Schedulable Utilization Bounds for EPDF Fair Multiprocessor Scheduling ⋆

UmaMaheswari C. Devi and James H. Anderson

Department of Computer Science,
The University of North Carolina, Chapel Hill, NC 27599, U.S.A.
Email: {uma,anderson}@cs.unc.edu

Abstract. The earliest-pseudo-deadline-first (EPDF) algorithm is less expensive than other known Pfair algorithms, but is not optimal for scheduling recurrent real-time tasks on more than two processors. Prior work established sufficient per-task weight (*i.e.*, utilization) restrictions that ensure that tasks either do not miss their deadlines or have bounded tardiness when scheduled under EPDF. Implicit in these restrictions is the assumption that total system utilization may equal the total available processing capacity (*i.e.*, the total number of processors). This paper considers an orthogonal issue — that of determining a sufficient restriction on the total utilization of a task set for it to be schedulable under EPDF, assuming that there are no per-task weight restrictions. We prove that a task set with total utilization at most $\frac{3M+1}{4}$ is correctly scheduled under EPDF on M processors, regardless of how large each task's weight is. At present, we do not know whether this bound is tight. However, we provide a conterexample that shows that it cannot be improved to exceed 86% of the total processing capacity. Our schedulability test is expressed in terms of the maximum weight of any task, and hence, if this value is known, may be used to schedule task sets with total utilization greater than $\frac{3M+1}{4}$.

⋆ Work supported by NSF grants CCR 9988327, ITR 0082866, CCR 0204312, and CCR 0309825.

1 Introduction

We consider the scheduling of recurrent (*i.e.*, periodic, sporadic, or rate-based) real-time task systems on multiprocessor platforms comprised of M identical, unit-capacity processors. Pfair scheduling, originally introduced by Baruah *et al.* [8], is the only known way of optimally scheduling such multiprocessor task systems. Under Pfair scheduling, each task must execute at an approximately uniform rate, while respecting a fixed-size allocation quantum. A task's execution rate is defined by its *weight* (*i.e.*, *utilization*). Uniform rates are ensured by subdividing each task into quantum-length *subtasks* that are subject to intermediate deadlines, called *pseudo-deadlines*. Subtasks are then scheduled on an earliest-pseudo-deadline-first basis. However, to avoid deadline misses, ties among subtasks with the same deadline must be broken carefully. In fact, tie-breaking rules are of crucial importance when devising optimal Pfair scheduling algorithms.

Srinivasan and Anderson observed that overheads associated with tie-breaking rules may be unnecessary or unacceptable for many *soft* real-time task systems [17]. A soft real-time task differs from a hard real-time task in that its deadlines may *occasionally* be missed. If a job (*i.e.*, task instance) or a subtask with a deadline at time d completes executing at time t, then it is said to have a *tardiness* of $\max(0, t-d)$. Overheads associated with tie-breaking rules motivated Srinivasan and Anderson to consider the viability of scheduling soft real-time task systems using the simpler *earliest-pseudo-deadline-first* (EPDF) Pfair algorithm, which uses no tie-breaking rules. They succeeded in showing that EPDF is *optimal* on up to two processors [2], and that if each task's weight is at most $\frac{q}{q+1}$, then EPDF guarantees a tardiness of at most q quanta for every subtask [17]. In later work [10], we showed that this condition can be improved to $\frac{q+1}{q+2}$. If M denotes the total number of processors, then with either condition, the total utilization of a task set may equal M.

In this paper, we address an orthogonal question: If individual tasks cannot be subject to weight restrictions, then what would be a sufficient restriction on the total utilization of a task set for it to be correctly scheduled under EPDF? We answer this question by providing a sufficient *utilization-based* schedulability test for EPDF. Such a test is specified by establishing a schedulable utilization bound. If $\mathcal{U}(M)$ is a schedulable utilization bound for scheduling algorithm $\mathcal{A}$, then $\mathcal{A}$ can correctly schedule any set of recurrent tasks with total utilization at most $\mathcal{U}(M)$ on M processors [13]. If it is also the case that no schedulable utilization bound for $\mathcal{A}$ can exceed $\mathcal{U}(M)$, then $\mathcal{U}(M)$ is an *optimal* schedulable utilization bound for $\mathcal{A}$.

Schedulability tests can generally be classified as being either *utilization-based* or *demand-based*. Though utilization-based tests are usually less accurate than demand-based tests, they can be evaluated in time that is polynomial in the number of tasks. In dynamic systems in which tasks may leave or join at arbitrary times, constant time is sufficient to determine whether a new task may be allowed to join if a utilization-based test is used. On the other hand, demand-based tests require either exponential time, or, at best, pseudo-polynomial time, and hence, when timeliness is a concern, as in online admission-control tests, utilization-based tests are usually preferred. Therefore, devising utilization-based tests is of considerable value and interest.

Optimal schedulable utilization bounds are known for several scheduling algorithms. In the domain of uniprocessor scheduling, a bound of 1.0 is optimal for preemptive earliest-deadline-first (EDF) scheduling, while one of $N(2^{1/N}-1)$ is optimal for preemptive rate-monotonic (RM) scheduling, where N is the number of tasks [12]. The RM bound converges to $\ln 2 \approx 0.69$ as $N \to \infty$.

Multiprocessor scheduling algorithms use either a *partitioned* or *global* scheduling approach. Under partitioning, tasks are assigned to processors by defining a many-to-one mapping (a surjective function) from the set of tasks to the set of processors. Thus, each task is bound to a single processor, and every instance of that task may execute upon that processor only. A separate instance of a uniprocessor scheduling algorithm is then used to schedule the tasks assigned to a processor. If $W_{\max}$, where $0 < W_{\max} \leq 1$, denotes the maximum weight of any task, then a scheduleable utilization bound of $\frac{\beta M+1}{\beta+1}$, where $\beta = \left\lfloor \frac{1}{W_{\max}} \right\rfloor$ is optimal for the partitioned approach, if EDF is the per-processor scheduling algorithm used [14]. This bound approaches $\frac{M+1}{2}$ as $W_{\max} \to 1.0$. Because EDF is an optimal uniprocessor scheduling algorithm, a higher bound is not possible with any other per-processor scheduling algorithm.

Under global scheduling, a task may execute on any processor. This approach can be further differentiated based upon whether a preempted instance is allowed to resume execution on a different processor. If each job is bound to a single processor only, then migrations are said to be *restricted*; otherwise, they are *unrestricted*. Under global scheduling, among job-level fixed-priority algorithms, such as EDF, a schedulable utilization bound exceeding $\frac{M+1}{2}$ is impossible, regardless of the nature of migrations [6, 7]. Among static-priority scheduling algorithms, such as RM, a schedulable utilization bound exceeding $\frac{M}{2}$ is impossible for the unrestricted-migrations case [4, 5]. Observe that each of the multiprocessor schedulable utilization bounds considered so far converges to 50% of the total processing capacity.

Pfair scheduling algorithms also fall under the global scheduling category. However, as mentioned earlier, optimal scheduling on multiprocessors is possible with Pfair scheduling. Therefore, each of the optimal Pfair algorithms PF [8], PD [9], and PD^2 [16], has an optimal schedulable utilization bound of M.

Contributions. In this paper, we show that $\frac{(k(k-1)M+1)(k+(k-1)W_{\max})-1}{k^2(k-1)(1+W_{\max})}$, where $k = \left\lfloor \frac{1}{W_{\max}} \right\rfloor + 1$, is a schedulable utilization bound for the simpler EPDF Pfair scheduling algorithm on $M > 2$ processors.[1] For $W_{\max} > \frac{1}{2}$, *i.e.*, $k = 2$, this bound reduces to $\frac{(2M+1)(2+W_{\max})-1}{4(1+W_{\max})}$, and as $W_{\max} \to 1.0$, it approaches $\frac{3M+1}{4}$, which approaches $\frac{3M}{4}$, *i.e.*, 75% of the total processing capacity, as $M \to \infty$. Note that this bound is greater than that of every known non-Pfair algorithm by 25%. At present, we do not know if this bound is optimal. However, we provide a counterexample that shows that the bound with $W_{\max} = 1$ cannot exceed 86%. Finally, we extend this bound to allow a tardiness of q quanta.

The rest of the paper is organized as follows. Sec. 2 provides an overview of Pfair scheduling. In Sec. 3, the schedulable utilization bound for EPDF mentioned above is derived. Sec. 4 concludes.

---

[1] EPDF is optimal on up to two processors [3]. Therefore, its optimal schedulable utilization bound on $M \leq 2$ processors is M.

2 Pfair Scheduling

In this section, we summarize relevant Pfair scheduling concepts and state the required definitions and results from [1–3, 8, 16, 17]. Initially, we limit attention to periodic tasks that begin execution at time 0. Such a task T has an integer *period* $T.p$, an integer *execution cost* $T.e$, and a *weight* $wt(T) = T.e/T.p$, where $0 < wt(T) \leq 1$. A task is *light* if its weight is less than $\frac{1}{2}$, 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). The sequence of allocation decisions over time defines a *schedule* S. Formally, $S : \tau \times \mathcal{N} \mapsto \{0, 1\}$, where τ is a task set and $\mathcal{N}$ is the set of nonnegative integers. $S(T, t) = 1$ iff T is scheduled in slot t. On 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. Deviation from the fluid schedule is formally captured by the concept of *lag*. Formally, the *lag of task T at time t* is $lag(T, t) = wt(T) \cdot t - \sum_{u=0}^{t-1} S(T, u)$. (For conciseness, we leave the schedule implicit and use $lag(T, t)$ instead of $lag(T, t, S)$.) 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 i^{th} subtask of task T as T_i, where $i \geq 1$. As in [8], we associate a *pseudo-release* $r(T_i)$ and a *pseudo-deadline* $d(T_i)$ with each subtask T_i, as follows. (For brevity, we often drop the prefix "pseudo-.")

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

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 given by

$$|w(T_i)| = d(T_i) - r(T_i). \tag{3}$$

As an example, consider subtask T_1 in Fig. 1(a). Here, we have $r(T_1) = 0$, $d(T_1) = 2$, and $|w(T_1)| = 2$.

Note that, by (2), $r(T_{i+1})$ is either $d(T_i) - 1$ or $d(T_i)$. Thus, consecutive windows either overlap by one slot, or are disjoint. The "b-bit," denoted by $b(T_i)$, distinguishes between these possibilities. Formally,

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

For example, in Fig. 1(a), $b(T_i) = 1$ for $1 \leq i \leq 7$ and $b(T_8) = 0$. We often overload function S (described earlier) and use it to denote the allocation status of *subtasks*. Thus, $S(T_i, t) = 1$ iff subtask T_i is scheduled in slot t.

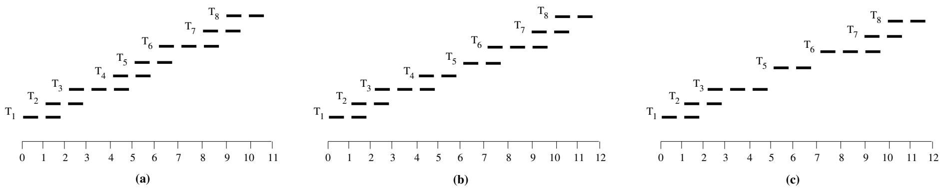

Fig. 1. **(a)** Windows of the first job of a periodic task T with weight $8/11$. This job consists of subtasks $T_1, \ldots, T_8$, each of which must be scheduled within its window, or else a lag-bound violation will result. (This pattern repeats for every job.) **(b)** The Pfair windows of an IS task. Subtask T_5 becomes eligible one time unit late. **(c)** The Pfair windows of a GIS task. Subtask T_4 is absent and subtask T_6 becomes eligible one time unit late.

Algorithm EPDF. Most Pfair scheduling algorithms schedule tasks by choosing subtasks to schedule at the beginning of every quantum. As its name suggests, the earliest-pseudo-deadline-first (EPDF) Pfair algorithm gives higher priority to subtasks with earlier deadlines. A tie between subtasks with equal deadlines is broken arbitrarily. As mentioned earlier, EPDF is optimal on at most two processors, but not on an arbitrary number of processors [3].

Task models. In this paper, we consider the *intra-sporadic* (IS) task model and the *generalized-intra-sporadic* (GIS) task model [2, 16], which provide a general notion of recurrent execution that subsumes that found in the well-studied periodic and sporadic task models. The *sporadic* model generalizes the periodic model by allowing jobs to be released "late"; the IS model generalizes the sporadic model by allowing 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 shifted right from where they would appear if the task were periodic.

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

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

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,

$$k > i \Rightarrow \theta(T_k) \geq \theta(T_i). \tag{6}$$

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. In particular, a subtask can become eligible before its "release" time. It is required that

$$(\forall i \geq 1 :: e(T_i) \leq r(T_i) \ \wedge \ e(T_i) \leq e(T_{i+1})). \tag{7}$$

Intervals $[r(T_i), d(T_i))$ and $[e(T_i), d(T_i))$ are called the *PF-window* and *IS-window* of T_i, respectively. A schedule for an IS system is *valid* iff each subtask is scheduled in its IS-window. (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.)

b-bits for IS tasks are defined in the same way as for periodic tasks (refer to (4)). $r(T_i)$ is defined as follows.

$$r(T_i) = \begin{cases} e(T_i), & \text{if } i = 1 \\ \max(e(T_i), d(T_{i-1}) - b(T_{i-1})), & \text{if } i \geq 2 \end{cases} \tag{8}$$

Thus, if T_i is eligible *during* T_{i-1}'s PF-window, then $r(T_i) = d(T_{i-1}) - b(T_{i-1})$, and hence, the spacing between $r(T_{i-1})$ and $r(T_i)$ is exactly as in a periodic task system. On the other hand, if T_i becomes eligible *after* T_{i-1}'s PF-window, then T_i's PF-window begins when T_i becomes eligible. Note that (8) implies that consecutive PF-windows of the same task are either disjoint, or overlap by one slot, as in a periodic system.

T_i's deadline $d(T_i)$ is defined to be $r(T_i) + |w(T_i)|$. PF-window lengths are given by (3), as in periodic systems. Thus, by (5), we have $|w(T_i)| = \left\lceil \frac{i}{wt(T)} \right\rceil - \left\lfloor \frac{i-1}{wt(T)} \right\rfloor$ and $d(T_i) = r(T_i) + \left\lceil \frac{i}{wt(T)} \right\rceil - \left\lfloor \frac{i-1}{wt(T)} \right\rfloor$.

Generalized intra-sporadic task systems. A *generalized* intra-sporadic task system is obtained by removing subtasks from a corresponding IS task system. Specifically, in a GIS task system, a task T, after releasing subtask T_i, may release subtask T_k, where $k > i + 1$, instead of T_{i+1}, with the following restriction: $r(T_k) - r(T_i)$ is at least $\left\lfloor \frac{k-1}{wt(T)} \right\rfloor - \left\lfloor \frac{i-1}{wt(T)} \right\rfloor$. In other words, $r(T_k)$ is not smaller than what it would have been if $T_{i+1}, T_{i+2}, \ldots, T_{k-1}$ were present and released as early as possible. For the special case where T_k is the first subtask released by T, $r(T_k)$ must be at least $\left\lfloor \frac{k-1}{wt(T)} \right\rfloor$. Fig. 1(c) shows an example. If T_i is the most recently released subtask of T, then T may release T_k, where $k > i$, as its next subtask at time t, if $r(T_i) + \left\lfloor \frac{k-1}{wt(T)} \right\rfloor - \left\lfloor \frac{i-1}{wt(T)} \right\rfloor \leq t$. 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.

As shown in [2], a valid schedule exists for a GIS task set τ on M processors iff $\sum_{T \in \tau} wt(T) \leq M$.

Shares and lags in IS and GIS task systems. The lag of T at time t is defined in the same way as for periodic tasks [16]. Let $ideal(T, t)$ denote the processor share that T receives in an ideal fluid (processor-sharing) schedule in $[0, t)$. Then,

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

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.

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

Using (10), it is not difficult to see that

$$(\forall i > 0, u \geq 0 :: f(T_i, u) \leq wt(T)). \tag{11}$$

Given f, $share(T, u)$ can be defined in terms of f as

$$share(T, u) = \sum_i f(T_i, u). \tag{12}$$

$share(T, u)$ usually equals $wt(T)$, but in certain slots, it may be less than $wt(T)$, so that the total allocation that a subtask T_i receives in the slots that span its window is exactly one in the ideal system. These and similar properties have been formally proved in [15]. Later in this paper, we will use (13) and (14) given below.

$$(\forall u \geq 0 :: share(T, u) \leq wt(T)) \tag{13}$$

$$(\forall T_i :: \sum_{u=r(T_i)}^{d(T_i)-1} f(T_i, u) = 1) \tag{14}$$

Having defined $share(T, u)$, $ideal(T, t)$ can then be defined as $\sum_{u=0}^{t-1} share(T, u)$. Hence, from (9),

$$\begin{aligned} lag(T, t+1) &= \sum_{u=0}^{t}(share(T, u) - S(T, u)) \\ &= lag(T, t) + share(T, t) - S(T, t). \end{aligned} \tag{15}$$

The total lag for a task system τ with respect to a schedule $\mathcal{S}$ at time t, denoted $LAG(\tau, t)$ is then given by

$$LAG(\tau, t) = \sum_{T \in \tau} lag(T, t). \tag{16}$$

From (15) and (16), $LAG(\tau, t+1)$ can be expressed as follows. ($LAG(\tau, 0)$ is defined to be 0.)

$$LAG(\tau, t+1) = LAG(\tau, t) + \sum_{T \in \tau}(share(T, t) - S(T, t)). \tag{17}$$

The rest of this section presents some additional definitions and results that will be used in the rest of this paper.

Active tasks. It is possible for a GIS (or IS) task to have no eligible subtasks and a share of zero during certain time slots, if subtasks are absent or are released late. Tasks with and without subtasks at time t are distinguished using the following definition of an *active* task. (A task that is active at t is not necessarily scheduled at t.)

Definition 1: A GIS task U is *active* at time t if it has a subtask U_j such that $e(U_j) \leq t < d(U_j)$.

Task classification[16]**.** Tasks in τ may be classified as follows with respect to a schedule $\mathcal{S}$ and time t. [2]

$A(t)$**:** Set of all tasks that are scheduled at t.
$B(t)$**:** Set of all tasks that are not scheduled at t, but are active at t.
$I(t)$**:** Set of all tasks that are neither active nor are scheduled at t.

$A(t)$, $B(t)$, and $I(t)$ form a partition of τ, *i.e.*,

$$\begin{aligned} &A(t) \cup B(t) \cup I(t) = \tau, \\ &A(t) \cap B(t) = B(t) \cap I(t) = I(t) \cap A(t) = \emptyset. \end{aligned} \tag{18}$$

Using (16) and (18) above, we have the following.

$$LAG(\tau, t+1) = \sum_{T \in A(t)} lag(T, t+1) + \sum_{T \in B(t)} lag(T, t+1) + \sum_{T \in I(t)} lag(T, t+1) \tag{19}$$

The next definition identifies the last-released subtask at t of any task U.

Definition 2: Subtask U_j is the *critical subtask of* U *at* t iff $e(U_j) \le t < d(U_j)$ holds, and no other subtask U_k of U, where $k > j$, satisfies $e(U_k) \le t < d(U_k)$.

Holes. If fewer than M tasks are scheduled at time t in $\mathcal{S}$, then one or more processors would be idle at t. If k processors are idle during t, then we say that there are k *holes* in S at t. The following lemma, proved in [16], relates an increase in the total lag of τ, LAG, to the presence of holes.

Lemma 1. [16] *If* $LAG(\tau, t+1) > LAG(\tau, t)$*, then there are one or more holes in* t*.*

Intuitively, if there are no idle processors in slot t, then the total allocation to τ in $\mathcal{S}$ is at least the total allocation to τ in the ideal system in slot t. Therefore, LAG cannot increase.

Displacements. In our proof, we consider task systems obtained by removing subtasks. If $\mathcal{S}$ is a schedule for a GIS task system τ, then removing a subtask from τ results in another GIS system τ', and may cause other subtasks to shift earlier in $\mathcal{S}$, resulting in a schedule $\mathcal{S}'$ that is valid for τ'. Such a shift is called a *displacement* and is denoted by a 4-tuple $\langle X^{(1)}, t_1, X^{(2)}, t_2 \rangle$, where $X^{(1)}$ and $X^{(2)}$ represent subtasks. This is equivalent to saying that subtask $X^{(2)}$ originally scheduled at t_2 in $\mathcal{S}$ displaces subtask $X^{(1)}$ scheduled at t_1 in $\mathcal{S}$. A displacement $\langle X^{(1)}, t_1, X^{(2)}, t_2 \rangle$ is *valid* iff $e(X^{(2)}) \le t_1$. Because there can be a cascade of shifts, we may have a chain of displacements. This chain is represented by a sequence of 4-tuples.

The next lemma concerns displacements and is proved in [16]. It states that a subtask removal can only cause left shifts.

Lemma 2. [16] *Let* $X^{(1)}$ *be a subtask that is removed from* τ*, and let the resulting chain of displacements in an EPDF schedule for* τ *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* $i \in [1, k]$*.*

---

[2] For brevity, we let the task system τ and schedule $\mathcal{S}$ be implicit in these definitions.

3 Sufficient Schedulability Test for EPDF

In this section, we establish a sufficient schedulability test for EPDF by deriving a schedulable utilization bound for it, given by the following theorem.

Theorem 1. $\frac{(k(k-1)M+1)((k-1)W_{\max}+k)-1}{k^2(k-1)(1+W_{\max})}$, *where* $W_{\max}$ *is the maximum weight of any task in* τ *and* $k = \left\lfloor \frac{1}{W_{\max}} \right\rfloor + 1$, *is a schedulable utilization bound of EPDF for scheduling a GIS task system* τ *on* $M > 2$ *processors.*

As a shorthand, we define $U(M, W_{\max})$ as follows.
Definition 3: $U(M, W_{\max}) \stackrel{\text{def}}{=} \frac{(k(k-1)M+1)((k-1)W_{\max}+k)-1}{k^2(k-1)(1+W_{\max})}$, where $k = \left\lfloor \frac{1}{W_{\max}} \right\rfloor + 1$.
For simplicity, we prove the theorem for $k = 2$, *i.e.*, when the following holds.

$$\frac{1}{2} < W_{\max} \leq 1 \tag{20}$$

Later, we show how to extend the proof for $k > 2$. When $k = 2$, $U(M, W_{\max})$ reduces to $\frac{(2M+1)(2+W_{\max})-1}{4(1+W_{\max})}$.

We use the proof technique developed by Srinivasan and Anderson in [17] to prove the above theorem. If Theorem 1 does not hold, then t_d and τ defined as follows exist. (In these definitions, we assume that τ is scheduled on M processors.)

Definition 4: t_d is the earliest time that any task system (with each task weight at most $W_{\max}$ and total utilization at most $U(M, W_{\max})$) has a deadline miss under EPDF, *i.e.*, some such task system misses a subtask deadline at t_d, and no such system misses a subtask deadline prior to t_d.
Definition 5: τ is a task system with the following properties.

(T1) t_d is the earliest time that a subtask in τ misses its deadline under $\mathcal{S}$, an EPDF schedule for τ.
(T2) The weight of every task in τ is at most $W_{\max}$ and the total utilization of τ is at most $U(M, W_{\max})$.
(T3) No other task system satisfying (T1) and (T2) releases fewer subtasks in $[0, t_d)$ than τ.
(T4) No other task system satisfying (T1), (T2), and (T3) has a larger rank than τ at t_d, where the *rank* of a system τ at t is the sum of the eligibility times of all subtasks with deadlines at most t, *i.e.*, $rank(\tau, t) = \sum_{\{T_i : T \in \tau \, \wedge \, d(T_i) \leq t\}} e(T_i)$.

By (T1) and (T3), exactly one subtask in τ misses its deadline: if several such subtasks exist, then all but one can be removed and the remaining subtask will still miss its deadline, contradicting (T3).

In what follows, we use the shorthand notation given by Defs. 6 and 7 below.

Definition 6: α denotes the total utilization of τ, expressed as a fraction of M, *i.e.*, $\sum_{T \in \tau} wt(T) = \alpha M$.

Definition 7: $\delta \stackrel{\text{def}}{=} \frac{W_{\max}}{1+W_{\max}}$.

The lemma below follows from the definitions of U and α, (T2), and Lemma 12.

Lemma 3. $0 \leq \alpha \leq \frac{U(M,W_{\max})}{M} < 1$.

The next lemma is immediate from the definition of δ and (20).

Lemma 4. $\frac{1}{3} < \delta \leq \frac{1}{2}$.

We now prove some properties about τ and $\mathcal{S}$. In proving some of these properties, we make use of the following three lemmas established in prior work by Srinivasan and Anderson.

Lemma 5. [16] *If* $LAG(\tau, t+1) > LAG(\tau, t)$*, then* $B(t) \neq \emptyset$.

The following is an intuitive explanation for why Lemma 5 holds. Recall from Sec. 2 that $B(t)$ is the set of all tasks that are active and not scheduled at t. By Def. 1, (10), and (7), only tasks that are active at t may have non-zero shares at t in the ideal fluid schedule. Therefore, if every task that is active at t is scheduled at t, then the total allocation in $\mathcal{S}$ cannot be less than the total allocation in the ideal schedule, and hence, by (17), LAG cannot increase across slot t.

Lemma 6. [15] *Let* $t < t_d$ *be a slot with holes and let* $T \in B(t)$*. Then, the critical subtask at* t *of* T *is scheduled before* t.

To see that the above lemma holds, let T_i be the critical subtask of T at t. By its definition, the IS-window of T_i overlaps slot t, but T is not scheduled at t. Also, there is at least a hole in t. Because EPDF does not idle a processor while there is a task with an outstanding execution request, it should be the case that T_i is scheduled before t.

Lemma 7. [16] *Let* U_j *be a subtask that is scheduled in slot* t'*, where* $t' \leq t < t_d$*, in* $\mathcal{S}$*, where there is a hole in* t*. Then,* $d(U_j) \leq t+1$.

This lemma is true because it can be shown that if $d(U_j) > t+1$ holds, then U_j has no impact on the deadline miss at t_d. In other words, it can be shown that if the lemma does not hold, then the task system obtained from τ by removing U_j also has a deadline miss at t_d, which contradicts (T3).

Lemma 8. *The following properties hold for* τ *and* $\mathcal{S}$.

(**a**) *For all* T_i, $d(T_i) \leq t_d$.
(**b**) *Exactly one subtask of* τ *misses its deadline at* t_d.
(**c**) $LAG(\tau, t_d) = 1$.
(**d**) $(\forall T_i :: d(T_i) < t_d \;\Rightarrow\; (\exists t :: e(T_i) \leq t < d(T_i) \;\wedge\; S(T_i, t) = 1))$.
(**e**) *Let* U_k *be the subtask that misses its deadline at* t_d*. Then,* U *is not scheduled at* $t_d - 1$.
(**f**) *There are no holes in slot* $t_d - 1$.
(**g**) *There exists a time* $v \leq t_d - 2$ *such that the following both hold.*
  (**i**) *There are no holes in* $[v, t_d - 2)$.
  (**ii**) $LAG(\tau, v) \geq (t_d - v)(1 - \alpha)M + 1$.
(**h**) *There exists a time* $u \in [0, t_d - 3]$ *such that* $LAG(\tau, u) < 1$ *and* $LAG(\tau, u+1) \geq 1$.

Parts (a), (b), and (c) are proved in [16]. Part (d) follows directly from (T1). The remaining are proved in [11].

Overview of the rest of the proof of Theorem 1. By Lemma 8(h), if t_d and τ as defined by Defs. 4 and 5, respectively, exist, then there exists a time slot $u < t_d - 2$ across which LAG increases to at least one. To prove Theorem 1, we show that for every such u, either **(i)** there exists a time u', where $u+1 < u' \leq t_d$, such that $LAG(\tau, u') < 1$, and thereby derive a contradiction to Lemma 8(c), or **(ii)** there does not exist a $v \leq t_d - 2$ such that there are no holes in $[v, t_d-2)$ and $LAG(\tau, v) > (t_d - v)(1-\alpha)M$, deriving a contradiction to Lemma 8(g). In what follows, we state and prove several other lemmas that are required to accomplish this.

The first lemma shows that LAG does not increase across slot zero.

Lemma 9. $LAG(\tau, 1) \leq LAG(\tau, 0) = 0$.

Proof. Assume to the contrary that $LAG(\tau, 1) > LAG(\tau, 0)$. Then, by Lemma 5, $B(0) \neq \emptyset$ holds. Let T be a task in $B(0)$ and let T_i be its critical subtask at time zero. Then, by Lemma 6, T_i is scheduled before time zero, which is impossible. Therefore, our assumption that $LAG(\tau, 1) > LAG(\tau, 0)$ holds is incorrect. □

By Lemma 1, one or more holes in slot t are necessary for LAG to increase across t. The next lemma shows that there are no holes in slot $t-1$ in this case.

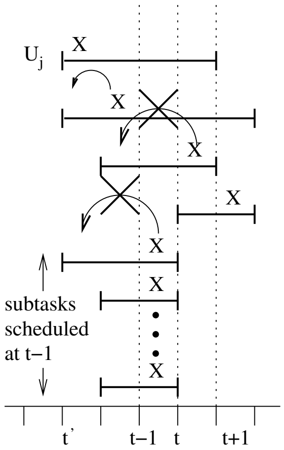

Fig. 2. Illustration for Lemma 10. Subtask U_j is as specified in the proof. Directed arcs depict some displacements that may be possible if U_j is removed. If there is a hole in $t-1$, then the crossed out displacements are not possible, and hence, the displacements chain cannot extend beyond slot $t-1$.

Lemma 10. *If* $LAG(\tau, t+1) > LAG(\tau, t)$*, where* $1 \leq t < t_d - 2$*, then there are no holes in slot* $t-1$.

Proof. Contrary to the statement of the lemma, assume the following.

(B) There is a hole in slot $t-1$.

By Lemma 7, this assumption implies that the deadline of every subtask T_i scheduled at $t-1$ is at most t. Because $t < t_d$ holds, by Lemma 8(d), T_i does not miss its deadline. Therefore, we have the following.

$$(\forall T_i :: S(T_i, t-1) = 1 \ \Rightarrow \ d(T_i) = t) \tag{21}$$

Refer to Fig. 2 for an illustration.

Because $LAG(\tau, t+1) > LAG(\tau, t)$ holds (by the statement of the Lemma), by Lemma 5, $B(t)$ is not empty. Let U be any task in $B(t)$ and let U_j be its critical subtask at t. Then, by Lemma 6, U_j is scheduled before t in $\mathcal{S}$, say at t', *i.e.*,

$$S(U_j, t') = 1 \ \wedge \ t' < t. \tag{22}$$

Also, by Def. 2,

$$d(U_j) \geq t+1. \tag{23}$$

Let τ' be the task system obtained by removing U_j from τ, and let $\mathcal{S}'$ be the schedule that results due to the left shifts caused in $\mathcal{S}$ by U_j's removal. We show that the left shifts do not extend beyond slot $t-1$, which would imply that a deadline is still missed at t_d in $\mathcal{S}'$. This in turn would imply that τ', with one subtask fewer than τ, also has a deadline miss at t_d, contradicting (T3).

Let $\Delta_1, \Delta_2, \ldots, \Delta_n$ be the chain of displacements in $\mathcal{S}$ caused by removing U_j, where $\Delta_i = \langle X^{(i)}, t_i, X^{(i+1)}, t_{i+1} \rangle$, $1 \le i \le n$, and $X^{(1)} = U_j$. By Lemma 2, $t_i < t_{i+1}$ holds for all $1 \le i \le n$. By (22), $t_1 = t' < t$ holds, as illustrated in Fig. 2. Because EPDF scheduled $X^{(i)}$ at t_i in preference to $X^{(k)}$ in $\mathcal{S}$, where $i < k \le n+1$, the priority of $X^{(i)}$ is at least as high as that of $X^{(k)}$. In other words, we have $d(X^{(i)}) \le d(X^{(k)})$, for all $1 \le i < k \le n+1$. Because $X^{(1)} = U_j$, by (23), this implies the following.

$$(\forall k : 1 \le k \le n+1 :: d(X^{(k)}) \ge t+1) \tag{24}$$

We next show that the chain of displacements does not extend beyond slot $t-1$. Suppose that the displacements extend beyond slot $t-1$. Let Δ_h, $1 \le h \le n$ be the displacement with the smallest index such that $t_h \le t-1$ and $t_{h+1} \ge t$ holds. Because Δ_h is valid, $e(X^{(h+1)}) \le t_h$ holds. Now, if $t_h < t-1$, then since there is a hole in slot $t-1$, $X^{(h+1)}$ should have been scheduled at $t-1$ in $\mathcal{S}$ and not at $t_{h+1} \ge t$. Therefore, $t_h = t-1$, *i.e.*, $X^{(h)}$ is scheduled at $t-1$ in $\mathcal{S}$. Hence, by (21), we have $d(X^{(h)}) = t$, which contradicts (24). This is illustrated in Fig. 2. Thus, the displacements do not extend beyond $t-1$, which implies that a deadline is still missed at t_d, contradicting (T3). Therefore, our assumption in (B) is false. $\square$

The next lemma, proved in [11], bounds the lag of each task at time $t+1$, where t is a slot with one or more holes.

Lemma 11. [11] *If $t < t_d$ is a slot with one or more holes, then the following hold.*

(**a**) $(\forall T \in A(t) :: lag(T, t+1) < wt(T))$
(**b**) $(\forall T \in B(t) :: lag(T, t+1) \le 0)$
(**c**) $(\forall T \in I(t) :: lag(T, t+1) = 0)$

In proving the remaining lemmas, we make use of the following three lemmas. Their proofs are omitted here due to space constraints, and can be found in the full version of this paper [11].

Lemma 12. [11] $\frac{(k(k-1)M+1)((k-1)W_{\max}+k)-1}{Mk^2(k-1)(1+W_{\max})} < 1$, *where* $k = \left\lfloor \frac{1}{W_{\max}} \right\rfloor + 1$, *for all* $0 < W_{\max} \le 1$ *and* $M > 1$.

Lemma 13. [11] *A solution to the recurrence*

$$L_0 < \delta + \delta \cdot \alpha M$$
$$L_k \le \delta \cdot L_{k-1} + \delta \cdot (2\alpha M - M),$$

where $0 \le \delta < 1$, *is given by*

$$L_k < \delta^{k+1}(1+\alpha M) + (1-\delta^k)\left(\frac{\delta}{1-\delta}\right)(2\alpha M - M), \quad \textit{for all } k \ge 0.$$

Lemma 14. [11] $\frac{M(2-\delta-\delta^{n+1})+\delta^{n+2}-\delta^{n+1}+1-\delta}{M(2-\delta^{n+1}-\delta^{n+2})} \geq \frac{M(2-\delta)+\frac{1}{2}}{2M}$ *holds for all* $n \geq 0, 0 \leq \delta \leq 1/2$, *and* $M \geq 1$.

Having shown that a hole in slot $t-1$ is necessary for LAG to increase across t (Lemma 10), we next show how to derive an upper bound on LAG at $t+1$ in terms of LAG at t and $t-1$.

Lemma 15. *Let* t*, where* $1 \leq t < t_d - 2$*, be a slot such that there is at least a hole in slot* t *and there is no hole in slot* $t-1$*. Then* $LAG(\tau, t+1) \leq LAG(\tau, t) \cdot \delta + \alpha M \cdot \delta$ *and* $LAG(\tau, t+1) \leq LAG(\tau, t-1) \cdot \delta + (2\alpha M - M) \cdot \delta$.

Proof. By the statement of the lemma, there is at least one hole in slot t. Therefore, by Lemma 11, only tasks that are scheduled in slot t, *i.e.*, tasks in set $A(t)$, may have a positive lag at $t+1$. Let x denote the number of tasks scheduled at t, *i.e.*, $x = \sum_{T \in \tau} S(T,t) = |A(t)|$. Then, by (19), we have

$$\begin{aligned}
LAG(\tau, t+1) &\leq \sum_{T \in A(t)} lag(T, t+1) \\
&< \sum_{T \in A(t)} wt(T) && \text{, by Lemma 11} \\
&\leq \sum_{T \in A(t)} W_{\max} \\
&= x \cdot W_{\max}. && , |A(t)| = x \qquad (25)
\end{aligned}$$

Using (17), $LAG(\tau, t+1)$ can be expressed as follows.

$$\begin{aligned}
LAG(\tau, t+1) &= LAG(\tau, t) + \sum_{T \in \tau} (share(T,t) - S(T,t)) \\
&= LAG(\tau, t) + \sum_{T \in \tau} share(T,t) - x && , \textstyle\sum_{T \in \tau} S(T,t) = x \\
&\leq LAG(\tau, t) + \sum_{T \in \tau} wt(T) - x && \text{, by (13)} \\
&= LAG(\tau, t) + \alpha M - x && \text{, by Def. 6} \qquad (26)
\end{aligned}$$

By (25) and (26), we have

$$LAG(\tau, t+1) \leq \min(x \cdot W_{\max}, LAG(\tau, t) + \alpha M - x). \qquad (27)$$

Because $x \cdot W_{\max}$ increases with increasing x, whereas $LAG(\tau, t) + \alpha M - x$ decreases, $LAG(\tau, t+1)$ is maximized when $x \cdot W_{\max} = LAG(\tau, t) + \alpha M - x$, *i.e.*, when $x = \frac{LAG(\tau,t)}{1+W_{\max}} + \frac{\alpha M}{1+W_{\max}}$. Therefore, using either (25) or (26), we have

$$\begin{aligned}
LAG(\tau, t+1) &\leq LAG(\tau, t) \cdot \left(\frac{W_{\max}}{1+W_{\max}}\right) + \alpha M \cdot \left(\frac{W_{\max}}{1+W_{\max}}\right) \\
&= LAG(\tau, t) \cdot \delta + \alpha M \cdot \delta && \text{, by Def. 7.} \qquad (28)
\end{aligned}$$

By the statement of the lemma again, there is no hole in slot $t-1$. (Also, $t \geq 1$, and hence, $t-1$ exists.) Therefore, using (17), $LAG(\tau, t)$ can be expressed as follows.

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

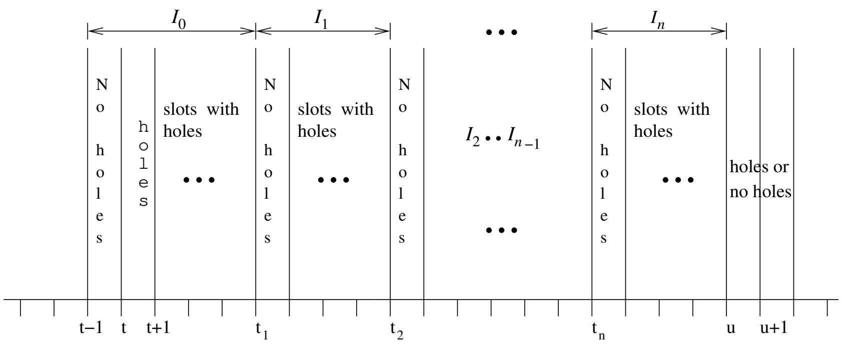

Fig. 3. Lemma 16. $LAG(\tau, t) < 1$ and $LAG(\tau, t+1) \geq 1$. No two consecutive slots in $[t-1, u)$ are without holes. The objective is to determine a bound on $LAG(\tau, u)$.

$$= LAG(\tau, t-1) + \sum_{T \in \tau} share(T, t-1) - M$$

$$, \textstyle\sum_{T \in \tau} S(T, t-1) = M \text{ (there are no holes in } t-1\text{)}$$

$$\leq LAG(\tau, t-1) + \alpha M - M \qquad \text{, by (13) and Def. 6} \tag{29}$$

Substituting (29) in (28), we have $LAG(\tau, t+1) \leq LAG(\tau, t-1) \cdot \delta + (2\alpha M - M) \cdot \delta$. □

The next lemma shows how to bound LAG at the end of an interval that does not contain two consecutive slots without holes.

Lemma 16. *Let $1 \leq t < t_d - 2$ be a slot across which LAG increases to one, i.e.,*

$$1 \leq t < t_d - 2 \ \wedge \ LAG(\tau, t) < 1 \ \wedge \ LAG(\tau, t+1) \geq 1. \tag{30}$$

Let u, where $t < u < t_d$, be such that there is at least one hole in $u - 1$ and there are no two consecutive slots without holes in the interval $[t+1, u)$. Then, $LAG(\tau, u) < (2 - 2\alpha)M + 1$.

Proof. Because (30) holds, by Lemmas 1 and 10, we have (C1) and (C2), respectively.

(C1) There is at least one hole in slot t.
(C2) There is no hole in slot $t - 1$.

By (C1) and the definition of u, we have the following.

$$(\forall t' : t \leq t' \leq u - 1 :: \text{ there is a hole in } t' \text{ or } t' + 1) \tag{31}$$

Let $t_1, t_2, \ldots, t_n$, where $t < t_1 < t_2 < \ldots < t_n < u - 1$ be the slots without holes in $[t, u)$. Then, by (C1) and (31), there is at least one hole in each of $t_i - 1$ and $t_i + 1$ for all $1 \leq i \leq n$.

We divide the interval $[t - 1, u)$ into $n + 1$ non-overlapping subintervals using the slots without holes $t - 1, t_1, \ldots, t_n$, as shown in Fig. 3. The subintervals denoted $I_0, I_1, \ldots, I_n$ are defined as follows.

$$I_0 \stackrel{\text{def}}{=} \begin{cases} [t-1, t_1), & \text{if } t_1 \text{ exists} \\ [t-1, u), & \text{otherwise} \end{cases} \tag{32}$$

$$I_n \stackrel{\text{def}}{=} [t_n, u) \tag{33}$$

$$I_k \stackrel{\text{def}}{=} [t_k, t_{k+1}), \quad \text{for all } 1 \le k < n \tag{34}$$

Because $t > 1$ and $u < t_d$ hold (by the statement of the Lemma), I_0 exists, and hence,

$$n \ge 0. \tag{35}$$

Before proceeding further, the following notation is in order. We denote the start and end times of I_k, where $0 \le k \le n$, by t_s^k and t_f^k, respectively, *i.e.*, I_k is denoted as follows.

$$I_k \stackrel{\text{def}}{=} [t_s^k, t_f^k), \qquad \text{for all } k = 0, 1, \ldots, n. \tag{36}$$

LAG at $t_s^k + 2$ is denoted L_k, *i.e.*,

$$L_k \stackrel{\text{def}}{=} LAG(\tau, t_s^k + 2), \qquad \text{for all } k = 0, 1, \ldots, n. \tag{37}$$

Note that the end of each subinterval is defined so that the following property holds.

(C3) For all k, $0 \le k \le n$, there is no hole in slot t_s^k and there is at least one hole in every slot $\hat{t}$, where $t_s^k + 1 \le \hat{t} < t_f^k$.

Our goal now is to derive bounds for LAG at t_f^k, for all $0 \le k \le n$. Towards this end, we first establish the following claim.

Claim 1. $(\forall k, t' : 0 \le k \le n, t_s^k + 2 \le t' \le t_f^k :: LAG(\tau, t') \le L_k)$.
The proof is by induction on t'.
Base Case: $t' = t_s^k + 2$. The claim holds by (37).
Induction Step: Assuming that the claim holds at all times in the interval $[t_s^k + 2, t']$, where $t_s^k + 2 \le t' < t_f^k$, we show that it holds at $t' + 1$. By this induction hypothesis, we have

$$LAG(\tau, t') \le L_k. \tag{38}$$

Because $t' < t_f^k$ and $t' \ge t_s^k + 2$ hold (by the induction hypothesis), by (C3), there is at least one hole in both t' and $t' - 1$. Therefore, by the contrapositive of Lemma 10, $LAG(\tau, t' + 1) \le LAG(\tau, t')$, which by (38), is at most L_k. $\square$

Having shown that $LAG(\tau, t_f^k)$ is at most L_k, we now bound L_k. We start by determining a bound for L_0. From (32) and (36), we have $t_s^0 = t - 1$. Therefore, $t_s^0 + 2 = t + 1$. Because (C1) and (C2) hold, by Lemma 15,

$$\begin{aligned} L_0 &= LAG(\tau, t + 1) \\ &\le \delta \cdot LAG(\tau, t) + \delta \cdot \alpha M \\ &< \delta + \delta \cdot \alpha M, \qquad , LAG(\tau, t) < 1 \text{ by (30).} \end{aligned} \tag{39}$$

We next determine an upper bound for L_k, where $1 \le k \le n$. Notice that by our definition of I_k in (34), we have $t_s^k = t_f^{k-1}$. Thus, $LAG(\tau, t_s^k) = LAG(\tau, t_f^{k-1})$, and hence, by Claim 1, we have

$$LAG(\tau, t_s^k) \le L_{k-1}. \tag{40}$$

By (C3), there is a hole in slot $t_s^k + 1$ and no hole in slot t_s^k. Therefore, by Lemma 15, $LAG(\tau, t_s^k + 2) \le \delta \cdot LAG(\tau, t_s^k) + \delta \cdot (2\alpha M - M)$, which by (37) and (40) implies

$$L_k = LAG(\tau, t_s^k + 2) \le \delta \cdot L_{k-1} + \delta \cdot (2\alpha M - M). \quad (41)$$

By (33) and (36), we have $u = t_f^n$. Therefore, by Claim 1 and (35), we have

$$LAG(\tau, u) = LAG(\tau, t_f^n) \le L_n, \quad (42)$$

and hence, an upper bound on $LAG(\tau, u)$ can be determined by solving the recurrence given by (39) and (41), which is restated below for convenience.

$$L_0 < \delta + \delta \cdot \alpha M$$
$$L_k \le \delta \cdot L_{k-1} + \delta \cdot (2\alpha M - M)$$

By Lemma 13, a solution to the above recurrence is given by

$$L_k < \delta^{k+1}(1 + \alpha M) + (1 - \delta^k)\left(\frac{\delta}{1-\delta}\right)(2\alpha M - M). \quad (43)$$

Therefore, $LAG(\tau, u) \le L_n < \delta^{n+1}(1 + \alpha M) + (1 - \delta^n)\left(\frac{\delta}{1-\delta}\right)(2\alpha M - M)$.

If L_n is at least $(2-2\alpha)M+1$, then $\delta^{n+1}(1+\alpha M)+(1-\delta^n)\left(\frac{\delta}{1-\delta}\right)(2\alpha M - M) > (2 - 2\alpha)M + 1$, which on rearranging terms implies that

$$\begin{aligned}
\alpha &> \frac{M(2 - \delta - \delta^{n+1}) + \delta^{n+2} - \delta^{n+1} + 1 - \delta}{M(2 - \delta^{n+1} - \delta^{n+2})} \\
&\ge \frac{M(2-\delta) + \frac{1}{2}}{2M} && \text{, by Lemma 14, and } 0 \le \delta \le 1/2 \text{ (by Lemma 4)} \\
&= \frac{(2M+1)(2 + W_{\max}) - 1}{4M(1 + W_{\max})} && \text{, by Def. 7} \\
&= \frac{U(M, W_{\max})}{M} && \text{, by Def. 3 and (20).} \quad (44)
\end{aligned}$$

Because (44) is in contradiction to Lemma 3, we conclude that $L_n < (2 - 2\alpha)M + 1$. Hence, by (42), $LAG(\tau, u) \le L_n < (2 - 2\alpha)M + 1$. □

Lemma 17. *Let $t < t_d - 2$ be a slot such that $LAG(\tau, t) < 1 \wedge LAG(\tau, t+1) \ge 1$ and let u be the earliest time after t such that $u = t_d - 2$ or there no no holes in each of u and $u + 1$. (Note that this implies that no two consecutive slots in $[t + 1, u)$ are without holes.) Then, at least one of the following holds.*

$u \le t_d - 2$, *there are no holes in both u and $u + 1$, and $LAG(\tau, u + 2) < 1$.* (45)

$u = t_d - 2$, *there is at least a hole in $t_d - 3$, and $LAG(\tau, t_d - 2) < 2(1 - \alpha)M$.* (46)

$u = t_d - 2$, *there is no hole in $t_d - 3$, at least a hole in $t_d - 4$,*
$LAG(\tau, t_d - 3) < 3(1 - \alpha)M$, and $LAG(\tau, t_d - 2) < 2(1 - \alpha)M$. (47)

Proof. Because $LAG(\tau, t+1) > LAG(\tau, t)$ holds (by the statement of the lemma), by Lemma 1, we have the following.

(D1) There is at least a hole in t.

We consider two cases depending on u.
Case 1: $u \leq t_d - 2$ **and there is no hole in** u**.**
We first prove that there is at least one hole in slot $u - 1$. If $u = t + 1$ holds, then by (D1), there is a hole in $t = u - 1$; if $u \neq t + 1$, then the absence of holes in $u - 1$ would contradict the fact that u is the earliest time after t such that either there is no hole in both u and $u + 1$ or $u = t_d - 2$. Thus, there is at least one hole in $u - 1$, and by the definition of u, no two consecutive slots in the interval $[t + 1, u)$ are without holes. Therefore, by Lemma 16, we have $LAG(\tau, u) < (2 - 2\alpha)M + 1$.

To show that $LAG(\tau, u + 2) < 1$ holds, we next show that there are no holes in $u + 1$. If $u < t_d - 2$ holds, then there are no holes in $u + 1$ by the definition of u. On the other hand, if $u = t_d - 2$, then there are no holes in $u + 1 = t_d - 1$ by Lemma 8(f). By the assumption for this case, there are no holes in u either. Therefore, by (17), we have

$$\begin{aligned}
LAG(\tau, u + 2) &= LAG(\tau, u) + \sum_{v=u+1}^{u+2} \sum_{T \in \tau} (share(T, v) - S(T, v)) \\
&= LAG(\tau, u) + \sum_{v=u+1}^{u+2} \sum_{T \in \tau} share(T, v) - 2M \\
&\qquad \text{, there are no holes in } u \text{ and } u + 1 \\
&< (2 - 2\alpha)M + 1 + \sum_{v=u+1}^{u+2} \sum_{T \in \tau} share(T, v) - 2M \\
&\qquad , LAG(\tau, u) < (2 - 2\alpha)M + 1 \\
&\leq (2 - 2\alpha)M + 1 + \sum_{v=u+1}^{u+2} \sum_{T \in \tau} wt(T) - 2M \qquad \text{, by (13)} \\
&= (2 - 2\alpha)M + 1 + 2\alpha M - 2M \qquad \text{, by Def. 6} \\
&= 1.
\end{aligned}$$

Thus, condition (45) holds for this case.
Case 2: $u = t_d - 2$ **and there is a hole in slot** $t_d - 2$**.**
Because $u = t_d - 2$ holds for this case, by the definition of u, the following holds.

(D2) No two consecutive slots in $[t + 1, t_d - 2)$ are without holes.

Let $u' < t_d - 2$ denote the last slot with no hole in $[t + 1, t_d - 2)$. We consider the following two subcases.
Subcase 2a: $t < u' < t_d - 3$ **or there is at least a hole in every slot in** $[t + 1, t_d - 2)$**.**
If $t < u' < t_d - 3$ holds, then by the definition of u', there is at least a hole in $t_d - 3$. On the other hand, if there is no slot without a hole in $[t + 1, t_d - 2)$, then, because $t < t_d - 2$ holds (by the statement of the lemma), by (D1), there is a hole in $t_d - 3$. Therefore, by (D2) and because $LAG(\tau, t + 1) > LAG(\tau, t)$ holds (by the statement of the lemma), Lemma 16 applies with $u = t_d - 2$. Hence, by Lemma 16, we have $LAG(\tau, t_d - 2) < (2 - 2\alpha)M + 1$. Therefore, for this case, (46) is satisfied.
Subcase 2b: $t < u' = t_d - 3$**.** Because there is no hole in slot $t_d - 3$ (by the assumption of this subcase), (D2) implies that there is at least a hole in slot $t_d - 4$. (Because $t_d - 3 > t$ holds (by the assumption of this subcase again), $t_d - 4$ exists.) Therefore, because

$LAG(\tau, t+1) > LAG(\tau, t)$ holds, Lemma 16 applies with $u = t_d - 3$. Hence,

$$\begin{aligned} LAG(\tau, t_d - 3) &< (2-2\alpha)M + 1 && \text{, by Lemma 16} \quad (48) \\ &< (3-3\alpha)M + 1 && \text{, by Lemma 3.} \quad (49) \end{aligned}$$

Further, by (17), we have

$$\begin{aligned} LAG(\tau, t_d - 2) &= LAG(\tau, t_d - 3) + \sum_{T \in \tau} (share(T, t_d - 3) - S(T, t_d - 3)) \\ &< (2-2\alpha)M + \sum_{T \in \tau} (share(T, t_d - 3) - S(T, t_d - 3)) && \text{, by (48)} \\ &= (2-2\alpha)M + \sum_{T \in \tau} share(T, t_d - 3) - M \\ & \qquad \text{, there are no holes in } t_d - 3 \text{ (by the assumption of this subcase)} \\ &\le (2-2\alpha)M + \sum_{T \in \tau} wt(T) - M && \text{, by (13)} \\ &= (2-2\alpha)M + \alpha M - M && \text{, by Def. 6} \\ &< (2-2\alpha)M. && \text{, by Lemma 3} \quad (50) \end{aligned}$$

By (D2), (49), and (50), condition (47) holds. □

By part (h) of Lemma 8, there exists a u, where $0 \le u < t_d - 2$, such that $LAG(\tau, u) < 1$ and $LAG(\tau, u+1) \ge 1$. Let t be the largest such u. Then, by Lemma 17, one of the following holds.

(**a**) There exists a t', where $t' \le t_d$, such that $LAG(\tau, t') < 1$.
(**b**) There does not exist a $v \le t_d - 2$ such that there are no holes in $[v, t_d - 2)$ and $LAG(\tau, v) \ge (t_d - v)(1-\alpha)M + 1$. (This is implied by both (46) and (47).)

If (a) holds, and $t' < t_d$ holds, then this contradicts the maximality of t. On the other hand, if $t' = t_d$, then it contradicts part (c) of Lemma 8. If (b) holds, then part (g) of the same lemma is contradicted. Therefore, our assumption that τ misses its deadline at t_d is incorrect, which in turn proves Theorem 1, for $k = 2$.

As a corollary to Theorem 1, we have the following utilization-based schedulability test for EPDF.

Corollary 1. *A GIS task set τ is schedulable on $M > 2$ processors under EPDF if the total utilization of τ is at most $U(M, W_{\max})$, where $W_{\max}$ is the maximum weight of any task in τ and $U(M, W_{\max})$ is given by Def.* 3.

Generalizing the proof. The proof of Theorem 1 given above for $k = 2$ can be generalized to $k > 2$ as follows. If $W_{\max} \le 1/2$, then it can be shown that for $LAG(\tau, t+1) > LAG(\tau, t)$ to hold, there should be no holes in slots $t-1$ and $t-2$, which is a generalization of Lemma 10. In general, if $W_{\max} \le \frac{1}{k-1}$, then it can be shown that at least $k-1$ slots preceding t are without holes. Similarly, parts (f) and (g) of Lemma 8 can be generalized as follows. There are no holes in the last $k-1$ slots (slots $[t_d - k + 1, t_d)$), and there exists a $v \le t_d - k$ such that there are no holes in every slot in $[v, t_d - k)$ and $LAG(\tau, v) \ge (t_d - v)(1-\alpha)M + 1$. A formal proof is omitted

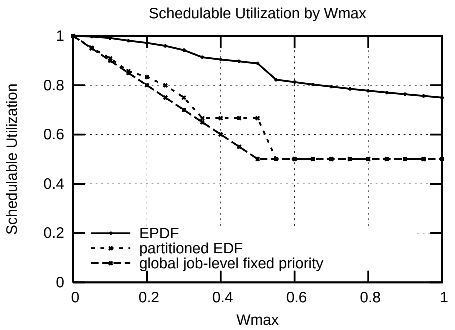

Fig. 4. Schedulable utilization by $W_{\max}$.

due to space constraints. Fig. 4 shows the plot of the schedulable utilizaton of EPDF (computed using the bound in Theorem 1 with a sufficiently large M, and expressed as a percentage of the total processing capacity) with respect to $W_{\max}$. For comparison, plots of optimal schedulable utilization for the partitioned approach with EDF [14] and the global approach that assigns a fixed priority to each job [6] (computed using their best known bounds expressed in terms of $W_{\max}$), are also shown in the same figure.

Is the schedulable utilization bound given by Theorem 1 optimal? As yet, we do not know the answer to this question. However, as the following example shows, the general bound cannot be improved to exceed $0.86M$.

Counterexample. Consider a task set comprised of $2n + 1$ tasks of weight $\frac{1}{2}$, n tasks of weight $\frac{3}{4}$, and n tasks of weight $\frac{5}{6}$ scheduled on $3n$ processors. There is an EPDF schedule for this task set in which a deadline miss occurs at time 12. (The schedule is not shown here due to space constraints.) The total utilization of this task set is $\frac{31n}{12} + \frac{1}{2}$, which approaches 86.1% of $3n$, the total processing capacity, as $n \to \infty$. Given that devising counterexamples for Pfair scheduling algorithms is somewhat hard, we believe that it may not be possible to significantly improve the bound of Theorem 1, which is asymptotically 75% of the total processing capacity.

Utilization restriction for a tardiness of q quanta. Having determined a sufficient utilization restriction for schedulability under EPDF, we were interested in determining a sufficient utilization restriction for a tardiness of q quanta. Extending the technique used above, we found that if the total utilization of τ is at most $\frac{(5q+6)M}{5q+8}$, then no subtask of τ misses its deadline by more than q quanta. For a tardiness of at most one, this imposes a sufficient utilization restriction of 84.6%. We feel that this is somewhat restrictive and that it can be improved *significantly* by identifying and exploiting the right properties of a system with a tardiness of q. We have deferred this for future work.

4 Conclusion

We have determined a schedulable utilization bound for the earliest-pseudo-deadline-first (EPDF) Pfair scheduling algorithm, and thereby, presented a sufficient schedulability test for EPDF. In general, this test allows any task set with total utilization not exceeding $\frac{3M+1}{4}$ to be scheduled on M processors. Our schedulability test is expressed in terms of the maximum weight of any task, and hence, may be used to schedule task sets with total utilization greater than $\frac{3M+1}{4}$. We have also presented a counterexample that suggests that a significant improvement to the test may not be likely. Finally, we have extended the test to allow a tardiness of q quanta.

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. *Journal of Computer and System Sciences*, 68(1):157–204, 2004.
4. B. Andersson, S. Baruah, and J. Jonsson. Static priority scheduling on multiprocessors. In *Proc. of the 22nd Real-time Systems Symposium*, pages 193–202, December 2001.
5. B. Andersson and J. Jonsson. The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%. In *Proc. of the 15th Euromicro Conference on Real-time Systems*, pages 33–40, July 2003.
6. S. Baruah. Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors. *IEEE Transactions on Computers*. To appear.
7. S. Baruah. Multiprocessor fixed-priority scheduling with restricted inter-processor migrations. In *Proc. of the 15th Euromicro Conference on Real-time Systems*, pages 195–202, July 2003.
8. S. Baruah, N. Cohen, C.G. Plaxton, and D. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15:600–625, 1996.
9. 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.
10. U. Devi and J. Anderson. Improved conditions for bounded tardiness under EPDF fair multiprocessor scheduling. In *Proc. of the 12th International Workshop on Parallel and Distributed Real-time Systems*, April 2004. To Appear.
11. U. Devi and J. Anderson. Schedulable utilization bounds for EPDF fair multiprocessor scheduling (full paper). Available at http://www.cs.unc.edu/˜anderson/papers.html, April 2004.
12. C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, 1973.
13. J.W.S. Liu. *Real-time Systems*. Prentice Hall, 2000.
14. J.M. Lopez, M. Garcia, J.L. Diaz, and D.F. Garcia. Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems. In *Proc. of the 12th Euromicro Conference on Real-time Systems*, pages 25–34, June 2000.
15. A. Srinivasan. *Efficient and Flexible Fair Scheduling of Real-time Tasks on Multiprocessors*. PhD thesis, University of North Carolina at Chapel Hill, Dec. 2003.
16. A. Srinivasan and J. Anderson. Optimal rate-based scheduling on multiprocessors. In *Proc. of the 34th ACM Symposium on Theory of Computing*, pages 189–198, May 2002.
17. A. Srinivasan and J. Anderson. Efficient scheduling of soft real-time applications on multiprocessors. In *Proc. of the 15th Euromicro Conference on Real-time Systems*, pages 51–59, July 2003.