(U_j)$. Let C denote the set of such subtasks for all the tasks in B. Then, by Lemma 12,

$$\text{for all } U_j \in C,\ d(U_j) = t+1 \ \wedge\ b(U_j) = 1. \tag{18}$$

If U is heavy, then this would imply that $D(U_j) > t+1$. (It is easy to show that for any subtask T_i of a heavy task T, $D(T_i) = d(T_i)$ holds iff $b(T_i) = 0$.) Thus, the leave condition in (C2) is not satisfied at time $t+1$ for any task in B, and hence no such task leaves at time $t+1$.

Let A' (respectively, I') denote the set of tasks in A (respectively, I) that are active at time $t+1$. Then, the set of active tasks at time $t+1$ is $A' \cup I' \cup B$. Therefore, by the join condition in (C2),

$$\sum_{T \in A' \cup I' \cup B} wt(T) \leq M. \tag{19}$$

Also, $\sum_{T \in \tau} \mathit{flow}(T, t+1) = \sum_{T \in A' \cup I' \cup B} \mathit{flow}(T, t+1)$. By (F1), this implies that $\sum_{T \in \tau} \mathit{flow}(T, t+1) \leq \sum_{T \in A' \cup I'} wt(T) + \sum_{T \in B} \mathit{flow}(T, t+1)$. Therefore, by (17),

$$\sum_{T \in \tau} (\mathit{flow}(T, t) + \mathit{flow}(T, t+1)) \leq M - h + \sum_{T \in A' \cup I'} wt(T) + \sum_{T \in B} (\mathit{flow}(T, t) + \mathit{flow}(T, t+1)) \tag{20}$$

Consider $U_j \in C$ (hence, $U \in B$). Let U_k denote the successor of U_j. Because U_j is the subtask with the largest index such that $e(U_j) \leq t < d(U_j)$, we have $e(U_k) \geq t+1$. Hence, $r(U_k) \geq t+1$. By (18), we have $d(U_j) = t+1$. Therefore, by (F2), $\mathit{flow}(U, t) + \mathit{flow}(U, t+1) \leq wt(U)$ for each $U \in B$. By (20), this implies that $\sum_{T \in \tau} (\mathit{flow}(T, t) + \mathit{flow}(T, t+1)) \leq M - h + \sum_{T \in A' \cup I' \cup B} wt(T)$. Therefore, from (19), it follows that

$$\sum_{T \in \tau} (\mathit{flow}(T, t) + \mathit{flow}(T, t+1)) \leq M - h + M = 2M - h. \tag{21}$$

By the statement of the lemma, B contains at least one light task. Therefore, by Lemma 13, there is no hole in slot $t+1$. Because there are h holes in slot t, we have $\sum_{T \in \tau} (S(T, t) + S(T, t+1)) = M - h + M = 2M - h$.

Hence, by (21), $\sum_{T \in \tau} (\mathit{flow}(T, t) + \mathit{flow}(T, t+1)) \leq \sum_{T \in \tau} (S(T, t) + S(T, t+1))$. Using this relation in the identity (obtained from (12)), $\mathit{LAG}(\tau, t+2) = \mathit{LAG}(\tau, t) + \sum_{T \in \tau} (\mathit{flow}(T, t) + \mathit{flow}(T, t+1)) - \sum_{T \in \tau} (S(T, t) + S(T, t+1))$, and the fact that $\mathit{LAG}(\tau, t) < 1$, we obtain $\mathit{LAG}(\tau, t+2) < 1$. □

The following lemma generalizes Lemma 15 by allowing B to consist solely of heavy tasks.

Lemma 16 *There exists $v \in \{t+2, \ldots, t_d\}$ such that $\mathit{LAG}(\tau, v) < 1$.*

Proof sketch: By Lemma 11, B is non-empty. If B contains a light task, then by Lemma 15, $t+2$ satisfies the stated requirement. Suppose that B consists solely of heavy tasks. Let U_j be the subtask of task $U \in B$ with the largest index such that $e(U_j) \leq t < d(U_j)$. Let L_i be the lowest-priority subtask among all such subtasks. By Lemma 14, there is a slot in $\{t, \ldots, \min(D(L_i), t_d) - 1\}$ with no hole. Let u be the earliest such slot. Because there is a hole in t (by 13), $u > t$. It can be shown that the following holds for every $v \in \{t+1, \ldots, u-1\}$.

- All tasks in B are inactive in slot v.
- $\mathit{LAG}(\tau, v+1) \leq \mathit{LAG}(\tau, v)$, *i.e.*, LAG decreases monotonically over $t+1, \ldots, u$.

(Also, by the leave condition in (C2), no tasks in B can leave at or before $u < D(L_i)$.) Using these properties and conducting an analysis similar to that used in the proof of Lemma 15, we can show that $\mathit{LAG}(t_d, u+1) < 1$. Because $t+2 \leq u+1 \leq t_d$, we obtain the required result. □

Recall our assumption that t is the latest time such that $LAG(\tau, t) < 1$ and $LAG(\tau, t+1) \geq 1$. Because $t \leq t_d - 2$ (by (13)), we have $t+2 \leq t_d$. By Lemma 16, $LAG(\tau, v) \leq 0$ for some $v \in \{t+2, \ldots, t_d\}$. By Lemma 5, parts (e) and (f), v cannot be t_d or $t_d - 1$. Thus, $v \leq t_d - 2$. Because $LAG(\tau, t_d) \geq 1$, this contradicts the maximality of t. Therefore, t_d and τ as defined cannot exist. Thus, we have the following theorem.

Theorem 4 *PD^2 correctly schedules any dynamic GIS task set satisfying* (C2).

6 Conclusions

In this paper, we have addressed the problem of scheduling a dynamic GIS task system on multiprocessors. We have shown that if the sum of the weights of the $M-1$ heaviest tasks is at most 1, then the uniprocessor join/leave conditions presented previously [6, 13] are sufficient to avoid deadline misses on M processors when EDPF is used. This result applies to any EPDF-based algorithm, and hence to PD^2, as well. We have also provided join/leave conditions for the general case in which weights are not restricted in this way and when the dynamic task system is scheduled using PD^2. We have further shown that, in general, it is not possible to improve upon these conditions.

Acknowledgments: We are grateful to Philip Holman for valuable comments on an earlier draft of this paper.

References

[1] 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.

[2] J. Anderson and A. Srinivasan. Pfair scheduling: Beyond periodic task systems. In *Proc. of the 7th International Conference on Real-Time Computing Systems and Applications*, pages 297–306, Dec. 2000.

[3] J. Anderson and A. Srinivasan. Mixed pfair/erfair scheduling of asynchronous periodic tasks. In *Proc. of the 13th Euromicro Conference on Real-Time Systems*, pages 76–85, June 2001.

[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] S. Baruah, J. Gehrke, C. G. 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*, 26(1):43–51, Jan. 1998.

[7] A. Chandra, M. Adler, P. Goyal, and P. Shenoy. Surplus fair scheduling: A proportional-share cpu scheduling algorithm for symmetric multiprocessors. In *Proc. of the Fourth ACM Symposium on Operating System Design and Implementation (OSDI)*, Oct 2000.

[8] A. Chandra, M. Adler, and P. Shenoy. Deadline fair scheduling: Bridging the theory and practice of proportionate-fair scheduling in multiprocessor servers. In *Proc. of the 7th IEEE Real-Time Technology and Applications Symposium*, May 2001.

[9] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[10] A. K. Mok. Fundamental design problems of distributed systems for the hard-real-time environment. Technical Report MIT/LCS/TR-297, Massachussets Institute of Technology, 1983.

[11] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham. Mode change protocols for priority-driven pre-emptive scheduling. *Real-Time Systems*, 1(3):244–264, 1989.

[12] A. Srinivasan and J. Anderson. Optimal rate-based scheduling on multiprocessors. In *Proc. of the 34th Annual ACM Symposium on Theory of Computing*, May 2002. *To appear.*

[13] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. Baruah, J. Gehrke, and C. G. 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.

[14] K. W. Tindell, A. Burns, and A. J. Wellings. Mode changes in priority pre-emptively scheduled systems. In *Proc. of the IEEE Real-Time Systems Symposium*, pages 100–109, Dec 1992.