Sanjoy Baruah, Giorgio Buttazzo, Sergey Gorinsky, and Giuseppe Lipari. Scheduling periodic task systems to minimize output jitter. Proceedings of the International Conference on Real-Time Computing Systems and Applications, pp 62-69, Hong Kong. December 1999. IEEE Computer Society Press.

Scheduling periodic task systems to minimize output jitter

Sanjoy Baruah* **Giorgio Buttazzo†** **Sergey Gorinsky‡** **Giuseppe Lipari†**

Abstract

Output jitter — the variation in the inter-completion times of successive jobs of the same task — is studied in the context of the preemptive uniprocessor scheduling of periodic real-time tasks. A formal quantitative model for output jitter is proposed. A list of properties that are desirable in any jitter-minimization schedule is enumerated. Algorithms are presented for generating such schedules, and bounds proved for the maximum jitter in schedules generated by these algorithms.

Keywords. Scheduling: uniprocessor, preemptive; periodic tasks; output jitter.

1 Introduction

In many real-time control applications, periodic activities represent the major computational demand on the system. Such periodic activities typically arise from sensory data acquisition, low-level servoing, control loops, action planning, and system monitoring, which need to be cyclically executed at specific rates (derived from the application requirements). These periodic activities are often formally modelled as *periodic tasks*: each periodic task $T = (e, p)$ is characterized by two parameters – an *execution requirement* e and a *period* p – with the interpretation that T generates an infinite sequence of *jobs* arriving at time-instants $0, p, 2p, 3p, \ldots$, respectively, with the job arriving at time-instant $k \cdot p$ needing to execute for e time units over the interval $[k \cdot p, (k+1) \cdot p)$ (i.e., we assume that each job has a *deadline* p time units after its arrival time). In this paper, we restrict our attention to the uniprocessor, preemptive, scheduling of systems of such tasks — by *uniprocessor*, we mean that all the jobs generated by all the tasks in the system are to execute on a single shared processor; by *preemptive*, we mean that a job executing on the shared processor may be interrupted at any instant in time, and its execution resumed later, with no cost or penalty.

A *schedule* for a system of periodic tasks specifies which task executes on the shared processor at each instant in time. Let $\Gamma = \{T_1, T_2, \ldots, T_n\}$ denote a system of n periodic tasks; $T_i = (e_i, p_i)$ for $1 \leq i \leq n$. With respect to a particular schedule S for Γ, the *completion time* $c_i^{(k)}$ of the k'th job of T_i is the earliest time-instant $\geq k \cdot p_i$ such that T_i has executed for e_i time units in S over the interval $[k \cdot p_i, c_i^{(k)})$. Schedule S is a *correct* schedule for Γ if and only if for all $T_i \in \Gamma$ and for all integer $k \geq 0$, $c_i^{(k)} \leq (k+1) \cdot p_i$; i.e., the completion time of the k'th job of T_i is no larger than the associated deadline.

*Department of Computer Science, The University of North Carolina, Chapel Hill, NC 27599–3175, USA. Email: baruah@cs.unc.edu

†Scuola Superiore S. Anna, Via Carducci, 40, 56100 Pisa, Italy. Email: {giorgio, lipari}@sssup.it

‡Department of Computer Science, The University of Texas, Austin, TX 78712, USA. Email: gorinsky@cs.utexas.edu

The primary goal of a scheduling algorithm is to generate correct schedules whenever possible. That is, a scheduling algorithm for scheduling systems of periodic tasks should accept as input a system Γ of periodic tasks, and attempt to generate as output a correct schedule S for Γ. (In this respect, the Earliest Deadline First scheduling algorithm (EDF) [5, 9] is known to be *optimal*, in the sense that if there exists a correct schedule for a system Γ of periodic tasks, then it is guaranteed that EDF will generate a correct schedule for Γ.) Assuming that this primary goal of meeting all deadlines can be met, we may in general desire that our scheduling algorithms be able to optimize certain *secondary* objectives. One such secondary objective is the focus of this paper: minimizing *output jitter*.

Output jitter refers to the variation between the inter-completion times of successive jobs of the same task. More formally, with respect to a particular (correct) schedule for Γ let

$$p_i^{(\min)} \stackrel{\text{def}}{=} \min_{k \geq 0} \left\{ c_i^{(k+1)} - c_i^{(k)} \right\}$$
$$p_i^{(\max)} \stackrel{\text{def}}{=} \max_{k \geq 0} \left\{ c_i^{(k+1)} - c_i^{(k)} \right\};$$

i.e., $p_i^{(\min)}$ and $p_i^{(\max)}$ denote the minimum and maximum separation between successive completions of T_i, $1 \leq i \leq n$. An *output jitter-free* (henceforth, simply "jitter-free") schedule is one in which $p_i^{(\min)} = p_i^{(\max)}$ (in which case it follows that $p_i^{(\min)} = p_i^{(\max)} = p_i$). Schedules that have as their secondary objective the minimization of output jitter attempt to minimize the variation of $p_i^{(\min)}$ and $p_i^{(\max)}$ from p_i.

In a sense, scheduling to minimize output jitter in periodic task systems is a non-issue. Assuming that the system under study is feasible (i.e., there exist correct schedules for the system), a schedule can be rendered virtually jitter-free by postponing execution of the last ϵ units of each job's execution requirement until its deadline, for some very small $\epsilon \to 0$ (equivalently, by postponing notification of the completion of a job until its deadline — if it completes execution prior to its deadline, it is simply buffered for the time remaining until its deadline is reached). Indeed, this is the approach adopted in [6] to handle the jitter problem in packet-switched networks — each packet is held back in a special buffer by each internal node of the network for the maximum possible delay in that node.

From a real-time systems point of view, however, such a solution is far from ideal. As stated above, EDF is an optimal scheduling algorithm for scheduling systems of periodic real-time tasks. In addition to this desirable property of optimality, EDF possesses many other features that render it particularly suitable for scheduling uniprocessor real-time systems. These include: low scheduling overhead; a reasonable bound on the number of preemptions (no more than twice the number of jobs); satisfying the *integral boundary constraint* (IBC) — if all the execution requirements and periods are integers, then all processor preemptions must take place at integer boundaries only; etc. We would like any jitter-minimization algorithm to possess these properties as well — below, we attempt to identify some of the properties that we would like any jitter-minimization algorithm to satisfy (in addition to the primary goal of meeting all deadlines):

1. The scheme is not allowed to buffer a job that is ready to execute. That is, if a job is ready to be released at time t, we are NOT allowed to delay its release to some time after t.
2. Perhaps equivalently, we will not allow for inserted idle time in the schedule — if there is some job ready to execute in the system, we are NOT allowed to idle the processor.
3. The scheme is not allowed to buffer a job that has completed. That is, if a job has completed execution at time t, we are NOT allowed to hold back from notifying the external world of

this fact.

4. The schedule generated should satisfy the integral boundary constraint (IBC).
5. The scheme should not require too many preemptions. We would like to see no increase in the number of preemptions over current scheduling schemes, in the worst-case. That is, the bound on the number of preemptions proved for the Earliest Deadline First scheduling algorithm (EDF) [9] — the total number of preemptions is no more than twice the number of jobs — should continue to hold.
6. The on-line scheduling overhead should continue to be as small as in the case of EDF.

Informally speaking, what we would like to do is continue to use EDF as our scheduling algorithm, but to perhaps change the "control" information that we use in doing this scheduling. To this end, we describe in this paper a transformation on a system of periodic tasks such that the schedule generated by EDF on the transformed system has better jitter performance than the schedule generated by EDF on the original system, while continuing to be a correct schedule for the original system.

Related work. Considerable research has previously been performed on the topic of jitter in a dynamic-scheduling context. For example, Lin and Herkert [8] have studied a *distance constrained task model* in which the difference between any two consecutive finishing times of the same task is required to be bounded by a specified value. This approach attempts to minimize the outgoing jitter caused by the scheduler. Stankovic and Di Natale [11] have studied jitter-minimization in the context of providing end-to-end timing guarantees in distributed real-time systems. Researchers in network communications have also devoted considerable attention to the phenomenon of jitter. They have considered the problem of providing deterministic timing guarantees in circuit-switched networks where the transmission of a video stream takes a route consisting of several hops, each of which may add to the jitter. Jitter-control schemes to minimize the end-to-end jitter in such networks have been proposed, for example by Ferrari [7] and Verma, Zhang, and Ferrari [6]. The research presented in [1, 3, 4, 12] addresses the problem of accomodating *input* jitter — i.e., scheduling systems of periodic tasks in which the ready-times of jobs cannot be predicted exactly a priori, but are subject to timing uncertainties at runtime.

Organization. The remainder of this paper is organized as follows. In Section 2, we formally state our model of periodic tasks, and provide a quantitative definition of output jitter. In Section 3, we describe a polynomial-time algorithm for transforming a periodic task system, such that the transformed task system can be scheduled by EDF to generate a schedule that exhibits significantly less jitter than would be exhibited by EDF-scheduling the original task system. In Section 4, we describe a *pseudo*polynomial-time algorithm for transforming a periodic task system into another that can be scheduled by EDF with less jitter — although this algorithm is computationally more expensive than the one in Section 3, it can in general guarantee better jitter behaviour. We have implemented both our jitter-control algorithms, and have performed experiments testing their effectiveness — we describe these results in Section 5. In Section 6, we illustrate our techniques by tracing their behaviour on some simple periodic task systems.

2 System model

As stated in the introduction, we are concerned here with the problem of minimizing output jitter in EDF-generated schedules of systems of periodic tasks. We will assume that each periodic task T_i

is characterized by a *jitter tolerance factor* ϕ_i, with the interpretation that tasks with large ϕ_i are more tolerant to jitter, in addition to the execution requirement e_i and the period p_i. Let Γ denote a periodic task system consisting of n tasks $T_1, T_2, \ldots, T_n$. Let $\rho_i \stackrel{\text{def}}{=} e_i/p_i$, and $\rho \stackrel{\text{def}}{=} \sum_{i=1}^{n} \rho_i$. It has been proved [9] that a necessary an sufficient condition for EDF to generate a correct schedule for Γ is that ρ be at most one.

With respect to a particular schedule for Γ, let $p_i^{(\min)}$ and $p_i^{(\max)}$ denote the minimum and maximum separation between successive completions of T_i, $1 \leq i \leq n$. The quantity

$$\mathsf{AbsJitter}(T_i) \stackrel{\text{def}}{=} \max\left(p_i^{(\max)} - p_i, p_i - p_i^{(\min)}\right)$$

is defined to be the (absolute) **jitter** of task T_i; observe that this can also be computed as $\max_{k \geq 0} |(c_i^{(k+1)} - c_i^{(k)}) - p_i|$; i.e., $\mathsf{AbsJitter}(T_i)$ measures the maximum variation of the intercompletion time from the task period. In the remainder of this paper, we will make use of the following inequality:

$$\mathsf{AbsJitter}(T_i) \leq \max_{k \geq 0}\{c_i^{(k)} - k \cdot p_i\} \quad - \quad \min_{k \geq 0}\{c_i^{(k)} - k \cdot p_i\} \tag{1}$$

i.e., the difference between the largest response time[1] and the smallest response time of a job of T_i is an upper bound on the variation of intercompltion times of jobs of T_i. (In general, this bound need not be tight: consider, for example, a task T_i with period 5 and executio requirement 1, and a schedule in which $c_i^{(0)} = 1$, $c_i^{(1)} = 7$, $c_i^{(2)} = 13$, $c_i^{(3)} = 19$, and $c_i^{(k)} = 5k + 5$ for all $k \geq 4$. While $\mathsf{AbsJitter}(T_i)$ for this schedule is 1, the bound $\max_{k \geq 0}\{c_i^{(k)} - k \cdot p_i\} - \min_{k \geq 0}\{c_i^{(k)} - k \cdot p_i\}$ equals $5 - 1 = 4$.)

The (absolute) jitter of task system Γ is defined to be largest absolute jitter of any task in Γ:

$$\mathsf{AbsJitter}(\Gamma) \stackrel{\text{def}}{=} \max_{T_i \in \Gamma}\{\mathsf{AbsJitter}(T_i)\}$$

The **weighted jitter** of T_i is defined to be

$$\mathsf{WtdJitter}(T_i) \stackrel{\text{def}}{=} \frac{\mathsf{AbsJitter}(T_i)}{\phi_i} \ .$$

The **weighted jitter** of Γ is defined to be

$$\mathsf{WtdJitter}(\Gamma) \stackrel{\text{def}}{=} \max_{T_i \in \Gamma}\{\frac{\mathsf{AbsJitter}(T_i)}{\phi_i}\} \ . \tag{2}$$

The goal of this research is to devise scheduling algorithms that accept as input periodic task systems Γ, and generate schedules for Γ that meet all deadlines *and* attempt to minimize $\mathsf{WtdJitter}(\Gamma)$.

The general model described above can represent several interesting forms of jitter which may be of relevance to the application system designer. For instance, if $\phi_i = 1$ for all i, then $\mathsf{WtdJitter}$ is the same as $\mathsf{AbsJitter}$. Setting $\phi_i = p_i$ for all i in Equation 2, we obtain the notion of **relative jitter** — the jitter of a task is measured as a fraction of its period (as an example, a schedule with a relative jitter of 30 % guarantees that the intercompletion times between successive jobs of T_i is at least $0.7p_i$, and at most $1.3p_i$, for all i). Similarly, we can model systems in which only a subset of the tasks are jitter-sensitive by setting the jitter-tolerance factor for the other (non jitter-sensitive) tasks to infinity.

[1] The *response time* of the k'th job of T_i is defined to be the difference between the completion time of the job — $c_i^{(k)}$ — and its release time — $k \cdot p_i$.

3 Jitter control in polynomial time

3.1 Computing jitter bounds

Since a job of T_i completes no sooner than e_i time units after its ready time and no later than its deadline in any correct schedule for Γ, it is easily seen that e_i and $2p_i - e_i$ are "quick-and-dirty" bounds on the maximum and minimum intercompletion times respectively of T_i; i.e., the jitter for task T_i is clearly $\leq (p_i - e_i)$; equivalently,

$$\mathsf{WtdJitter}(\Gamma) \leq \max_{T_i \in \Gamma}\{\frac{p_i - e_i}{\phi_i}\} = \max_{T_i \in \Gamma}\{\frac{e_i}{\phi_i}(\frac{1}{\rho_i} - 1)\} \tag{3}$$

With a little more effort, we can compute tighter bounds for the case when ρ – the system utilization – is less than 1:

Theorem 1 In an EDF-generated schedule for Γ, the separation between the completion times of successive jobs of task T_i lies in the interval

$$[p_i - e_i \cdot (\rho/\rho_i - 1), p_i + e_i \cdot (\rho/\rho_i - 1)] \ ;$$

i.e., $\mathsf{AbsJitter}(T_i) \leq e_i \cdot (\frac{\rho}{\rho_i} - 1)$; equivalently, $\mathsf{WtdJitter}(T_i) \leq \frac{e_i}{\phi_i}(\frac{\rho}{\rho_i} - 1)$.

Proof: Consider a scenario where the k'th job of T_i arrives at time t_o and completes at the earliest possible time, the $(k+1)$'th job completes at the latest possible time, and the $(k+2)$'th job completes, once again, at the earliest possible time.

- The completion time of the k'th job is at least $t_o + e_i$.
- Let the completion time of the $(k+1)$'th job be t_1, and let $[t_o + 2p_i - \ell, t_1]$ denote the longest contiguous interval preceding t_1 during which the processor only executes jobs with deadlines at or before $t_o + 2p_i$. Since all such jobs have arrival-times $\geq t_o + 2p_i - \ell$, and deadlines $\leq t_o + 2p_i$, and they all complete by t_1, it follows that $t_1 - (t_o + 2p_i - \ell)$ is at most $\rho \cdot \ell$. Hence, $t_1 \leq t_o + 2p_i - \ell(1-\rho)$; since ℓ is at least p_i, this implies that

$$t_1 \leq t_o + 2p_i - p_i(1-\rho) \ .$$

- The completion time of the $(k+2)$'th job is, once again, at least $t_o + 2p_i + e_i$.

The maximum intercompletion time, as defined by the difference between the completion times of the $(k+1)$'th and k'th jobs, is thus at most

$$\begin{aligned} & t_o + 2p_i - p_i(1-\rho) - (t_o + e_i) \\ = \ & 2p_i - p_i(1-\rho) - e_i \\ = \ & p_i(1+\rho) - e_i \\ = \ & p_i + p_i\rho - e_i \\ = \ & p_i + e_i(\frac{p_i}{e_i}\rho - 1) \\ = \ & p_i + e_i(\frac{\rho}{\rho_i} - 1) \end{aligned}$$

Similarly, the minimum intercompletion time, as defined by the difference between the completion times of the $(k+2)$th and the $(k+1)$'th jobs, is at least

$$t_o + 2p_i + e_i - (t_o + 2p_i - p_i(1-\rho))$$

$$
\begin{aligned}
&= e_i + p_i(1-\rho) \\
&= p_i - (p_i\rho - e_i) \\
&= p_i - e_i(\frac{p_i}{e_i}\rho - 1) \\
&= p_i - e_i(\frac{\rho}{\rho_i} - 1)
\end{aligned}
$$

■

Thus, given a system Γ of periodic tasks, we can determine an upper bound on its WtdJitter by the following relationship:

$$\mathsf{WtdJitter}(\Gamma) \leq \max_{T_i \in \Gamma} \left\{ \frac{e_i}{\phi_i}(\frac{\rho}{\rho_i} - 1) \right\} \tag{4}$$

Is this bound tight? Unfortunately, no. Tighter bounds can be obtained by actually simulating EDF on the given task-set, or by making use of the actual parameters — the execution requirements and periods — of all the tasks in the system. However, such approaches take exponential (or at least pseudo-polynomial) time as opposed to the computation suggested by the above relationship, which takes time linear in the number of tasks.

3.2 Jitter-minimization in polynomial time

In attempting to decrease $\mathsf{AbsJitter}(\Gamma)$, we can transform task system Γ by *increasing* the processor-share reserved for the use of each task T_i to $\theta_i \geq \rho_i$ (while ensuring that $\sum_{i=1}^{n} \theta_i \leq 1$ holds). Having done so, we can now "assign" a new deadline to the job of T_i, released at $k \cdot p_i$, to be $k \cdot p_i + (e_i/\theta_i)$ — since $\theta_i \geq \rho_i$, this assigned deadline will be no larger than the job's actual deadline at $(k+1) \cdot p_i$. These assigned deadlines are the ones that will be used by EDF is scheduling Γ, and it is guaranteed that each job will complete by its new assigned deadline: this follows from the fact [5] that EDF is an *optimal* uniprocessor scheduling algorithm in the sense that if there is a correct schedule for a set of jobs, then EDF will generate a correct schedule for this set of jobs. The existence of a correct schedule for the transformed system follows from the observation that the task T_i can be assigned a fraction θ_i of the processor during each time-slot — over the interval $[kp_i, kp_i + \frac{e_i}{\theta_i})$, therefore, T_i is assigned the processor for a total of $\frac{e_i}{\theta_i} \cdot \theta_i = e_i$ slots. Indeed, if the input parameters — the e_i's and the p_i's — are integers, it is easily seen that the "assigned" deadline of the k'th job of T_i can in fact be set equal to $\lfloor k \cdot p_i + (e_i/\theta_i) \rfloor$.

With this new assigned deadline, it can be shown (using techniques virtually identical to the ones used in the proof of Theorem 1) that the minimum and maximum separation between successive jobs is now bounded by

$$p_i \pm e_i \cdot (\theta/\theta_i - 1) \ , \tag{5}$$

where $\theta \stackrel{\text{def}}{=} \sum_{i=1}^{n} \theta_i$. We therefore conclude the following:

Theorem 2 Let Γ be a periodic task system with processor-shares $(\theta_1, \theta_2, \ldots, \theta_n)$. The maximum output jitter of Γ is bounded as follows:

$$\mathsf{WtdJitter}(\Gamma) \leq \max_{T_i \in \Gamma} \{ \frac{e_i}{\phi_i} \cdot (\frac{\theta}{\theta_i} - 1) \} \tag{6}$$

■

In order to minimize the weighted jitter experienced by Γ, we will attempt to choose the processor-shares $(\theta_1, \theta_2, \ldots, \theta_n)$ such that $\max_{T_i \in \Gamma} \{(e_i/\phi_i) \cdot (\theta/\theta_i - 1)\}$ is minimized. Of course,

any such choice will have to preserve $\theta \leq 1$ (the total processor-shares allocated does not exceed unity) and $\theta_i \geq \rho_i$ for each i (the processor-share allocated is at least as much as the amount the task needs). In addition, the resulting schedule should satisfy the list of desirable properties for jitter-control schemes enumerated in the Introduction.

The above formulation represents the problem as an optimization problem. We will now prove some properties about optimal solutions. But first, a definition:

Definition 1 *Processor shares* $(\theta_1, \theta_2, \ldots, \theta_n)$ *are defined to be* **optimal processor shares** *for task system* $\Gamma = \{T_1, T_2, \ldots, T_n\}$ *if and only if the following conditions hold:*

P1: *For each* i, $1 \leq i \leq n$, $\theta_i \geq \rho_i$.

P2: $\theta \leq 1$, *where* $\theta \stackrel{def}{=} \sum_{i=1}^{n} \theta_i$.

P3: *For any processor shares* $(\theta'_1, \theta'_2, \ldots, \theta'_n)$ *such that*

1. *for each* i, $1 \leq i \leq n$, $\theta'_i \geq \rho_i$, *and*
2. $\theta' \leq 1$, *where* $\theta' \stackrel{def}{=} \sum_{i=1}^{n} \theta'_i$,

it is the case that

$$\max_{T_i \in \Gamma} \{(e_i/\phi_i) \cdot (\theta/\theta_i - 1)\} \leq \max_{T_i \in \Gamma} \{(e_i/\phi_i) \cdot (\theta'/\theta'_i - 1)\}$$

Lemma 1 If $(\theta_1, \theta_2, \ldots, \theta_n)$ are optimal processor shares for Γ, then so are $(\theta_1/\theta, \theta_2/\theta, \ldots, \theta_n/\theta)$, where $\theta \stackrel{\text{def}}{=} \sum_{j=1}^{n} \theta_j$. (Observe that $\sum_{j=1}^{n}(\theta_j/\theta)$ equals one.)

Proof: Let $\hat{\theta}_i \stackrel{\text{def}}{=} (\theta_i/\theta)$, and let $\hat{\theta} \stackrel{\text{def}}{=} \sum_{i=1}^{n} \hat{\theta}_i$. We will prove below that $(\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_n)$ satisfies the properties enumerated above for optimal processor shares.

P1: Since $\theta \leq 1$, it follows that $\hat{\theta}_i = (\theta_i/\theta) \geq \theta_i$. Therefore, $\hat{\theta}_i \geq \rho_i$.

P2: $\hat{\theta} = (\sum_{i=1}^{n} \hat{\theta}_i) = (\sum_{i=1}^{n}(\theta_i/\theta)) = ((\sum_{i=1}^{n} \theta_i)\theta) = (\theta/\theta) = 1$.

P3: Since $\hat{\theta} = 1$, it follows that $(\hat{\theta}/\hat{\theta}_i) = (1/\hat{\theta}_i) = (1/(\theta_i/\theta)) = \theta/\theta_i$. Therefore,

$$\frac{e_i}{\phi_i}(\frac{\hat{\theta}}{\hat{\theta}_i} - 1) = \frac{e_i}{\phi_i}(\frac{\theta}{\theta_i} - 1)$$

for all i, from which it follows that

$$\max_{T_i \in \Gamma} \left\{ \frac{e_i}{\phi_i}(\frac{\hat{\theta}}{\hat{\theta}_i} - 1) \right\} = \max_{T_i \in \Gamma} \left\{ \frac{e_i}{\phi_i}(\frac{\theta}{\theta_i} - 1) \right\}$$

■

Recall that our goal was to determine optimal processor shares, and hence to be able to generate EDF schedules which minimize output jitter. Lemma 1 above suggests that we can narrow our search to only those processor shares that sum to one; i.e., we need only look for $(\theta_1, \theta_2, \ldots, \theta_n)$ for which $\sum_{i=1}^{n} \theta_i = 1$.

But suppose an additional goal was to determine optimal processor shares in which the cumulative processor shares assigned to all the tasks — i.e., the quantity θ — were as small as possiblle. Lemma 2 below suggests how we can obtain such optimal processor shares from one in which $\theta = 1$.

Lemma 2 Let $(\theta_1, \theta_2, \ldots, \theta_n)$ be optimal processor shares, with $\sum_{i=1}^{n} \theta_i = 1$. Let $\lambda \stackrel{\text{def}}{=} \max_{i=1}^{n}(\rho_i/\theta_i)$. Then $(\lambda \cdot \theta_1, \lambda \cdot \theta_2, \ldots, \lambda \cdot \theta_n)$ are optimal processor shares.

Proof: Let $\hat{\theta}_i \stackrel{\text{def}}{=} (\theta_i \cdot \lambda)$, and let $\hat{\theta} \stackrel{\text{def}}{=} \sum_{i=1}^{n} \hat{\theta}_i$. We will prove below that $(\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_n)$ satisfies the properties enumerated above for optimal processor shares.

P1: Since $\lambda \geq \frac{\rho_i}{\theta_i}$, it follows that $\hat{\theta}_i = (\theta_i \cdot \lambda) \geq \theta_i \cdot \frac{\rho_i}{\theta_i}$. Therefore, $\hat{\theta}_i \geq \rho_i$.

P2: Observe first that $\lambda \leq 1$. Now, $\hat{\theta} = (\sum_{i=1}^{n} \hat{\theta}_i) = (\sum_{i=1}^{n}(\theta_i \cdot \lambda)) = (\lambda \cdot (\sum_{i=1}^{n} \theta_i)) = \lambda$. Therefore $\hat{\theta} \leq 1$.

P3: $(\hat{\theta}/\hat{\theta}_i) = (\lambda/(\theta_i \cdot \lambda)) = (1/\theta_i) = \theta/\theta_i$. Therefore,

$$\frac{e_i}{\phi_i}(\frac{\hat{\theta}}{\hat{\theta}_i} - 1) = \frac{e_i}{\phi_i}(\frac{\theta}{\theta_i} - 1)$$

for all i, from which it follows that

$$\max_{T_i \in \Gamma} \left\{ \frac{e_i}{\phi_i}(\frac{\hat{\theta}}{\hat{\theta}_i} - 1) \right\} = \max_{T_i \in \Gamma} \left\{ \frac{e_i}{\phi_i}(\frac{\theta}{\theta_i} - 1) \right\}$$

■

Lemma 1 suggests that we can look for an optimal solution with the processor-shares assigned the various tasks summing to one. Should we be looking for an optimal solution that uses the minimum possible processor-shares, Lemma 2 tells us how we can get to such a solution from a solution in which the processor-shares assigned equals one.

We now describe how we can use the above results to determine processor-shares for a given task system Γ, such that these processor shares yield the best jitter bounds obtainable using our method. Suppose that $\mathsf{WtdJitter}(\Gamma) \leq J$ when the processor-shares are $(\lambda_1, \lambda_2, \ldots, \lambda_n)$. By Equation 6, we have for all i, $1 \leq i \leq n$,

$$\left(\frac{e_i}{\phi_i}(\frac{1}{\lambda_i} - 1) \leq J\right) \equiv \left(\frac{1}{\lambda_i} \leq \frac{J\phi_i}{e_i} + 1\right) \equiv \left(\lambda_i \geq \frac{e_i}{e_i + J\phi_i}\right) \ .$$

Since we also require $\lambda_i \geq \rho_i$, we have

$$\lambda_i \geq \max\left(\rho_i, \frac{e_i}{e_i + J\phi_i}\right) \ . \tag{7}$$

Hence, to determine whether a given J can be a value for $\mathsf{WtdJitter}(\Gamma)$, we can compute each λ_i according to Equation 7 above, and ensure that $\sum_{i=1}^{n} \lambda_i$ does not exceed one. If the λ_i's sum to at most one, then this value of $\mathsf{WtdJitter}$ is achievable; if not, then a jitter bound of J cannot be achieved using this particular method.

We can use a binary search to determine the tightest jitter bound obtainable by this method. Let J_{init} denote the initial jitter bound yielded by Inequality 4 for the system:

1. $J_{\mathsf{high}} \leftarrow J_{\mathsf{init}}$
2. $J_{\mathsf{low}} \leftarrow 0$
3. repeat
4. $\quad J_{\mathsf{mid}} \leftarrow (J_{\mathsf{high}} + J_{\mathsf{low}})/2$

5. Compute the shares λ_i, $1 \leq i \leq n$, according to Equation 7.
6. if $(\sum_{i=1}^{n} \lambda_i \leq 1)$ $J_{\mathsf{high}} \leftarrow J_{\mathsf{mid}}$
7. else $J_{\mathsf{low}} \leftarrow J_{\mathsf{mid}}$
8. until $(J_{\mathsf{high}} - J_{\mathsf{low}} \leq \mathsf{thres})$

(where thres is defined to be the degree of accuracy desired.)

4 Jitter control in pseudopolynomial time

A further generalization to the periodic task model has each task $T = (e, d, p)$ characterized by a **relative deadline** $d \leq p$ in addition to the period p and the execution requirement e. The issue of feasibility determination for a system of such tasks has been studied [2]; for *bounded-utilization* task systems (systems where the ratios e/p of the execution-requirement to period of all the tasks sum to no more than a constant strictly less than one), this can be efficiently done in time pseudo-polynomial in the size of the problem instance.

Consider periodic task system $\Gamma = \{T_i = (e_i, p_i)\}_{i=1}^{n}$. If we assign a relative deadline $d_i = e_i + J\phi_i$ to each task T_i, *and the resulting task system* $\Gamma' = \{T_i' = (e_i, e_i + J\phi_i, p_i)\}_{i=1}^{n}$ *is feasible*, then the maximum separation between successive completions of T_i' is at most $p_i + J\phi_i$ while the minimum separation is at least $p_i - J\phi_i$, for each i, $1 \leq i \leq n$. Hence to obtain a run-time schedule of Γ with a maximum weighted jitter of J, it suffices to

Step 1: determine whether Γ' is feasible,

Step 2: if so, then schedule Γ' rather than Γ at run-time.

Using the method of [2], Step 1 can be done in time

$$\mathcal{O}(\log n \cdot \frac{\rho}{1-\rho} \cdot \max_{i=1}^{n}\{p_i - d_i\}) \ . \tag{8}$$

This bound is pseudo-polynomial in the size of the problem instance. However, we expect to apply jitter-minimization primarily to systems with a relatively small value of ρ (intuitively speaking, this is because systems with a large value of ρ are quite constrained and we have less freedom to modify parameters to obtain better jitter behavior); consequently, we would expect the time bound of (8) to be quite reasonable in practice. For example, a ρ of 0.9 would permit a processor utilization of 90 %, yet allow feasibility-analysis to be completed in time proportional to nine times the largest period, multiplied by the binary logarithm of the number of tasks.

In general, the jitter-minimization procedure of this section yields a tighter bound than the procedure in Section 3.2, in the sense that the smallest jitter bound which this method can guarantee is smaller than the smallest bound obtainable using the processor-share method of Section 3.2. The downside is that, due to the pseudo-polynomiality of (8), the time-complexity is no longer polynomial. A reasonable approach in practice is to first execute the method of Section 3.2, and use the bound $\hat{J}$ thus obtained as an upper bound for performing binary search between 0 and $\hat{J}$ to determine the smallest J which passes the pseudo-polynomial feasibility test of this section.

A note on implementation. The feasibility test from [2] essentially consists of simulating the behavior of EDF on the periodic task system starting at time zero, and terminating when either (i) a deadline is missed, at which point the system is known to be infeasible, or (ii) the processor is idled, which would imply that the system is feasible. (See [2] for a proof of correctness.) The bound of (8) follows from the fact (proved in [2]) that one of these two events — a missed deadline or an idle slot — must occur within the first $\frac{\rho}{1-\rho} \cdot \max_{i=1}^{n}\{p_i - d_i\}$ time units.

Implementing this jitter-minimization procedure therefore reduces to implementing EDF for periodic task systems, for which very efficient techniques are known [10].

A note on initial offsets. The feasibility of a periodic task system in which each task is characterized by an execution requirement and a period (no relative deadlines) is unaffected by the presence of *initial offsets* – i.e., if each task T_i is characterized by an additional offset parameter a_i with the interpretation that the k'th job of T_i arrives at time $a_i + k \cdot p_i$, and must receive service by $a_i + (k+1) \cdot p_i$, for all integer $k \geq 0$. The jitter-minimization procedure of Section 3.2 enjoys the same property — jitter analysis is unaffected by whether tasks have initial offsets or not.

With respect to the jitter minimization procedure of this section, a slightly weaker statement can be made. If the method outlined above guarantees a jitter bound of J, then this jitter bound is obtainable even in the presence of offsets. However, a system with offsets can sometimes be scheduled with far smaller jitter than can be shown using this method. Attempting to incorporate offsets into the methodology of this section involves feasibility-analysis of periodic task systems in which each task is characterized by an offset, an execution requirement, a relative deadline, and a period; unfortunately, this feasibility-analysis problem has been shown [2] to be co-NP-complete in the strong sense, indicating that we are unlikely to be able to perform it in pseudopolynomial time. We have therefore not attempted to take advantage of initial offsets to reduce jitter even when these offsets are present.

In related ongoing research, we are however attempting to minimize jitter in task-systems of the model in [9] (where each task is characterized by execution requirements and periods only), by introducing offsets to aid in jitter-control when application system semantics would permit us to do so. The basic idea is to use the methodology proposed in this paper to obtain an initial jitter bound, and then to use heuristics to search for phasings which may further reduce the jitter. Although such an approach would likely prove intractable in the worst case and may sometimes result in no improvement in jitter-behaviour, we expect that substantial improvement may be obtained on average.

5 Experimental evaluation

Given a system Γ of periodic tasks that are to be scheduled using EDF, Equation 3 represents a naive bound on the jitter experienced by the system — $\mathsf{WtdJitter}(\Gamma) \leq \max_{T_i \in \Gamma}\{\frac{e_i}{\phi_i}(\frac{1}{\rho_i} - 1)\}$. In Theorem 1 (Equation 4), we derived a somewhat tighter bound for systems where the utilization ρ is strictly less than one — $\mathsf{WtdJitter}(\Gamma) \leq \max_{T_i \in \Gamma}\{\frac{e_i}{\phi_i}(\frac{\rho}{\rho_i} - 1)\}$. Then in Sections 3 and 4, we presented algorithms for generating EDF-schedules that exhibit lower jitter than implied by these bounds — the technique in Section 3 takes time polynomial in the representation of Γ, while the one in Section 4 takes pseudopolynomial time but may in general result in schedules with better jitter behaviour.

We have conducted a series of experiments to evaluate the effectiveness of our jitter-minimization schemes. We have implemented both our jitter-minimization schemes and have used them to minimize the relative jitter[2] of randomly-generated systems of periodic tasks — some of the results are tabulated in Figures 1–5. In each figure, the system jitter $\mathsf{WtdJitter}(\Gamma)$ is plotted as a function of the total system utilization (also known as system *load*). For each value of the system load, we generated 900 task sets. Each task set was generated by randomly choosing the tasks' computation

[2]Recall that the *relative jitter* of a periodic task measures its jitter as a fraction of its period, and is obtained by setting $\phi_i = p_i$.

times as integers between 1 and 10, and then randomly choosing the periods such that the total system load be approximately equal to the desired load.

For each set we calculated the bound obtained by Equation 4 (the line labelled "Bound" in the figures); then we applied both the polynomial-time transformation of Section 3 (labelled "Method 1" in the figures) and the pseudopolynomial-time transformation of Section 4 (labelled "Method 2" in the figures). Only average values are depicted in the figures. (For each point we also calculated the 90% confidence interval — these are not shown in the figures, but were small enough to confirm the validity of our results.) The five figures Figures 1–5 represent the results of five different sets of experiments:

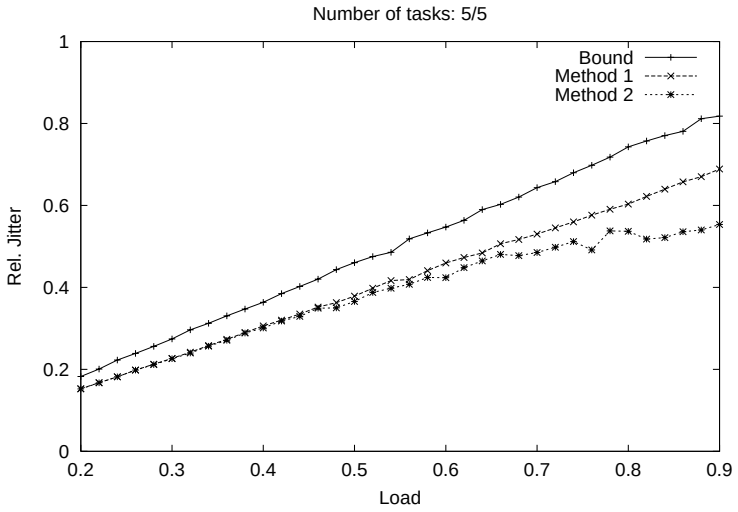

Figure 1: Five tasks – minimize jitter of the entire task set

Figure 1: Γ consists of five tasks, and the goal is to minimize the relative jitter over the complete task set. This is obtained by setting $\phi_i = p_i$ for all i.

Figure 2: Γ consists of ten tasks, and the goal is to minimize the relative jitter over the complete task set. This is obtained by setting $\phi_i = p_i$ for all i.

Figure 3: Γ consists of ten tasks, and the goal is to minimize the relative jitter of just one of the tasks. This is obtained by setting $\phi_1 = p_1$, and $\phi_i = \infty$ for all $i > 1$.

Figure 4: Γ consists of ten tasks, and the goal is to minimize the relative jitters of three of the tasks. This is obtained by setting $\phi_i = p_i$ for $i = 1, 2, 3$, and $\phi_i = \infty$ for all $i > 3$.

Figure 5: Γ consists of ten tasks, and the goal is to minimize the relative jitters of five of the tasks. This is obtained by setting $\phi_i = p_i$ for $i = 1, 2, \ldots, 5$, and $\phi_i = \infty$ for all $i > 5$.

Among the conclusions we are able to draw from our simulations:

- For low system loads, EDF schedules with reasonably low relative jitter can be obtained by both Method 1 and Method 2. As system load increases, Method 2 generally outperforms Method 1.
- If the relative jitters of *all* tasks is to be minimized (as in Figures 1 and 2), the methods presented in this paper are not very effective. In fact, if the system load is low, the results

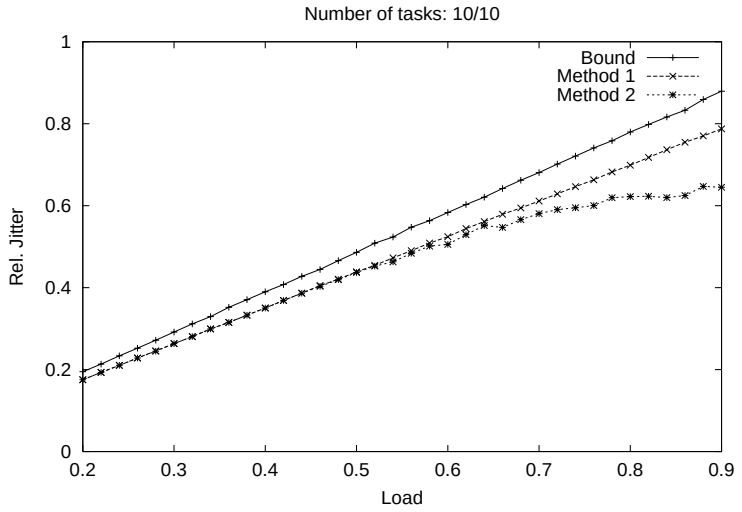

Figure 2: Ten tasks – minimize jitter of the entire task set

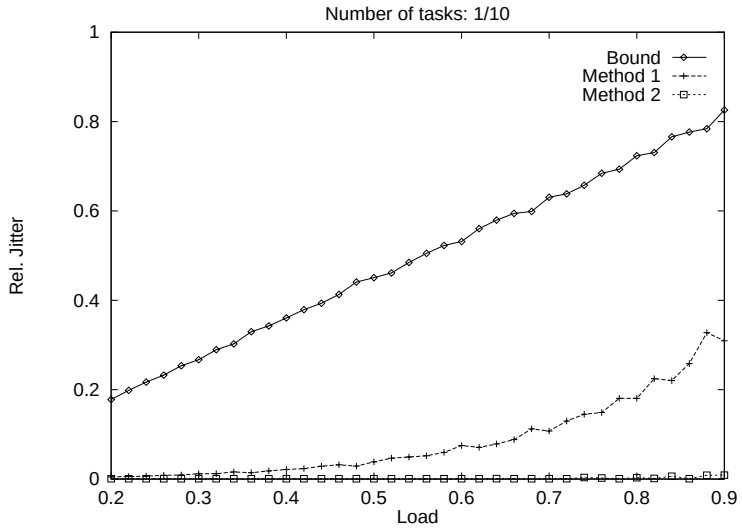

Figure 3: Ten tasks – minimize jitter of one task

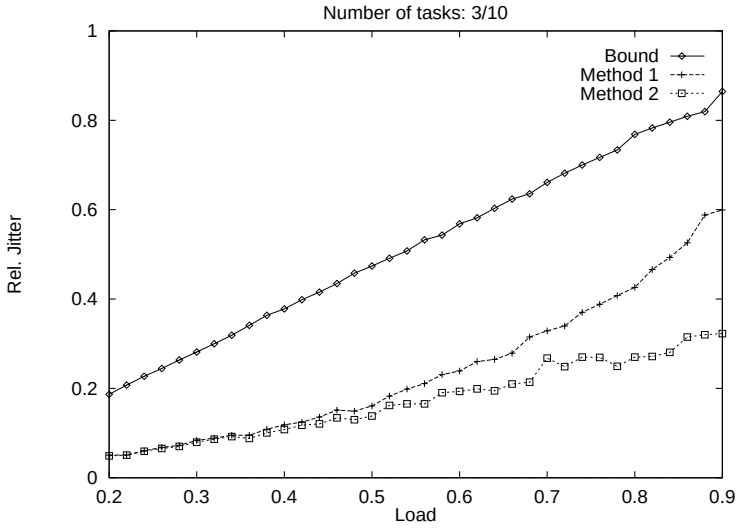

Figure 4: Ten tasks – minimize jitter of three tasks

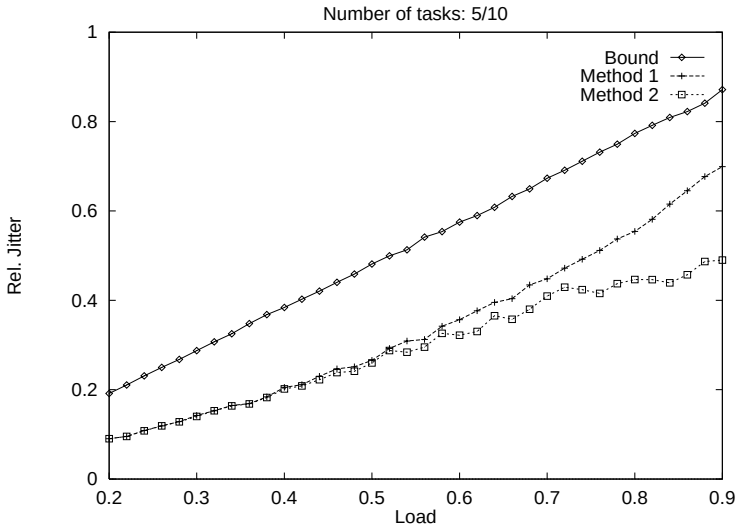

Figure 5: Ten tasks – minimize jitter of five tasks

obtained by both methods are very close to the initial bound (which is already low); if the system load is high, Method 2 can obtain some improvement over the initial bound; however the jitter remains high.

This is not surprising — intuitively speaking, minimizing the jitter of an entire task system is a highly-constrained optimization problem. Indeed, decreasing the jitter of some task can significantly increase the jitter of other tasks in the system.

- As Figures 3–5 illustrate, jitter minimization is more effective when just a few of the tasks are jitter-sensitive. Intuitively speaking, this is because, once the primary goal of meeting deadlines have been satisfied, all the freedom remaining in the schedule can be devoted to minimizing the jitters of a few tasks. Especially in the case in which we wish to minimize the jitter of only one task, Method 2 seems to provide a very low relative jitter (Figure 3) regardless of the overall system load. This is a potentially useful lesson in application system design — jitter minimization is a precious resource, which should be attempted only upon those periodic tasks that really need it.
- Although not illustrated by Figures 1–5, another noteworthy observation is that it is, in general, easier to minimize relative jitter of tasks with low utilization. Once again, this lends itself to an intuitive explanation: increasing the processor share of T_i k times (i.e., $\theta_i \leftarrow k \cdot \rho_i$), while keeping every other task's processor shares the same, decreases $\mathsf{WtdJitter}(T_i)$ by a factor of k. Since spare capacity (i.e., $1 - \rho$) can be used to increase the processor shares of lower-utilization tasks by a larger factor than it can be used to increase the processor shares of higher-utilization tasks, it is more efficient to use the excess capacity to influence the jitter of lower-utilization tasks.

6 Examples

In this section, we illustrate the techniques presented in Sections 3 and 4 by means of some examples.

Example 1 Consider a system of three tasks: $\Gamma = \{T_1 = (2, 10), T_2 = (3, 15), T_3 = (2, 20)\}$ with $\phi_1 = \phi_2 = \phi_3 = 1$; observe that $\rho_1 = \rho_2 = 0.2$, $\rho_3 = 0.1$, and $\rho = 0.5$. Theorem 1 yields a jitter bound $\mathsf{WtdJitter}(\Gamma) = 8$. This is the initial value for J_{high}; the initial value for J_{low} is zero. Observe that, since $\phi_1 = \phi_2 = \phi_3 = 1$, the integer boundary constraint (IBC) guarantees that all jitter values will be integers. Hence we may use *integer* binary search in this example — J_{high}, J_{mid}, and J_{low} all need take on integer values only. (Equivalently, the quantity thres in the pseudocode at the end of Section 3.2 can be set equal to one.)

Choosing $J_{\mathsf{mid}} = (8 + 0)/2 = 4$ and plugging into Equation 7, we determine that $\lambda_1 = 0.333$, $\lambda_2 = 0.428$, and $\lambda_3 = 0.333$. Since $\lambda_1 + \lambda_2 + \lambda_3 > 1$, this is an infeasible solution. Hence, J_{low} is set equal to 4.

Next, choosing $J_{\mathsf{mid}} = (8 + 4)/2 = 6$ and applying the same techniques yields $(\lambda_1, \lambda_2, \lambda_3) = (0.250, 0.333, 0.250)$, which is a feasible solution since $0.250 + 0.333 + 0.250 \leq 1$. Hence, J_{high} is set equal to 6.

Next, choosing $J_{\mathsf{mid}} = (6 + 4)/2 = 5$ and applying the same techniques yields $(\lambda_1, \lambda_2, \lambda_3) = (0.286, 0.375, 0.286)$, which is once again a feasible solution since $0.286 + 0.375 + 0.286 \leq 1$. Hence, J_{high} is set equal to 5.

Next, we compute $J_{\mathsf{mid}} = (5 + 4)/2 = 4$. But we've already checked that this results in an infeasible solution; hence, the tightest jitter bound obtainable using the processor-share method is 5.

We now try using the pseudo-polynomial time test of Section 4, performing binary search with initial "high" of 5 and initial "low" of 0. The test fails for $J = (5+0)/2 = 2$, and succeeds for $J = (5+2)/2 = 3$. Since $(3+2)/2 = 2$ and we've already determined that the test fails for $J = 2$, we can stop now, having determined that the tightest bound obtainable using this method is $J = 3$.

Example 2 Consider a system of three tasks: $\Gamma = \{T_1 = (2,9), T_2 = (4,15), T_3 = (2,12)\}$ with $\phi_1 = \phi_2 = \phi_3 = 1$. Using essentially the same approach as above, Theorem 1 yields an initial jitter bound of 7; the polynomial-time method of Section 3.2 refines this to 6; and the pseudopolynomial-time method of Section 4 further improves this to 4.

Example 3 Consider a system of three tasks: $\Gamma = \{T_1 = (2,10), T_2 = (3,15), T_3 = (20,200)\}$ with $\phi_1 = \phi_2 = \phi_3 = 1$. Once again going through the same steps as above, Theorem 1 yields an initial jitter bound of 80; the polynomial-time method of Section 3.2 refines this to 14; and the pseudopolynomial-time method of Section 4 further improves this to 9.

Example 4 Consider the same system of tasks $\Gamma = \{T_1 = (2,10), T_2 = (3,15), T_3 = (2,20)\}$. Suppose that task T_3 is the only jitter-sensitive task. This can be modelled by setting $\phi_1 = \phi_2 = \infty$, and $\phi_3 = 1$. Several kinds of questions can now be asked; for example:

- Is it possible to schedule Γ such that the maximum jitter for task T_3 never exceeds 3? This is equivalent to determining a θ_3 such that

$$\begin{aligned} e_3(\frac{1}{\theta_3} - 1) &\leq 3 \\ \equiv \quad 2(\frac{1}{\theta_3} - 1) &\leq 3 \\ \equiv \quad \frac{1}{\theta_3} &\leq \frac{3}{2} + 1 \\ \equiv \quad \theta_3 &\geq \frac{2}{5} \end{aligned}$$

  Since assigning a share of 2/5 to T_3 does not cause the total processor-share assigned to exceed 1, we conclude that it *is* possible to schedule Γ such that T_3's maximum jitter does not exceed 3.

- What is the minimum value for the maximum jitter of task T_3? Since the largest value that θ_3 can have is $1.0 - (0.2 + 0.2)$, this is equivalent to computing

$$\begin{aligned} & e_3(\frac{1}{0.6_3} - 1) \\ &= 2\frac{10-6}{6} \\ &= 1.33 \end{aligned}$$

  Since the schedule will obey the IBC, this implies that the maximum jitter of task T_3 will be at most 1 — i.e., the intercompletion time between consecutive jobs of task T_3 will be either 19, 20, or 21.

  Using the method of Section 4, we are reduced to determining the smallest J such that the periodic task system $\{(2,10,10), (3,15,15), (2,2+J,20)\}$ is feasible. Since the utilization of this periodic system is 0.5, we expect the pseudo-polynomial test to be quite efficient.

This turns out to indeed be the case, and the smallest J satisfying the desired condition is $J = 0$ (i.e., the system $\{(2, 10, 10), (3, 15, 15), (2, 2, 20)\}$ is feasible). This permits us to conclude that the system can be scheduled such that T_3 suffers absolutely no jitter — i.e., the intercompletion time between consecutive jobs of task T_3 will always be exactly 20.

7 Conclusions

As observed in the Introduction, output-jitter minimization is not really an issue in the scheduling-theoretical sense, in that notification of the completion of jobs can always be postponed until the jobs' deadlines. In this paper, we have decided that such an artificial solution is unacceptable from a practical point of view, and have enumerated a series of conditions we believe should be satisfied by any reasonable jitter-minimization scheme. Within the context of these conditions, we have proposed two scheduling algorithms for minimizing jitter. One scheme is polynomial-time while the other provides better jitter bounds but takes time pseudo-polynomial in the size of the problem instance; however, even the pseudo-polynomial algorithm is very efficiently implementable. In practice, a reasonable approach to jitter minimization seems to be to use the polynomial-time algorithm to efficiently obtain an upper bound on jitter, and then use the pseudo-polynomial algorithm to further improve this bound. We have implemented and tested our schemes — the algorithms are simple and easy to code, and the run-time behavior is very satisfactory.

References

[1] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, 8(5):285–292, 1993.

[2] S. Baruah, R. Howell, and L. Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems: The International Journal of Time-Critical Computing*, 2:301–324, 1990.

[3] Sanjoy Baruah, Deji Chen, and Aloysius Mok. Jitter concerns in periodic task systems. In *Proceedings of the Eighteenth Real-Time Systems Symposium*, pages 68–77, San Francisco, CA, December 1997. IEEE Computer Society Press.

[4] A. Burns, M. Nicholson, K. Tindell, and N. Zhang. Allocating and scheduling hard real-time tasks on a point-to-point distributed system. In *Proceedings of the Workshop on Parallel and Distributed Real-Time Systems*, pages 11–20, April 1993.

[5] M. Dertouzos. Control robotics : the procedural control of physical processors. In *Proceedings of the IFIP Congress*, pages 807–813, 1974.

[6] D.Verma, H. Zhang, and Domenico Ferrari. Delay jitter control for real-time communication in a packet switching network. In *Proceedings of the IEEE Conference on Communications Software: Communications for Distributed Applications and Systems*, pages 35–43, Chapel Hill, NC, 1991.

[7] D. Ferrari. Client requirements for real-time communications services. *IEEE Communications*, 28(11), November 1990.

[8] K.J. Lin and A. Herkert. Jitter control in time-triggered systems. In *Proceedings of the 29th Hawaii International Conference on System Sciences*, Maui, Hawaii, January 1996.

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

[10] A. Mok. Task management techniques for enforcing ED scheduling on a periodic task set. In *Proc. 5th IEEE Workshop on Real-Time Software and Operating Systems*, pages 42–46, Washington D.C., May 1988.

[11] J. Stankovic and M. DiNatale. Dynamic end-to-end guarantees in distributed real-time systems. In *Proceedings of the Real-Time Systems Symposium*, San Juan, Puerto Rico, December 1994.

[12] K. W. Tindell, A. Burns, and A. J. Wellings. An extendible approach for analysing fixed priority hard real-time tasks. *Real-Time Systems: The International Journal of Time-Critical Computing*, 6:133–151, 1994.