A multiprocessor implementation of the Total Bandwidth Server[*]

Sanjoy Baruah
The University of North Carolina
Chapel Hill, NC. USA

Giuseppe Lipari
Scuola Superiore S. Anna
Pisa, Italy

Abstract

If a periodic task system is scheduled upon an identical multiprocessor platform using the Earliest Deadline First scheduling algorithm, it is known that the "schedulable utilization" – the largest bound such that any periodic task system with cumulative utilization no larger than this bound is guaranteed to be successfully scheduled – is strictly less than the capacity of the platform. The issue of using the excess processing capacity (the difference between the platform capacity and the schedulable utilization) is addressed here, and an algorithm is presented, and proven correct, that uses this excess capacity to provide guaranteed real-time service to aperiodic jobs.

Keywords: *Real-time systems; multiprocessor scheduling; periodic task systems; aperiodic jobs; earliest deadline first.*

1 Introduction

Over the years, the **preemptive periodic** model has proven remarkably useful for representing recurring processes that occur in hard-real-time (HRT) application systems. The scheduling of systems of such tasks has therefore been the subject of much research, particularly upon uniprocessor platforms. In many real-time application systems, it is the case that there are occasional non-recurring jobs that need to be executed in addition to the HRT jobs generated by the periodic tasks. Such **aperiodic** jobs are typically handled by an **aperiodic server**, which makes use of the processor capacity left over by the HRT jobs to execute these aperiodic jobs.

In uniprocessor systems, all the processor capacity that is left unused by the HRT jobs is available to the aperiodic server; hence, designing an aperiodic server for uniprocessor platforms is essentially a problem of accurately measuring the amount of such unused capacity, and then allocating it appropriately to the aperiodic jobs in order to optimize some desired metric. This can be a complex problem, particularly if it is desired (as it usually is) that each aperiodic job complete as quickly as possible. In multiprocessor systems, however, the complexity of designing aperiodic servers is further compounded by the fact that, in addition to measuring the *amount* of processor capacity left unused by the periodic tasks, we must also take into account the *manner* in which this unused capacity occurs. For example, if $k > 1$ processors are left idle at some instant by the periodic tasks' jobs and there is only one aperiodic job to be served, then this aperiodic job is able to execute on only one of these idle processors, and hence a fraction$(k-1)/k$ of the idle capacity is not usable by the aperiodic server. In this paper, we study the issue of designing aperiodic servers for such multiprocessor platforms, that make efficient use of the processor capacity left unused by the periodic tasks' jobs.

Organization of this document. In Section 2 below, we define the terminology and notation that will be used in this paper, and present prior results that are needed in the following sections. In Section 3 we design, and establish the correctness of, a multiprocessor aperiodic server that is able to make efficient use of the capacity left unused by periodic tasks. We conclude in Section 4 with a summary of the results presented here, and discussion on interesting future research directions.

2 System model and Background

2.1 The periodic task model

In the preemptive periodic model [10, 9] of recurring real-time tasks[1], a **periodic task** $\tau_i = (C_i, P_i)$ is characterized by two parameters: a (worst-case) execution requirement C_i and a minimum inter-arrival

*Supported in part by the National Science Foundation (Grant Nos. ITR-0082866, CCR-0204312, and CCR-0309825).

[1]Note that what we call a periodic task here is sometimes referred to in the literature as a *sporadic* task.

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

separation parameter P_i (often referred to as the *period* of the task). Such a periodic task generates an infinite sequence of **hard-real-time jobs** (HRT jobs). An HRT job j is characterized by three parameters – an arrival time $a(j)$, an execution requirement $e(j)$, and a deadline $d(j)$ – with the interpretation that it must execute for an amount equal to its execution requirement $e(j)$ over the time-interval $[a(j), d(j))$. The HRT jobs generated by periodic task τ_i each have an execution requirement of C_i and a deadline P_i time units after its arrival time. The first HRT job generated by τ_i may arrive at any time-instant; successive arrivals are separated by at least P_i time units. We use the term *utilization* to denote the ratio of the execution requirement parameter of a task to its inter-arrival separation parameter — the utilization U_i of τ_i is equal to C_i/P_i. Intuitively speaking, the utilization of a periodic task denotes the fraction of the computing capacity of a unit-capacity processor that may need to be devoted to executing jobs of this periodic task, in the worst case. A periodic task system consists of several such periodic tasks that are to execute on a shared platform. The jobs are assumed to be *independent* in the sense that each job does not interact in any manner (accessing shared data, exchanging messages, etc.) with other jobs of the same or another task. It is also assumed that the model allows for job *preemption*; i.e., a job executing on a processor may be preempted prior to completing execution, and its execution resumed later on the same or a different processor, at no cost or penalty. Let $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ denote a periodic task system. For any such periodic task system τ, $U_{\mathsf{sum}}(\tau)$ will denote the cumulative utilizations of all tasks in τ ($U_{\mathsf{sum}}(\tau) \stackrel{\text{def}}{=} \sum_{i=1}^{n} U_i$); $U_{\mathsf{max}}(\tau)$ will denote the largest utilization of any task in τ ($U_{\mathsf{max}}(\tau) \stackrel{\text{def}}{=} \max_{i=1}^{n} U_i$).

The basic periodic task model described above can be generalized in several ways. These generalizations better represent the kinds of recurring processes occurring in a larger class of real-time systems, and hence are likely to prove more useful to the designers of real-time application systems.

§1. As our first generalization, we no longer require that all the jobs generated by a periodic task have identical execution requirements and relative deadlines. Rather, we consider the utilization parameter of a periodic task to be a "first-class" concept, and derive conditions on the execution requirement and relative deadline from this utilization parameter. This generalization is motivated by the following abstraction: Each generalized periodic task τ_i represents a sequence of HRT jobs that are known to meet their time constraints when executing upon a processor of computing capacity equal to the tasks' utilization. The periodic task system $\tau = \{\tau_1, \ldots, \tau_n\}$ is therefore comprised of n such sequences of jobs that are multiprogrammed onto a common platform. For technical reasons that we will elaborate upon later, we add the restriction that each individual job's execution not exceed an *a priori* specified upper bound. Formally, then, each generalized periodic task τ_i is characterized by a utilization U_i, and an upper bound upon the execution requirements of each of its jobs $C_i^{(\mathsf{ub})}$. The jobs $j_1, j_2, \ldots$, generated by τ_i are required to satisfy the properties enumerated in Equation 1:

$$\begin{array}{llcll} \textbf{(1a)} & a(j_1) & \geq & 0 & \\ \textbf{(1b)} & e(j_\ell) & \leq & C_i^{(\mathsf{ub})} \text{ for all } \ell & \\ \textbf{(1c)} & a(j_\ell) & \geq & d(j_{\ell-1}) \text{ for all } \ell \geq 2 & \\ \textbf{(1d)} & d(j_\ell) & = & a(j_\ell) + \dfrac{e(j_\ell)}{U} \text{ for all } \ell \geq 1 & \end{array} \qquad (1)$$

We will refer to periodic task systems that satisfy the constraints represented in Equation 1 above as Partially Generalized Periodic Task Systems, or **PGPTS**'s (this model is "partially generalized" in contrast to the more fully generalized model described below). In contrast, we will refer to periodic task systems defined using the (C, P) model described above as **basic** periodic task systems.

§2. As a further generalization, we get rid of the requirement that successive jobs of the same task not have overlapping intervals. That is, we replace the constraints (1c) and (1d) with the weaker constraint (2c) below:

$$\begin{array}{ll} \textbf{(2a)} & a(j_1) \geq 0 \\ \textbf{(2b)} & e(j_\ell) \leq C_i^{(\mathsf{ub})} \text{ for all } \ell \\ \textbf{(2c)} & d(j_\ell) = \max\{a(j_\ell), d(j_{\ell-1})\} + \dfrac{e(j_\ell)}{U} \text{ for all } \ell \geq 2 \end{array} \qquad (2)$$

We will refer to periodic task systems that satisfy the constraints represented in Equation 2 above as *generalized periodic task systems*, or **GPTS**'s. (Note that this generalization places some restrictions upon the implementation of the multiprocessor system-level scheduler – since there may be several jobs of the same task active simultaneously, it is incumbent upon the scheduler to ensure that several jobs of a task are not simultaneously scheduled upon different processors.)

The various PTS models described above are summarized in Table 1

Some further notation: for any (partially) generalized periodic task $\tau_i = (C_i^{(\mathsf{ub})}, U_i)$, let $P_i^{(\mathsf{ub})}$ denote the relative deadline of a job of τ_i that has the maximum permissible execution requirement:

$$P_i^{(\mathsf{ub})} \stackrel{\text{def}}{=} \frac{C_i^{(\mathsf{ub})}}{U_i}\ .$$

Basic (BPTS). Each periodic task τ_i is specified by an execution requirement C_i and a period P_i, and generates a sequence of jobs each with execution requirement C_i and relative deadline P_i, with successive jobs arriving at least P_i time units apart.

Partially Generalized (PGPTS). Each periodic task τ_i is specified by a utilization U_i and an upper bound on execution requirement $C_i^{(\mathsf{ub})}$, and generates a sequence of jobs each with execution requirement $\leq C_i^{(\mathsf{ub})}$, with the arrival time of job j being after the deadline of the previous job generated by τ_i, and the deadline of job j being $e(j)/U_i$ time units after its arrival.

Generalized (GPTS). Each periodic task τ_i is specified by a utilization U_i and an upper bound on execution requirement $C_i^{(\mathsf{ub})}$, and generates a sequence of jobs each with execution requirement $\leq C_i^{(\mathsf{ub})}$. Job j can arrive at any time after the arrival of the previous job generated by τ_i, and has a deadline $e(j)/U_i$ time units after the *later* of its arrival-time and the deadline of the previous job generated by τ_i.

Table 1. The three kinds of Periodic Task System (PTS) models discussed in this paper.

For any (P)GPTS τ, let $P_{\mathsf{max}}(\tau)$ denote the largest relative deadline of any task in τ:

$$P_{\mathsf{max}}(\tau) \stackrel{\text{def}}{=} \max_{i=1}^{n}\{P_i\} \ .$$

Aperiodic jobs. In addition to the periodic tasks' jobs, there may be certain aperiodic jobs that also need to be scheduled. An *aperiodic job* may arrive at any time, and is characterized by a (worst-case) execution requirement and a deadline. Nothing about an aperiodic job – its arrival time, execution requirement, or deadline – is known prior to the instant it arrives; at that instant, all its parameters are completely known.

Some further definitions:

Definition 1 Let τ denote a periodic task system specified using the basic, partially generalized, or generalized model, which is to be scheduled upon a multiprocessor platform comprised of m unit-capacity processors. Let J denote a (finite or infinite) collection of HRT jobs generated by τ.

1. For any m-processor scheduling algorithm A, let $\underline{A(J)}$ denote the schedule generated when J is scheduled upon m unit-capacity processors by scheduling algorithm A.

2. Let $t \in \mathbb{R}$ denote any time-instant, and A any m-processor scheduling algorithm. For any $J' \subseteq J$, the expression $\underline{\mathcal{W}(J', A(J), t)}$ denotes the total amount that jobs in J' have been executed in schedule $A(J)$ over the time-interval $[0, t)$.

3. For any periodic task $\tau_i \in \tau$, let $\underline{I(\tau_i)}$ denote any collection of jobs that could legally be generated by task τ_i. Let $\underline{I(\tau)}$ denote any collection of jobs that could legally be generated by the tasks in τ: $I(\tau) \stackrel{\text{def}}{=} \bigcup_{\tau_i \in \tau} I(\tau_i)$.

4. We are interested in periodic task systems that only generate collections of HRT jobs that are *feasible*; hence, we assume that τ satisfies these two properties: **(i)** $U_{\mathsf{sum}}(\tau) \leq m$; and **(ii)** $U_{\mathsf{max}}(\tau) \leq 1$. It is straightforward to observe that any τ not satisfying both these properties can generate collections of jobs which are infeasible. On the other hand if τ satisfies both these properties, any $I(\tau)$ is feasible upon a multiprocessor platform comprised of m unit-capacity processors. To see this, let j denote any job generated by τ_i in $I(\tau)$, and let j' denote the previous job generated by τ_i if any (if j is the first job generated by τ_i, then $d(j')$ is assumed equal to zero). A processor-sharing schedule which assigns a fraction U_i of a processor to job j during the interval $[\max\{a(j)d(j')\}, d(j))$, will meet job j's deadline. Let $\underline{\mathsf{opt}}$ denote a scheduling algorithm that generates such a schedule, and hence let $\underline{\mathsf{opt}(I(\tau))}$ denote this schedule for $I(\tau)$ — i.e., $\mathsf{opt}(I(\tau))$ ***assigns each job in*** $I(\tau_i)$ ***a fraction*** U_i ***of a processor over the time-interval between the larger of its arrival-time and its predecessor's deadline, and its own deadline.***

2.2 Background and related work

The **Total Bandwidth Server** (TBS) abstraction was introduced by Buttazzo et al. [12, 14, 13] for servicing aperiodic jobs in a uniprocessor hard-real-time environment, in which a set of (basic) periodic tasks is scheduled using the preemptive Earliest Deadline First scheduling algorithm (Algorithm EDF) [9, 7]. When an aperiodic job arrives, the TBS algorithm makes a determination on whether to accept the aperiodic job or not – accepting the job is tantamount to guaranteeing that it will be executed for an amount of time equal to its execution requirement before its deadline. The uniprocessor TBS variants proposed in [12, 14, 13] achieve full processor utilization while simultaneously guaranteeing the timely execution of all HRT jobs.

We have recently [8, 15, 6] been studying the scheduling of (generalized) periodic task systems upon multiprocessor platforms comprised of several identical processors. For periodic task systems that are scheduled using Algorithm EDF upon m processors, we have shown that the *schedulable utilization*[2] is not equal to the system capacity (as is the case with uniprocessor EDF); rather, it depends upon the largest utilization of any periodic task in the task system being scheduled. More specifically, we have shown:

1. *A sufficient condition for ensuring that periodic task system τ is successfully scheduled upon m unit-capacity processors by* EDF *is that*
$$U_{\mathsf{sum}}(\tau) \leq m - (m-1) \times U_{\mathsf{max}}(\tau) \ . \qquad (3)$$

2. *For all μ, $0 < \mu < 1$ and for all positive ϵ , there are periodic task systems τ' that have $U_{\mathsf{max}}(\tau') = \mu$ and $U_{\mathsf{sum}}(\tau') = m - (m-1)\mu + \epsilon$, which* EDF *fails to successfully schedule upon m unit-capacity processors.*

In the uniprocessor TBS algorithm [12, 14, 13], aperiodic jobs are serviced by a server which is assigned a capacity $(1 - U_{\mathsf{sum}}(\tau))$, where τ denotes the periodic task system being scheduled. When an aperiodic job arrives, the server determines whether it possesses sufficient capacity prior to the job's deadline to complete it. If so, the job is admitted and guaranteed to meet its deadline; otherwise, it is rejected. Intuitively, this server approach works for the following reason: from the optimality of EDF upon uniprocessors [9, 7], it follows that <u>all</u> the capacity of the processor can be used for scheduling the periodic plus the aperiodic jobs. Hence the processor capacity not needed by the periodic tasks – a quantity equal to $(1 - U_{\mathsf{sum}}(\tau))$ — may be devoted to executing aperiodic jobs.

Contributions in this paper. In this paper, we present Algorithm M-TBS, a *multiprocessor* version of the total bandwidth server. As with the uniprocessor TBS, M-TBS schedules the HRT jobs of the periodic tasks with EDF, and has an aperiodic server to handle the aperiodic jobs. Our objective is to obtain an aperiodic server that uses not just the amount of the usable capacity unused by the periodic tasks (i.e., an amount $[m - (m-1)\,U_{\mathsf{max}}(\tau) - U_{\mathsf{sum}}(\tau)]$) for serving aperiodics — such a server would essentially mimic, upon multiple processors, the behavior of TBS's uniprocessor server — but also the additional amount $(m-1)\,U_{\mathsf{max}}$ that is, according to Equation 3, essentially unavailable to multiprocessor EDF for making real-time performance guarantees (to the periodic tasks' jobs). So we are, in a sense, fundamentally *reclaiming* for real-time use by our aperiodic server the capacity that is not available for serving periodic tasks. Stated differently, Algorithm M-TBS, executing upon m processors, is able to service aperiodic jobs even when the periodic task system τ scheduled upon the m processors maximally utilizes the processors by having $U_{\mathsf{sum}}(\tau)$ equal to $(m - (m-1)U_{\mathsf{max}}(\tau))$. Consequently, the aperiodic server in Algorithm M-TBS is rather more complex than the aperiodic server of uniprocessor TBS [12, 14, 13], and new techniques are introduced for analyzing its behavior.

[2]The **schedulable utilization** of a scheduling algorithm is defined as follows [11] : *"A scheduling algorithm can [successfully] schedule any set of periodic tasks [...] if the total utilization of the tasks is equal to or less than the <u>schedulable utilization</u> of the algorithm."*

3 Design and analysis of a multiprocessor aperiodic server

In this section we derive, and establish the correctness of, the aperiodic server associated with Algorithm M-TBS that is responsible for scheduling aperiodic jobs. For ease of exposition, we first (Section 3.1) assume that (i) there is exactly one aperiodic job in the system at any instant in time; (ii) the aperiodic server simply executes in the background — a waiting aperiodic job is executed only if there is no HRT job awaiting execution; and (iii) HRT jobs are generated by a partially generalized periodic task system (PG-PTS). We then (Section 3.2) get rid of the first assumption, and permit arbitrarily many aperiodic jobs. Next, (Sections 3.3 and 3.4), we remove the second assumption. Finally (Section 3.5), we permit our recurring HRT processes to be modelled using the generalized periodic task model, and describe the aperiodic server in complete detail.

For ease of presentation, we will assume in the remainder of this paper that the global multiprocessor EDF scheduling algorithm is used for scheduling the HRT jobs which are generated by the [generalized] periodic task systems. However, we will show that our results hold if *any* multiprocessor scheduling algorithm A is used for this purpose, provided that this algorithm satisfies the following condition:

$$\forall I(\tau) : \forall t \in \mathbb{R} : \mathcal{W}\left(I(\tau), A(I(\tau)), t\right) \geq \mathcal{W}\left(I(\tau), \mathsf{opt}(I(\tau)), t\right) \qquad (4)$$

I.e., *for all collections of jobs $I(\tau)$ that could legally be generated by generalized periodic task system τ, it is the case that jobs in $I(\tau)$ have received at least as much execution over the time-interval $[0, t)$ under Algorithm A as they have in the schedule* $\mathsf{opt}(I(\tau))$ (described in Definition 1), in which all deadlines are met, for all time-instants t. (Of course, one would first need to ensure that the scheduling algorithm used is able to meet the deadlines of all the HRT jobs; however, this issue is orthogonal to the one of designing an aperiodic server.)

Lemma 1 below asserts that EDF satisfies Condition 4 above, and is hence a suitable algorithm to use with our reclamation scheme:

Lemma 1 ([8]) *Let τ denote a generalized periodic task system satisfying Inequality 3 above:*

$$U_{\mathsf{sum}}(\tau) \leq m - (m-1)\, U_{\mathsf{max}}(\tau) \ .$$

Algorithm EDF *satisfies Condition 4 above:*

$$\forall I(\tau) : \forall t \in \mathbb{R} : \mathcal{W}\left(I(\tau), \mathsf{EDF}(I(\tau)), t\right) \geq \mathcal{W}\left(I(\tau), \mathsf{opt}(I(\tau)), t\right) \tag{5}$$

Although we restrict our attention in this paper to systems that use EDF for scheduling the HRT jobs generated by the (generalized) periodic task system, we would like to reiterate that our results hold if any multiprocessor scheduling algorithm that satisfies Inequality 4 is used. For example, it can be shown that the *pfair* scheduling algorithms PF [4], PD [5], and PD^2 [1] all satisfy Inequality 4 upon periodic task systems τ for which $U_{\mathsf{sum}}(\tau) \leq m$ and $U_{\mathsf{max}}(\tau) \leq 1$ hold; hence the results presented here hold for systems in which these algorithms are used for scheduling the HRT jobs. Also, it is known [8] that any *work-conserving* scheduling algorithm A satisfies Inequality 4, provided τ satisfies Inequality 3 above; hence, recently-proposed work-conserving multiprocessor scheduling algorithms such as RM-US [2], fpEDF [3], etc. can be used on such systems.

3.1 Background scheduling a single aperiodic job

Suppose that a collection of HRT jobs $I(\tau)$, generated by PGPTS τ, is being scheduled by EDF upon a multiprocessor platform comprised of m unit-capacity processors. (Note that we do not require that the arrival-times of the jobs in $I(\tau)$ be known beforehand; however, the parameters of all tasks in τ is assumed to be known *a priori*.) Suppose further that τ satisfies Condition 3, and all jobs in $I(\tau)$ consequently meet their deadlines in the EDF-generated schedule $\mathsf{EDF}(I(\tau))$.

Suppose that a single aperiodic job $J = (A, E)$ now arrives at time-instant A, with execution-requirement equal to E. Suppose that this job has a ***response time*** of f when executed as a background job — i.e., it completes execution as a background job at time-instant $A + f$. In the remainder of this section, we compute an upper bound on this response time f.

Let $I'(\tau)$ denote all the jobs in $I(\tau)$ that arrive before time-instant $A+f$; for each i, $1 \leq i \leq n$, let $I'(\tau_i)$ denote the jobs in $I'(\tau)$ generated by τ_i. That is,

$$I'(\tau_i) \ \stackrel{\text{def}}{=} \ \left\{ j | j \in I(\tau_i) \bigwedge a(j) \leq A + f \right\}$$

$$I'(\tau) \ \stackrel{\text{def}}{=} \ \bigcup_{\tau_i \in \tau} I'(\tau_i)$$

Observe that over the interval $[A, A+f)$, aperiodic job J is executing whenever all m processors are not busy executing HRT jobs. Hence, *f is maximized when all the HRT work that arrives before $A+f$ and that has not been executed prior to time-instant A executes with maximum parallelism* – i.e., on all m processors, thereby leaving no processor available for J.

How much such work can there possibly be? To compute this, observe that the amount of such work in $I'(\tau_i)$ — i.e., generated by task τ_i — is at most

$$\left(\sum_{j \in I'(\tau_i)} e(j) \right) - \mathcal{W}(I'(\tau_i), \mathsf{EDF}(I(\tau)), A) \ .$$

Hence, the total amount of such work in $I'(\tau)$ is at most

$$\begin{aligned}
&\sum_{\tau_i \in \tau} \left[\left(\sum_{j \in I'(\tau_i)} e(j) \right) - \mathcal{W}(I'(\tau_i), \mathsf{EDF}(I(\tau)), A) \ . \right] \\
\equiv \quad & \left(\sum_{j \in I'(\tau)} e(j) \right) - \mathcal{W}(I'(\tau), \mathsf{EDF}(I(\tau)), A) \\
\equiv \quad & \text{(Since } I'(\tau) \text{ includes } \textit{all} \text{ jobs that arrive before } A) \\
& \left(\sum_{j \in I'(\tau)} e(j) \right) - \mathcal{W}(I(\tau), \mathsf{EDF}(I(\tau)), A) \\
\leq \quad & \text{(By Equation 5, since } \tau \text{ satisfies Condition 3)} \\
& \left(\sum_{j \in I'(\tau)} e(j) \right) - \mathcal{W}(I(\tau), \mathsf{opt}(I(\tau)), A) \qquad (6)
\end{aligned}$$

Hence, the total amount of HRT work in $I(\tau)$ that, by executing in parallel upon all m processors, can delay the completion of the aperiodic job $J = (A, E)$, is upper-bounded by Expression 6 above. In Lemma 2 below, we derive a succinct upper bound on Expression 6.

Lemma 2

$$\begin{aligned}
& \sum_{j \in I'(\tau)} e(j) - \mathcal{W}(I(\tau), \mathsf{opt}(I(\tau)), A) \qquad (7) \\
\leq \quad & \left(f \times U_{\mathsf{sum}}(\tau) + \sum_{\tau_i \in \tau} P_i^{(\mathsf{ub})} U_i (1 - U_i) \right)
\end{aligned}$$

Proof: All jobs in $I(\tau_i)$ that have a deadline *before* time-instant A complete prior to A in the schedule $\mathsf{opt}(I(\tau))$; hence, they do not contribute to the left-hand side of Inequality 7. We partition the remaining jobs in $I(\tau_i)$ into three categories:

1. At most one job j in $I(\tau_i)$ may arrive prior to A and have deadline after A.

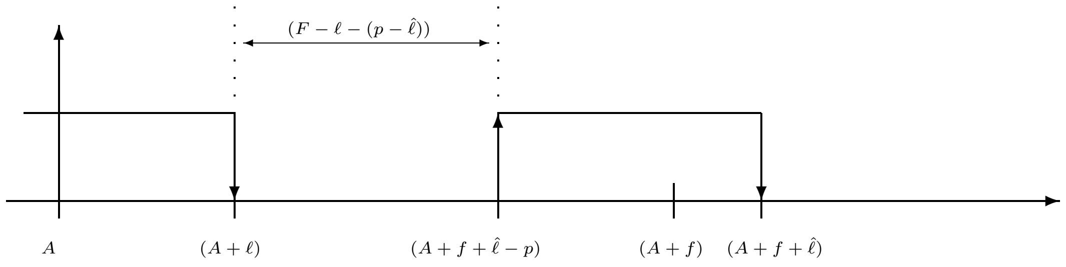

Figure 1. Contribution of jobs of task τ_i to the left-hand side of Inequality 7 (Lemma 2).

- Let us assume that this job's deadline is at time-instant $A+\ell$ (see Figure 1).
- The contribution of this job to the left-hand side of Inequality 7 is exactly equal to $\ell \cdot U_i$ — this follows from the fact that the schedule $\mathsf{opt}(I(\tau))$ assigns a fraction U_i of a processor to τ_i between the arrival and deadline of each of its jobs.

2. At most one job j in $I(\tau_i)$ may arrive prior to $A+f$ and have deadline after $A+f$.

   - Let us assume that this job's deadline is at time-instant $A+f+\hat{\ell}$ (see Figure 1), and that its relative deadline is p where $p \leq P_i^{(\mathsf{ub})}$.
   - Then, its arrival time is at time-instant $A+f+\hat{\ell}-p$.
   - Let us assume that its execution requirement is C_i', which is, by Equation 1, no more than $p \cdot U_i$. Note that, since this job completes execution *before* time-instant $A+f$, it must be the case that $(p-\hat{\ell}) \geq C_i'$; i.e., $\hat{\ell} \leq (p-C_i')$.

3. The maximum contribution of jobs in $I(\tau_i)$ that have arrival instant after $A+\ell$, and deadline no larger than $A+f+\hat{\ell}-p$, is bounded from above by $((f-\ell-(p-\hat{\ell}))\cdot U_i)$.

Hence, the total contribution of τ_i to the left-hand side of Inequality 7 is bounded from above by

$$\begin{aligned}
&(f-\ell-(p-\hat{\ell}))\cdot U_i + \ell\cdot U_i + C_i' \\
&= \quad f\cdot U_i - (p-\hat{\ell})\cdot U_i + C_i' \\
&\leq \quad f\cdot U_i - (p-(p-C_i'))\cdot U_i + C_i' \ \text{(Since } \hat{\ell} \leq (p-C_i')) \\
&= \quad f\cdot U_i - C_i'\cdot U_i + C_i' \\
&= \quad f\cdot U_i + C_i'\cdot(1-U_i) \\
&= \quad f\cdot U_i + pU_i\cdot(1-U_i) \\
&\leq \quad f\cdot U_i + P_i^{(\mathsf{ub})}\cdot(1-U_i) \ \text{(Since } p \leq P_i^{(\mathsf{ub})}) \qquad (8)
\end{aligned}$$

To bound the left-hand side of Inequality 7, we sum Expression 8 over all tasks $\tau_i \in \tau$ to obtain

$$\left(\sum_{\tau_i\in\tau}\left(f\cdot U_i + P_i^{(\mathsf{ub})}U_i\cdot(1-U_i)\right)\right),$$

which simplifies to

$$\left(fU_{\mathsf{sum}}(\tau) + \sum_{\tau_i\in\tau} P_i^{(\mathsf{ub})}U_i\cdot(1-U_i)\right),$$

which is the right-hand side of Inequality 7.

In Expression 6, we had obtained an upper bound on the total amount of HRT work in $I(\tau)$ arriving no later than $A+f$ that had not yet been executed by time-instant A. In Lemma 2, we derived a succinct upper bound on Expression 6; i.e., we have shown that the total amount of HRT work in $I(\tau)$ arriving no later than $A+f$ that, by executing in parallel upon all m processors, can delay the completion of the aperiodic job $J=(A,E)$, is bounded from above by

$$f\times U_{\mathsf{sum}}(\tau) + \sum_{\tau_i\in\tau} P_i^{(\mathsf{ub})}U_i(1-U_i)$$

If this work executes with maximum parallelism, then it will consume all m processors over time-intervals of cumulative length

$$\frac{f\cdot U_{\mathsf{sum}}(\tau) + \sum_{\tau_i\in\tau} P_i^{(\mathsf{ub})}U_i(1-U_i)}{m}.$$

Consequently, the aperiodic job $J=(A,E)$ completes execution no later than time-instant

$$A + \frac{f\cdot U_{\mathsf{sum}}(\tau) + \sum_{\tau_i\in\tau} P_i^{(\mathsf{ub})}U_i(1-U_i)}{m} + E.$$

We can therefore solve for f:

$$
\begin{aligned}
& A + \frac{f \cdot U_{\mathsf{sum}}(\tau) + \sum_{\tau_i \in \tau} P_i^{(\mathsf{ub})} U_i(1-U_i)}{m} + E \geq A + f \\
\equiv \quad & f \cdot (m - U_{\mathsf{sum}}(\tau)) \leq mE + \sum_{\tau_i \in \tau} P_i^{(\mathsf{ub})} U_i(1-U_i) \\
\equiv \quad & f \leq \left(\frac{mE + \sum_{\tau_i \in \tau} P_i^{(\mathsf{ub})} U_i(1-U_i)}{m - U_{\mathsf{sum}}(\tau)} \right) \qquad (9)
\end{aligned}
$$

Inequality 9 above is the desired upper bound on the response-time f of the aperiodic job $J = (A, E)$: it is guaranteed that Algorithm M-TBS will complete job J at or before time-instant $A + f$. Observe that this bound depends only upon the parameters of periodic task-system τ, and the worst-case execution requirement of J; hence, it can be computed at time-instant A when job J arrives.

3.2 Multiple aperiodic jobs

Above, we have considered the situation where exactly one aperiodic job must be scheduled along with all the periodic HRT jobs in $I(\tau)$. A more general formulation would have an entire series of aperiodic jobs $J_1, J_2, \ldots$, with each $J_i = (A_i, E_i)$ characterized by an arrival time and an execution requirement, that need to be executed. Below, we describe how upper bounds on the completion times for all these aperiodic jobs may be computed. In the remainder of this section, we assume that these aperiodic jobs are indexed according to non-decreasing arrival times (i.e., $A_i \leq A_{i+1}$ for all $i \geq 1$), and that the aperiodic server considers these jobs in non-decreasing index order (i.e., J_i is considered prior to J_{i+1}) by the aperiodic server.

The aperiodic server would maintain an additional variable E_R, which keeps track of the remaining execution requirement of all aperiodic jobs. Let $E_R(t)$ denote the value of this variable at time-instant t, $t \geq 0$.

- E_R is initially equal to zero: $E_R(0) \leftarrow 0$.
- When aperiodic job $J_i = (A_i, E_i)$ is processed by the aperiodic server, E_R is incremented by an amount equal to E_i: $E_R(A_i) \leftarrow E_R(A_i) + E_i$.
- At each instant, E_R is decremented at a rate equal to the number of processors that are executing aperiodic job at that instant: $\frac{d}{dt} E_R(t) = -k$, where k denotes the number of processors ($0 \leq k \leq m$) executing aperiodic jobs at time-instant t.

Using arguments virtually identical to those above, it can be shown that $A_i + f_i$, the latest completion-time for aperiodic job $J_i = (A_i, E_i)$, can be obtained by solving the following for f_i:

$$
\begin{aligned}
& A_i + \frac{f_i \cdot U_{\mathsf{sum}}(\tau) + \sum_{\tau_i \in \tau} P_i^{(\mathsf{ub})} U_i(1-U_i) + E_R(A_i)}{m} + E_i \geq A_i + f_i \\
\equiv \quad & f_i \cdot (m - U_{\mathsf{sum}}(\tau)) \leq mE_i + \sum_{\tau_i \in \tau} P_i^{(\mathsf{ub})} U_i(1-U_i) + E_R(A_i) \\
\equiv \quad & f_i \leq \left(\frac{mE_i + \sum_{\tau_i \in \tau} P_i^{(\mathsf{ub})} U_i(1-U_i) + E_R(A_i)}{m - U_{\mathsf{sum}}(\tau)} \right) \qquad (10)
\end{aligned}
$$

Comparing Inequality 10 with Inequality 9, we see that the presence of not-yet-completed aperiodic work increases the numerator of the expression by an amount equal to the outstanding work $E_R(A_i)$, and is hence essentially equivalent to having the single aperiodic job's execution requirement increased by an amount equal to $(E_R(A_i)/m)$.

3.3 EDF scheduling of the aperiodic jobs

Rather than scheduling the aperiodic job as a background job, we can instead have it scheduled by the EDF scheduler that is responsible for scheduling the HRT jobs, by assigning it an appropriate deadline. To determine an appropriate value for this deadline $A + D$ (see Figure 2), observe that

1. the presence of this aperiodic job in the EDF queue has no effect upon the scheduling of HRT jobs with deadline less than $(A + D)$; and
2. no HRT job with deadline greater than $(A + f + P_{\mathsf{max}}(\tau))$ has arrived prior to the instant $A + f$ when the aperiodic job completes execution.

Hence, assigning the aperiodic job a deadline equal to

$$
A + \left(\frac{mE + \sum_{\tau_i \in \tau} P_i^{(\mathsf{ub})} U_i(1-U_i)}{m - U_{\mathsf{sum}}(\tau)} \right) + P_{\mathsf{max}}(\tau) \quad (11)
$$

ensures that it effectively executes in the background with respect to the HRT jobs, and does not effect the scheduling of the HRT jobs in any manner.

It can similarly be shown that, in the case of multiple aperiodic jobs (as considered above in Section 3.2), each job J_i could be assigned a deadline D_i equal to

$$
\max \left(D_{i-1}, \frac{mE_i + \sum_{\tau_i \in \tau} P_i^{(\mathsf{ub})} U_i(1-U_i) + E_R(A_i)}{m - U_{\mathsf{sum}}(\tau)} + P_{\mathsf{max}}(\tau) \right) \quad (12)
$$

with the "max" is due to the fact that deadlines are assigned to aperiodic jobs in non-decreasing order: since the deadlines of earlier aperiodic jobs were computed

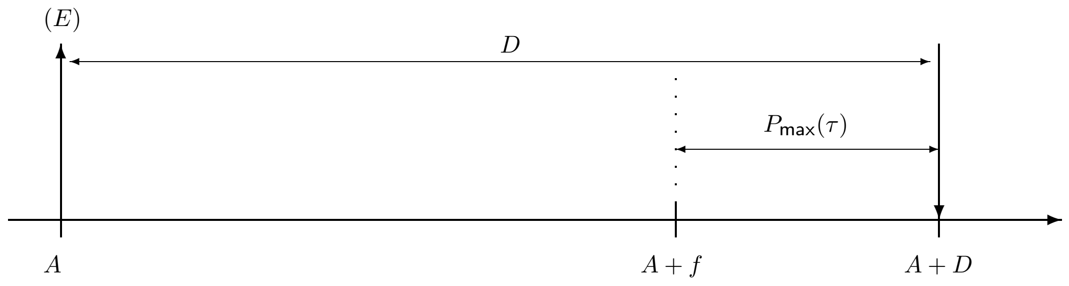

Figure 2. An aperiodic job with execution requirement E arrives at time-instant A, and is assigned a deadline equal to $A + D$. It completes execution at time-instant $A + f$. The deadline assigned must be such that it results in $(D - f)$ being at least P_{max}.

with no knowledge of the i'th aperiodic job, the presence of this i'th job should not effect the scheduling of earlier jobs. (For the same reason, we also assume that in case of tied deadlines, the EDF implementation breaks ties in favor of earlier-admitted jobs.)

3.4 Admission control

Thus far in this section, we have assumed that aperiodic jobs are not subject to any real-time constraints. An alternative model formulation – the one described in the Introduction (Section 1) – would have each aperiodic job also subject to a real-time constraint. In this model, an aperiodic job $J_i = (A_i, E_i, \eta_i)$ is characterized by a response-time constraint η_i in addition to its arrival time A_i and its execution requirement E_i (as above). For such systems, the aperiodic server must perform **admission control** when job J_i arrives at time-instant A_i: admit this job only if it is possible to execute it for E_i units by time-instant $A_i + \eta_i$ without compromising the schedulablity of HRT jobs or any previously admitted aperiodic jobs. That is,

- job J_i is admitted if and only if $f_i \leq \eta_i$, where f_i is as defined in Expression 10 above; and
- E_R is incremented at time-instant A_i if and only if J_i is admitted.

Algorithm M-TBS's acceptance test for aperiodic jobs is given in Figure 3. A variable E_R, referenced in this acceptance test, is maintained by Algorithm M-TBS, and stores the cumulative remaining execution requirement of all aperiodic jobs admitted thus far. Initially, $E_R \leftarrow 0$. When an aperiodic job is admitted, E_R is incremented by its execution requirement. If k aperiodic jobs are being executed, $k \leq m$, then E_R is decremented at a rate equal to k.

3.5 Handling generalized periodic tasks

In Sections 3.1–3.4 above, we derived an algorithm (summarized in Figure 3 above) for servicing aperiodic jobs in an environment in which HRT jobs generated by a partially generalized periodic task system (PGPTS) are to be scheduled to meet all deadlines upon a multiprocessor platform. We now consider the situation in which the HRT jobs are generated by a generalized periodic task system (GPTS). (The reader may wish to consult Table 1 to refresh his/ her memory concerning the precise difference between the two models.)

In this case, the arguments in Section 3.1 no longer hold. In brief, this is because the crucial step in that section is the computation of an upper bound on the completion time of an aperiodic job when it executes in the background. This upper bound was determined by computing an upper bound on the amount of work that could be generated by HRT jobs of the PGPTS. If the HRT jobs are generated by a GPTS, though, scheduling an aperiodic job in the background could result in a potentially unbounded response time for the aperiodic job since each periodic task τ_i could generate an arbitrarily large number of jobs, each with execution requirement at most $(U_i \times P_i^{(\mathsf{ub})})$, all arriving at the same time instant as the aperiodic job. All these HRT jobs would execute prior to surrendering the processor to the background server responsible for scheduling the aperiodic job, which would therefore see its execution pushed back by an arbitrary amount of time. Hence the analysis of Sections 3.1 and 3.2 – computing upper

```
**Admission control**: Should aperiodic job $J_i = (A_i, E_i, \zeta_i)$ be admitted?
if $\zeta \geq f_i$, where $f_i$ is computed according to Inequality 10, then
  then accept $J_i$ and assign it a deadline according to Equation 12, and update $E_R$ accordingly
  else reject $J_i$
end if
```

Figure 3. Algorithm M-TBS's acceptance test for aperiodic jobs.

bounds on the response time of a single and of multiple aperiodic job[s] respectively – no longer hold. However, we will show below that the results of Sections 3.3 and 3.4 continue to hold; in particular, the admission control/ deadline assignment algorithm presented in Section 3.4 (and summarized in Figure 3 above) remains correct for this generalized task model.

In brief, the reasoning that leads us to this conclusion is as follows.

- Recall that it is required that the multiprocessor scheduling algorithm that is used for scheduling the HRT jobs (assumed in this paper to be multiprocessor EDF) guarantee to meet deadlines of all the HRT jobs. For the case of a GPTS τ comprised of n generalized periodic tasks $\tau_1, \tau_2, \ldots, \tau_n$, this implies that the scheduling algorithm meet all HRT jobs' deadlines when the generalized periodic tasks generate *any* legal collection of jobs – i.e., each τ_i generates any sequence of jobs satisfying Equation 2.

- Consider now any collection of jobs $I(\tau)$ generated by GPTS $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$, with $I(\tau_i)$ denoting the jobs generated by τ_i.

- Suppose that an aperiodic job J is admitted at time instant A with execution requirement E and is assigned a deadline $A + D$, based upon a response-time upper bound computation of $A + f$ (see Figure 1). Let us examine the potential effect of this admitted aperiodic job upon the scheduling of jobs in $I(\tau_i)$. Let $A + f_o$ denote the actual completion time of the aperiodic job J when scheduled using deadline $A + D$; since f was a computed *upper* bound on the response time, it follows that $(A + f_o) \leq (A + f)$.

- It is possible that some job j in $I(\tau_i)$, which has arrival time $a(j) < A + f_o$ but deadline $d(j) > A + D$ (and hence lower deadline-priority than the admitted aperiodic job), is denied execution at some instant that it would have been executed had J not been admitted. However, this in itself cannot result in future missed deadlines for any HRT jobs in $I(\tau)$. To see why,

  1. Suppose that we were to modify $I(\tau)$, to *delay* job j's arrival until the instant that the aperiodic job J completes execution. More formally, $\hat{I}(\tau)$ is obtained from $I(\tau)$ by replacing j with a new job $\hat{j}$, with $a(\hat{j}) = A + f_o$, $e(\hat{j}) = e(j)$, and $d(\hat{j}) = d(j)$.
  2. Note that
  $$\begin{aligned} & d(\hat{j}) - a(\hat{j}) \\ & \quad \geq \quad (A + D) - (A + f_o) \\ & \quad \geq \quad D - f_o \\ & \quad \geq \quad D - f \\ & \quad = \quad P_{\mathsf{max}}(\tau) \quad \text{(by Equation 12)} \\ & \geq P_i^{(\mathsf{ub})} \ ; \end{aligned}$$
  hence, the value assigned to $e(\hat{j})$ does not violate Equation (2c), and $\hat{I}(\tau)$ is consequently a legal collection of jobs generated by GPTS τ.
  3. Therefore, the multiprocessor scheduling algorithm that is used for scheduling the HRT jobs (assumed in this paper to be multiprocessor EDF) guarantees to meet deadlines of all the HRT jobs in $\hat{I}(\tau)$.

- By repeated applications of the above argument to each job in $I(\tau)$ that is delayed due to the admission of each aperiodic job, we conclude that all HRT deadlines are met by Algorithm M-TBS even when the HRT jobs are generated by generalized periodic tasks satisfying the constraints enumerated in Equation 2.

4 Conclusions

In many hard-real-time application systems, there are occasional "aperiodic" jobs that need to be serviced in addition to the jobs that are generated by recurring ("periodic") tasks. In this paper, we have designed a scheduling algorithm that provides guaranteed performance to such aperiodic real-time jobs. While our goal is to provide upon multiprocessor platforms the same kind of service that is provided by the uniprocessor Total Bandwidth Server (TBS) of Buttazzo et al. [12, 14, 13], our approach is quite different, due in large part to the fact that while the uniprocessor TBS essentially uses capacity that is available for, but unused by, periodic tasks, our multiprocessor Algorithm M-TBS actually uses processor capacity that is not available to periodic tasks.

For ease of exposition, we have restricted our attention in this paper to sytems that use EDF for scheduling the jobs that are generated by the periodic task system. However, our techniques are applicable, and our results hold, for a much larger class of scheduling algorithms than merely EDF: if any multiprocessor scheduling algorithm A is able to guarantee that Condition 4 is satisfied, our reclamation techniques will hold for systems that use this algorithm A to schedule these periodic tasks' jobs.

References

[1] Anderson, J., and Srinivasan, A. Early release fair scheduling. In *Proceedings of the EuroMicro Conference on Real-Time Systems* (Stockholm, Sweden, June 2000), IEEE Computer Society Press, pp. 35–43.

[2] Andersson, B., Baruah, S., and Jansson, J. Static-priority scheduling on multiprocessors. In *Proceedings of the IEEE Real-Time Systems Symposium* (December 2001), IEEE Computer Society Press, pp. 193–202.

[3] Baruah, S. Utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors. Tech. Rep. TR03-022, Department of Computer Science, The University of North Carolina, June 2003.

[4] Baruah, S., Cohen, N., Plaxton, G., and Varvel, D. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica 15*, 6 (June 1996), 600–625.

[5] Baruah, S., Gehrke, J., and Plaxton, G. Fast scheduling of periodic tasks on multiple resources. In *Proceedings of the Ninth International Parallel Processing Symposium* (April 1995), IEEE Computer Society Press, pp. 280–288. Extended version available via anonymous ftp from `ftp.cs.utexas.edu`, as Tech Report TR–95–02.

[6] Baruah, S., Goossens, J., and Lipari, G. Implementing constant-bandwidth servers upon multiprocessor platforms. In *Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium* (San Jose, California, September 2002), IEEE Computer Society Press, pp. 154–163.

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

[8] Goossens, J., Funk, S., and Baruah, S. Priority-driven scheduling of periodic task systems on multiprocessors. *Real Time Systems 25*, 2–3 (2003), 187–205.

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

[10] Liu, C. L. Scheduling algorithms for multiprocessors in a hard real-time environment. *JPL Space Programs Summary 37-60 II* (1969), 28–31.

[11] Liu, J. W. S. *Real-Time Systems*. Prentice-Hall, Inc., Upper Saddle River, New Jersey 07458, 2000.

[12] Spuri, M., and Buttazzo, G. Efficient aperiodic service under earliest deadline scheduling. In *Proceedings of the Real-Time Systems Symposium* (San Juan, Puerto Rico, 1994), IEEE Computer Society Press, pp. 228–237.

[13] Spuri, M., and Buttazzo, G. Scheduling aperiodic tasks in dynamic priority systems. *Real-Time Systems: The International Journal of Time-Critical Computing 10*, 2 (1996).

[14] Spuri, M., Buttazzo, G., and Sensini, F. Robust aperiodic scheduling under dynamic priority systems. In *Proceedings of the Real-Time Systems Symposium* (Pisa, Italy, 1995), IEEE Computer Society Press, pp. 210–221.

[15] Srinivasan, A., and Baruah, S. Deadline-based scheduling of periodic task systems on multiprocessors. *Information Processing Letters 84*, 2 (2002), 93–98.