Tardiness Bounds for FIFO Scheduling on Multiprocessors*

Hennadiy Leontyev and James H. Anderson
Department of Computer Science, University of North Carolina at Chapel Hill
leontyev@cs.unc.edu, anderson@cs.unc.edu

Abstract

FIFO *scheduling is often considered to be inappropriate for scheduling workloads that are subject to timing constraints. However,* FIFO *is implemented in many general-purpose OSs, and is more widely supported than other priority-based scheduling methods. In this paper, we show that, when the global* FIFO *scheduling algorithm is used to schedule sporadic real-time tasks on a multiprocessor, deadline tardiness is bounded. This result shows that global* FIFO *may in fact be useful for scheduling soft real-time workloads.*

1. Introduction

Historically, FIFO (first-in, first-out) scheduling was one of the first scheduling policies to be implemented in time-sharing systems. Due to its simplicity, it is widely supported. For example, it is implemented in the Linux kernel to schedule tasks with high priority [1]. FIFO is inferior to other methods as a means for scheduling recurring activities with hard timing constraints [9]. Under FIFO, tasks start missing deadlines on a uniprocessor when total utilization is approximately 45% in the average case, and 5% in the worst case [4]. In contrast, the EDF (earliest-deadline-first) algorithm correctly schedules any task set with total utilization at most 100% on uniprocessors. Nonetheless, FIFO is a starvation-free policy, *i.e.*, no pending job is denied execution, and thus is of use in time-sharing and packet-processing systems [1, 10].

In this paper, we consider the problem of scheduling soft real-time sporadic tasks using the global FIFO algorithm on a multiprocessor. This focus is driven by two factors. First, although variants of FIFO scheduling are widely implemented, they are often dismissed as an option for scheduling real-time tasks — we wish to know if this dismissive attitude is warranted. Second, with the emergence of multicore technologies, the common platform in the future in many settings will be a multiprocessor. These platforms will likely be deployed in many settings where soft real-time constraints are required [2]. On multicore platforms, task migration costs are usually much lower than on conventional SMPs, due to the presence of shared on-chip caches. Thus, as kernels evolve to be more "multicore-aware," a strong case can be made that global scheduling policies should be supported. As in the past, we expect FIFO scheduling to be more widely deployed (at least, in general-purpose OSs) than other priority-based policies. FIFO is better understood by many system designers, and other policies that have been proposed for scheduling soft real-time workloads on multiprocessors (*e.g.* [12, 6]) would require more effort to integrate into existing kernels.

In this paper, we show that global FIFO scheduling (henceforth, referred to simply as FIFO) ensures bounded deadline tardiness when scheduling sporadic real-time task systems on a multiprocessor. The term *tardiness* refers to the extent by which deadlines may be missed. If bounded tardiness can be ensured for a sporadic task, then its long-term processor share can be guaranteed — such guarantees are sufficient for many soft real-time applications. Under FIFO, jobs with earlier release times are given higher priority, and as such, execute non-preemptively. Non-preemptive execution has an advantage of simplifying WCET analysis.

Like global EDF [6, 7, 13], FIFO does not require restrictive caps on total utilization to ensure bounded tardiness. In contrast, such caps are required for ensuring timing constraints (even bounded tardiness) under the most widely-studied alternative, partitioning schemes. The need for such caps arises due to connections to bin-packing. Because of these connections, there exist real-time workloads with total utilization of approximately $m/2$ that cannot be scheduled via partitioning on m processors. Although

*Work supported by a grant from Intel Corp., by NSF grants CNS 0408996, CCF 0541056, and CNS 0615197 and by ARO grant W911NF-06-1-0425.

this weakness is counterbalanced by the avoidance of task migrations, under FIFO, tasks can only migrate at job boundaries, and overall utilization can be as high as m (if tardiness is allowed).

Prior work. In [9], George and Minet considered the use of FIFO in hard real-time distributed systems with consistency constraints. They presented a pseudo-polynomial time necessary and sufficient condition for the feasibility of a sporadic task system scheduled under FIFO. They also showed that FIFO with deadline-monotonic tie-breaks is optimal in the class of uniprocessor FIFO scheduling algorithms.

The analysis of FIFO in our paper uses ideas from two prior papers by Devi and Anderson [6, 7]. In [6], tardiness bounds for preemptive and non-preemptive global EDF (henceforth referred as to EDF and NP-EDF, respectively) are established. These bounds apply to any sporadic task system with total utilization at most m scheduled in an EDF manner on m symmetric processors. The analysis from [6] is improved in [7]. In EDF and NP-EDF, jobs are prioritized by their absolute deadlines. In FIFO, jobs are prioritized by their release times. In all three algorithms, the priority of a job does not change throughout its existence. This type of job prioritization is called *job-level restricted dynamic* [8]. Hence, results obtained for EDF and NP-EDF are of relevance to our work.

Summary of contributions. The main contribution of this paper is to establish tardiness bounds for FIFO. We also present an experimental evaluation of FIFO's effectiveness in limiting tardiness compared to EDF and NP-EDF. In the workloads considered in these experiments, *maximum* tardiness (both observed and via theoretical bounds) was higher under FIFO than under EDF and NP-EDF. However, the three schemes exhibited comparable *average* tardiness. To our knowledge, this paper is the first to show that tardiness for sporadic tasks scheduled under FIFO is bounded (on multiprocessors or uniprocessors). In what follows, we first present some necessary definitions in Sec. 2, and then present our formal and experimental results in Secs. 3–4.

2. System Model

We consider the problem of scheduling a set τ of sporadic tasks on $m \geq 2$ unit-speed processors. (Tardiness is bounded for $m = 1$, but due to space constraints, we only consider the case $m \geq 2$.) We assume that τ consists of n independent tasks, $T_1, \ldots, T_n$. Each task is invoked or *released* repeatedly, with each such invocation called a *job*. Associated with each task T_i are two parameters, e_i and p_i: e_i gives the maximum *execution time* of one job of T_i, while, p_i, called the *period* of T_i, gives the minimum time between two consecutive job releases of T_i. For brevity, T_i's parameters are sometimes denoted using the notation $T_i = (e_i, p_i)$. We assume that each job of each task T_i executes for *exactly* e_i time units. This assumption can be eased to treat e_i as an upper bound, at the expense of more cumbersome notation. The *utilization of task* T_i is defined as $u_i = e_i/p_i$, and the *utilization of the task system* τ as $U_{sum} = \sum_{T_i \in \tau} u_i$.

The k^{th} job of T_i, where $k \geq 1$, is denoted $T_{i,k}$. A task's first job may be released at any time at or after time zero. The release time of the job $T_{i,k}$ is denoted $r_{i,k}$ and its (absolute) deadline $d_{i,k}$ is defined as $r_{i,k} + p_i$. If a job $T_{i,j}$ with a deadline at $d_{i,j}$ completes at time t, then its *tardiness* is defined as $max(0, t - d_{i,j})$. A *task*'s tardiness is the maximum of the tardiness of any of its jobs. We require $u_i \leq 1$ and $U_{sum} \leq m$. Otherwise, tardiness can grow unboundedly. When a job of a task misses its deadline, the release time of the next job of that task is not altered. This ensures that each task receives a processor share in accordance with its utilization in the long term (if tardiness is bounded). Each task is sequential, so at most one job of a task may execute at any time, even if deadlines are missed.

We assume that released jobs are placed into a single global ready queue and are prioritized by their release times, with deadline-monotonic tie-breaking. That is, $T_{i,j}$ has higher priority than $T_{k,l}$ if $(r_{i,j} < r_{k,l}) \vee ((r_{i,j} = r_{k,l}) \wedge (p_i < p_k))$. We further assume that if two jobs have equal priority and one is executing, then the tie is broken in its favor. Finally, (as reiterated in Def. 3 below) a job is *ready* for execution if it has been released and its predecessor has completed execution. When choosing a new job to schedule, the scheduler selects (and dequeues) the ready job of highest priority.

3. Tardiness Bound

In this section, we derive a tardiness bound for FIFO using techniques presented in [6, 7] for EDF and NP-EDF. Some results proved in these two prior papers also hold for FIFO.

3.1. Definitions

The system start time is assumed to be zero. For any time $t > 0$, t^- denotes the time $t - \epsilon$ in the limit $\epsilon \to 0+$.

Definition 1. (active jobs) A task T_i is *active* at time t if there exists a job $T_{i,j}$ (called T_i's *active job* at t) such that $r_{i,j} \leq t < d_{i,j}$. By our task model, every task has at most one active job at any time.

Definition 2. (pending jobs) $T_{i,j}$ is *pending* at t in a schedule $\mathcal{S}$ if $r_{i,j} \leq t$ and $T_{i,j}$ has not completed execution by t in $\mathcal{S}$.

Definition 3. (ready jobs) A pending job $T_{i,j}$ is *ready* at t in a schedule $\mathcal{S}$ if $t \geq r_{i,j}$ and all prior jobs of T_i have completed execution by t in $\mathcal{S}$.

Claim 1. *Under* FIFO*, jobs execute non-preemptively.*

Proof. Suppose that job $T_{i,j}$ begins execution at time t. Then, at that time, at most $m-1$ tasks have ready jobs with priority higher than $T_{i,j}$. We claim that this property remains true until $T_{i,j}$ completes execution, which implies that it executes non-preemptively. To see that this property holds, note that any job that becomes ready after t does so either because it was just released, or its predecessor job has just completed execution. In the former case, the new job clearly has lower priority than $T_{i,j}$, since it has a later release. This same reasoning holds in the latter case as well, unless the new job is released before $T_{i,j}$. However, in this case, the new job's predecessor also has an earlier release, so the number of tasks with jobs of higher priority than $T_{i,j}$ remains unchanged. □

Before continuing, we consider an example, which illustrates some of the difference between EDF, NP-EDF, and FIFO.

Example 1. Consider a task set τ to be scheduled under EDF, NP-EDF, and FIFO that is comprised of four tasks: $T_1 = (1, 2)$, $T_2 = (2, 6)$, $T_3 = (2, 8)$, and $T_4 = (11, 12)$. The first jobs of T_1, T_2, T_3, and T_4 are released at times 2, 1, 0, and 0, respectively. Total utilization is two, so this system can be scheduled on two processors. Consider the three schedules in Fig. 1 for τ. In the EDF and FIFO schedules, in insets (a) and (c), respectively, deadline misses occur. For example, the job $T_{4,1}$ misses its deadline at time 12 by one time unit under EDF because it cannot execute in parallel, and hence, cannot benefit from the spare processing capacity in the system during the intervals $[3, 4)$, $[5, 6)$, $[7, 8)$, and $[9, 10)$. Under FIFO, $T_{1,1}$ misses its deadline at time 4 by one time unit because it cannot preempt $T_{2,1}$ and $T_{4,1}$, which have earlier release times and later deadlines.

A task system is *concrete* if the release times of all jobs are specified, and *non-concrete*, otherwise. The tardiness bound established for FIFO is derived by comparing the allocations to a concrete task system τ in an ideal processor-sharing (PS) schedule to those in a FIFO schedule. In a *PS schedule*, each job of a task T_i is executed at a constant rate of u_i between its release and

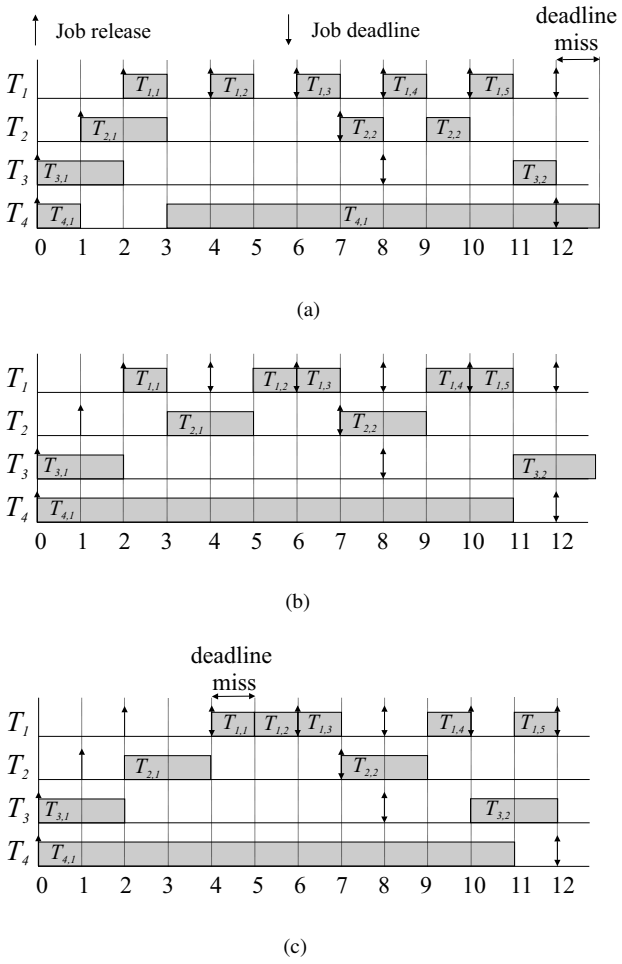


Figure 1. Schedules under (a) EDF, (b) NP-EDF, and (c) FIFO for the task set in Example 1.

deadline. As an example, consider Fig. 2, which shows the PS schedule for the task system τ in Example 1. Note that, in a PS schedule, each job completes exactly at its deadline. Thus, if a job misses its deadline, then it is "lagging behind" the PS schedule — this concept of "lag" is instrumental in the analysis and is formalized below.

So that we may compare allocations in different schedules, let $\mathsf{A}(T_{i,j}, t_1, t_2, \mathcal{S})$ denote the total allocation to the job $T_{i,j}$ in an arbitrary schedule $\mathcal{S}$ in $[t_1, t_2)$. Similarly, let $\mathsf{A}(T_i, t_1, t_2, \mathcal{S}) = \sum_{j \geq 1} \mathsf{A}(T_{i,j}, t_1, t_2, \mathcal{S})$ denote the total time allocated to all jobs of T_i in $[t_1, t_2)$ in $\mathcal{S}$.

Since, in a PS schedule, T_i executes with the rate u_i

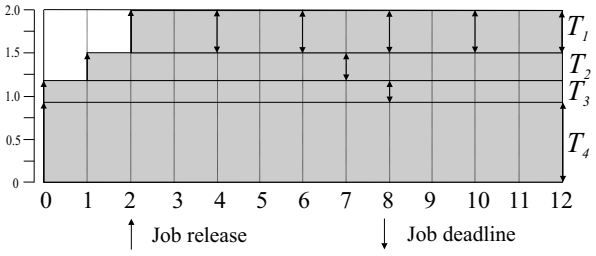


Figure 2. PS **schedule for τ in Example 1.**

at each instant when it is active in $[t_1, t_2)$, we have

$$\mathsf{A}(T_i, t_1, t_2, \mathsf{PS}) \leq (t_2 - t_1)u_i. \tag{1}$$

The difference between the allocations to a job $T_{i,j}$ up to time t in a PS schedule and an arbitrary schedule $\mathcal{S}$, termed the *lag of job $T_{i,j}$ at time t in schedule $\mathcal{S}$*, is given by

$$\mathsf{lag}(T_{i,j}, t, \mathcal{S}) = \mathsf{A}(T_{i,j}, 0, t, \mathsf{PS}) - \mathsf{A}(T_{i,j}, 0, t, \mathcal{S}). \tag{2}$$

The *lag of a task T_i at time t in schedule $\mathcal{S}$* is defined by

$$\begin{aligned} \mathsf{lag}(T_i, t, \mathcal{S}) &= \sum_{j \geq 1} \mathsf{lag}(T_{i,j}, t, \mathcal{S}) \\ &= \mathsf{A}(T_i, 0, t, \mathsf{PS}) - \mathsf{A}(T_i, 0, t, \mathcal{S}). \end{aligned} \tag{3}$$

Finally, the *lag for a finite job set Ψ at time t in the schedule $\mathcal{S}$* is defined by

$$\begin{aligned} &\mathsf{LAG}(\Psi, t, \mathcal{S}) \\ &= \sum_{T_{i,j} \in \Psi} \mathsf{lag}(T_{i,j}, t, \mathcal{S}) \\ &= \sum_{T_{i,j} \in \Psi} (\mathsf{A}(T_{i,j}, 0, t, \mathsf{PS}) - \mathsf{A}(T_{i,j}, 0, t, \mathcal{S})). \end{aligned} \tag{4}$$

Since $\mathsf{LAG}(\Psi, 0, \mathcal{S}) = 0$, the following hold for $t' \leq t$.

$$\begin{aligned} &\mathsf{LAG}(\Psi, t, \mathcal{S}) \\ &= \mathsf{LAG}(\Psi, t', \mathcal{S}) + \mathsf{A}(\Psi, t', t, \mathsf{PS}) - \mathsf{A}(\Psi, t', t, \mathcal{S}) \end{aligned} \tag{5}$$

$$\begin{aligned} &\mathsf{lag}(T_i, t, \mathcal{S}) \\ &= \mathsf{lag}(T_i, t', \mathcal{S}) + \mathsf{A}(T_i, t', t, \mathsf{PS}) - \mathsf{A}(T_i, t', t, \mathcal{S}) \end{aligned} \tag{6}$$

In essence, the concept of lag is important because, if it can be shown that lags remain bounded, then tardiness is bounded as well.

Definition 4. (busy and non-busy intervals) A time interval $[t_1, t_2)$, where $t_2 > t_1$, is said to be *busy* for any job set Ψ if all m processors are executing some job in Ψ at each instant in the interval, *i.e.*, no processor is ever idle in the interval or executes a job not in Ψ. An interval $[t_1, t_2)$ that is not busy for Ψ is said to be *non-busy* for Ψ. An interval $[t_1, t_2)$ is said to be *maximally non-busy* if it is non-busy at every instant within it and either t_1^- is a busy instant or $t_1 = 0$.

We are interested in non-busy intervals (for a job set) because total lag (for that job set) can increase only across such intervals. Such increases can lead to deadline misses. This fact is proved in Lemma 2 in [6]. The following example illustrates how lag can change across busy and non-busy intervals.

Example 2. Consider task set τ from Example 1. Let $\Psi = \{T_{1,1}, \ldots, T_{1,5}, T_{2,1}, T_{3,1}, T_{4,1}\}$ be the set of jobs with deadlines at most 12. The interval $[4, 7)$ in Fig. 1(c) is a busy interval for Ψ. We will show that the LAG for Ψ at the end of this interval is equal to the LAG for Ψ at the beginning of the interval. By (5), $\mathsf{LAG}(\Psi, 7, \mathcal{S}) = \mathsf{LAG}(\Psi, 4, \mathcal{S}) + \mathsf{A}(\Psi, 4, 7, \mathsf{PS}) - \mathsf{A}(\Psi, 4, 7, \mathcal{S})$, where $\mathcal{S}$ is the FIFO schedule. The allocation of Ψ in the PS schedule during the interval $[4, 7)$ is $\mathsf{A}(\Psi, 4, 7, \mathsf{PS}) = 3/2 + 6/6 + 6/8 + 33/12 = 6$. The allocation of Ψ in $\mathcal{S}$ throughout $[4, 7)$ is also 6.

Now let $\Psi = \{T_{1,1}\}$ be the set of jobs with deadlines at most 4. Because the jobs $T_{2,1}$ and $T_{4,1}$, which have deadlines after time 4, execute within the interval $[2, 4)$ in Fig. 1(c), this interval is non-busy for Ψ in $\mathcal{S}$. By (4), $\mathsf{LAG}(\Psi, 4, \mathcal{S}) = \mathsf{A}(\Psi, 0, 4, \mathsf{PS}) - \mathsf{A}(\Psi, 0, 4, \mathcal{S})$. The allocation of Ψ in the PS schedule throughout the interval $[0, 4)$ is $\mathsf{A}(\Psi, 0, 4, \mathsf{PS}) = 2 \cdot 1/2 = 1$. The allocation of Ψ in $\mathcal{S}$ is $\mathsf{A}(\Psi, 0, 4, \mathcal{S}) = 0$. Thus, $\mathsf{LAG}(\Psi, 4, \mathcal{S}) = 1 - 0 = 1$. Fig. 1(c) shows that at time 4, $T_{1,1}$ from Ψ is pending. This job has unit execution cost, which is equal to the amount of pending work given by $\mathsf{LAG}(\Psi, 4, \mathcal{S})$.

3.2. Tardiness Bound for FIFO

Given an arbitrary non-concrete task system τ^N, we want to determine the maximum tardiness of any job of any task in any concrete instantiation of τ^N. The approach for doing this is taken from [6, 7]. Let τ be a concrete instantiation of τ^N. Let $T_{\ell,j}$ be a job of a task T_ℓ in τ, let $t_d = d_{\ell,j}$, and let $\mathcal{S}$ be a FIFO schedule for τ with the following property.

(P) The tardiness of every job of every task T_k in τ with deadline less than t_d is at most $x + e_k$ in $\mathcal{S}$,

where $x \geq 0$.

Our goal is to determine the smallest x such that the tardiness of $T_{\ell,j}$ remains at most $x + e_\ell$. Such a result would by induction imply a tardiness of at most $x + e_k$ for all jobs of every task $T_k \in \tau$. Because τ is arbitrary, the tardiness bound will hold for every concrete instantiation of τ^N.

The objective is easily met if $T_{\ell,j}$ completes by its deadline, t_d, so assume otherwise. The completion time of $T_{\ell,j}$ then depends on the amount of work that can compete with $T_{\ell,j}$ after t_d. Hence, a value for x can be determined via the following steps.

1. Compute an upper bound on the work pending for tasks in τ (including that for $T_{\ell,j}$) that can compete with $T_{\ell,j}$ after t_d.
2. Determine the amount of such work necessary for the tardiness of $T_{\ell,j}$ to exceed $x + e_\ell$.
3. Determine the smallest x such that the tardiness of $T_{\ell,j}$ is at most $x + e_\ell$ using the upper bound and the necessary condition above.

To reason about the tardiness of $T_{\ell,j}$ we need to determine how other jobs delay the execution of $T_{\ell,j}$. We classify jobs based on the relation between their release times and deadlines to those of $T_{\ell,j}$, as follows.

$$\begin{aligned}
\mathbf{rd} &= \{T_{i,k} : (r_{i,k} \leq r_{\ell,j}) \wedge (d_{i,k} \leq t_d)\} \\
\mathbf{Rd} &= \{T_{i,k} : (r_{i,k} > r_{\ell,j}) \wedge (d_{i,k} \leq t_d)\} \\
\mathbf{rD} &= \{T_{i,k} : (r_{i,k} < r_{\ell,j}) \wedge (d_{i,k} > t_d)\} \\
\mathbf{RD} &= \{T_{i,k} : (r_{i,k} \geq r_{\ell,j}) \wedge (d_{i,k} > t_d)\}
\end{aligned}$$

In this notation, **d** denotes deadlines at most t_d, and **D** denotes deadlines greater than t_d. Also, **r** denotes release times at most or strictly less than $r_{\ell,j}$, and **R** denotes release times at least or strictly greater than $r_{\ell,j}$. For any t_d, the sets **rd**, **Rd**, and **rD** are finite. Note also that $T_{\ell,j} \in \mathbf{rd}$.

Example 3. Consider the task set $\tau = \{T_1(1,2), T_2(4,7), T_3(8,9)\}$ and the PS schedule for it in Fig. 3. Jobs $T_{1,1}$ and $T_{2,1}$ are released at time 1, and job $T_{3,1}$ is released at time 0. Consider job $T_{\ell,j} = T_{2,1}$, which has a deadline at time 8. With respect to $T_{2,1}$, the four sets mentioned above are $\mathbf{rd} = \{T_{1,1}, T_{2,1}\}$, $\mathbf{Rd} = \{T_{1,2}, T_{1,3}\}$, $\mathbf{rD} = \{T_{3,1}\}$, and $\mathbf{RD} = \{T_{1,4}, T_{1,5}, T_{2,2}, T_{3,2}\}$. (**RD** would also include any jobs released after those shown in the figure.)

The set of jobs with deadlines at most t_d is further referred to as $\Psi = \mathbf{rd} \cup \mathbf{Rd}$. We are interested in this set of jobs because these jobs do not execute beyond

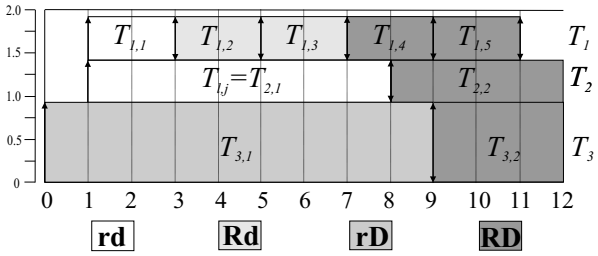


Figure 3. Job set partitioning.

t_d in the PS schedule. Because jobs in $\mathbf{rd} \cup \mathbf{rD}$ have priority at least that of $T_{\ell,j}$, the execution of $T_{\ell,j}$ will be postponed (in the worst case) until there are at most m ready jobs in Ψ including $T_{\ell,j}$ with priority at least that of $T_{\ell,j}$.

So that we can analyze the impact of jobs from **rD**, let the *carry-in* job $T_{k,j}$ of a task T_k be defined as the job, if any, for which $r_{k,j} \leq t_d < d_{k,j}$ holds. At most one such job could exist for each task T_k.

Determining an upper bound on competing work. Because jobs in $\mathbf{rd} \cup \mathbf{rD}$ have priority at least that of $T_{\ell,j}$, the competing work for $T_{\ell,j}$ beyond t_d is upper bounded by the sum of **(i)** the amount of work pending at t_d for jobs in **rd**, and **(ii)** the amount of work $\mathsf{D}(\mathbf{rD}, t_d, \mathcal{S})$ demanded by jobs in **rD** that can compete with $T_{\ell,j}$ after t_d.

For the pending work mentioned in (i), because jobs from Ψ have deadlines at most t_d, they do not execute in the PS schedule beyond t_d. Thus, the work pending for jobs in **rd** is given by $\mathsf{LAG}(\mathbf{rd}, t_d, \mathcal{S})$, which must be positive in order for $T_{\ell,j}$ to miss its deadline at t_d. We find it more convenient to reason about $\mathsf{LAG}(\Psi, t_d, \mathcal{S})$ instead. Note that $\mathsf{LAG}(\mathbf{Rd}, t_d, \mathcal{S})$ is non-negative because the jobs in **Rd** cannot perform more work by time t_d in $\mathcal{S}$ than they have performed in the PS schedule. Hence, $\mathsf{LAG}(\mathbf{rd}, t_d, \mathcal{S}) \leq \mathsf{LAG}(\Psi, t_d, \mathcal{S})$ Thus, the desired upper bound on the amount of competing work for job $T_{\ell,j}$ after t_d is given by $\mathsf{LAG}(\Psi, t_d, \mathcal{S}) + \mathsf{D}(\mathbf{rD}, t_d, \mathcal{S})$. We are left with determining upper bounds on $\mathsf{LAG}(\Psi, t_d, \mathcal{S})$ and $\mathsf{D}(\mathbf{rD}, t_d, \mathcal{S})$.

Upper bound on $\mathsf{LAG}(\Psi, t_d, \mathcal{S})$. In deriving this bound, we assume that all busy and non-busy intervals considered are with respect to Ψ and the FIFO schedule $\mathcal{S}$ unless stated otherwise. As mentioned above, LAG can increase only throughout a non-busy interval. Thus, $\mathsf{LAG}(\Psi, t_d, \mathcal{S}) \leq \mathsf{LAG}(\Psi, t', \mathcal{S})$, where t' is the end of the latest maximally non-busy interval at or before t_d. If such a non-busy interval does not exist, then the entire interval $[0, t_d)$ is busy. Hence, $\mathsf{LAG}(\Psi, t_d, \mathcal{S}) \leq 0$, so

all jobs in Ψ complete by t_d, which is contrary to our assumption that $T_{\ell,j}$ misses its deadline at t_d.

An interval could be non-busy for two reasons:

1. There are not enough ready jobs in Ψ to occupy all available processors. In this case, it is immaterial whether jobs from **rD** or **RD** execute during this interval. We call such an interval *non-busy non-displacing*.

2. Jobs in **rD** occupy one or more processors (because they have earlier release times), and there are ready jobs in Ψ. We call such an interval *non-busy displacing*.

Example 4. In the FIFO schedule in Fig. 1(c), for the task set τ from Example 1, the interval $[2, 4)$ is a non-busy displacing interval for $\Psi = \{T_{1,1}\}$. The ready job $T_{1,1}$, which has a deadline at time 4, does not execute. Jobs $T_{2,1}$ and $T_{4,1}$, which have deadlines at times 7 and 12, respectively, execute because they are released at time 0.

The displacement of jobs in Ψ is caused by jobs in **rD**, which have higher priority than $T_{\ell,j}$. **rD** consists of carry-in jobs, which have a release time before $r_{\ell,j}$ and a deadline after t_d. As noted earlier, only one carry-in job can exist for each task.

Definition 5. Let τ_H be the set of tasks that have jobs in **rD**.

Definition 6. Let δ_k be the amount of work performed by a carry-in job $T_{k,j}$ in the schedule $\mathcal{S}$ by time t_d.

In much of the rest of the analysis, we focus on a time t_n defined as follows: if there exists a non-busy non-displacing interval before t_d, across which LAG for Ψ increases, then t_n is the end of the latest such interval; otherwise, $t_n = 0$. In Lemma 1 below, we establish a relationship between $\mathsf{LAG}(\Psi, t_d, \mathcal{S})$ and $\mathsf{LAG}(\Psi, t_n, \mathcal{S})$. We then upper bound $\mathsf{LAG}(\Psi, t_n, \mathcal{S})$ in Lemma 3 by a function of the task parameters and x.

Lemma 1. $\mathsf{LAG}(\Psi, t_d, \mathcal{S}) \leq \mathsf{LAG}(\Psi, t_n, \mathcal{S}) + \sum_{T_k \in \tau_H} \delta_k(1 - u_k)$.

Proof. By (4),

$$\begin{aligned}&\mathsf{LAG}(\Psi, t_d, \mathcal{S})\\ &\quad\leq\ \mathsf{LAG}(\Psi, t_n, \mathcal{S})\\ &\qquad +\mathsf{A}(\Psi, t_n, t_d, \mathsf{PS}) - \mathsf{A}(\Psi, t_n, t_d, \mathcal{S}). \quad (7)\end{aligned}$$

We split $[t_n, t_d)$ into b non-overlapping intervals $[t_{p_i}, t_{q_i})$ $1 \leq i \leq b$ such that $t_n = t_{p_1}$, $t_{q_{i-1}} = t_{p_i}$, and $t_{q_b} = t_d$. Each interval $[t_{p_i}, t_{q_i})$ is either busy, non-busy displacing, or non-busy non-displacing (see Fig. 4). We

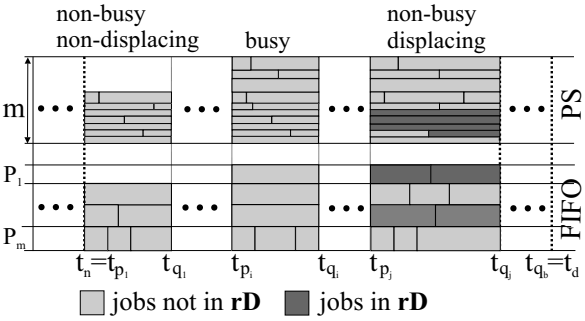


Figure 4. Splitting the interval $[t_n, t_d)$ into subintervals $[t_{p_i}, t_{q_i})$.

assume that any displacing interval $[t_{p_i}, t_{q_i})$ is defined so that if a task $T_k \in \tau_H$ executes at some point in the interval, then it executes continuously throughout the interval. Note that such a task T_k does not necessarily execute continuously throughout $[t_n, t_d)$. For each displacing interval $[t_{p_i}, t_{q_i})$, we define a subset of tasks $\alpha_i \subseteq \tau_H$ that execute continuously throughout $[t_{p_i}, t_{q_i})$. The allocation difference for Ψ throughout the interval $[t_n, t_d)$ is thus $\mathsf{A}(\Psi, t_n, t_d, \mathsf{PS}) - \mathsf{A}(\Psi, t_n, t_d, \mathcal{S}) = \sum_{i=1}^{b} \mathsf{A}(\Psi, t_{p_i}, t_{q_i}, \mathsf{PS}) - \mathsf{A}(\Psi, t_{p_i}, t_{q_i}, \mathcal{S})$.

We now bound the difference between the work performed in the PS schedule and the FIFO schedule $\mathcal{S}$ across each of these intervals $[t_{p_i}, t_{q_i})$. The sum of these bounds gives us a bound on the total allocation difference throughout $[t_n, t_d)$. Depending on the nature of the interval $[t_{p_i}, t_{q_i})$, three cases are possible.

Case 1. $[t_{p_i}, t_{q_i})$ *is busy.* Because in $\mathcal{S}$ all processors are occupied by jobs in Ψ, $\mathsf{A}(\Psi, t_{p_i}, t_{q_i}, \mathcal{S}) = m(t_{q_i} - t_{p_i})$. In PS, $\mathsf{A}(\Psi, t_{p_i}, t_{q_i}, \mathsf{PS}) \leq U_{sum}(t_{q_i} - t_{p_i})$. Since $U_{sum} \leq m$, we have

$$\mathsf{A}(\Psi, t_{p_i}, t_{q_i}, \mathsf{PS}) - \mathsf{A}(\Psi, t_{p_i}, t_{q_i}, \mathcal{S}) \leq 0. \quad (8)$$

Case 2. $[t_{p_i}, t_{q_i})$ *is non-busy non-displacing.* By the selection of t_n, LAG does not increase for Ψ across $[t_{p_i}, t_{q_i})$. Therefore, from (5), we have

$$\mathsf{A}(\Psi, t_{p_i}, t_{q_i}, \mathsf{PS}) - \mathsf{A}(\Psi, t_{p_i}, t_{q_i}, \mathcal{S}) \leq 0. \quad (9)$$

Case 3. $[t_{p_i}, t_{q_i})$ *is non-busy displacing.* The cumulative utilization of all tasks $T_k \in \alpha_i$, which execute continuously throughout $[t_{p_i}, t_{q_i})$, is $\sum_{T_k \in \alpha_i} u_k$. The carry-in jobs of these tasks do not belong to Ψ, by the definition of Ψ. Therefore, by (1), the allocation of jobs in Ψ during $[t_{p_i}, t_{q_i})$ in PS is at most $\mathsf{A}(\Psi, t_{p_i}, t_{q_i}, \mathsf{PS}) \leq (t_{q_i} - t_{p_i})(m - \sum_{T_k \in \alpha_i} u_k)$.

All processors are occupied at every time instant in the interval $[t_{p_i}, t_{q_i})$, because it is displacing. Thus, $\mathsf{A}(\Psi, t_{p_i}, t_{q_i}, \mathcal{S}) = (t_{q_i} - t_{p_i})(m - |\alpha_i|)$. Therefore, the allocation difference for jobs in Ψ throughout the interval is

$$\begin{aligned}
&\mathsf{A}(\Psi, t_{p_i}, t_{q_i}, \mathsf{PS}) - \mathsf{A}(\Psi, t_{p_i}, t_{q_i}, \mathcal{S}) \\
&\leq (t_{q_i} - t_{p_i}) \left((m - \sum_{T_k \in \alpha_i} u_k) - (m - |\alpha_i|) \right) \\
&= (t_{q_i} - t_{p_i}) \sum_{T_k \in \alpha_i} (1 - u_k). \qquad (10)
\end{aligned}$$

To finish the proof, define $\alpha_i = \emptyset$ for all intervals $[t_{p_i}, t_{q_i})$ that are either busy or non-busy non-displacing. Then, summing the allocation differences for all the intervals $[t_{p_i}, t_{q_i})$ given by (8), (9), and (10), we have

$$\begin{aligned}
&\mathsf{A}(\Psi, t_n, t_d, \mathsf{PS}) - \mathsf{A}(\Psi, t_n, t_d, \mathcal{S}) \\
&\leq \sum_{i=1}^{b} \sum_{T_k \in \alpha_i} (t_{p_i} - t_{q_i})(1 - u_k).
\end{aligned}$$

For each task $T_k \in \tau_H$, the sum of the lengths of the intervals $[t_{p_i}, t_{q_i})$, in which the carry-in job of T_k executes continuously is at most δ_k. Thus, $\mathsf{A}(\Psi, t_n, t_d, \mathsf{PS}) - \mathsf{A}(\Psi, t_n, t_d, \mathcal{S}) \leq \sum_{T_k \in \tau_H} \delta_k(1 - u_k)$. Setting this value into (7), we get $\mathsf{LAG}(\Psi, t_d, \mathcal{S}) \leq \mathsf{LAG}(\Psi, t_n, \mathcal{S}) + \sum_{T_k \in \tau_H} \delta_k(1 - u_k)$. □

We now determine an upper bound on $\mathsf{LAG}(\Psi, t_n, \mathcal{S})$, where t_n is as defined earlier. We first establish the following

Lemma 2. *$\mathsf{lag}(T_k, t, \mathcal{S}) \leq x \cdot u_k + e_k$ for any task T_k and $t \in [0, t_d]$.*

Proof. Let $d_{k,j}$ be the deadline of the earliest pending job of T_k, $T_{k,j}$, in the FIFO schedule $\mathcal{S}$ at time t. Let γ_k be the amount of work $T_{k,j}$ performs before t. We prove the lemma for the case $d_{k,j} < t$, leaving the case $d_{k,j} \geq t$ to the reader. By (3) and the selection of $T_{k,j}$,

$$\begin{aligned}
&\mathsf{lag}(T_k, t, \mathcal{S}) \\
&= \sum_{h \geq j} \mathsf{lag}(T_{k,h}, t, \mathcal{S}) \\
&= \sum_{h \geq j} (\mathsf{A}(T_{k,h}, 0, t, \mathsf{PS}) - \mathsf{A}(T_{k,h}, 0, t, \mathcal{S})).
\end{aligned}$$

Because no job executes before its release time, $\mathsf{A}(T_{k,h}, 0, t, \mathcal{S}) = \mathsf{A}(T_{k,h}, r_{k,h}, t, \mathcal{S})$. Thus,

$$\begin{aligned}
&\mathsf{lag}(T_k, t, \mathcal{S}) \\
&= \sum_{h > j} (\mathsf{A}(T_{k,h}, r_{k,h}, t, \mathsf{PS}) - \mathsf{A}(T_{k,h}, r_{k,h}, t, \mathcal{S})) \\
&\quad + \mathsf{A}(T_{k,j}, r_{k,j}, t, \mathsf{PS}) - \mathsf{A}(T_{k,j}, r_{k,j}, t, \mathcal{S}) \qquad (11)
\end{aligned}$$

By the definition of PS, $\mathsf{A}(T_{k,j}, r_{k,j}, t, \mathsf{PS}) = e_k$ and $\sum_{h>j} \mathsf{A}(T_{k,h}, r_{k,h}, t, \mathsf{PS}) \leq u_k(t - d_{k,j})$. By the selection of $T_{k,j}$, $\mathsf{A}(T_{k,j}, r_{k,j}, t, \mathcal{S}) = \gamma_k$ and $\sum_{h>j} \mathsf{A}(T_{k,h}, r_{k,h}, t, \mathcal{S}) = 0$. Setting these values into (11), we have

$$\mathsf{lag}(T_k, t, \mathcal{S}) \leq u_k(t - d_{k,j}) + e_k - \gamma_k. \qquad (12)$$

By Property (P), $T_{k,j}$ has tardiness at most $x + e_k$, so $t + e_k - \gamma_k \leq d_{k,j} + x + e_k$. Thus, $t - d_{k,j} \leq x + \gamma_k$. From (12), we therefore have $\mathsf{lag}(T_k, t, \mathcal{S}) \leq u_k(t - d_{k,j}) + e_k - \gamma_k \leq u_k \cdot x + e_k$. □

The lemma above was originally proved in [7] for EDF, but since it only depends on Property (P), it also holds for FIFO.

Lemma 3 below upper bounds $\mathsf{LAG}(\Psi, t_n, \mathcal{S})$ in terms of two terms, E_L and U_L. Letting $U(\tau, y)$ be the set of at most y tasks of highest utilization from the task set τ, and $E(\tau, y)$ be the set of at most y tasks with the highest execution costs from τ, E_L and U_L are defined as follows.

$$E_L = \sum_{T_i \in E(\tau, m-1)} e_i \quad \text{and} \quad U_L = \sum_{T_i \in U(\tau, m-1)} u_i \qquad (13)$$

Lemma 3. $\mathsf{LAG}(\Psi, t_n, \mathcal{S}) \leq E_L + x \cdot U_L$.

Proof. To bound $\mathsf{LAG}(\Psi, t_n, \mathcal{S})$, we sum individual task lags at t_n. If $t_n = 0$, then $\mathsf{LAG}(\Psi, t_n, \mathcal{S}) = 0$ and the lemma holds trivially. So assume that $t_n > 0$. Consider the set of tasks $\beta = \{T_i : \exists T_{i,j} \in \Psi \text{ such that } T_{i,j} \text{ is pending at } t_n^-\}$. Because the instant t_n^- is non-busy non-displacing, $|\beta| \leq m - 1$. If a task has no pending jobs at t_n^-, then $\mathsf{lag}(T_i, t_n, \mathcal{S}) \leq 0$. Therefore, by (4) and Lemma 2, we have

$$\begin{aligned}
&\mathsf{LAG}(\Psi, t_n, \mathcal{S}) \\
&= \sum_{T_i:\, T_{i,j} \in \Psi} \mathsf{lag}(T_i, t_n, \mathcal{S}) \\
&\leq \sum_{T_i \in \beta} \mathsf{lag}(T_i, t_n, \mathcal{S}) \\
&\leq \sum_{T_i \in \beta} x \cdot u_i + e_i \leq E_L + x \cdot U_L.
\end{aligned}$$
□

Upper bound on LAG+D. The demand placed by jobs in **rD** after t_d is clearly $\mathsf{D}(\mathbf{rD}, t_d, \mathcal{S}) \leq \sum_{T_k \in \tau_H} e_k - \delta_k$. Since the tasks in τ_H have jobs in **rD**, they have

periods greater than p_l. From Lemmas 1 and 3, we therefore have

$$\begin{aligned}
&\mathsf{LAG}(\Psi, t_d, \mathcal{S}) + \mathsf{D}(\mathbf{rD}, t_d, \mathcal{S}) \\
&\quad \leq \quad E_L + x \cdot U_L + \sum_{T_k \in \tau_H} \delta_k(1 - u_k) + (e_k - \delta_k) \\
&\quad \leq \quad E_L + x \cdot U_L + \sum_{T_h : p_k > p_\ell} e_k. \qquad (14)
\end{aligned}$$

Necessary condition for tardiness to exceed $x + e_\ell$. We now find the amount of competing work for $T_{\ell,j}$ that is necessary for $T_{\ell,j}$ to miss its deadline by more than $x + e_\ell$.

Lemma 4. *If the tardiness of $T_{\ell,j}$ exceeds $x + e_\ell$, then* $\mathsf{LAG}(\mathbf{rd}, t_d, \mathcal{S}) + \mathsf{D}(\mathbf{rD}, t_d, \mathcal{S}) > mx + e_\ell$.

Proof. We prove the contrapositive: we assume that $\mathsf{LAG}(\mathbf{rd}, t_d, \mathcal{S}) + \mathsf{D}(\mathbf{rD}, t_d, \mathcal{S}) \leq mx + e_\ell$ holds and show that the tardiness of $T_{\ell,j}$ cannot exceed $x + e_\ell$. In $\mathcal{S}$, the jobs in **Rd** and **RD** do not affect the execution of jobs in **rd** and **rD**. We can thus concentrate on the scheduling of jobs in **rd** $\cup$ **rD**. Let γ_ℓ be the amount of work $T_{\ell,j}$ performs by time t_d in $\mathcal{S}$. Let $y = x + \frac{\gamma_\ell}{m}$. We consider two cases.

Case 1. $[t_d, t_d + y)$ *is a busy interval for* **rd** $\cup$ **rD.** In this case, the work performed by the system on the jobs in **rd** $\cup$ **rD** throughout $[t_d, t_d + y)$ is $my = mx + \gamma_\ell$. Let W be the amount of work that can compete with $T_{\ell,j}$ after $t_d + y$, including the work due for $T_{\ell,j}$. Then, $W = \mathsf{LAG}(\mathbf{rd}, t_d, \mathcal{S}) + \mathsf{D}(\mathbf{rD}, t_d, \mathcal{S}) - my \leq mx + e_\ell - my = e_\ell - \gamma_\ell$. Because **rd** and **rD** are finite, $T_{\ell,j}$ eventually completes execution at some time t_f. Because FIFO is work-conserving, at least one processor is busy until $T_{\ell,j}$ completes. Thus, the amount of work performed by the system for jobs in **rd** $\cup$ **rD** during the interval $[t_d + y, t_f)$ is at least $t_f - t_d - y$. Therefore, $t_f - t_d - y \leq W$. Because $W \leq e_\ell - \gamma_\ell$, this implies that $t_f - t_d - y \leq e_\ell - \gamma_\ell$, and hence, the tardiness of $T_{\ell,j}$ is $t_f - t_d \leq y + e_\ell - \gamma_\ell = x + \frac{\gamma_\ell}{m} + e_\ell - \gamma_\ell \leq x + e_\ell$.

Case 2. $[t_d, t_d + y)$ *is non-busy for* **rd** $\cup$ **rD.** Let $t_s > 0$ be the earliest non-busy instant in $[t_d, t_d + y)$. Since FIFO is work-conserving, all time instants after t_s are non-busy for **rd** $\cup$ **rD**. The job $T_{\ell,j}$ cannot start execution before the preceding job $T_{\ell,j-1}$ (if it exists) completes. Let t_p be the finish time of the job $T_{\ell,j-1}$, if it exists, or 0 otherwise. We consider three subcases.

Subcase 1. $T_{\ell,j}$ *executes at* t_s^-**.** By Claim 1, $T_{\ell,j}$ cannot be preempted after t_s, so it finishes by time $t_s + e_\ell - \gamma_\ell < t_d + y + e_\ell - \gamma_\ell = t_d + x + \frac{\gamma_\ell}{m} - \gamma_\ell + e_\ell \leq t_d + x + e_\ell$.

Subcase 2. $T_{\ell,j}$ *does not execute at* t_s^- *and* $t_p \leq t_s$**.** The latest time $T_{\ell,j}$ can commence execution is t_s ($\gamma_\ell = 0$ holds in this case). By Claim 1, $T_{\ell,j}$ is not preempted after it commences execution, and thus finishes by time $t_s + e_\ell$. Because $t_s < t_d + y$, $t_s + e_\ell < t_d + y + e_\ell = t_d + x + e_\ell$ (since $\gamma_\ell = 0$).

Subcase 3. $T_{\ell,j}$ *does not execute at* t_s^- *and* $t_p > t_s$**.** The latest time $T_{\ell,j}$ can commence execution is t_p. By Property (P) and the definition of t_d, the finish time of $T_{\ell,j-1}$ is $t_p \leq t_d - p_\ell + x + e_\ell \leq t_d + x$. Also, $\gamma_\ell = 0$. Because $t_p > t_s \geq t_d$ and all time instants after t_s are non-busy for **rd** $\cup$ **rD**, the finish time of $T_{\ell,j}$ is at most $t_d + x + e_\ell$. □

The lemma above was proved in [6] in the context of EDF and NP-EDF. However, the proof only relies on Property (P) and the fact that jobs have job-level restricted-dynamic priorities so it holds for FIFO as well.

By Lemma 4, setting the upper bound on $\mathsf{LAG}(\Psi, t_d, \mathcal{S}) + \mathsf{D}(\mathbf{rD}, t_d, \mathcal{S})$ as implied by (14) to be at most $mx + e_\ell$ will ensure that the tardiness of $T_{\ell,j}$ is at most $x + e_\ell$. By solving for the minimum x that satisfies the resulting inequality, we obtain a value of x that is sufficient for ensuring a tardiness of at most $x + e_\ell$. The inequality is as follows.

$$x \cdot U_L + E_L + \sum_{T_h : p_k > p_\ell} e_k \leq mx + e_\ell.$$

Solving for x, we have

$$x \geq \frac{E_L + (\sum_{p_k > p_\ell} e_k) - e_\ell}{m - U_L}. \qquad (15)$$

If x equals the right-hand side of (15), then the tardiness of $T_{\ell,j}$ will not exceed $x + e_\ell$. A value for x that is independent of the parameters of T_ℓ can be obtained by replacing $\sum_{p_k > p_\ell} e_k - e_\ell$ with $max_\ell(\sum_{p_k > p_\ell} e_k - e_\ell)$ in (15).

Theorem 1. *With x as defined above, the tardiness for a task T_k scheduled under global* FIFO *is at most $x + e_k$.*

Note that, for tardiness to be bounded under FIFO, the denominator in the right-hand-side expression in (15) must not be zero. This condition is satisfied because $U_L < U_{sum}(\tau) \leq m$ by the definition of U_L in (13).

Example 5. Consider the task set from Example 1. Applying Theorem 1 to the tasks in this set, we find that the tardiness bound for each task T_k is $23.08 + e_k$. For task T_1, for example, the computed bound is 24.08.

3.3. Generalized Tardiness Bound

The proofs of the lemmas presented in the previous section in the context of FIFO scheduling depend only on Property (P), and the definition of a carry-in job (particularly, the term δ_k). In particular, these proofs do not mention the exact manner in which jobs are scheduled. As a result, the bounds presented above can be generalized to apply to a wide range of other global scheduling algorithms. Such generalizations are considered in [11].

4. Experimental Evaluation

In this section, we present an experimental evaluation of FIFO. In our experiments, we compared observed average and maximum tardiness under FIFO, EDF, and NP-EDF, as computed from actual schedules, to their respective maximum tardiness bounds, as established in this paper and in [6]. As explained below, each task set had a maximum per-task execution cost of $e_{max} = 10$. The task sets were classified based on their maximum per-task utilization, u_{max}. In inset (a) of Fig. 5, both observed maximum tardiness values and those computed from the respective bounds are plotted versus $u_{max} \in \{0.05, 0.1, 0.3\}$ for each scheme, for $m = 4$ and $e_{max} = 10$. Inset (b) of Fig. 5 is similar except that observed average tardiness is plotted instead of observed maximum tardiness. Also, for clarity, in each inset, the bars are listed in the inset's legend in the same order as they appear in the graph from left to right.

For each value of u_{max} in each graph, 50 task sets were generated. Each such task set was generated by creating tasks with utilizations taken randomly from $[0, u_{max}]$ until total utilization exceeded four, and by then reducing the last task's utilization so that four processors were exactly required. The execution cost of the first task was then taken to be $e_{max} = 10$ and subsequent task execution costs were taken randomly from $[0, e_{max}]$. To determine observed tardiness values (average and maximum), a simulator was used for each scheduling scheme that schedules tasks in a synchronous, periodic fashion until time 20,000. In this set of experiments, costs due to scheduling and system overheads were not considered. (Since EDF preempts jobs more frequently, the performance of EDF would worsen, comparatively, if overheads were considered. In addition, scheduling costs under FIFO should be lower, because FIFO-ordered priority queues can be accessed in constant time, whereas deadline-ordered priority queues have linear or logarithmic access costs, depending on the implementation.)

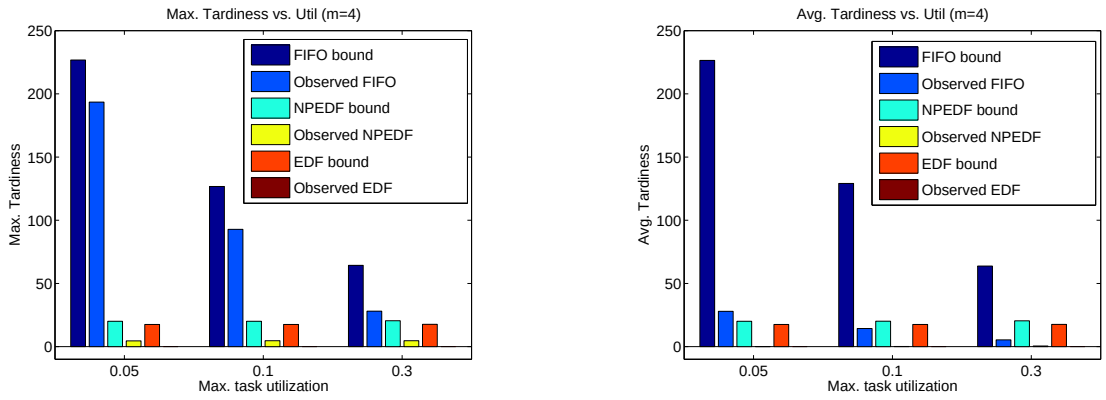
As seen in Fig.5, observed maximum tardiness under FIFO is fairly close to that given by the bound computed in this paper. Furthermore, maximum tardiness under FIFO (either observed or via the bound) is significantly higher than that under the two EDF schemes. However, maximum tardiness under FIFO improves relative to the other two schemes as u_{max} increases. These trends are a consequence of the fact that, under FIFO, tasks with later deadlines can postpone the execution of tasks with smaller deadlines, causing high tardiness values. The term $max(\sum_{p_k > p_l} e_k - e_l)$ in (15) describes this effect. When u_{max} is small, relative deadlines tend to be larger, which increases the likelihood of having high maximum tardiness.

In contrast to the maximum values considered above, when average observed tardiness is considered, as plotted in inset (b) of Fig.5, a different story emerges. As seen, average tardiness under FIFO is quite competitive with that under the two EDF schemes. We conclude that FIFO is a reasonable scheduling option to consider for applications in which good average-case tardiness is required and somewhat higher maximum tardiness is tolerable, as long as it is bounded. We remind the reader that, on some development platforms, FIFO might be the only option, as other real-time scheduling approaches are not as widely supported.

5. Conclusion

We have shown that the global FIFO scheduling algorithm ensures bounded task tardiness when scheduling sporadic soft real-time task systems on a symmetric multiprocessor platform. To our knowledge, this paper is the first attempt to establish soft real-time constraints for tasks scheduled in a FIFO manner (on multiprocessors or uniprocessors).

Several interesting avenues for further work exist. They include probabilistic analysis of task tardiness (that is, *analytically* determining average tardiness), considering multi-speed multicore architectures, and extending the results to include tasks with synchronization requirements. It would also be interesting to consider workloads that consist of a mix of soft real-time and non-real-time tasks.



(a) (b)

Figure 5. Tardiness bounds and (a) observed maximum tardiness and (b) observed average tardiness for EDF, NP-EDF, and FIFO vs. maximum per-task utilization.

References

[1] M. Cesati and D. Bovet. *Understanding the Linux Kernel, Second Edition*. O'REILLY, 2002.

[2] J. Calandrino, D. Baumberger, T. Li, S. Hahn, and J. Anderson. Soft real-time scheduling on performance asymmetric multicore platforms. In *Proc. of the 13th IEEE Real-Time and Embedded Technology and Applications Symp.*, pp. 101-110, Apr. 2007.

[3] J. Calandrino, H. Leontyev, A. Block, U. Devi, and J. Anderson. LITMUS$^{\mathrm{RT}}$: A testbed for empirically comparing real-time multiprocessor schedulers. In *Proc. of the 27th IEEE Real-Time Systems Symp.*, pages 111–123, Dec. 2006.

[4] J. Lehoczky, D. Cornhill, and L. Sha. Limitations of ada for real-time scheduling. In *ACM SIGAda Ada Letters, Proc. of the 1st International Workshop on Real-Time Ada Issues*, Vol. 8. ACM Press, Oct. 1987.

[5] U. Devi. *Soft Real-Time Scheduling on Multiprocessors*. PhD thesis, Oct. 2006, http://www.cs.unc.edu/~anderson/diss/devidiss.pdf.

[6] U. Devi and J. Anderson. Tardiness bounds for global EDF scheduling on a multiprocessor. In *Proc. of the 26th IEEE Real-Time Systems Symp.*, pages 330–341, Dec. 2005.

[7] U. Devi and J. Anderson. Flexible tardiness bounds for sporadic real-time task systems on multiprocessors. In *Proc. of the 20th IEEE International Parallel and Distributed Processing Symp.*, April 2006 (on CD ROM).

[8] J. Carpenter and S. Funk and P. Holman and A. Srinivasan and J. H. Anderson and S. Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms. In Joseph Y. Leung, editor, *Handbook on Scheduling Algorithms, Methods, and Models*, pages 30.1–30.19. Chapman Hall/CRC, Boca Raton, Florida, 2004.

[9] L. George and P. Minet. A FIFO worst case analysis for a hard real-time distributed problem with consistency constraints. In *Proc. of the 17th International Conference on Distributed Computing Systems*, pages 441–448, May 1997.

[10] J. Lehoczky. Scheduling communication networks carrying real-time traffic. In *Proc. of the 19th IEEE Real-Time Systems Symp.*, pages 470–479, Dec. 1998.

[11] H. Leontyev and J. H. Anderson. Tardiness bounds for generalized priority scheduling. Unpublished manuscript, 2007.

[12] A. Srinivasan and J. Anderson. Fair scheduling of dynamic task systems on multiprocessors. *Journal of Systems and Software*, 77(1):67–80, April 2005.

[13] P. Valente and G. Lipari. An upper bound to the lateness of soft real-time tasks scheduled by EDF on multiprocessors. In *Proc. of the 26th IEEE Real-Time Systems Symp.*, pages 311–320, Dec. 2005.