# Supporting Pipelines in Soft Real-Time Multiprocessor Systems*

Cong Liu and James H. Anderson
Department of Computer Science, University of North Carolina at Chapel Hill

## Abstract

*In work on multiprocessor real-time systems, processing pipelines have received little attention. In this paper, soft real-time periodic task systems are considered that include such pipelines. Conditions are presented for guaranteeing bounded deadline tardiness in such systems under global EDF or FIFO multiprocessor scheduling.*

## 1 Introduction

With the advent of multicore technologies, it is important that real-time scheduling theory be extended so that commonly-used multiprocessor programming techniques can be supported in systems with timing constraints. One technique of particular utility is pipelined execution, which is used to increase throughput by leveraging the parallelism inherent on multiprocessor platforms. In this paper, we consider the problem of supporting such pipelines on a multiprocessor where the workload is specified as a periodic task system with implicit deadlines (relative deadlines equal periods). If all deadlines in such a task system are considered to be hard, then pipelines can be easily supported by assigning a common period to all tasks in a pipeline and by adjusting job releases so that successive pipeline stages execute in sequence. Fig. 1 shows an example, where an ordinary periodic task $T_1$ executes on a two-processor system with three other tasks, $T_2^1$, $T_2^2$, and $T_2^3$, which form a three-stage pipeline (the *kth* job of $T_2^1$, $T_2^2$, and, $T_2^3$, respectively, must execute in sequence). As seen in this example, as long as no deadlines are missed, the timing guarantees provided by the periodic model ensure that any pipeline executes correctly.

Unfortunately, for multiprocessor systems, if all deadlines must be viewed as hard, then significant processing capacity must be sacrificed, due to either inherent schedulability-related utilization loss—which is unavoidable under most scheduling schemes [1]—or high runtime overheads—which typically arise in optimal schemes that avoid schedulability-related loss [2, 3]. In systems where less stringent notions of real-time correctness suffice, such capacity loss can be avoided by viewing deadlines as soft. This follows from recent work on global scheduling algorithms that has shown that bounded deadline tardiness can be ensured with no utilization loss under a wide variety of such algorithms, including algorithms that are less costly to implement than optimal algorithms [4, 5]. Unfortunately, if deadlines can be missed, then pipelines are not as easy to support. For example, if the first job of $T_2^1$ in Fig. 1 were to miss its deadline, then its execution might overlap that of the first job of $T_2^2$. This violates the requirement that successive pipeline stages must execute in sequence.

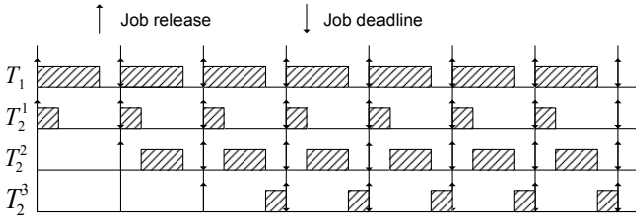


**Figure 1. Example task system.**

Motivated by the observations above, we consider in this paper the problem of supporting pipelined execution on a multiprocessor platform in systems where bounded deadline tardiness is acceptable. Our goal is to determine whether such a system can be scheduled without significant utilization loss. We focus specifically on two global scheduling algorithms that are capable of ensuring bounded deadline tardiness for ordinary periodic task systems (i.e., systems without pipelined tasks) with no utilization loss [4, 5], namely, the *global earliest-deadline-first* (GEDF) algorithm and the *global first-in first-out* (GFIFO) algorithm. We wish to know whether these algorithms can also ensure bounded deadline tardiness if pipelined tasks are present with no utilization loss; if this is not always possible, then we would like to know the conditions under which bounded tardiness can be guaranteed.

**Related work.** To our knowledge, pipelined execution under global scheduling algorithms has not been considered before. However, distributed systems (which must be scheduled by partitioning approaches) have been considered. For example, Jayachandran and Abedelzaher have presented delay composition rules that provide a bound on the end-to-end delay of jobs in partitioned distributed systems that include pipelines [6] or more general (acyclic) precedence constraints [7]. These rules permit a pipelined system to be transformed so that uniprocessor schedulability analysis can be applied.

*Work supported by IBM, Intel, and Sun Corps.; NSF grants CNS 0834270, CNS 0834132, and CNS 0615197; and ARO grant W911NF-06-1-0425.

Several off-line algorithms have also been proposed for scheduling tasks with precedence constraints in distributed real-time systems comprised of periodic tasks [7,8,9]. Schedulability test for pipelined distributed systems have also been proposed in which end-to-end deadlines are supported by deriving deadlines for individual pipeline stages [8, 9].

**Contributions.** We show that the ability to support pipelines with bounded deadline tardiness hinges upon a pipeline parameter that we call "stretch," with range $[0, 1]$. We present a general tardiness bound, which is applicable to either GEDF or GFIFO, that expresses tardiness as a function of stretch and other parameters. This bound shows that pipelines can be supported with bounded tardiness and no utilization loss if each pipelined task's stretch is less than $1 - \frac{U}{m}$, where $m$ is the number of processors and $U$ is sum of the $m(m-1)$ largest subtask utilizations, where by "subtask," we mean a task corresponding to one pipeline stage. In other cases, utilization loss is inherent. We prove this by presenting a counterexample with unbounded tardiness in which two two-stage pipelines execute on three processors.

**Organization.** The remainder of this paper is organized as follows. Sec. 2 describes the system model. In Sec. 3, the tardiness bound derivation that is our main result is presented. Sec. 4 concludes.

## 2 System Model

We consider the problem of scheduling a set $\tau = \{T_1, ..., T_n\}$ of $n$ independent periodic pipeline tasks on $m \geq 2$ identical processors. An $h$-stage pipeline task $T_l$, where $1 \leq h \leq m$, consists of $h$ subtasks, $T_l^1, ...T_l^h$. (If $h = 1$, then $T_l$ is an ordinary periodic task.) Each subtask is released repeatedly, with each such invocation called a *job*. We assume that each job of $T_l^h$ executes for *exactly* $e_l^h$ time units. This assumption can be eased to treat $e_l^h$ as an upper bound, at the expense of more cumbersome notation. For the class of scheduling algorithms we consider, reducing a job's execution cost cannot increase any job's tardiness. The $j^{th}$ job of $T_l^h$, denoted $T_{l,j}^h$, is released at time $r_{l,j}^h$ and has a deadline at time $d_{l,j}^h$. Associated with each pipeline task $T_l$ is a period $p_l$, which specifies both the time between two consecutive job releases of any subtask of $T_l$ and the relative deadline of each such job (i.e., $d_{l,j}^h = r_{l,j}^h + p_l$). The utilization of a subtask $T_l^h$ is defined as $u_l^h = e_l^h/p_l$, and the utilization of the task system $\tau$ as $U_{sum} = \sum_{T_i \in \tau} \sum_{T_i^j \in T_i} u_i^j$.

To ease the reasoning in our tardiness-bound derivation, we assume that each subtask $T_l^h$ releases a job every $p_l$ time units, starting at time 0. Note that this is different from the release pattern suggested by Fig. 1. This can be seen in

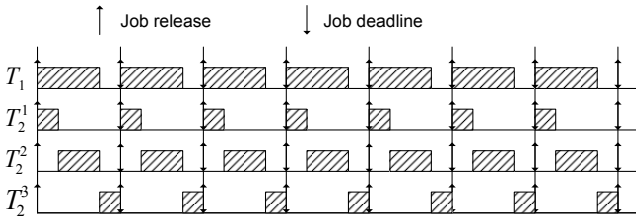


**Figure 2. Modified example task system.**

Fig. 2, which is a modification of Fig. 1, in which some "extra" initial jobs have been introduced, so that each subtask starts at time 0. Note that removing these "extra" jobs cannot increase any job's tardiness. Thus, it suffices to bound tardiness assuming each subtask begins at time 0.

Successive jobs of the same subtask are required to execute in sequence. Also, for $j, h > 1$, job $T_{l,j}^h$ cannot commence execution until job $T_{l,j-1}^{h-1}$ completes. To avoid confusion when discussing these precedence constraints, we will refer to $T_{l,j-1}^h$ as the *L-predecessor* of $T_{l,j}^h$ (assuming $j > 1$), and $T_{l,j-1}^{h-1}$ as the *U-predecessor* of $T_{l,j}^h$ (assuming $j, h > 1$). ("L" and "U" stand for "left" and "upper", respectively). For example, in Fig. 2, $T_{2,1}^1$ is the L-predecessor of $T_{2,2}^1$ and the U-predecessor of $T_{2,2}^2$.

If a job $T_{i,j}^k$ completes at time $t$, then its tardiness is defined as $max(0, t - d_{i,j}^k)$. A pipeline task's tardiness is the maximum of the tardiness of any job of any of its subtasks. We require $u_i^k \leq 1$ and $U_{sum} \leq m$; otherwise, tardiness can grow unboundedly. Note that, when a job of a subtask misses its deadline, the release time of the next job of that task is not altered. Despite this, it is still required that a job cannot execute in parallel with either of its predecessors.

Under GEDF (GFIFO), released jobs are prioritized by their deadlines (release times). So that our results can be applied to both algorithms, we consider a generic scheduling algorithm (GSA) where each job is prioritized by some time point between its release time and deadline. For any job $T_{i,k}^w$, its prioritization function, $\rho_{i,k}^w$, is defined as: $\rho_{i,k}^w = r_{i,k}^w + \kappa \cdot p_i$, where $0 \leq \kappa \leq 1$. We assume that for subtasks belonging to the same task, ties are broken in favor of earlier stages, and any remaining ties are broken by task ID. Note that GEDF and GFIFO are special cases of GSA where $\kappa$ is set to 1 and 0, respectively.

## 3 A Tardiness Bound for GSA

We derive a tardiness bound for GSA by comparing the allocations to a pipelined task system $\tau$ in a processor sharing (PS) schedule and an actual GSA schedule of interest for $\tau$, both on $m$ processors, and quantifying the difference between the two. We analyze task allocations on a per-subtask basis. According to our system model, each subtask can be treated as a periodic task.

The time interval $[t_1, t_2)$, where $t_2 > t_1$, consists of all time instances $t$, where $t_1 \leq t < t_2$, and is of length $t_2 - t_1$. For any time $t > 0$, the notation $t^-$ is used to denote the time $t - \varepsilon$ in the limit $\varepsilon \to 0+$, and the notation $t^+$ is used to denote the time $t + \varepsilon$ in the limit $\varepsilon \to 0+$

**Definition 1.** A subtask $T_i^w$ is *active* at time $t$ if there exists a job $T_{i,v}^w$ such that $r_{i,v}^w \leq t < d_{i,v}^w$. By our task model, every subtask has one active job at any time.

**Definition 2.** Job $T_{i,v}^w$ is *pending* at time $t$ if $r_{i,v}^w \leq t$ and $T_{i,v}^w$ has not completed execution by $t$.

**Definition 3.** Job $T_{i,v}^w$ is *ready* at $t$ if $t \geq r_{i,v}^w$ and its L-predecessor (if any) has completed execution at $t$. $T_{i,v}^w$ is *enabled* if it has been released and both its U-predecessor (if any) and L-predecessor (if any) have completed.

Let $A(T_{i,j}^k, t_1, t_2, S)$ denote the total allocation to the job $T_{i,j}^k$ in an arbitrary schedule $S$ in $[t_1, t_2)$. Then, the total time allocated to all jobs of $T_i^k$ in $[t_1, t_2)$ in $S$ is given by

$$A(T_i^k, t_1, t_2, S) = \sum_{j \geq 1} A(T_{i,j}^k, t_1, t_2, S).$$

Consider a PS schedule *PS*. In such a schedule, $T_i^k$ executes with the rate $u_i^k$ at each instant when it is active in $[t_1, t_2)$. Thus,

$$A(T_{i,j}^k, t_1, t_2, PS) = (t_2 - t_1)u_i^k. \quad (1)$$

The difference between the allocation to a job $T_{i,j}^k$ up to time $t$ in a PS schedule and an arbitrary schedule $S$, denoted *the lag of job $T_{i,j}^k$ at time $t$ in schedule $S$*, is defined by

$$\begin{aligned} & lag(T_i^k, t, S) = \textstyle\sum_{j \geq 1} lag(T_{i,j}^k, t, S) \\ = \quad & A(T_{i,j}^k, 0, t, PS) - A(T_{i,j}^k, 0, t, S). \end{aligned} \quad (2)$$

The concept of lag is important because, if it can be shown that lags remain bounded, then tardiness is bounded as well. The *LAG* for a finite job set $J$ at time $t$ in the schedule $S$ is defined by

$$\begin{aligned} & LAG(J, t, S) = \textstyle\sum_{T_{i,j}^k \in J} lag(T_{i,j}^k, t, S) \\ = \quad & \textstyle\sum_{T_{i,j}^k \in J} (A(T_{i,j}^k, 0, t, PS) - A(T_{i,j}^k, 0, t, S)). \end{aligned} \quad (3)$$

Our tardiness-bound derivation focuses on a given task system $\tau$. We order the jobs in $\tau$ based on their priorities: $T_{i,v}^w \prec T_{a,b}^c$ iff $\rho_{i,v}^w < \rho_{a,b}^c$ or $(\rho_{i,v}^w = \rho_{a,b}^c) \wedge (i = a) \wedge (w < c)$ or $(\rho_{i,v}^w = \rho_{a,b}^c) \wedge (i < a)$. Let $T_{l,j}^h$ be a job of a subtask $T_l^h$ in $\tau$, $t_d = d_{l,j}^h$, and $S$ be a GSA schedule for $\tau$ with the following property.

**(P)** The tardiness of every job $T_{i,k}^w$ such that $T_{i,k}^w \prec T_{l,j}^h$ is at most $x + e_i^w$ in $S$, where $x \geq 0$.

Our objective is to determine the smallest $x$ such that the tardiness of $T_{l,j}^h$ is at most $x + e_l^h$. This would by induction imply a tardiness of at most $x + e_i^k$ for all jobs of every subtask $T_i^k$ of $T_i \in \tau$. We assume that $T_{l,j}^h$ finishes after $t_d$, for otherwise, its tardiness is trivially zero. The steps for determining the value for $x$ are as follows.

1. Determine an upper bound on the work pending for tasks in $\tau$ that can compete with $T_{l,j}^h$ after $t_d$. This is dealt with in Lemmas 1, 2, and 3 in Sec. 3.1.

2. Determine a lower bound on the amount of work pending for tasks in $\tau$ that can compete with $T_{l,j}^h$ after $t_d$, required for the tardiness of $T_{l,j}^h$ to exceed $x + e_l^h$. This is dealt with in Lemma 4 in Sec. 3.2.

3. Determine the smallest $x$ such that the tardiness of $T_{l,j}^h$ is at most $x + e_l^h$, using the above upper and lower bounds.

**Definition 4.** We categorize jobs based on the relationship between their priorities and deadlines and those of $T_{l,j}^h$:

$$\mathbf{d} = \{T_{i,v}^w : (T_{i,v}^w \preceq T_{l,j}^h) \wedge (d_{i,v}^w \leq t_d)\}$$

$$\mathbf{D} = \{T_{i,v}^w : (T_{i,v}^w \prec T_{l,j}^h) \wedge (d_{i,v}^w > t_d)\}.$$

**d** is the set of jobs with deadlines at most $t_d$ with priority at least that of $T_{l,j}^h$. These jobs do not execute beyond $t_d$ in the PS schedule. Note that $T_{l,j}^h$ is in **d**. **D** is the set of jobs that have higher priorities than $T_{l,j}^h$ and deadlines greater than $t_d$. Let $D_H$ be the set of subtasks with jobs in **D**. **D** consists of *carry-in jobs*, which have a release time before $t_d$ and a deadline after $t_d$. Exactly one such job exists for each subtask in $D_H$. Note that **D** is empty under GEDF because jobs with later deadlines have lower priorities.

**Definition 5.** A time interval $[t_1, t_2)$ is defined to be *busy* for any job set $\theta$ if all $m$ processors are executing some job in $\theta$ at each instant in the interval. An interval $[t_1, t_2)$ that is not busy for $\theta$ is defined to be *non-busy* for $\theta$.

The following claim follows from the definition of $LAG$.

**Claim 1.** *If $LAG(\mathbf{d}, t_2, S) > LAG(\mathbf{d}, t_1, S)$, then $[t_1, t_2)$ is non-busy for **d**. In other words, LAG for **d** can increase only throughout a non-busy interval.*

An interval could be non-busy for **d** for two reasons:

1. There are not enough enabled jobs in **d** to occupy all available processors. Such an interval is called *non-busy non-displacing*.

2. Jobs in **D** occupy one or more processors and there are enabled jobs in **d**. Such an interval is called *non-busy displacing*.

**Definition 6.** Let $\delta_k^w$ be the amount of work performed by a carry-in job $T_{k,v}^w$ by time $t_d$.

**Definition 7.** Let $B(\mathbf{D}, t_d, S)$ be the amount of work due to jobs in $\mathbf{D}$ that can compete with $T_{l,j}^h$ after $t_d$.

**Definition 8.** If $T_{i,v}^w$ first starts execution at time $t$, then $t$ is called its *start time*, denoted $S(T_{i,v}^w)$. If $T_{i,v}^w$ completes execution at time $t'$, then $t'$ is called its *finish time*, denoted $F(T_{i,v}^w)$.

**Definition 9.** Let $\max_k = \max\{j \mid 1 \leq j \leq k \leq m \wedge (\forall w : 1 \leq w \leq k : e_i^j \geq e_i^w)\}$. That is, subtask $T_i^{\max_k}$ (where $\max_k \leq k$) has the maximum execution cost among the subtasks $\{T_i^1, T_i^2, ..., T_i^k\}$.

**Definition 10.** Let $s_i^w = \dfrac{e_i^{max_w} - e_i^w}{e_i^{max_w}}$. $s_i^w$ is called the *subtask stretch* of $T_i^w$. Let $s_i = \max\{s_i^1, s_i^2, ..., s_i^m\}$. $s_i$ is called the *task stretch* of $T_i$. Let $s_{max} = \max\{s_1, s_2, ..., s_n\}$. $s_i$ is called the *maximum stretch*.

Since $\mathbf{d} \cup \mathbf{D}$ includes all jobs of higher priority than $T_{l,j}^h$, the competing work for $T_{l,j}^h$ is given by the sum of **(i)** the amount of work pending at $t_d$ for jobs in $\mathbf{d}$, and **(ii)** the amount of work $B(\mathbf{D}, t_d, S)$ demanded by jobs in $\mathbf{D}$ that competes with $T_{l,j}^h$ after $t_d$. Since jobs from $\mathbf{d}$ have deadlines at most $t_d$, they do not execute in the PS schedule beyond $t_d$. Thus, the work pending for them is given by $LAG(\mathbf{d}, t_d, S)$. Therefore, the competing work for $T_{l,j}^h$ after $t_d$ is given by $LAG(\mathbf{d}, t_d, S) + B(\mathbf{D}, t_d, S)$. Let

$$Z = LAG(\mathbf{d}, t_d, S) + B(\mathbf{D}, t_d, S). \tag{4}$$

Note that jobs not in $\mathbf{d} \cup \mathbf{D}$ have lower priority than those in $\mathbf{d} \cup \mathbf{D}$ and thus do not affect the scheduling of jobs in $\mathbf{d} \cup \mathbf{D}$. For simplicity, we will henceforth assume that no job not in $\mathbf{d} \cup \mathbf{D}$ executes beyond $t_d$.

### 3.1 Upper Bound

In this section, we determine an upper bound on $Z$ in terms of the parameters of the tasks in $\tau$.

**Definition 11.** Let $t_n$ be the end of the latest non-busy non-displacing interval for $\mathbf{d}$ before $t_d$, if any; otherwise, $t_n = 0$.

The following two lemmas have been proved previously for both GEDF [4] and GFIFO [5]. Their proofs depend only on Property (P) and are not affected by precedence constraints, so they also hold for pipelined task systems. For completeness, proofs in our framework are given in an appendix.

**Lemma 1.** $LAG(\boldsymbol{d}, t_d, S) \leq LAG(\boldsymbol{d}, t_n, S) + \sum_{T_k^w \in D_H} \delta_k^w (1 - u_k^w)$, *where* $t \in [0, t_d]$.

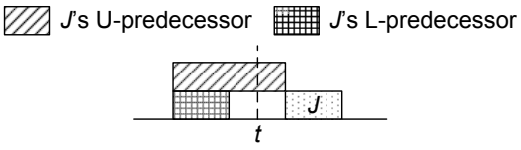


**Figure 3. Job $J$ is blocked at time $t$.**

**Lemma 2.** $lag(T_k^w, t, S) \leq u_k^w \cdot x + e_k^w$ *for any subtask* $T_k^w$ *and* $t \in [0, t_d]$.

Lemma 3 below upper bounds $LAG(\mathbf{d}, t_n, S)$.

**Definition 12.** Let $U(\tau, y)$ ($E(\tau, y)$) be the set of $\min(y, b)$ subtasks of highest utilization (execution cost) in $\tau$, where $b$ is the number of subtasks in $\tau$. Define $U$ and $\Gamma$ as follows.

$$U = \textstyle\sum_{T_i^w \in U(\tau, m(m-1))} u_i^w$$
$$\Gamma = \textstyle\sum_{T_i^w \in E(\tau, m(m-1))} e_i^w$$

**Definition 13.** If a released job's L-predecessor has completed, but its U-predecessor has not, then it is said to be *blocked*. Note that the first job of any subtask cannot be blocked because it has no L-predecessor or U-predecessor. Blocking is illustrated in Fig. 3.

**Lemma 3.** $LAG(\boldsymbol{d}, t_n, S) \leq U \cdot x + \Gamma$.

*Proof.* By summing individual subtask lags at $t_n$, we can bound $LAG(\mathbf{d}, t_n, S)$. If $t_n = 0$, then $LAG(\mathbf{d}, t_n, S) = 0$, so assume that $t_n > 0$. Consider the set of subtasks $\beta = \{T_i^w : \exists T_{i,v}^w \text{ in } \mathbf{d} \text{ such that } T_{i,v}^w \text{ is pending at } t_n^-\}$. Given that the instant $t_n^-$ is non-busy non-displacing, at most $m-1$ subtasks in $\beta$ have jobs executing at $t_n^-$. Due to the existence of blocked jobs, as defined in Def. 13, $\beta$ may contain more than $m-1$ subtasks. In the worst case, a subtask in $\beta$ that is executing at $t_n^-$ could be a first-stage subtask, $T_i^1$, and it may cause jobs of subtasks in $\{T_i^2, T_i^3, ..., T_i^m\}$ to be blocked at $t_n^-$ so that all $m$ subtasks of the corresponding pipeline task have pending jobs at $t_n^-$. Since there are at most $m-1$ subtasks in $\beta$ that are executing at $t_n^-$, $|\beta| \leq m(m-1)$. If subtask $T_i^w$ does not have pending jobs at $t_n^-$, then $lag(T_i^w, t_n, S) \leq 0$. Therefore, by (3), we have

$$\begin{aligned}
LAG(\mathbf{d}, t_n, S) &= \sum_{T_i^w : T_{i,v}^w \in \mathbf{d}} lag(T_i^w, t_n, S) \\
&\leq \sum_{T_i^w \in \beta} lag(T_i^w, t_n, S) \\
&\{\text{by Lemma 2}\} \\
&\leq \sum_{T_i^w \in \beta} (u_i^w \cdot x + e_i^w) \\
&\{\text{because } |\beta| \leq \mathrm{m(m-1)}\} \\
&\leq U \cdot x + \Gamma. \qquad \square
\end{aligned}$$

The demand placed by jobs in $\mathbf{D}$ after $t_d$ is $B(\mathbf{D}, t_d, S) = \sum_{T_i^w \in D_H}(e_i^w - \delta_i^w)$. Thus, by (4) and Lemmas 1 and 3, we have the following upper bound:

$$\begin{aligned} Z &\leq U \cdot x + \Gamma + \sum_{T_i^w \in D_H} (\delta_i^w(1-u_i^w) + (e_i^w - \delta_i^w)) \\ &\leq U \cdot x + \Gamma + \sum_{T_i \in \tau} \sum_{T_i^w \in T_i} e_i^w. \end{aligned} \tag{5}$$

### 3.2 Lower Bound

In the following lemma, we determine a lower bound on $Z$ that is necessary for the tardiness of $T_{l,j}^h$ to exceed $x + e_l^h$. Let $e_{max}$ be the maximum execution time among all subtasks.

**Lemma 4.** *If the tardiness of $T_{l,j}^h$ exceeds $x + e_l^h$, then $Z > (1 - s_{max}) \cdot mx - (m-1)e_l^h - me_{max}$.*

*Proof.* We prove the contrapositive: we assume that

$$Z \leq (1 - s_{max}) \cdot mx - (m-1)e_l^h - me_{max} \tag{6}$$

holds and show that the tardiness of $T_{l,j}^h$ cannot exceed $x + e_l^h$. Let $\eta_l^h$ be the amount of work $T_{l,j}^h$ performs by time $t_d$ in $S$. Define $y$ as follows.

$$y = (1 - s_{max}) \cdot x + \frac{\eta_l^h}{m} \tag{7}$$

Let $W$ be the amount of work due to jobs in $\mathbf{d} \cup \mathbf{D}$ that can compete with $T_{l,j}^h$ after $t_d + y$, including the work due for $T_{l,j}^h$. Let $t_f = F(T_{l,j}^h)$. We consider two cases.

**Case 1.** *$[t_d, t_d + y)$ is a busy interval for $\mathbf{d} \cup \mathbf{D}$.* In this case, by (6) and (7), $W = Z - my \leq (1 - s_{max}) \cdot mx - (m-1)e_l^h - me_{max} - my = (1 - s_{max}) \cdot mx - (m-1)e_l^h - me_{max} - (1 - s_{max}) \cdot mx - \eta_l^h = -(m-1)e_l^h - me_{max} - \eta_l^h < 0$. Because GSA is work-conserving (i.e., GSA idles a processor only when there is no enabled job), at least one processor is busy until $T_{l,j}^h$ completes. Thus, the amount of work performed by the system for jobs in $\mathbf{d} \cup \mathbf{D}$ during the interval $[t_d + y, t_f)$ is at least $t_f - t_d - y$. Hence, $t_f - t_d - y \leq W < 0$. Therefore, the tardiness of $T_{l,j}^h$ is $t_f - t_d < y = (1 - s_{max}) \cdot x + \frac{\eta_l^h}{m} \leq x + e_l^h$.

**Case 2.** *$[t_d, t_d + y)$ is a non-busy interval for $\mathbf{d} \cup \mathbf{D}$.* Let $t_s \geq t_d$ be the earliest non-busy instant in $[t_d, t_d + y)$. Job $T_{l,j}^h$ cannot start execution before its predecessors complete. Let $t_p$ be the latest finish time of $T_{l,j}^h$'s predecessors. We consider three subcases.

**Subcase 2.1.** *$t_p \leq t_s$ and $T_{l,j}^h$ is not preempted before its completion.* In this case, $T_{l,j}^h$ can start execution at $t_s$ because $t_s$ is non-busy. Thus, because $t_s < t_d + y$, by (7), $T_{l,j}^h$ finishes by time $t_s + e_l^h - \eta_l^h < t_d + y + e_l^h - \eta_l^h = t_d + (1 - s_{max}) \cdot x + \frac{\eta_l^h}{m} + e_l^h - \eta_l^h \leq t_d + x + e_l^h$.

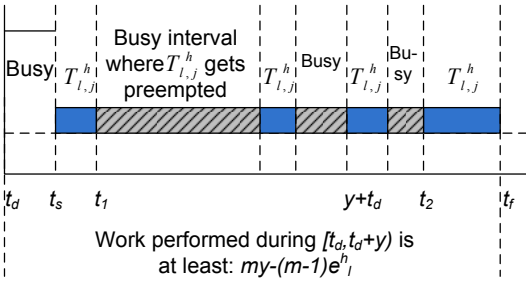


**Figure 4. Subcase 2.2**

**Subcase 2.2** *$t_p \leq t_s$ and $T_{l,j}^h$ is preempted before its completion.* Let $t_1 > t_s$ be the earliest time when $T_{l,j}^h$ is preempted. As shown in Fig. 4, by the definition of $t_s$ and $t_1$, $T_{l,j}^h$ executes continuously within $[t_s, t_1)$. Because $T_{l,j}^h$ is preempted at $t_1$, $t_1$ is busy with respect to $\mathbf{d} \cup \mathbf{D}$. Let $t_2$ be the last time $T_{l,j}^h$ resumes execution after being preempted. (Since a finite number of jobs have higher priority than $T_{l,j}^h$, $t_2$ exists.) Within $[t_1, t_2)$, $T_{l,j}^h$ could be preempted multiple times. All such intervals during which $T_{l,j}^h$ is preempted must be busy in order for the preemption to happen. Given that $t_f \leq t_2 + e_l^h - \eta_l^h$, if $t_2 \leq y + t_d$, then $t_f \leq y + t_d + e_l^h - \eta_l^h$, in which case, by (7), the tardiness of $T_{l,j}^h$ is $t_f - t_d \leq y + e_l^h - \eta_l^h \leq (1 - s_{max}) \cdot x + e_l^h \leq x + e_l^h$, as required. If $t_2 > t_d + y$, then the amount of work due to $\mathbf{d} \cup \mathbf{D}$ performed within $[t_d, t_d + y)$ is at least $my - (m-1) \cdot \min(e_l^h, y)$ because all intervals during which $T_{l,j}^h$ is preempted are busy, and $T_{l,j}^h$ can execute for at most $e_l^h$ time in $[t_d, y + t_d)$. (Within such intervals, at least one processor is occupied by $T_{l,j}^h$.) Thus, the amount of work that can compete with $T_{l,j}^h$ after $t_d + y$ is

$$\begin{aligned} W &\leq Z - (my - (m-1) \cdot \min(e_l^h, y)) \\ &\quad \{\text{by (6)}\} \\ &\leq (1 - s_{max}) \cdot mx - (m-1)e_l^h - me_{max} \\ &\quad -(my - (m-1) \cdot \min(e_l^h, y)) \\ &\leq (1 - s_{max}) \cdot mx - me_{max} - my \\ &\quad \{\text{by (7)}\} \\ &= -me_{max} - \eta_l^h \\ &< 0. \end{aligned}$$

Therefore, the tardiness of $T_{l,j}^h$ is $t_f - t_d \leq y + W < y = (1 - s_{max}) \cdot x + \frac{\eta_l^h}{m} \leq x + e_l^h$.

**Subcase 2.3**: *$t_p > t_s$.* The earliest time $T_{l,j}^h$ can commence execution is $t_p$, as shown in Fig. 5. If fewer than $m$ subtasks have ready jobs in $\mathbf{d} \cup \mathbf{D}$ at any time instant within $[t_s, t_p)$, then $T_{l,j}^h$ will start execution at $t_p$ and finish

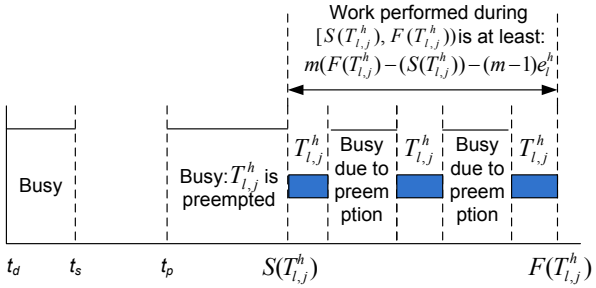


**Figure 5. Subcase 2.3**

at $t_p + e_l^h$. (Note that the number of ready jobs in $\mathbf{d} \cup \mathbf{D}$ after $t_d$ is non-increasing as time increases.) According to Property (P), $t_p \leq t_d - p_l + x + max\{e_l^h, e_l^{h-1}\} \leq t_d + x$. Thus, the tardiness of $T_{l,j}^h$ is $t_f - t_d = t_p + e_l^h - t_d \leq x + e_l^h$.

The remaining possibility (which requires a much lengthier argument) is: $t_p > t_s$ and at least $m$ subtasks have ready jobs in $\mathbf{d} \cup \mathbf{D}$ at each time instant within $[t_s, t_p)$. Our initial goal in this case is to prove that the amount of work performed within $[t_s, t_p)$ due to $\mathbf{d} \cup \mathbf{D}$ is at least $(1 - s_{max}) \cdot m(t_p - t_s) - me_{max}$ (see Claim 3 below).

In this case, given that at least $m$ subtasks have ready jobs in $\mathbf{d} \cup \mathbf{D}$ at $t_s$, $t_s$ is non-busy due to the existence of blocked jobs, as defined in Def. 13.

**Definition 14.** If $T_{i,v}^w$ is enabled at time $t$ but does not execute at $t$, then it is *preempted* at $t$. The total time for which $T_{i,v}^w$ is preempted is called its *preemption time*.

**Definition 15.** We define the set of *U-jobs* of any job $T_{i,v}^w$ to be $T_{i,v-1}^{w-1}, T_{i,v-2}^{w-2}, ..., T_{i,v-w+1}^1$, if $v \geq w$. Otherwise, if $w > v$, then the set of U-jobs of $T_{i,v}^w$ is $T_{i,v-1}^{w-1}, T_{i,v-2}^{w-2}, ..., T_{i,1}^{w-v+1}$.

A U-job of $T_{i,v}^w$ is a job of the same pipeline task that may impact the scheduling of $T_{i,v}^w$, directly or indirectly, through precedence constraints.

**Definition 16.** If $T_{i,v}^w$ is blocked at time $t$ and none of $T_{i,v}^w$'s U-jobs is preempted at $t$, then $T_{i,v}^w$ is *idle blocked* at $t$. The total time for which $T_{i,v}^w$ is idle blocked within a time interval $[t, t')$ is called its *idle blocking time* within $[t, t')$.

Note that, if some of $T_{i,v}^w$'s U-jobs are preempted while $T_{i,v}^w$ is blocked, then there can be no idle processors. Thus, $T_{i,v}^w$'s idle blocking time upper-bounds the total time for which some processor is idle while $T_{i,v}^w$ is blocked that could otherwise execute $T_{i,v}^w$ if it had not been blocked.

**Claim 2.** *The total idle blocking time of $T_{i,v}^k$ is at most $e_i^{max_w} - e_i^k$.*

*Proof.* We prove the claim by induction on $k$. The base case is trivial: $T_{i,v}^1$ has no U-predecessor and thus is not blocked. We shall now prove the induction step, $k > 1$. For

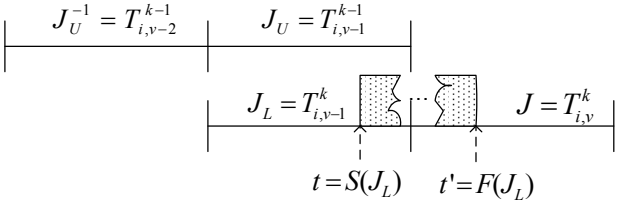


**Figure 6. Upper bounding idle blocking time.**

simplicity, let $J$ denote $T_{i,v}^k$, $J_L$ denote $T_{i,v-1}^k$, $J_U$ denote $T_{i,v-1}^{k-1}$, and $J_U^{-1}$ denote $T_{i,v-2}^{k-1}$. Let $t = S(J_L)$ and $t' = F(J_L)$, as shown in Fig. 6. Note that if $J_L$ does not exist, then $J$ is released at time zero and will not be blocked, by Def. 13. So, assume that $J_L$ exists.

**Case 1**: $J_U$'s U-predecessor is complete at $t$ (or, it does not exist). Observe that $J_U^{-1}$ (if it exists) is complete by time $t$ (otherwise, $J_L$ could not execute). Thus, $J_U$ is enabled at or before $t$, which implies that all U-jobs of $J$ other than $J_U$ complete at or before $t$. If $J_U$ does not complete by time $t'$, then in $[t, t')$, by our priority definition, $J_U$ executes whenever $J_L$ does, and $J_L$ is preempted whenever $J_U$ is. Therefore, after $t'$, $J_U$ has at most $\max(0, e_i^{k-1} - e_i^k)$ computation time left, which is at most $e_i^{max_w} - e_i^k$. Given that $J_U$ is the only U-job of $J$ that may not have completed by time $t'$, by Def. 16, $T_{i,v}^k$'s total idle blocking time is at most $e_i^{max_w} - e_i^k$.

**Case 2**: $J_U$'s U-predecessor is not complete at $t$. Then, $J_U$ is blocked at or after $t$. By the induction hypothesis, $J_U$ idle blocks for at most $e_i^{max_w} - e_i^{k-1}$ time at or after $t$. Thus, the amount of work due to $J_U$ and its U-jobs performed in $[t, F(J_U))$ is at most $J_U$'s idle blocking time plus an additional $e_i^{k-1}$ (for $J_U$), which is at most $e_i^{max_w} - e_i^{k-1} + e_i^{k-1} = e_i^{max_w}$. During $[t, t']$, if $J_L$ executes, then one of $J_U$'s U-jobs (if $J_U$ is blocked) or $J_U$ (if $J_U$ is not blocked) execute as well, because such jobs have higher priority than $J_L$. Thus, the amount of work due to $J_U$ and its U-jobs performed in $[t', F(J_U))$ is at most $e_i^{max_w} - e_i^k$. Therefore, by Def. 16, the total idle blocking time for $J$ is at most $e_i^{max_w} - e_i^k$. □

**Definition 17.** The *pre-stage subtasks* of any subtask $T_i^k$ are $T_i^1, T_i^2, ..., T_i^k$ (note that $T_i^k$ itself is included).

Let $W'$ be the amount of work due to $\mathbf{d} \cup \mathbf{D}$ performed during $[t_s, t_p)$. Let $I$ be the total idle time in $[t_s, t_p)$, where the idle time at each instant is the number of idle processors at that instant. Then, $W' + I = m \cdot (t_p - t_s)$. The following claim will be used to complete the proof of Subcase 2.3.1.

**Claim 3.** $W' \geq (1 - s_{max}) \cdot m(t_p - t_s) - me_{max}$.

*Proof.* We prove the claim by constructing $m$ disjoint sets of jobs, denoted $\lambda^{(1)}, \lambda^{(2)}, ..., \lambda^{(m)}$, that execute (at least partially) in $[t_s, t_p)$. The construction method is described by the algorithm *Schedule-scan* in Fig. 7. The jobs added

**Algorithm 1** *Schedule-scan*

1: Let $\lambda^{(1)}, \lambda^{(2)}, ..., \lambda^{(m)}$ be $m$ disjoint job sets, initially empty
2: $t := t_s$
3: Select $m$ jobs, $T_1^{(1)}, T_1^{(2)}, ..., T_1^{(m)}$, that are ready at $t$
4: **for** $k := 1$ **to** $m$ **do**
5: $\quad q_k := 1$
6: $\quad IT(T_1^{(k)}) := t_s$
7: $\quad$ Add $T_1^{(k)}$, where $1 \leq k \leq m$, to set $\lambda^{(k)}$
8: **end for**
9: **for** $t := t_s + \varepsilon$ **to** $t_p^-$ **do**
10: $\quad$ **for** $k := 1$ **to** $m$ **do**
11: $\quad\quad$ **if** $T_{q_k}^{(k)}$ completes at $t$ **then**
12: $\quad\quad\quad q_k := q_k + 1$
13: $\quad\quad\quad$ Select a job not in $\lambda^{(1)} \cup ... \cup \lambda^{(m)}$ that is ready at $t$
14: $\quad\quad\quad$ Let $T_{q_k}^{(k)}$ be this job and add it to $\lambda^{(k)}$
15: $\quad\quad\quad ET(T_{q_k-1}^{(k)}) := IT(T_{q_k}^{(k)}) := t$
16: $\quad\quad$ **end if**
17: $\quad$ **end for**
18: **end for**
19: **for** $k = 1$ **to** $m$ **do**
20: $\quad ET(T_{q_k}^{(k)}) := t_p$
21: **end for**

**Figure 7. Psuedocode of** *Schedule-scan***.**

to each $\lambda^{(k)}$ are selected at lines 3 and 12 in *Schedule-scan*. Each selected job exists because at least $m$ jobs in $\mathbf{d} \cup \mathbf{D}$ are ready at each time instant within $[t_s, t_p)$. For each selected job $T_i^{(k)}$, an *initial time*, $IT(T_i^{(k)})$, and an *end time*, $ET(T_i^{(k)})$, are computed. The corresponding intervals $[IT(T_i^{(k)}), ET(T_i^{(k)}))$, where $T_i^{(k)} \in \lambda^{(k)}$, satisfy $\sum_{T_i^{(k)} \in \lambda^{(k)}} (ET(T_i^{(k)}) - IT(T_i^{(k)})) = t_p - t_s$, as illustrated in Fig. 8. We call the interval $[IT(T_i^{(k)}), ET(T_i^{(k)}))$ the *presence interval* for $T_i^{(k)}$.

Note that a job $T_i^{(k)}$ may execute before $IT(T_i^{(k)})$, and in addition, if $T_i^{(k)}$ is the last selected job for $\lambda^{(k)}$, it may complete execution after $ET(T_i^{(k)})$. We are primarily interested in the computation $T_i^{(k)}$ performs in its presence interval. As such intervals do not overlap for different jobs in $\lambda^{(k)}$, we assume, without loss of generality, that each job in $\lambda^{(k)}$ executes on processor $k$ during its presence interval.

We now upper-bound the idleness within $[t_s, t_p)$ on processor $k$. Consider a job $J$ in $\lambda^{(k)}$. We first upper bound the idle time on processor $k$ within $[IT(J), ET(J))$, and then sum over all such intervals as defined in constructing $\lambda^{(k)}$.

If processor $k$ is idle at any time in $[IT(J), E(J))$, then $J$ is blocked at that time. By the discussion following Def. 16, it follows that $J$'s idle blocking time in $[IT(J), ET(J))$, denoted $I(J)$, upper-bounds the actual idle time on processor $k$ in $[IT(J), ET(J))$. By Claim 2, $I(J) \leq e^{max(J)} - e(J)$, where $e^{max(J)}$ is the maximum execution time among all pre-stage subtasks of $J$, and $e(J)$ is $J$'s execution time.

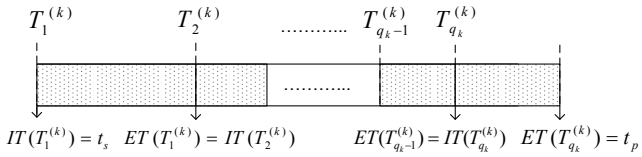


**Figure 8. Job intervals within $[t_s, t_p]$ defined by** *schedule-scan***.**

Because each selected job may execute partially in its presence interval (as discussed above), jobs in $\lambda^{(k)}$ can be classified into three sets: **(i)** *IN*, which consists of jobs that execute fully within their presence intervals; **(ii)** *BE* (for *Before*), which consists of jobs that execute partially in their presence intervals but complete before $t_p$ (and also the end of their presence intervals); and **(iii)** *AF* (for *After*), which consists of jobs that execute partially in their presence intervals but complete after $t_p$. Note that *AF* consists of at most one job, which is the last selected job in $\lambda^{(k)}$. In order to upper-bound the idleness within $[t_s, t_p)$ on processor $k$, we prove that for any job $J$ in *IN* and *BE*, the idleness within $[IT(J), ET(J))$ is at most $s_{max} \cdot (ET(J) - IT(J))$, and for the (at most) one job $J$ in *AF*, the idleness within $[IT(J), ET(J))$ is at most $s_{max} \cdot (ET(J) - IT(J)) + e_{max}$. Since each job in $\lambda^{(k)}$ belongs to one of the above three sets, we consider three cases.

**Case 1:** $J \in$ *IN*. If $J$ is idle blocked at $t \in [IT(J), ET(J))$, then by Def. 16, $J$ and $J$'s U-jobs are not preempted at $t$. Similarly, if $J$ or one of $J$'s U-jobs is preempted at $t$, then $J$ is not idle blocked at $t$. Thus, because $J$ executes fully during its presence interval, $ET(J) - IT(J) = I(J) + \Delta(J) + e(J)$, where $I(J)$ is $J$'s idle blocking time in $[IT(J), ET(J))$, $\Delta(J)$ is the total preemption time of $J$ and $J$'s U-jobs within $[IT(J), ET(J))$, and $e(J)$ is $J$'s execution time. Thus,

$$\begin{aligned} I(J) &= \frac{I(J)}{ET(J) - IT(J)} \cdot (ET(J) - IT(J)) \\ &= \frac{I(J)}{I(J) + \Delta(J) + e(J)} \cdot (ET(J) - IT(J)) \\ &\leq \frac{I(J)}{I(J) + e(J)} \cdot (ET(J) - IT(J)) \\ &\quad \{\text{by Claim2}\} \\ &\leq \frac{e^{max(J)} - e(J)}{e^{max(J)}} \cdot (ET(J) - IT(J)) \\ &\leq s_{max} \cdot (ET(J) - IT(J)). \end{aligned}$$

**Case 2:** $J \in$ *BE*. In this case $J$ starts execution before $IT(J)$. Thus, $J$ either executes or is preempted during $[IT(J), ET(J)]$. Therefore, $I(J) = 0$.

**Case 3:** $J \in$ *AF*. In this case $J$ may not have completed execution at $t_p$, or may not even start execution at $t_p$. We prove that $I(J) \leq s_{max} \cdot (ET(J) - IT(J)) + e_{max}$. Let

$e'(J)$ denote the amount of execution $J$ performs within $[IT(J), ET(J))$. Thus, $ET(J) - IT(J) = I(J) + \Delta(J) + e'(J)$, where $I(J)$ and $\Delta(J)$ are as defined in Case 1. We consider two subcases.

**Subcase 3.1:** $\Delta(J) \geq e(J) - e'(J)$. In this case, $ET(J) - IT(J) = I(J) + \Delta(J) + e'(J) \geq I(J) + (e(J) - e'(J)) + e'(J) = I(J) + e(J)$. Thus, we have

$$\begin{aligned} I(J) &= \frac{I(J)}{ET(J) - IT(J)} \cdot (ET(J) - IT(J)) \\ &\leq \frac{I(J)}{I(J) + e(J)} \cdot (ET(J) - IT(J)) \\ &\quad \{\text{reasoning as in Case 1}\} \\ &\leq s_{max} \cdot (ET(J) - IT(J)) \\ &< s_{max} \cdot (ET(J) - IT(J)) + e_{max}. \end{aligned}$$

**Subcase 3.2:** $\Delta(J) < e(J) - e'(J)$. In this case, given that $I(J) \leq e^{max(J)} - e(J)$, by Claim 2, we have $ET(J) - IT(J) = I(J) + \Delta(J) + e'(J) < (e^{max(J)} - e(J)) + (e(J) - e'(J)) + e'(J) = e^{max(J)} \leq e_{max}$. Thus, we have $I(J) \leq ET(J) - IT(J) < e_{max} \leq s_{max} \cdot (ET(J) - IT(J)) + e_{max}$.

Because there is at most one job in *AF*, the idleness within $[t_s, t_p)$ on processor $k$, denoted $I^k$, satisfies

$$\begin{aligned} I^k &\leq \sum_{J \in \lambda^{(k)}} s_{max} \cdot (ET(J) - IT(J)) + e_{max} \\ &= s_{max} \cdot (t_p - t_s) + e_{max} \end{aligned}$$

Given that there are $m$ $\lambda$ sets, the idleness within $[t_s, t_p)$ on all $m$ processors, $I$, is at most $m \cdot \max\{I^k\} \leq m \cdot (s_{max} \cdot (t_p - t_s) + e_{max}) = s_{max} \cdot m(t_p - t_s) + me_{max}$. Thus, $W' = m(t_p - t_s) - I \geq (1 - s_{max}) \cdot m(t_p - t_s) - me_{max}$. This completes the proof of Claim 3. □

We now complete the proof of Subcase 3.2 (and thereby Lemma 4).

As shown in Fig. 5, $[t_d, t_s)$ and $[t_p, S(T^h_{l,j}))$ are busy for $\mathbf{d} \cup \mathbf{D}$. By Claim 3, the amount of work due to $\mathbf{d} \cup \mathbf{D}$ performed in $[t_s, t_p)$ is at least $(1 - s_{max}) \cdot m(t_p - t_s) - m \cdot e_{max}$. Also, the amount of work due to $\mathbf{d} \cup \mathbf{D}$ performed in $[S(T^h_{l,j}), F(T^h_{l,j}))$ is at least $m(F(T^h_{l,j}) - S(T^h_{l,j})) - (m - 1)e^h_l$ (see Subcase 2.2). Thus, we have

$$\begin{aligned} Z \geq\ & m(t_s - t_d) + (1 - s_{max}) \cdot m(t_p - t_s) \\ & - m \cdot e_{max} + m(S(T^h_{l,j}) - t_p) \\ & + m(F(T^h_{l,j}) - S(T^h_{l,j})) - (m - 1)e^h_l. \end{aligned}$$

By (6), we therefore have

$$\begin{aligned} & (1 - s_{max}) \cdot mx - (m - 1) \cdot e^h_l - me_{max} \\ \geq\ & m(t_s - t_d) + (1 - s_{max}) \cdot m(t_p - t_s) \\ & - m \cdot e_{max} + m(S(T^h_{l,j}) - t_p) \\ & + m(F(T^h_{l,j}) - S(T^h_{l,j})) - (m - 1)e^h_l, \end{aligned}$$

which gives,

$$F(T^h_{l,j}) - t_d \leq (1 - s_{max}) \cdot x + s_{max} \cdot (t_p - t_s).$$

According to Property (P), $t_p - t_s \leq t_p - t_d \leq x - p_l + max\{e^h_l, e^{h-1}_l\} \leq x$. Therefore, $F(T^h_{l,j}) - t_d \leq (1 - s_{max}) \cdot x + s_{max} \cdot x < x + e^h_l$. □

### 3.3 Determining $x$

Setting the upper bound on $LAG(\mathbf{d}, t_d, S) + B(\mathbf{D}, t_d, S)$ in (5) to be at most the lower bound in Lemma 4 will ensure that the tardiness of $T^h_{l,j}$ is at most $x + e^h_l$. 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 $x + e^h_l$. By (5) and Lemma 4, this inequality is

$$\begin{aligned} & U \cdot x + \Gamma + \textstyle\sum_{T_i \in \tau} \sum_{T^k_i \in T_i} e^k_i \\ \leq\ & (1 - s_{max}) \cdot mx - (m - 1)e^h_l - me_{max}. \end{aligned}$$

Solving for $x$, we have

$$x \geq \frac{\Gamma + \sum_{T_i \in \tau} \sum_{T^k_i \in T_i} e^k_i + (m - 1)e^h_l + me_{max}}{(1 - s_{max}) \cdot m - U}. \tag{8}$$

If $x$ equals the right-hand side of (8), then the tardiness of $T^h_{l,j}$ will not exceed $x + e^h_l$. A value for $x$ that is independent of the parameters of $T^h_l$ can be obtained by replacing $(m - 1)e^h_l$ with $max_{l,h}((m - 1)e^h_l)$ in (8).

**Theorem 1.** *With $x$ as defined above, the tardiness for any subtask $T^h_l$ scheduled under GSA is at most $x + e^h_l$, provided $U < (1 - s_{max}) \cdot m$, where $U$ is sum of the $m(m-1)$ largest subtask utilizations.*

For GFIFO, the bound in Theorem 1 can be improved slightly.

**Corollary 1.** *The tardiness for any subtask $T^h_l$ scheduled under GFIFO is at most $x + e^h_l$, where $x = \frac{\Gamma + (m - 1)e^h_l + me_{max} + \sum_{p^k_i > p^h_l} e^k_i}{(1 - s_{max}) \cdot m - U}$, provided $U < (1 - s_{max}) \cdot m$.*

*Proof.* Under GFIFO, $D_H$ consists of carry-in jobs that are released before $r^h_{l,j}$ and have deadlines later than $t_d$, which implies that these jobs have periods greater than $p_l$. Thus, an upper bound on the LAG possible for $\tau$ at $t_d$ under GFIFO becomes $LAG(\mathbf{d}, t_d, S) + B(\mathbf{D}, t_d, S) \leq U \cdot x + \Gamma + \sum_{p^k_i > p^h_l} e^k_i$. Using this upper bound to solve for $x$, the corollary follows. □

For GEDF, the bound in Corollary 1 can be further improved.

**Corollary 2.** *The tardiness for any subtask $T_l^h$ scheduled under GEDF is at most $x + e_l^h$, where $x = \frac{\Gamma + (m-1)e_l^h + me_{max}}{(1 - s_{max}) \cdot m - U}$, provided $U < (1 - s_{max}) \cdot m$.*

*Proof.* Under GEDF, the demand placed by jobs in $\mathbf{D}$ after $t_d$ is zero because $\mathbf{D} = \emptyset$. Thus, an upper bound on the $LAG$ possible for $\tau$ at $t_d$ under GEDF becomes $LAG(\mathbf{d}, t_d, S) + B(\mathbf{D}, t_d, S) \leq U \cdot x + \Gamma$. Using this upper bound to solve for $x$, the corollary follows. □

**Definition 18.** A pipeline task $T_l$ with $h$ stages is *monotonically increasing* if $(\forall j : 1 \leq j < h :: e_l^j \leq e_l^{j+1})$.

**Corollary 3.** *If all pipeline tasks are monotonically increasing, then the tardiness for any subtask $T_l^h$ scheduled under GSA is at most $x + e_l^h$, where $x = \frac{\Gamma + \sum_{T_i \in \tau} \sum_{T_i^k \in T_i} e_i^k + (m-1)e_l^h + me_{max}}{m - U}$. In this case, because $U \leq U_{sum}$, we only need to constrain total utilization by $U_{sum} < m$.*

*Proof.* By Defs. 10 and 18, $s_{max} = 0$ in this case. □

**Corollary 4.** *For two-processor systems, the tardiness for any subtask $T_l^h$ scheduled under GSA is at most $x + e_l^h$, where $x = \frac{\Gamma + \sum_{T_i \in \tau} \sum_{T_i^k \in T_i} e_i^k + (m-1)e_l^h + me_{max}}{m - U}$ (again, only $U_{sum} < m$ is required).*

*Proof.* If $m = 2$, then the lower bound as stated in Lemma 4 becomes $mx - (m-1)e_l^h - me_{max}$. In this case, Lemma 4 holds trivially because $[t_d, t_d + y)$ is a busy interval for $\mathbf{d} \cup \mathbf{D}$ (details are left to the reader). By solving $x$ using this lower bound, the corollary holds. □

### 3.4 A Counterexample

Previous research has shown that every periodic task system under GFIFO or GEDF scheduling has bounded tardiness [5]. We show that it is possible for a pipeline task system to have unbounded tardiness under GFIFO or GEDF scheduling, if the utilization cap in Theorem 1 is violated.

Consider a task set $\tau$, to be scheduled under GFIFO or GEDF on three processors, that consists of two two-stage pipeline tasks: $T_1^1 = (9, 10), T_1^2 = (7, 10), T_2^1 = (5, 5)$, and $T_2^2 = (2, 5)$. For this task system, $s_{max} = 0.6$ and $U = 3$, which violates the condition stated in Theorem 1. Fig. 9 shows the tardiness of both pipeline tasks scheduled under GFIFO by job instance. These graphs were obtained by simulating the execution of this system. We have verified analytically that the tardiness growth rate seen in these graphs continues indefinitely (this is also true for GEDF).

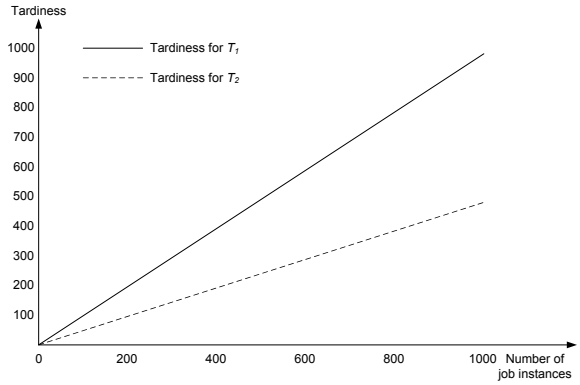


**Figure 9. Tardiness growth rates for counterexample under GFIFO.**

This counterexample shows that, on three or more processors, overall utilization must (generally) be constrained. As seen in Corollary 4, on two-processor systems, only $U_{sum} < m$ is required.

### 3.5 Experimental Evaluation

In this section, we describe the results of two sets of experiments conducted using randomly-generated task sets to evaluate the accuracy and the applicability of the tardiness bounds for GFIFO and GEDF derived in Sec. 3.3.

The goal of the first set of experiments is to examine how restrictive the utilization cap stated in Theorem 1 is. Task sets were generated as follows. The maximum per-subtask utilization $u_{max}$ was chosen uniformly over $\{0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5\}$. $s_{max}$ varied over $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$ and was determined by setting $e_{max} = 10$, where $e_{max}$ is the maximum per-subtask execution cost. We selected each subtask's execution cost uniformly over $[\mu \cdot e_{max}, e_{max}]$, where $\mu$ is a coefficient with range $\{0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$. For each combination of parameters $(u_{max}, s_{max})$, 1,000 task sets were generated for a four-processor system.

As seen in Fig. 10, when $u_{max} < 0.3$, all task sets have bounded tardiness. This is because when per-subtask utilizations are not high, $U$ becomes small even if total utilization is $m$. When $u_{max}$ exceeds 0.5 (note that 0.5 is a very high per-subtask utilization, given that each pipeline task may contain four subtasks), approximately 80% of all task sets are still schedulable with bounded tardiness if $s_{max} < 0.5$. Even when $s_{max} = 0.6$ and $u_{max} = 0.5$, approximately 65% of all task sets have bounded tardiness.

The second set of experiments was conducted to compare computed and observed tardiness for both ordinary and pipelined task systems under both GFIFO and GEDF. We used the same method as used in the first experiment to generate task systems. (By simply removing all precedence constraints, we obtain ordinary task systems.) 1,000

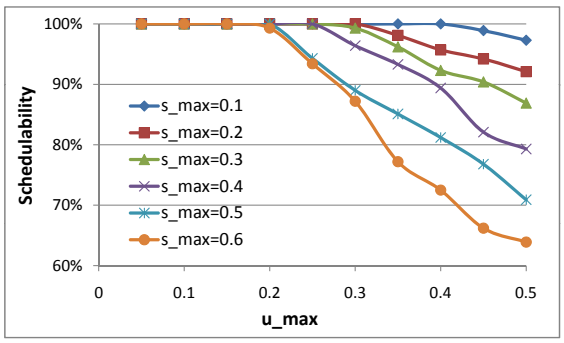


**Figure 10. Applicability.**

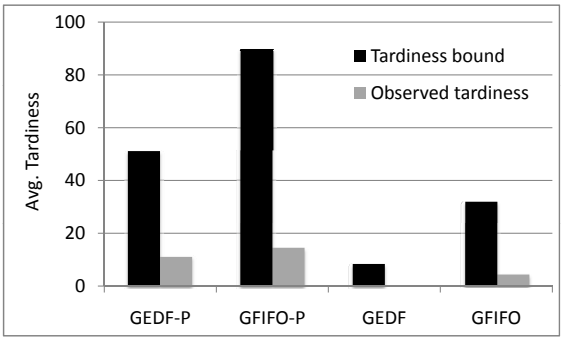


**Figure 11. Tardiness.**

task sets were generated for a four-processor system with $u_{max} = 0.1$ and $\mu = 0.8$. When generating task sets, we dropped any task set that violates the condition as stated in Thereom 1 (and those for which bounded tardiness cannot be guaranteed). As shown in Fig. 11, the tardiness bound of pipeline task systems scheduled under GEDF (denoted GEDF-P) or GFIFO (denoted GFIFO-P) is much higher than that of ordinary task systems. This is due to the denominator $(1 - s_{max})m - U$ in the derived bound, which is smaller than that appearing in the bound for ordinary task systems [4, 5]. However, the observed tardiness of pipeline task systems under GEDF or GFIFO is fairly close to that of ordinary task systems. These results (which for lack of space are all that we can present) suggest that GEDF and GFIFO are reasonable global scheduling options to consider for pipeline task systems.

## 4 Conclusion

We have derived a tardiness bound that can be applied to globally-scheduled periodic pipeline task systems. This bound is applicable to a class of global algorithms that includes GEDF and GFIFO. The derived tardiness bound requires overall utilization to be constrained in some systems. However, only $U_{sum} < m$ is required for any two-processor system or for any system where all pipeline tasks are monotonically increasing. For other systems, utilization constraints are fundamental, as we have shown via a counterexample. Nonetheless, for these systems, the required constraint is quite liberal.

The results of this paper are not applicable to sporadic task systems. Our tardiness proof relies crucially on the tie-breaking rule we assumed concerning subtasks within the same pipeline task. In a sporadic task system, job releases can be slightly jittered to produce a schedule that is "almost" periodic in which no two jobs have the same priority. In such cases, tie-breaking rules are of no utility. By exploiting this fact, we have been able to derive a number of counterexamples in which tardiness grows unboundedly in sporadic systems. However, in recent work [10], we have shown that such systems can be dealt with by using periodic server tasks to service sporadic workflows.

## References


[1] J. Carpenter, S. Funk, P. Holman, J. H. Anderson, and S. Baruah. *Handbook on Scheduling Algorithms, Methods, and Models*. Chapman Hall/CRC, Boca, 2004.

[2] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. In *Algorithmica*, Vol.15, pp. 600-625, 1996.

[3] H. Cho, B. Ravindran, and E. D. Jensen. An optimal real-time scheduling algorithm for multiprocessors. In *Proc. of the 27th IEEE Int'l Real-Time Systems Symp.*, pp. 101-110, 2006.

[4] U. C. Devi and J. H. Anderson. Tardiness bounds under global edf scheduling on a multiprocessor. In *Proc. of the 26th IEEE Int'l Real-Time Systems Symp.*, pp. 330-341, 2005.

[5] H. Leontyev and J. H. Anderson. Tardiness bounds for fifo scheduling on multiprocessors. In *Proc. of the 19th Euromicro Conf. on Real-Time Systems*, pp. 71-80, 2007.

[6] P. Jayachandran and T. Abdelzaher. A delay composition theorem for real-time pipelines. In *Proc. of the 19th Euromicro Conf. on Real-Time Systems*, pp. 29-38, 2007.

[7] P. Jayachandran and T. Abdelzaher. Transforming distributed acyclic systems into equivalent uniprocessors under preemptive and non-preemptive scheduling. In *Proc. of the 20th Euromicro Conf. on Real-Time Systems*, pp. 233-242, 2008.

[8] J.C. Palencia and M.G. Harbour. Offset-based response time analysis of distributed systems scheduled under edf. In *Proc. of the 15th Euromicro Conf. on Real-Time Systems*, pp. 3-12, 2003.

[9] R. Pellizzoni and G. Lipari. Improved schedulability analysis of real-time transactions with earliest deadline scheduling. In *Proc. of the 11th IEEE Int'l Real Time and Embedded Technology and Applications Symp.*, pp. 66-75, 2005.

[10] C. Liu and J. H. Anderson. Supporting sporadic pipelined tasks with early-releasing in soft real-time multiprocessor systems. In Submission. Available at:http://www.cs.unc.edu/~anderson/papers.

## 5 Appendix

Let $A(\chi, t_1, t_2, S)$ denote the total time allocated to all jobs in the job set $\chi$ in $[t_1, t_2)$ in $S$.

**Lemma 1.** $LAG(\mathbf{d}, t_d, S) \leq LAG(\mathbf{d}, t_n, S) + \sum_{T_k^w \in D_H} \delta_k^w (1 - u_k^w)$, where $t \in [0, t_d]$.

*Proof.* By (3), we have

$$\begin{aligned} LAG(\mathbf{d}, t_d, S) &\leq LAG(\mathbf{d}, t_n, S) + A(\mathbf{d}, t_n, t_d, PS) \\ &\quad - A(\mathbf{d}, t_n, t_d, S). \end{aligned} \tag{9}$$

We split $[t_n, t_d)$ into $z$ non-overlapping intervals $[t_{p_i}, t_{q_i}), 1 \leq i \leq z$ such that $t_n = t_{p_1}, t_{q_{i-1}} = t_{p_i}$, and $t_{q_z} = t_d$. Each interval $[t_{p_i}, t_{q_i})$ is either busy or non-busy displacing for $\mathbf{d}$, by the selection of $t_n$. We assume that the intervals are defined so that for each non-busy displacing interval $[t_{p_i}, t_{q_i})$, if a subtask in $D_H$ executes in $[t_{p_i}, t_{q_i})$ then it executes continuously throughout $[t_{p_i}, t_{q_i})$; we let $\alpha_i$ denote the set of such subtasks.

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

**Case 1.** $[t_{p_i}, t_{q_i})$ is busy. Since in $S$ all processors are occupied by jobs in $\mathbf{d}$, we have $A(\mathbf{d}, t_{p_i}, t_{q_i}, PS) - A(\mathbf{d}, t_{p_i}, t_{q_i}, S) \leq U_{sum}(t_{q_i}, t_{p_i}) - m(t_{q_i} - t_{p_i}) \leq 0$.

**Case 2.** $[t_{p_i}, t_{q_i})$ is non-busy displacing. The cumulative utilization of all subtasks $T_k^w \in \alpha_i$ is $\sum_{T_k^w \in \alpha_i} u_k^w$. The carry-in jobs of these subtasks do not belong to $\mathbf{d}$, by the definition of $\mathbf{d}$. Therefore, the allocation of jobs in $\mathbf{d}$ during $[t_{p_i}, t_{q_i})$ in $PS$ is $A(\mathbf{d}, t_{p_i}, t_{q_i}, PS) \leq (t_{q_i} - t_{p_i})(m - \sum_{T_k^w \in \alpha_i} u_k^w)$. All processors are occupied at every time instant in the interval $[t_{p_i}, t_{q_i})$, because it is displacing. Thus, $A(\mathbf{d}, t_{p_i}, t_{q_i}, S) = (t_{q_i} - t_{p_i})(m - |\alpha_i|)$. Therefore, the allocation difference for jobs in $\mathbf{d}$ thoughout the interval is

$$\begin{aligned} & A(\mathbf{d}, t_{p_i}, t_{q_i}, PS) - A(\mathbf{d}, t_{p_i}, t_{q_i}, S) \\ \leq\ & (t_{q_i} - t_{p_i}) \left( (m - \sum_{T_k^w \in \alpha_i} u_k^w) - (m - |\alpha_i|) \right) \\ =\ & (t_{q_i} - t_{p_i}) \sum_{T_k^w \in \alpha_i} (1 - u_k^w). \end{aligned} \tag{10}$$

For each subtask $T_k^w$ in $D_H$, the sum of the lengths of the intervals $[t_{p_i}, t_{q_i})$ in which the carry-in job of $T_k^w$ executes continuously is at most $\delta_k^w$. Thus, summing the allocation differences for all the intervals $[t_{p_i}, t_{q_i})$ given by (10), we have

$$\begin{aligned} & A(\mathbf{d}, t_n, t_d, PS) - A(\mathbf{d}, t_n, t_d, S) \\ \leq\ & \sum_{i=1}^{z} \sum_{T_k^w \in D_H} (t_{q_i} - t_{p_i})(1 - u_k^w) \\ \leq\ & \sum_{T_k^w \in D_H} \delta_k^w (1 - u_k^w). \end{aligned} \tag{11}$$

Setting this value into (9), we get $LAG(\mathbf{d}, t_d, S) \leq LAG(\mathbf{d}, t_n, S) + A(\mathbf{d}, t_n, t_d, PS) - A(\mathbf{d}, t_n, t_d, S) \leq LAG(\mathbf{d}, t_n, S) + \sum_{T_k^w \in D_H} \delta_k^w (1 - u_k^w)$. □

**Lemma 2.** $lag(T_k^w, t, S) \leq u_k^w \cdot x + e_k^w$ for any subtask $T_k^w$ and $t \in [0, t_d]$.

*Proof.* Let $d_{k,j}^w$ be the deadline of the earliest pending job of $T_k^w, T_{k,j}^w$, in the schedule $S$ at time $t$. If such a job does not exist, then $lag(T_k^w, t, S) \leq 0$, and the lemma holds trivially. Let $\gamma_k^w$ be the amount of work $T_{k,j}^w$ performs before $t$.

By the selection of $T_{k,j}^w$, we have

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

Given that no job executes before its release time, $A(T_{k,h}^w, 0, t, S) = A(T_{k,h}^w, r_{k,h}^w, t, S)$. Thus,

$$\begin{aligned} lag(T_k^h, t, S) &= A(T_{k,j}^w, r_{k,h}^w, t, PS) - A(T_{k,j}^w, r_{k,j}^w, t, S) \\ &\quad + \sum_{h > j} (A(T_{k,h}^w, r_{k,h}^w, t, PS) \\ &\quad - A(T_{k,h}^w, r_{k,h}^w, t, S)). \end{aligned} \tag{12}$$

By the definition of $PS$, $A(T_{k,j}^w, r_{k,h}^w, t, PS) \leq e_k^w$, and $\sum_{h>j} A(T_{k,h}^w, r_{k,h}^w, t, PS) \leq u_k^w \cdot (t - d_{k,j}^w)$. By the selection of $T_{k,j}^w$, $A(T_{k,j}^w, r_{k,j}^w, t, S) = \gamma_k^w$, and $\sum_{h>j} A(T_{k,h}^w, r_{k,h}^w, t, S) = 0$. By setting these values into (12), we have

$$lag(T_k^w, t, S) \leq e_k^w - \gamma_k^w + u_k^w \cdot (t - d_{k,j}^w). \tag{13}$$

There are two cases to consider.

**Case 1.** $d_{k,j}^w \geq t$. In this case, (13) becomes $lag(T_k^w, t, S) \leq e_k^w - \gamma_k^w$, which implies $lag(T_k^w, t, S) \leq u_k^w \cdot x + e_k^w$.

**Case 2.** $d_{k,j}^w < t$. By Property (P), $T_{k,j}^w$ has tardiness at most $x + e_k^w$, so $t + e_k^w - \gamma_k^w \leq d_{k,j}^w + x + e_k^w$. Thus, $t - d_{k,j}^w \leq x + \gamma_k^w$. Setting this value into (13), we have $lag(T_k^w, t, S) \leq u_k^w \cdot x + e_k^w$. □