Supporting Sporadic Pipelined Tasks with Early-Releasing in Soft Real-Time Multiprocessor Systems[*]

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

Abstract

Soft real-time sporadic multiprocessor task systems are considered that include processing pipelines. Conditions are presented for guaranteeing bounded deadline tardiness in such systems under global EDF or FIFO scheduling. "Early-releasing" is applied to make pipeline scheduling work-conserving. This lessens job response times in lightly-loaded systems.

1 Introduction

With the advent of multicore technologies, it is important for programming frameworks for real-time applications to provide support for commonly-used multiprocessor programming techniques. One such technique 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 in real-time multiprocessor systems where the workload is specified as a sporadic 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 sporadic 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 k^{th} 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 sporadic model ensure that any pipeline executes correctly.

Unfortunately, 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 recent work [4], we showed that such loss can often be avoided in periodic task systems if bounded deadline tardiness is allowed. In particular, pipelines can be supported in such systems if certain utilization constraints are enforced; the required constraints are quite liberal. This result holds for a class of global scheduling algorithms that includes the *global earliest-deadline-first* (GEDF) and *global first-in first-out* (GFIFO) algorithms. (Note that these two algorithms are also capable of ensuring bounded deadline tardiness for ordinary sporadic task systems—i.e., systems without pipelined tasks—with no utilization loss [5, 6].) In this paper, we show how to adapt this result for use in sporadic pipeline task systems.

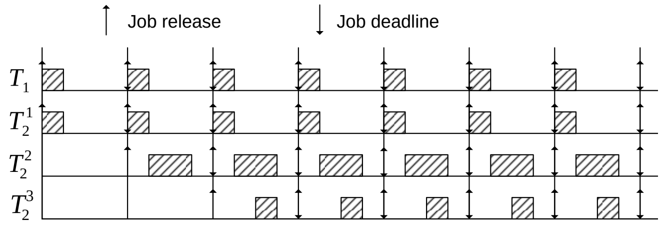


Figure 1. Example pipeline task system.

The above-mentioned result concerning periodic tasks hinges crucially upon the assumption that priority ties involving successive stages of the same pipeline are broken in favor of earlier stages. However, in a sporadic pipeline 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. Thus, tie-breaking rules are of little or no utility. Another limitation of this prior work is that pipeline scheduling is not work-conserving. This can cause unnecessarily long job response times, particularly in lightly-loaded systems. (A scheduler is *work conserving* if and only if it never leaves a processor idle when uncompleted work exists.) Work-conserving behavior can be ensured by "early-releasing" successive pipeline stages. A job that is early-released is allowed to become eligible for execution before its "actual" release time as dictated by the periodic or sporadic model. In the case of a pipeline, we wish to allow the j^{th} job of stage k (where $k > 1$) to become eligible once both $(j-1)^{st}$ job of stage k (if $j > 1$) and the j^{th} job of stage $k-1$ have completed execution. This can be accomplished by defining the early-release time of the j^{th} job of stage k to equal the release time of the j^{th} job of the first stage. Fig. 2 shows the impact of this technique on the

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

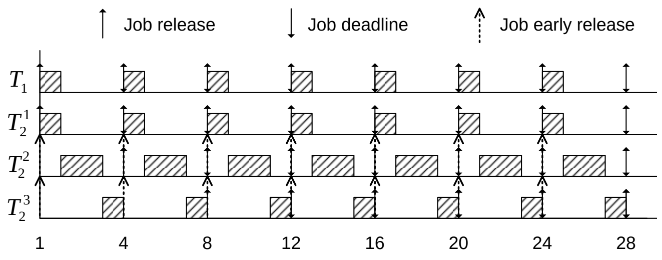


Figure 2. Example pipeline task system with early-releasing.

system considered earlier in Fig. 1. For example, the second job of T_2^2 is early-released at time 4 becasue the second job of T_2^1 is released at that time. In this paper, we show that by allowing early-releasing, pipelines can be supported in a work-conserving way, not only in periodic systems, but sporadic systems as well. In addition, we show that rate-based pipeline task systems can also be supported. In a rate-based system, job release times may experience significant jitter (see Sec. 4).

Related work. To our knowledge, we are the first to consider the problem of supporting pipelined execution in globally-scheduled soft real-time multiprocessor systems. However, pipelined execution has been considered in distributed systems (which must be scheduled by partitioning approaches). For example, Jayachandran and Abdelzaher have presented delay composition rules that provide a bound on the end-to-end delay of jobs in partitioned distributed systems that include pipelines [7] or more general (acyclic) precedence constraints [8]. These rules permit a pipelined system to be transformed so that uniprocessor schedulability analysis can be applied.

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 tests for pipelined distributed systems have also been proposed in which end-to-end deadlines are supported by deriving deadlines for individual pipeline stages [9, 10].

Contributions. We derive a general tardiness bound that can be applied to globally-scheduled sporadic pipeline task systems with early-releasing. 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. We show via counterexamples that such constraints are generally required. However, nontrivial constraints (beyond either $U_{sum} < m$ or $U_{sum} \leq m$ where U_{sum} is the total utilization and m is the processor count) are not required if all pipeline tasks are monotonically increasing (see Def. 7) or on two-processor systems. For systems where utilization must be further constrained, the required constraint is quite liberal. As we show via experimental results, early-releasing can significantly improve response times (and hence tardiness) in both sporadic and rate-based systems.

Organization. The remainder of this paper is organized as follows. Sec. 2 describes our system model. In Sec. 3, a tardiness bound for periodic tasks with early-releasing is derived. In Sec. 4, this result is extended to apply to sporadic and rate-based systems. In Sec. 5, experimental results are presented. The paper concludes in Sec. 6.

2 System Model

We consider the problem of scheduling a set $\tau = \{T_1, ..., T_n\}$ of n independent sporadic pipeline tasks on $m \geq 2$ identical processors. A z-stage pipeline task T_l, where $1 \leq z \leq m$, consists of z subtasks, $T_l^1, ...T_l^z$. (If $z = 1$, then T_l is an ordinary sporadic task.) Each subtask is released repeatedly, with each such invocation called a *job*. We assume that each job of T_l^h ($1 \leq h \leq z$) 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 response time. 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 minimum time between two consecutive job releases of any subtask of T_l and the relative deadline of each such job (i.e., $d_{l,j}^h = r_{l,j}^h + p_l$). If, for each task T_l, p_l specifies the *exact* time between two consecutive job releases of any subtask of T_l, then τ is called a *periodic* task system. 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* τ as $U_{sum} = \sum_{T_i \in \tau} \sum_{T_i^j \in T_i} u_i^j$.

Given the way pipelines usually function (the k^{th} jobs of all subtasks of a pipeline task represent a sequential computation that is initiated at the first stage), we assume that "sporadic separations" originate only within first-stage subtasks, i.e., for any $j > 1$, $r_{l,j}^1 \geq d_{l,j-1}^1$, and for any $h > 1$, $r_{l,j}^h = d_{l,j}^{h-1}$. Our proofs remain valid if this assumption is removed, but we find it natural to assume in the examples we present.

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

If a job $T_{i,j}^k$ completes at time t, then its *tardiness* is defined as $max(0, t - d^k_{i,j})$. A pipeline task's tardiness is the maximum of the tardiness of any job of any of its subtasks. We require $u^k_i \leq 1$ 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.

We allow jobs to execute before their actual release times by being "early-released." The earliest time at which job $T^h_{l,j}$ may execute is defined by its *early-release time* $\varepsilon(T^h_{l,j})$, where $\varepsilon(T^h_{l,j}) \leq r^h_{l,j}$. An unfinished job $T^h_{l,j}$ is *eligible* for execution at time t if both its L- and U-predecessors (if they exist) have completed by time t and $t \geq \varepsilon(T^h_{l,j})$.

The common case for pipelines is that first-stage jobs are released due to external events. In this case, it is not possible to early-release any first-stage job before its actual release time. However, any non-first-stage job $T^k_{l,j}$ $(k > 1)$ should be able to start execution whenever its L- and U-predecessors complete, even if both predecessors complete before $T^k_{l,j}$'s actual release time. This can be accomplished by defining the early-release time of a non-first-stage job to equal the release time of the corresponding first-stage job, i.e., $\varepsilon(T^h_{l,j}) = r^1_{l,j}$ where $1 \leq h \leq z$. This particular early-releasing method is not required of our proofs (as presented in Sec. 3), but it is assumed in the examples we present.

Under GEDF (GFIFO), released jobs are prioritized by their deadlines ((actual) 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 $T^w_{i,k}$, its prioritization function, $\rho^w_{i,k}$, is defined as: $\rho^w_{i,k} = r^w_{i,k} + \kappa \cdot p_i$, where $0 \leq \kappa \leq 1$. Jobs in τ are ordered based on their priorities: $T^w_{i,v} \prec T^c_{a,b}$ if and only if $\rho^w_{i,v} < \rho^c_{a,b}$ or $(\rho^w_{i,v} = \rho^c_{a,b}) \wedge (i = a) \wedge (w < c)$ or $(\rho^w_{i,v} = \rho^c_{a,b}) \wedge (i < a)$. $T^w_{i,v}$ has higher priority than $T^c_{a,b}$ if and only if $T^w_{i,v} \prec T^c_{a,b}$. Note that GEDF and GFIFO are special cases of GSA where κ is set to 1 and 0, respectively.

3 A Tardiness Bound for GSA with Early-Releasing

In this section, we derive a tardiness bound for GSA with early-releasing for periodic pipeline task systems. Based upon this, we then present an approach for dealing with sporadic task systems in Sec. 4. The tardiness bound is given in Theorem 1 below. The definitions that follow are used in the statement of the theorem.

Definition 1. Let $U(\tau, y)$ $(E(\tau, y))$ be the set of $\min(y, b)$ subtasks of highest utilization (execution cost) in τ, where b is the number of subtasks in τ. Define U and Γ as follows.

$$U = \textstyle\sum_{T^w_i \in U(\tau, m(m-1))} u^w_i$$
$$\Gamma = \textstyle\sum_{T^w_i \in E(\tau, m(m-1))} e^w_i$$

Definition 2. Let $\max_k = \max\{j \mid 1 \leq j \leq k \leq m \wedge (\forall w : 1 \leq w \leq k : e^j_i \geq e^w_i)\}$.

Definition 3. Let $s^v_i = \dfrac{e^{max_v}_i - e^w_i}{e^{max_v}_i}$. s^v_i is called the *subtask stretch* of T^v_i. Let $s_i = \max\{s^1_i, s^2_i, ..., s^z_i\}$. 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*.

Definition 4. Let e_{max} be the maximum execution time among all subtasks.

Theorem 1. *The tardiness for any subtask T^h_l scheduled under GSA (or GEDF or GFIFO) with early-releasing is at most $x + e^h_l$, where $x = \dfrac{\Gamma + \sum_{T_i \in \tau} \sum_{T^k_i \in T_i} e^k_i + (m-1)e^h_l + me_{max}}{(1 - s_{max}) \cdot m - U}$, provided $U < (1 - s_{max}) \cdot m$, where U is sum of the $m(m-1)$ largest subtask utilizations.*

The above theorem has been proved in [4] for GSA without early-releasing for periodic pipeline task systems. We now sketch the proof of this theorem (as presented in [4]) and explain why it remains valid if early-releasing is allowed. In the proof of Theorem 1, allocations to a periodic pipelined task system τ are compared in a processor sharing (PS) schedule and an actual GSA schedule for τ, both on m processors. The value of x given in the theorem is derived by considering a job $T^h_{l,j}$ of a subtask T^h_l in τ, where $t_d = d^h_{l,j}$, in a GSA schedule S for τ with the following property.

(P) The tardiness of every job $T^w_{i,k}$ such that $T^w_{i,k} \prec T^h_{l,j}$ is at most $x + e^w_i$ in S, where $x \geq 0$.

Determining the smallest x such that the tardiness of $T^h_{l,j}$ is at most $x + e^h_l$ inductively ensures a tardiness of at most $x + e^k_i$ for all jobs of every subtask T^k_i of $T_i \in \tau$. A value for x is derived via three steps:

1. Determine an upper bound on the work pending for tasks in τ that can compete with $T^h_{l,j}$ after t_d. This is dealt with in Lemma 1, given later.

2. Determine a lower bound on the amount of work pending for tasks in τ that can compete with $T^h_{l,j}$ after t_d, required for the tardiness of $T^h_{l,j}$ to exceed $x + e^h_l$. This is dealt with in Lemma 2.

3. Determine the smallest x such that the tardiness of $T^h_{l,j}$ is at most $x + e^h_l$, using the above upper and lower bounds.

To formally describe these steps, some additional notation is required. Let $A(T_{i,k}^w, t_1, t_2, S)$ denote the total allocation to the job $T_{i,k}^w$ in an arbitrary schedule S in $[t_1, t_2)$. Then, the total time allocated to all jobs of T_i^w in $[t_1, t_2)$ in S is given by

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

Consider a PS schedule *PS*. In such a schedule, $T_{i,k}^w$ executes at the rate u_i^w within $[r_{i,k}^w, r_{i,k}^w + p_i)$, i.e., for any subinterval $[t_1, t_2)$ of this interval, $A(T_{i,k}^w, t_1, t_2, PS) = (t_2 - t_1)u_i^w$.

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

$$lag(T_{i,k}^w, t, S) = A(T_{i,k}^w, 0, t, PS) - A(T_{i,k}^w, 0, t, S).$$

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

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

Definition 5. 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** (in fact, it is the only job of equal priority). **D** is the set of jobs that have higher priority than $T_{l,j}^h$ and deadlines greater than t_d. Note that **D** is empty under GEDF because jobs with later deadlines have lower priorities.

Definition 6. Let $B(\mathbf{D}, t_d, S)$ be the amount of work due to jobs in **D** that can compete with $T_{l,j}^h$ after t_d.

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 **d**, and **(ii)** the amount of work $B(\mathbf{D}, t_d, S)$ demanded by jobs in **D** that competes with $T_{l,j}^h$ after t_d. Since jobs from **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 (including that due to $T_{l,j}^h$ itself) is given by $LAG(\mathbf{d}, t_d, S) + B(\mathbf{D}, t_d, S)$.

Upper bound. The upper bound on the work pending for tasks in τ that can compete with $T_{l,j}^h$ after t_d is given by the following lemma, which was proved in [4] for systems without early-releasing.

Lemma 1. $LAG(\boldsymbol{d}, t_d, S) + B(\boldsymbol{D}, t_d, S) \leq U \cdot x + \Gamma + \sum_{T_i \in \tau} \sum_{T_i^w \in T_i} e_i^w$.

Note that the value of $LAG(\mathbf{d}, t_d, S) + B(\mathbf{D}, t_d, S)$ depends on allocations in the PS schedule PS and allocations to jobs in $\mathbf{d} \cup \mathbf{D}$ in the actual schedule S by time t_d. The PS schedule is not impacted by early-releasing—each job $T_{i,v}^w$ is allocated processor time only in $[r_{i,v}^w, d_{i,v}^w)$. Also, Property (P) alone is sufficient for determining how much work any job in $\mathbf{d} \cup \mathbf{D}$ other than $T_{l,j}^h$ completes before t_d—it does not matter if such a job is early-released. For these reasons, Lemma 1 continues to hold if early-releasing is allowed.

Lower bound. A lower bound on $LAG(\mathbf{d}, t_d, S) + B(\mathbf{D}, t_d, S)$ that is necessary for the tardiness of $T_{l,j}^h$ to exceed $x + e_l^h$ was established in [4] via the following lemma.

Lemma 2. *If the tardiness of* $T_{l,j}^h$ *exceeds* $x + e_l^h$*, then* $LAG(\boldsymbol{d}, t_d, S) + B(\boldsymbol{D}, t_d, S) > (1 - s_{max}) \cdot mx - (m - 1)e_l^h - me_{max}$.

Proving this lemma involves reasoning about how higher-priority jobs that may impact $T_{l,j}^h$ are scheduled beyond time t_d. As before, Property (P) largely dictates how much work these jobs can perform beyond t_d. Furthermore, note that all of the higher-priority jobs that must be analyzed are released before t_d. Thus, none of these jobs is early-released in the portion of the schedule that must be analyzed beyond time t_d. For this reason, early-releasing has no impact on the proof of Lemma 2 and thus it holds if early-releasing is allowed.

The value of x stated in Theorem 1 is obtained by setting the upper bound on $LAG(\mathbf{d}, t_d, S) + B(\mathbf{D}, t_d, S)$ in Lemma 1 to be at most the lower bound in Lemma 2 and solving for the minimum x that satisfies the resulting inequality.

Two corollaries of Theorem 1 were also stated in [4], and they also continue to hold if early-releasing is allowed. They are as follows.

Definition 7. A pipeline task T_l with z stages is *monotonically increasing* if $(\forall j : 1 \leq j < z :: e_l^j \leq e_l^{j+1})$.

Corollary 1. *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* $x = \dfrac{\Gamma + \sum_{T_i \in \tau} \sum_{T_i^k \in T_i} e_i^k + (m-1)e_l^h + me_{max}}{m - U}$. *In this*

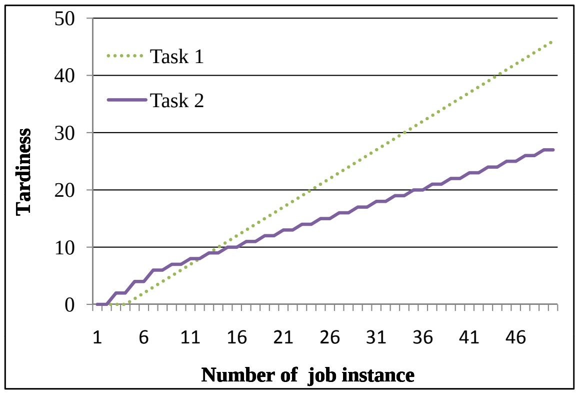


Figure 3. Tardiness growth rates for Counterexample 1 under GFIFO.

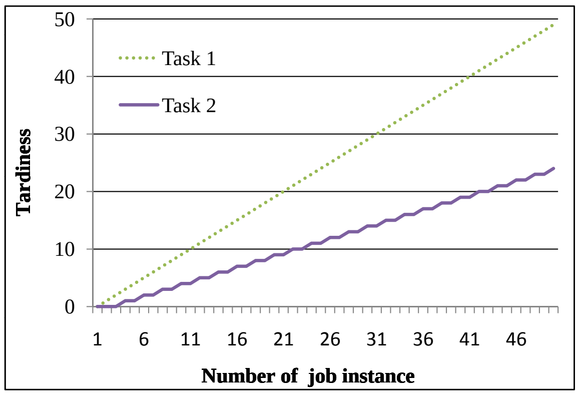


Figure 4. Tardiness growth rates for Counterexample 1 under GEDF.

case, because $U \leq U_{sum}$, we only need to constrain total utilization by $U_{sum} < m$.

Corollary 2. *For two-processor systems, the tardiness for any subtask T_l^h scheduled under GSA is at most $x + e_l^h$, where $x = \dfrac{\Gamma + \sum_{T_i \in \tau} \sum_{T_i^k \in T_i} e_i^k + (m-1)e_l^h + me_{max}}{m - U}$ (no utilization constraint other than $U_{sum} \leq m$ is required).*

As stated in Theorem 1, the condition $U < (1-s_{max}) \cdot m$ is required in order to ensure bounded tardiness for periodic pipeline task systems. It has been shown in [4] that some such constraint is needed via a counterexample where a periodic pipeline task system has unbounded tardiness. We now show that early-releasing does not eliminate the need for such a constraint by showing that tardiness grows unboundedly in the same counterexample if jobs are early-released.

Counterexample 1. Consider a task set τ, 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 $T_2^2 = (2, 5)$. Each job is assumed to be early-released in the way described in Sec. 2. For this task system, $s_{max} = \dfrac{e_2^1 - e_2^2}{e_2^1} = 0.6$ and $U = 3$. Thus, $(1 - s_{max}) \cdot m = 1.2 < U$, which violates the condition stated in Theorem 1. Figs. 3 and 4 show the tardiness of both pipeline tasks scheduled under GFIFO and GEDF by job instance. We have verified analytically that the tardiness growth rate seen in these graphs continues indefinitely.

4 Handling Sporadic Task Systems

The proof of Theorem 1 relies crucially on the assumption that if there is a priority tie between any two subtasks of a pipeline task, the earlier stage is favored. However, in a sporadic pipeline 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. Thus, tie breaking rules are of little or no utility for sporadic task systems. In fact, there exist sporadic task systems with unbounded tardiness for which tardiness is guaranteed by Theorem 1 to be bounded if all tasks are periodic. An example of such a system is given next.

Counterexample 2. Consider a task set τ, to be scheduled under GFIFO or GEDF on three processors, that consists of two two-stage pipeline tasks: $T_1^1 = (69, 100), T_1^2 = (70, 100), T_2^1 = (40, 50)$, and $T_2^2 = (40, 50)$. Note that all pipeline tasks in this task system are monotonically increasing. Thus, by Corollary 1, tardiness is bounded, assuming the system is periodic. Consider a schedule of this task set in which job releases are defined as follows:

$$\begin{aligned}
r_{1,j}^1 &= j \cdot p_1 + (j-1) \cdot \delta \\
r_{1,j}^2 &= (j+1) \cdot p_1 + (j-1) \cdot \delta \\
r_{2,j}^1 &= j \cdot p_2 + (j-1) \cdot \delta \\
r_{2,j}^2 &= (j+1) \cdot p_2 + (j-1) \cdot \delta,
\end{aligned}$$

where j denotes the j^{th} job instance of a subtask and $\delta = 0.01$. δ is used to jitter job releases to produce a sporadic schedule that is "almost" periodic in which no two jobs of any pipeline task have the same priority. This release pattern is shown in Fig. 5.

Figs. 6 and 7 show the tardiness of both pipeline tasks scheduled under GFIFO and GEDF in this example by job instance. We have verified analytically that the tardiness growth rate seen in these graphs continues indefinitely.

An approach for scheduling sporadic systems. Because proper tie-breaking appears to be so fundamental, and tie-breaking is rendered ineffective by non-periodic releases, we propose to schedule sporadic task systems by forcing job

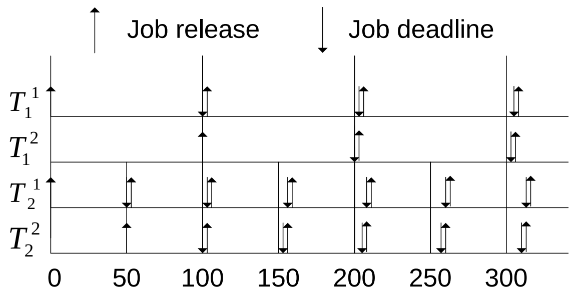


Figure 5. Job release pattern of Counterexample 2.

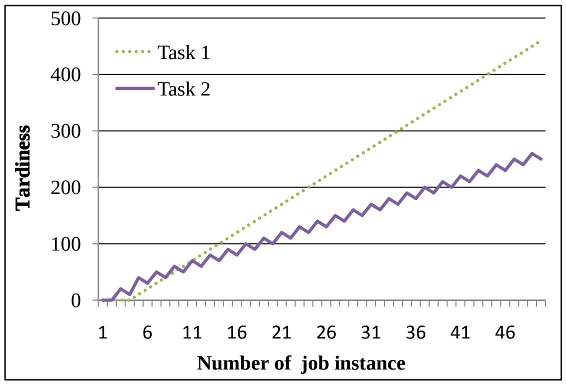


Figure 6. Tardiness growth rates for Counterexample 2 under GFIFO.

releases to be periodic. While such an approach may seem unreasonable in other contexts, in our case, it can be applied in a way that causes tardiness to increase only marginally. If a first-stage job $T_{l,j}^1$ is released at time $t \in ((k-1)\cdot p_l, k\cdot p_l]$, then job $T_{l,j}^h$'s ($h \geq 1$) early-release time, release time, deadline, and priority are re-defined according to rules, **S1-S4**, below (applied in the order specified).

S1 $\varepsilon(T_{l,j}^h) = t$.

S2 $r_{l,j}^h = (k+h-1)\cdot p_l$.

S3 $d_{l,j}^h = r_{l,j}^h + p_l$.

S4 $\rho_{l,j}^h = r_{l,j}^h + \kappa \cdot p_l$.

Example 1. To motivate these rules, consider Fig. 8 (a), which depicts the schedule of a sporadic task set with a three-stage pipeline task: $T_1^1 = (2,4)$, $T_1^2 = (2,4)$, and $T_1^3 = (2,4)$ under GEDF or GFIFO (the schedule is the same for both) on a two-processor system. Jobs are released in a sporadic manner. (Recall that "sporadic separations" originate at the first stage, as described in Sec. 2.) Every job in this task set is forced to be scheduled as if it were periodic

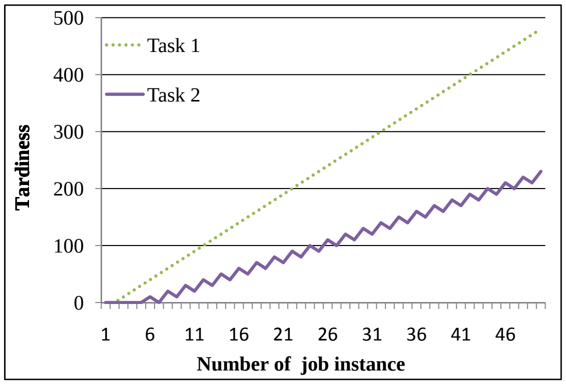


Figure 7. Tardiness growth rates for Counterexample 2 under GEDF.

by applying Rules **S2-S4**. For example, $T_{1,2}^1$ is released at time 6 (which is not a multiple of T_1's period), and hence by S2, its release time is re-set to be $r_{1,2}^1 = 2\cdot 4 = 8$ and its deadline to be $d_{1,2}^1 = 3\cdot 4 = 12$. Fig. 8 (b) shows the schedule that results by forcing periodic behavior without applying Rule **S1** to enable early-releasing. Note that, without early-releasing, a job's execution may be unnecessarily delayed. For instance, $T_{1,2}^1$ is actually released at time 6 and could start execution then. Rule **S1** enables each job to start execution at its early-release time. As shown in Fig. 8 (c), when Rule **S1** is applied, $T_{1,2}^1$ can start execution at time 6, although its priority and deadline are still determined by its newly-set release time, which is time 8. By applying this approach, each job could be delayed for at most one additional period in comparison to the periodic case. Thus, for each task, we must merely increase the tardiness bound given earlier for periodic pipeline task systems by the task's period.

Theorem 2. *With x as defined in Theorem 1, the tardiness for any subtask T_l^h in a sporadic pipeline task system scheduled under GSA (or GEDF or GFIFO) with early-releasing is at most $x + e_l^h + p_l$, where $x = \dfrac{\Gamma + \sum_{T_i\in\tau}\sum_{T_i^k\in T_i} e_i^k + (m-1)e_l^h + me_{max}}{(1-s_{max})\cdot m - U}$, provided $U < (1-s_{max})\cdot m$, where U is sum of the $m(m-1)$ largest subtask utilizations.*

Corollary 3. *If all pipeline tasks are monotonically increasing, then the tardiness for any subtask T_l^h in a sporadic pipeline task system with early-releasing scheduled under GSA is at most $x + e_l^h + p_l$. As in Corollory 1, we require $U_{sum} < m$.*

Corollary 4. *For two-processor systems, the tardiness for any subtask T_l^h in a sporadic pipeline task system with early-releasing scheduled under GSA is at most $x + e_l^h + p_l$ (no utilization constraint other than $U_{sum} \leq m$ is required).*

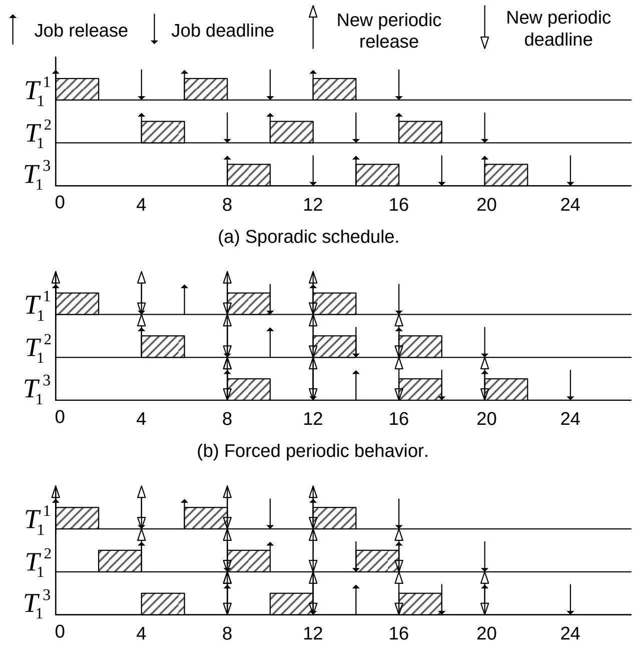


Figure 8. An example of forcing periodic behavior on sporadic workflows.

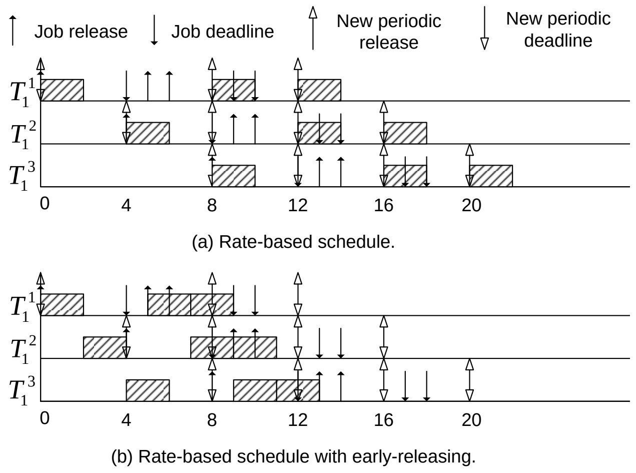


Figure 9. Rate-based schedules.

The above-mentioned early-releasing technique can also be used to support rate-based arrivals. In rate-based task models [11], a minimum inter-arrival separation between jobs is not ensured (which is different from the periodic/sporadic task model where a task's period specifies the exact/minimum inter-arrival separation between any two of its consecutive jobs). We assume that rate-based release patterns originate in first-stage-subtasks. The release time of any non-first-stage job $T^c_{a,b}$ where $c > 1$ is set to be $d^{c-1}_{a,b}$, which is the deadline of $T^c_{a,b}$'s U-predecessor. In order to support rate-based arrivals, if a first-stage job $T^1_{l,j}$ is released at time $t \in ((k-1)\cdot p_l, k\cdot p_l]$, then job $T^h_{l,j}$'s ($h \geq 1$) early-release time, release time, deadline, and priority are modified according to rules, **R1**-**R4**, below.

R1 $\varepsilon(T^h_{l,j}) = t$.

R2 $r^h_{l,j} = \max\{(k+h-1)\cdot p_l, d^h_{l,j-1}\}$.

R3 $d^h_{l,j} = r^h_{l,j} + p_l$.

R4 $\rho^h_{l,j} = r^h_{l,j} + \kappa \cdot p_l$.

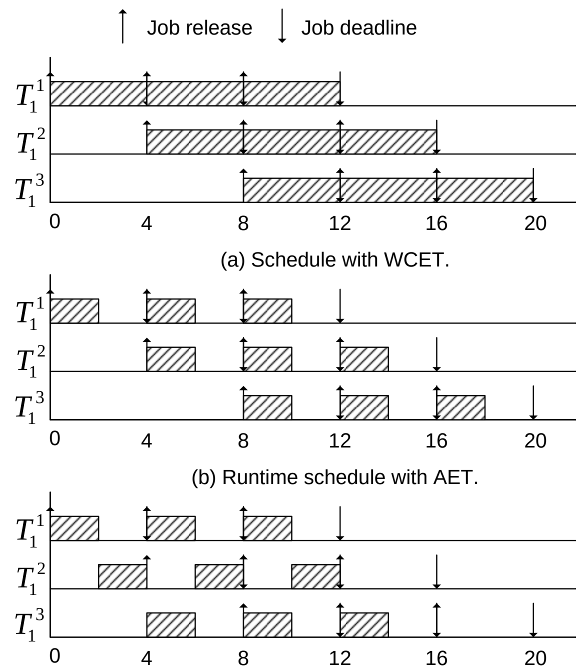
Example 2. Consider Fig. 9 (a), which depicts a schedule of the same task set considered in Eample 1, but with rate-based arrivals. In this task system, $T^1_{1,1}$, $T^1_{1,2}$, and $T^1_{1,3}$ are released at times 0, 5, and 6 respectively. According to rule **R2**, the release times of $T^1_{1,2}$ and $T^1_{1,3}$ are modified to be 8 and 12. Note that, without early-releasing, a job's execution may be unnecessarily delayed. For instance, as shown in Fig. 9 (a), $T^1_{1,3}$ is actually released at time 6 and could start execution then. Rule **R1** enables each job to start execution at its early-release time. As shown in Fig. 9 (b), when Rule **R1** is applied, $T^1_{1,2}$ can start execution at time 5. Observe that job response times can be significantly reduced by early-releasing.

Early-releasing could be very effective for applications where a job's actual execution time (AET) may be substantially less than its worst case execution time (WCET). Whenever a job executes for less than its WCET, early-releasing allows its successive-stage job to receive such unused capacity and start execution immediately.

Example 3. Fig. 10 (a) dipicts a schedule for a three-stage pipeline task, $T^1_1 = (4,4)$, $T^2_1 = (4,4)$, and $T^3_1 = (4,4)$, scheduled under GEDF or GFIFO on a three-processor system. Suppose at runtime, every job executes for only two time units, as shown in Fig. 10 (b). Fig. 10 (c) shows how such jobs are scheduled if early-releasing is enabled.

5 Experimental Evaluation

In this section, we describe the results of several sets of experiments conducted using randomly-generated task sets to evaluate the effectiveness of early-releasing with respect to both deadline tardiness and job response times on sporadic task systems, rate-based task systems, and task systems with various AET/WCET ratios. Due to space constraints, we do not present experimental results that assess the accuracy and the applicability of the tardiness bounds for GFIFO and GEDF derived in this paper, as such results



(a) Schedule with WCET.

(b) Runtime schedule with AET.

(c) Runtime early-releasing schedule with AET.

Figure 10. Impact by early-releasing on systems with runtime execution cost.

have been given earlier [4].

In these experiments, GEDF and GFIFO with and without early-releasing were compared assuming 4, 8, and 16 processors (m). For each m, 1,000 task sets were generated. For each task set, new tasks were added until total utilization exceeded m, and then the last task's utilization was reduced if needed so that total utilization equaled m. The utilization of each subtask was selected uniformly over $[0.01, 0.5]$. The execution cost of each subtask was chosen uniformly over (0,20]. The sporadic job release pattern was defined as follows. For any pipeline task T_l, the first job of its first task, $T_{l,1}^1$, was assumed to be released at time 0. The release time of any subsequent job, $T_{l,k}^1$ where $k > 1$, was chosen uniformly over $[d_{l,k-1}^1, d_{l,k-1}^1 + p_l]$. The release time of any other job, $T_{l,j}^h$ where $h > 1$, was determined based on the system model described in Sec. 2. When generating task sets, we dropped any task set that violates the utilization constraint as stated in Thereom 2. To determine average observed tardiness values, a simulator was used for each scheduling scheme that schedules tasks until time 50,000.

The first set of experiments evaluates the impact early-releasing has on deadline tardiness. In Fig. 11, the average observed tardiness is plotted for $m = 4$, $m = 8$, and $m = 16$, where $U_{sum} = m$. As seen, early-releasing substantially lowered tardiness in all cases.

The second set of experiments was conducted to evaluate the impact of early-releasing on end-to-end pipeline response times in sporadic task systems scheduled under GEDF. The *average response time* (ART) of a pipeline task T_i, denoted rt_i, is defined as

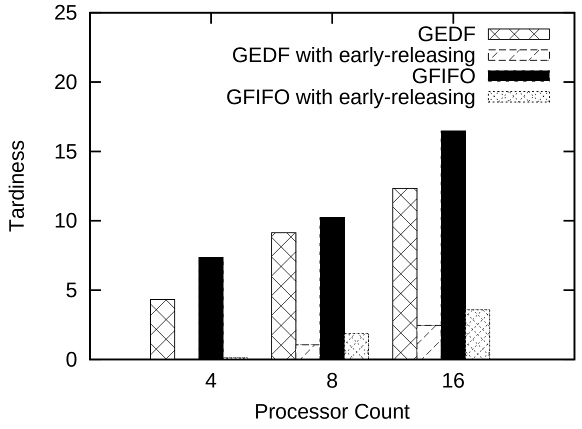


Figure 11. Impact on deadline tardiness by early-releasing .

$$rt_i = \frac{\sum_{k=1}^{L}(CT(T_{i,k}^z) - r_{i,k}^1)}{L},$$

where L is the number of jobs released by any subtask of T_i in the schedule under consideration (note that each such subtask releases the same number of jobs), z is the number of stages of T_i, and $r_{i,k}^1$ is the actual release time of $T_{i,k}^1$ before applying Rule **S2** or **R2**. Let $rt_i(s1)$ denote the ART of T_i scheduled under some scheduling policy A without early-releasing, and let $rt_i(s2)$ denote the ART of T_i scheduled under the same scheduling policy A but with early-releasing. The *average response time improvement* (ARTI) by early-releasing for a task system τ with N pipeline tasks is defined by $\dfrac{\sum_{T_i \in \tau}\left(\dfrac{rt_i(s1) - rt_i(s2)}{rt_i(s2)} \cdot 100\%\right)}{N}$.

Figs. 12-14 depict scatter plots for $m = 4$, $m = 8$, and $m = 16$, respectively, where each point denotes the ARTI for one task set under GEDF. Observe that ARTI can be significant, particularly in lightly-loaded systems. For example, in Fig. 14, when $U_{sum} = 2$ and $m = 16$, ARTI ranges between 400% and 800%. On the other hand, for heavily-loaded systems, early-releasing yields limited improvement.

In the third set of experiments, the impact of early-releasing on job response times in rate-based task systems scheduled under GEDF was evaluated. In a rate-based task system, a job may arrive any time after the release of its L-predecessor. In these experiments, task sets were randomly generated in the same way as mentioned above, except that they obey a rate-based arrival pattern, which was defined as follows. For any pipeline task T_l, the first job of its first

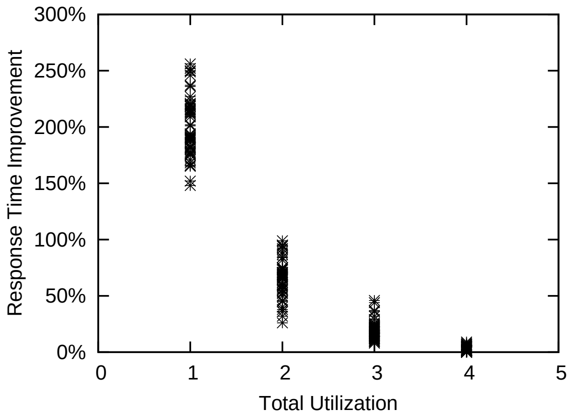


Figure 12. Average response time improvement for sporadic systems on four processors.

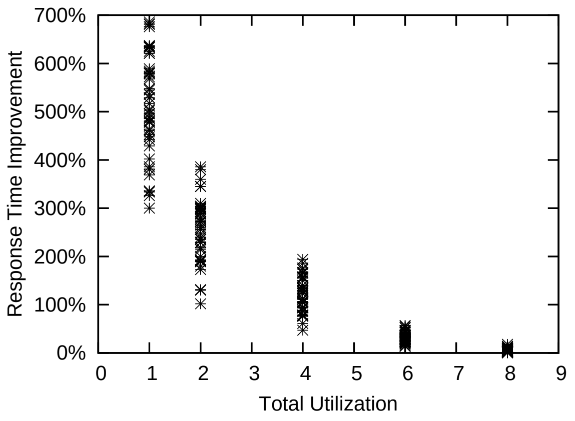


Figure 13. Average response time improvement for sporadic systems on eight processors.

task, $T_{l,1}^1$, was assumed to be released at time 0. The release time of any subsequent job, $T_{l,k}^1$ where $k > 1$, was chosen randomly over $(r_{l,k-1}^1, d_{l,k-1}^1]$ with probability v and over $(d_{l,k-1}^1, d_{l,k-1}^1 + p_l]$ with probability $1 - v$ (thus, "sporadic separations" are more likely if v is lower). For any non-first-stage job, $T_{l,j}^h$, where $h > 1$, $r_{l,j}^h = d_{l,j}^{h-1}$.

Figs. 15-17 depict scatter plots for $v = 25\%$, $v = 50\%$, and $v = 75\%$, respectively, where $m = 8$ and each point denotes the ARTI for one task set under GEDF. These figures suggest that early-releasing can have a significant impact in rate-based task systems. Unlike the sporadic case, significant impact is seen even in heavily-loaded systems. As seen in Figs. 15-17, ARTI can be as high as 120% even if $U_{sum} = m$. Another interesting observation is that as v increases, ARTI does as well. This is because as v increases, the probability of a job being released before the deadline of its L-predecessor also increases. Hence, without early-releasing, the probability of a job being scheduled

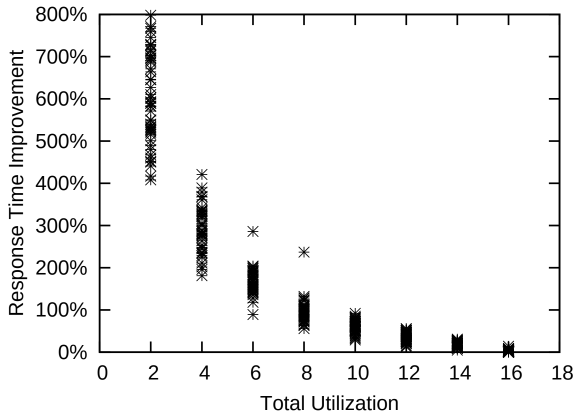


Figure 14. Average response time improvement for sporadic systems on sixteen processors.

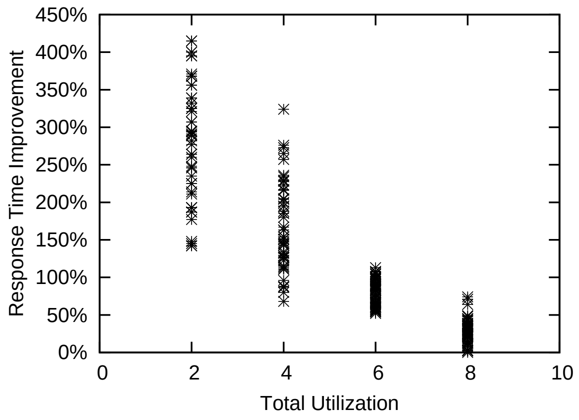


Figure 15. Average response time improvement for rate-based systems on eight processors where $v = 25\%$.

much later is increased, according to Rules **R2**-**R4**. Early-releasing, by Rule **R1**, allows jobs to execute much earlier.

The fourth set of experiments evaluates the impact of early-releasing on job response times in periodic task systems where tasks' AETs may be shorter than their WCETs. In these experiments, task sets were randomly generated in the same way as mentioned above, except that they obey a periodic arrival pattern. We varied the AET/WCET ratio, denoted ω, in order to study the impact of early-releasing with different AET/WCET ratios in periodic task systems. As seen in Fig. 18, when ω equals 25%, 50%, and 75%, ARTIs range within $[190\%, 380\%]$, $[30\%, 100\%]$, and $[10\%, 60\%]$, respectively.

6 Conclusion

We have derived a tardiness bound that can be applied to globally-scheduled sporadic pipeline task systems. This

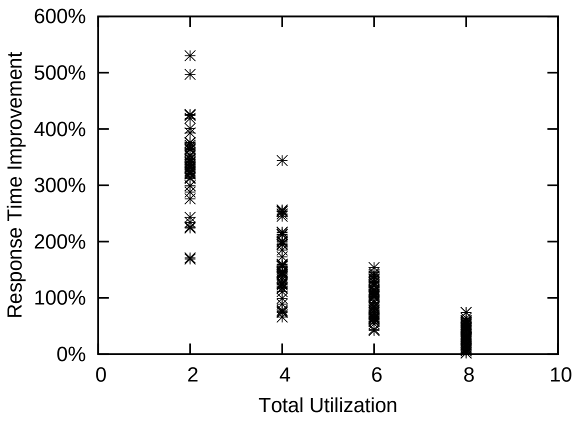


Figure 16. Average response time improvement for rate-based systems on eight processors where $v = 50\%$.

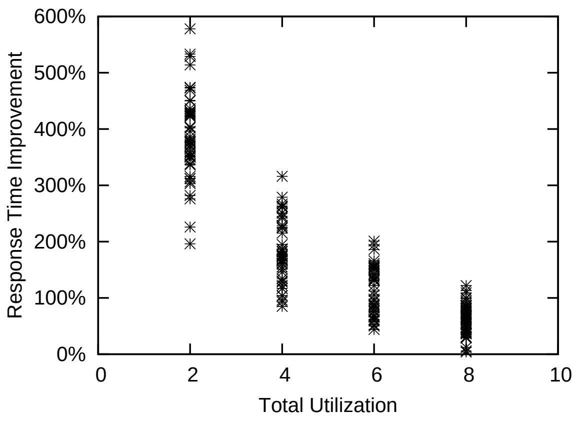


Figure 17. Average response time improvement for rate-based systems on eight processors where $v = 75\%$.

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 system where all pipeline tasks are monotonically increasing and no utilization constraint (other than $U_{sum} \leq m$) is required for any two-processor system. For other systems, utilization constraints are fundamental, as we have shown via counterexamples. Nonetheless, for these systems, the required constraint is quite liberal. To enable the work-conserving scheduling of pipelines, we introduced the technique of early-releasing. Early-releasing can greatly reduce job response times, particularly in lightly-loaded systems, as demonstrated in the presented experimental evaluation.

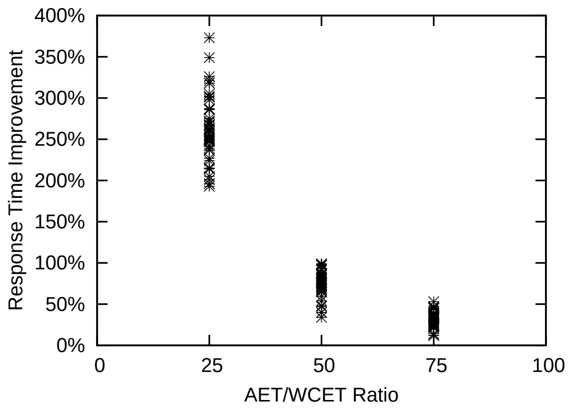


Figure 18. Average response time improvement for periodic systems with various AET/WCET ratios on eight processors where $U_{sum} = 8$.

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] C. Liu and J. H. Anderson. Supporting pipelines in soft real-time multiprocessor systems. In *Proc. of the 21st Euromicro Conf. on Real-Time Systems*, 2009 (to appear).

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] K. Jeffay and S. Goddard. A theory of rate-based execution. In *Proc. of the 20th IEEE Int'l Real-Time Systems Symp.*, pp. 304-314, 1999.