Executing aperiodic jobs in a multiprocessor constant-bandwidth server implementation

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

Giuseppe Lipari
Scuola Superiore S. Anna
Pisa, Italy

Abstract

The Constant Bandwidth Server (CBS) framework can be implemented on a preemptive uniprocessor platform to make full use of the computing capacity of the platform; when implemented upon a preemptive multiprocessor platform, however, it is known that the schedulable utilization is strictly less than the capacity of the platform. The issue of using the excess processing capacity 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.

1 Introduction

In many real-time application systems, the hard-real-time (HRT) jobs are often generated by relatively simple processes that are embedded within infinite loops, and hence essentially run “for ever.” The scheduling of collections of such recurring processes (also known as tasks) has therefore been the study of much research, particularly upon preemptive uniprocessor platforms. In addition, it is often the case that there are occasional non-recurring jobs that need to be executed along with the HRT jobs generated by the recurring tasks. Such aperiodic jobs are typically handled by an aperiodic server, that makes use of the processor capacity left over by the HRT jobs – those generated by the recurring tasks – 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 [9, 11, 10] 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, there is an additional constraint that must be considered – at each instant in time, an aperiodic job may be executing upon at most one processor. Hence in multiprocessor systems, 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 recurring tasks, we must also take into account the manner in which this unused capacity occurs. We have recently begun studying the issue of designing aperiodic servers for such multiprocessor platforms, that make efficient use of the processor capacity left unused by the recurring tasks’ jobs. In [3], we designed an aperiodic server for use in systems in which the recurring tasks are modelled using a minor generalization of the periodic task model of Liu and Layland [7, 6]; in this paper, we extend our aperiodic server to be applicable to systems where the recurring tasks are modelled using the far more general Constant Bandwidth Server (CBS) model of Abeni and Buttazzo [1]. That is, we design (and establish the correctness of) Multiprocessor Aperiodic Server (MAS) (Figure 1), a server for scheduling aperiodic jobs in real-time systems that consist of HRT recurring processes implemented upon a multiprocessor platform.

The remainder of this document is organized as follows. In Section 2 we describe related prior work on scheduling aperiodic and periodic jobs in multiprocessor environments. In Section 3, we describe the Constant Bandwidth Server (CBS) model of Abeni and Buttazzo [1]. In Section 4 we present MAS, an aperiodic server that, when used in conjunction with a real-time workload represented using the CBS model, is able to make efficient use of the excess processor capacity. We conclude in Section 5 with a summary of the results presented here.

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

The Total Bandwidth Server (TBS) abstraction was introduced by Buttazzo et al. [9, 11, 10] for servicing aperiodic jobs in a uniprocessor hard-real-time environment, in which a set of periodic tasks is scheduled using the preemptive Earliest Deadline First scheduling algorithm (Algorithm EDF) [6, 4]. 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 [9, 11, 10] achieve full processor utilization while simultaneously guaranteeing the timely execution of all HRT jobs.

In [3], we generalized TBS to be applicable to multiprocessor platforms. Specifically, we studied the EDF scheduling of periodic tasks and aperiodic jobs in multiprocessor environments, in which the periodic task model used is a generalization of the model proposed by Liu and Layland [6]. In this generalized model, a periodic task\(^1\) 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 \([a(j), d(j)]\). Each periodic task \(\tau_i\) is characterized by a utilization \(U_i\), and an upper bound upon the execution requirements of each of its jobs \(C^{(UB)}_i\). The jobs \(j_1, j_2, \ldots\), generated by \(\tau_i\) are required to satisfy the following properties:

\[
\begin{align*}
(1a) & \quad a(j_\ell) \geq 0; \quad a(j_{\ell+1}) = a(j_\ell) \text{ for all } \ell \\
(1b) & \quad e(j_\ell) \leq C^{(UB)}_i \text{ for all } \ell \\
(1c) & \quad d(j_\ell) = \max\{a(j_\ell), d(j_{\ell-1})\} + \frac{e(j_\ell)}{U_i} \text{ for all } \ell \geq 2
\end{align*}
\]

We will refer to periodic task systems that satisfy the constraints represented in Equation 1 above as generalized periodic task systems, or GPTS’s. (Note that this generalization places some requirements upon the implementation of the multiprocessor system-level scheduler – since there may be several jobs of the same task active simultaneously while our multiprocessor scheduling model allows each task to execute upon at most one processor at each instant in time, it is incumbent upon the scheduler to ensure that several jobs of a task are not simultaneously scheduled upon different

---

\(^1\)Note that what we call a periodic task here is sometimes referred to in the literature as a sporadic task.
Some further notation: for any generalized periodic task $\tau_i = (C_i^{(ub)}, U_i)$, let $P_i^{(ub)}$ denote the relative deadline of a job of $\tau_i$ that has the maximum permissible execution requirement: $P_i^{(ub)} \triangleq C_i^{(ub)}/U_i$. For any GPTS $\tau$, let $P_{\text{max}}(\tau)$ denote the largest relative deadline of any task in $\tau$ ($P_{\text{max}}(\tau) \triangleq \max_{i=1}^n \{P_i\}$); let $U_{\text{sum}}(\tau)$ denote the cumulative utilizations of all tasks in $\tau$ ($U_{\text{sum}}(\tau) \triangleq \sum_{i=1}^n U_i$); and let $U_{\text{max}}(\tau)$ denote the largest utilization of any task in $\tau$ ($U_{\text{max}}(\tau) \triangleq \max_{i=1}^n U_i$).

We have recently [5, 12, 2] 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\footnote{\textit{The schedulable utilization} of a scheduling algorithm is defined as follows [8]: \textit{"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 schedulable utilization of the algorithm."}} 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 $\tau$ is successfully scheduled upon $m$ unit-capacity processors by EDF is that

$$U_{\text{sum}}(\tau) \leq m \times (m-1) \times U_{\text{max}}(\tau).$$

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

In the remainder of this paper, we restrict our attention to task systems $\tau$ satisfying Inequality 2 above. For such systems, it is guaranteed that EDF will schedule the system to always meet all deadlines.

**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 $\tau$ denote a generalized periodic task system, 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 $\tau$.

1. For any $m$-processor scheduling algorithm $A$, let $A(J)$ 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 $\mathcal{W}(J', A(J), t)$ 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 $I(\tau_i)$ denote any collection of jobs that could legally be generated by task $\tau_i$. Let $I(\tau)$ denote any collection of jobs that could legally be generated by the tasks in $\tau$: $I(\tau) \triangleq \bigcup_{\tau_i \in \tau} I(\tau_i)$.

4. If $\tau$ satisfies Inequality 2, then 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 $\tau_i$ in $I(\tau)$, and let $j'$ denote the previous job generated by $\tau_i$ if any (if $j$ is the first job generated by $\tau_i$, 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 opt denote a scheduling algorithm that generates such a schedule, and hence let $\text{opt}(I(\tau))$ denote this schedule for $I(\tau)$ — i.e., $\text{opt}(I(\tau))$ assigns each job in $I(\tau)$ 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.

The following property of schedules generated using EDF upon multiprocessor platforms comprised of $m$ unit-capacity processors was proved in [5]:

**Lemma 1** ([5]) Let $\tau$ denote a task system satisfying Inequality 2 above:

$$U_{\text{sum}}(\tau) \leq m \times (m-1) \times U_{\text{max}}(\tau).$$

Algorithm EDF satisfies the following condition:

$$\forall I(\tau) : \forall t \in \mathbb{R} : \mathcal{W}(I(\tau), \text{EDF}(I(\tau)), t) \geq \mathcal{W}(I(\tau), \text{opt}(I(\tau)), t)$$

\[ (3) \]
An aperiodic server. In [3], we designed an aperiodic server for use in systems in which the recurring tasks are modeled using the GPTS model described above; in the remainder of this section, we describe the design of this aperiodic server.

Let \( J_1, J_2, \ldots \), denote the series of aperiodic jobs that arrive at the aperiodic server, with each \( J_i = (A_i, E_i) \) characterized by an arrival time and an execution requirement. 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 maintains an additional variable \( E_R \), which keeps track of the remaining execution requirement of all aperiodic jobs that have been accepted for execution. Let \( E_R(t) \) denote the value of this variable at time-instant \( t \), \( t \geq 0 \).

- \( E_R \) is initially equal to zero: \( E_R(0) = 0 \).
- When an aperiodic job \( J_i = (A_i, E_i) \) is admitted 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 \( t \).

If the aperiodic server were to execute in the background only — i.e., aperiodic jobs are executed only if there is no HRT job awaiting execution — it was shown [3] that the latest completion time for aperiodic job \( J_i \) is bounded from above by

\[
A_i + \left( \frac{mE_i + \sum_{\tau \in \tau_i} E_\tau^{(ub)} U_i(1 - U_i) + E_R(A_i)}{m - U_{\sum}(\tau)} \right)
\]

(4)

Rather than scheduling the aperiodic jobs as background jobs, we can instead have them scheduled by the EDF scheduler that is responsible for scheduling the HRT jobs, by assigning them appropriate deadlines. It was shown [3] that each job \( J_i \) could be assigned a deadline \( D_i \) equal to

\[
\max \left( D_{i-1}, \frac{mE_i + \sum_{\tau \in \tau_i} E_\tau^{(ub)} U_i(1 - U_i) + E_R(A_i)}{m - U_{\sum}(\tau)} + p_{\text{max}}(\tau) \right)
\]

(5)

Above, we have assumed that aperiodic jobs are not subject to any real-time constraints. An alternative model formulation 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 \( \eta_i \) 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 schedulability of HRT jobs or any previously admitted aperiodic jobs. That is,

- job \( J_i \) is admitted if and only if the upper bound on its completion time, as given by Inequality 4, is \( \leq \eta_i \); and
- \( E_R \) is incremented at time-instant \( A_i \) if and only if \( J_i \) is admitted.

3 The Constant Bandwidth Server

The Constant Bandwidth Server (CBS) scheduling framework was proposed by Abeni and Buttazzo [1] as a means of achieving the twin goals of per-task performance guarantees and inter-task isolation in certain kinds of multi-tasked real-time computer systems. In this framework, each CBS task \( \tau_i = (U_i, P_i) \) is characterized by two parameters — a (worst case) utilization \( U_i \), and a period \( P_i \). The utilization \( U_i \) denotes the amount of processor capacity that is to be devoted to the CBS task \( \tau_i \) (loosely speaking, it should seem to task \( \tau_i \) as though it were executing on a dedicated virtual processor of computing capacity \( U_i \)). The period \( P_i \) is an indication of the granularity of time from task \( \tau_i \)'s perspective — while this will be elaborated upon later, it suffices for the moment to assume that the smaller the value of \( P_i \), the more fine-grained the notion of real time for \( \tau_i \). It is assumed that each CBS task \( \tau_i \) generates a sequence of jobs \( j_1^i, j_2^i, j_3^i, \ldots \), with job \( j_k^i \) becoming ready for execution ("arriving") at time \( a(j_k^i) \) (with \( a(j_k^i) \leq a(j_k^i) + 1 \) for all \( i, k \)), and having an execution requirement equal to \( e(j_k^i) \) time units. The objective is to complete these jobs at about the same time that they would have completed, if the sequence were executing in a FCFS manner upon a dedicated processor (a "server") of computing capacity ("bandwidth") \( U_i \). The CBS model does not require that the execution requirement of a job be a priori known — the only way to know \( e(j_k^i) \) is to actually execute \( j_k^i \) to completion. Given a collection \( \tau \) of such tasks, whose parameters satisfy certain conditions specified in [1] that are to be executed on a single shared preemptable processor, the CBS scheduling algorithm presented in [1] makes the following performance guarantee: Let \( F_t^k \) denote the time instant at which job \( j_k^i \) would complete execution, if all jobs of task \( \tau_i \) were executed on a
Let $f^k_i$ denote the time instant at which $j^k_i$ completes execution under the CBS scheduling algorithm [1]. It is guaranteed that

$$f^k_i < F^k_i + P_i;$$

i.e., each job of each task $\tau_i$ is guaranteed to complete under the CBS scheduling algorithm no more than $P_i$ time units later than the time it would complete if executing on a dedicated processor. (This is what we mean when we refer to the period $P_i$ of a task $\tau_i$ as a measure of the "granularity" of time from the perspective of task $\tau_i$ — jobs of $\tau_i$ complete under the CBS scheduling algorithm within a margin of $P_i$ of the time they would complete on a dedicated processor.)

In [2], the M-CBS scheduling algorithm, a multiprocessor implementation of the CBS framework, was presented. In scheduling collection of CBS servers $\tau = \{\tau_1, \tau_2, \ldots, \tau_m\}$ upon $m$ unit-capacity processors, M-CBS makes the same performance guarantee as CBS provided the parameters of the CBS tasks in $\tau$ satisfy the condition given in Inequality 2 (i.e., $U_{\text{sum}}(\tau) \leq m - (m - 1) \times U_{\text{max}}(\tau)$).

In order to achieve its goal, the M-CBS algorithm maintains two variables: a deadline $D_i$ and a virtual time $V_i$, for each task $\tau_i$ in the system.

- The value of $D_i$ at each instant is a measure of the priority that the M-CBS server accords task $\tau_i$ at that instant — M-CBS performs earliest deadline first (EDF) scheduling based upon these $D_i$ values.

- The value of $V_i$ at any time is a measure of how much of task $\tau_i$'s "reserved" service has been consumed by that time. More precisely, the value of $V_i$ will be updated by Algorithm M-CBS in such a manner that, at each instant in time, task $\tau_i$ has received the same amount of service that it would have received by time $V_i$ if executing on a dedicated processor of capacity $U_i$. And, this can be achieved by incrementing $V_i$ at a rate $1/U_i$ whenever a job of $\tau_i$ is executing:

$$\frac{d}{dt} V_i = \begin{cases} 1/U_i, & \text{if } \tau_i \text{ is executing} \\ 0, & \text{otherwise} \end{cases}$$

—intuitively, executing $\tau_i$ for one time unit is equivalent to executing it for $1/U_i$ time units on a dedicated processor of capacity $U_i$.

**Task States.** At any instant in time during run-time, each task $\tau_i$ is in one of three states: inactive, activeContending, or activeNonContending. The initial state of each task is inactive. Intuitively at time $t_o$ a task is in the activeContending state if it has some jobs awaiting execution at that time; in the activeNonContending state if it has completed all jobs that arrived prior to $t_o$, but in doing so has "used up" its share of the processor until beyond $t_o$ (i.e., its virtual time is greater than $t_o$); and in the inactive state if it has no jobs awaiting execution at time $t_o$, and it has not used up its processor share beyond $t_o$.

Whenever a processor becomes available, the M-CBS algorithm chooses for execution some task that is in its activeContending state but is not currently executing upon another processor (if there is no such task, then the processor is idled). From among all the tasks that are in their activeContending state, the next job needing execution of the task $\tau_i$, whose deadline parameter $D_i$ is the smallest, is chosen for execution.

When the first job of $\tau_i$ arrives, $V_i$ is set equal to this arrival time. While (a job of) $\tau_i$ is executing, its virtual time $V_i$ is increased at a rate $(1/U_i)$ (as specified above, in Equation 7). If at any time this virtual time becomes equal to the deadline, then the deadline parameter is incremented by $P_i$ ($D_i \leftarrow D_i + P_i$). Notice that this may cause $T_i$ to no longer be the earliest-deadline active task, in which case it may surrender control of the processor upon which it is executing to an earlier-deadline task.

**State Transitions.** Certain (external and internal) events cause a task to change its state (see Figure 2):

1. If task $\tau_i$ is in the inactive state and a job $j^k_i$ arrives (at time-instant $a(j^k_i)$), then the following code is executed

$$V_i \leftarrow a(j^k_i)$$

$$D_i \leftarrow V_i + P_i$$

and task $\tau_i$ enters the activeContending state.
2. When a job \( j_k^i \) of \( \tau_i \) completes (at time-instant \( f_k^i \)) notice that \( \tau_i \) must then be in its activeContending state—the action taken depends upon whether the next job \( j_k^{i+1} \) of \( \tau_i \) has already arrived.

   (a) If so, then the deadline parameter \( D_i \) is updated as follows:
   \[
   D_i \leftarrow V_i + P_i
   \]
   and the task remains in the activeContending state.

   (b) If there is no job of \( \tau_i \) awaiting execution and \( V_i > f_k^i \) (i.e., the current value of \( V_i \) is greater than the current time) then task \( \tau_i \) changes state, and enters the activeNonContending state.

   (c) If there is no job of \( \tau_i \) awaiting execution and \( V_i \leq f_k^i \) (i.e., the current value of \( V_i \) is no larger than the current time) then, too, task \( \tau_i \) changes state and enters the inactive state.

3. For task \( \tau_i \) to be in the activeNonContending state at any instant \( t \), it is required that \( V_i > t \). When this ceases to be true, because time has elapsed since \( \tau_i \) entered the activeNonContending state but \( V_i \) does not change for tasks in this state, then the task enters the inactive state.

4. If a new job \( j_k^i \) arrives while task \( T_i \) is in the activeNonContending state, then the deadline parameter \( D_i \) is updated as follows:
   \[
   D_i \leftarrow V_i + P_i
   \]
   and task \( T_i \) returns to the activeContending state.

M-CBS’s performance guarantee. The performance guarantee that Algorithm M-CBS makes when scheduling a collection of Constant-Bandwidth Servers is the obvious analogue of the performance guarantee made by multiprocessor EDF when scheduling generalized periodic tasks (Lemma 1):

Let \( \tau \) denote a collection of Constant Bandwidth Servers satisfying Inequality 2:

\[
U_{\text{sum}}(\tau) \leq m \cdot (m - 1) U_{\text{max}}(\tau)
\]

Algorithm M-CBS (i) meets the CBS timing constraint (Inequality 6); and (ii) satisfies Condition 3:

\[
\forall (\tau) : \forall t \in R : W(f(\tau), \text{M-CBS}(f(\tau)), t) \geq W(f(\tau), \text{opt}(f(\tau)), t)
\]

It therefore follows that, if Algorithm MAS is used in an environment in which the HRT tasks are represented as Constant Bandwidth Servers (rather than according to the GPTS model) and satisfy Inequality 2, then Condition 3 holds for MAS as well:

\[
\forall (\tau) : \forall t \in R : W(f(\tau), \text{MAS}(f(\tau)), t) \geq W(f(\tau), \text{opt}(f(\tau)), t)
\]

4 Integrating an aperiodic server with CBS

We now describe how the jobs generated by a collection \( \tau \) of Constant Bandwidth Servers are equivalent, in a sense that is explained below, to those generated by a collection of generalized periodic tasks. As a consequence of this equivalence, the results we presented in Section 2 above concerning the scheduling of aperiodic jobs when recurring processes are modelled as generalized periodic tasks holds even when the recurring processes are modelled as Constant Bandwidth Servers.

Let \( j_1^i, j_2^i, \ldots, j_k^i, \ldots \) denote the sequence of jobs that are generated by Constant Bandwidth Server \( \tau_i \) during any specific run of the system. At the instant at which the M-CBS algorithm first considers jobs \( j_k^i \), server \( \tau_i \) could be in any of the three states (Figure 2).

- If \( \tau_i \) is inactive at this instant, then it undertakes Transition 1 in Figure 2. That is, M-CBS sets \( V_i \) to \( a(j_k^i) \), and \( D_i \) to \( a(j_k^i) + P_i \). This is equivalent, in the GPTS model, to generating a job with arrival time \( a(j_k^i) \), relative deadline \( P_i \), and execution requirement at most \( (U_i \times P_i) \).

- If \( \tau_i \) is either activeContending or activeNonContending at this instant, then it undertakes Transition 2a or Transition 4 respectively, in Figure 2. In either case, M-CBS sets \( D_i \) to \( V_i + P_i \). This is equivalent, in the GPTS model, to generating a job with arrival time at the current instant, a relative deadline at most \( P_i \) time-units beyond the previous job's relative deadline, and an execution requirement at most \( (U_i \times P_i) \).

The bound of \( (U_i \times P_i) \) on the execution requirement follows from the facts that (i) \( D_i \) is always set to \( V_i + P_i \); and (ii) whenever \( \tau_i \) executes, \( V_i \) increases at a rate \( 1/U_i \). Notice that EDF (and consequently, Algorithm MAS, since it is an EDF-based algorithm) does not make any use of the actual computation requirements of jobs in making scheduling decisions, but only needs to know whether a computation requirement is zero or not — i.e., whether a job is active or not. Hence, the lack of knowledge of the exact execution requirement of a job does not impact the algorithm.
As a consequence of this equivalence between CBS and GPTS tasks, the results derived in Section 4 above, concerning the performance of MAS in scheduling aperiodic jobs in an environment in which HRT jobs are generated by GPTS tasks, all remain valid when the HRT jobs are instead generated by CBS tasks. Specifically, Inequality 4 (the response time computation) and Inequality 5 (deadline assignment) remain valid when the HRT jobs are generated by CBS tasks rather than GPTS tasks. However, both these inequalities may be improved by MAS, as we now show, by keeping track of some additional information during run-time.

Let $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ denote a collection of Constant Bandwidth Servers, and consider the scheduling of a collection of HRT jobs $I(\tau)$ upon $m$ unit-capacity processors. Let $m(t), 0 \leq m(t) \leq m$, denote the number of processors that are used by the HRT scheduler at any time-instant $t \geq 0$ to schedule HRT jobs. Clearly,

$$W(I(\tau), \text{MAS}(I(\tau)), t_o) = \int_{t=0}^{t_o} m(t) \, dt. \quad (8)$$

At all time-instants $t$, let $U(t)$ denote the cumulative utilizations of all the HRT tasks in $I(\tau)$ that are currently active at that time-instant, and hence the only ones that are contributing to the work done by the optimal scheduling algorithm $\text{opt}$. That is, for all $t_o \geq 0$,

$$W(I(\tau), \text{opt}(I(\tau)), t_o) = \int_{t=0}^{t_o} U(t) \, dt. \quad (9)$$

Form Equations 8 and 9 above, we obtain

$$W(I(\tau), \text{MAS}(I(\tau)), t_o) - W(I(\tau), \text{opt}(I(\tau)), t_o) = \int_{t=0}^{t_o} (m(t) - U(t)) \, dt.$$

Using the short-hand notation $\Delta W(t_o) \equiv \int_{t=0}^{t_o} (m(t) - U(t)) \, dt$, to denote the "excess" execution that MAS has over $\text{opt}$ at time-instant $t_o$, we can state

$$\Delta W(t_o) = W(I(\tau), \text{MAS}(I(\tau)), t_o) - W(I(\tau), \text{opt}(I(\tau)), t_o). \quad (10)$$

Inequality 3 asserts that $\Delta W(t)$ is non-negative at all time-instants $t$, provided $\tau$ satisfies Inequality 2. In deriving the results described in Section 2 above, we have made use of this fact to bound the amount of higher-priority HRT work that may interfere with Algorithm MAS's (background) execution of an aperiodic job. However if Algorithm MAS were to use Equation 10 to keep track of the exact value of $\Delta W(I(\tau), t)$ for all $t$ during run-time, then we may use a more accurate value for $\Delta W(I(\tau), t)$ than this trivial lower-bound of zero to sometimes provide a tighter upper bound on the completion time of aperiodic jobs, and hence perhaps admit some aperiodic jobs that would not be admitted without this refinement.

**Computing $\Delta W(\cdot, \cdot)$**: In order to compute $\Delta W(I(\tau), t)$ during run-time, Algorithm MAS must keep track of $m(t)$ – the number of processors executing HRT jobs – and $U(t)$ – the cumulative utilization of all active GPTS tasks – at all time-instants $t$. The value of $m(t)$ is easily determined by MAS. The value of $U$ is initially set to zero. The following table shows the additional actions that must be taken by Algorithm MAS to maintain the value of $\Delta W(I(\tau), t_{\text{cur}})$ when CBS task $\tau_i$ undergoes a state transition at time-instant $t_{\text{cur}}$:

<table>
<thead>
<tr>
<th>Transition # (Figure 2)</th>
<th>Action Taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$U \leftarrow U - U_i$</td>
</tr>
<tr>
<td>2 (c)</td>
<td>$U \leftarrow U + U_i$</td>
</tr>
<tr>
<td></td>
<td>$\Delta W \leftarrow \Delta W + (t_{\text{cur}} - V_i) \cdot U_i$</td>
</tr>
<tr>
<td>3</td>
<td>$U \leftarrow U + U_i$</td>
</tr>
</tbody>
</table>

The incrementing/ decrementing of $U$ merely reflect the fact that the CBS task $\tau_i$ undergoing the respective transitions are entering/ leaving an inactive state (as is evident from Figure 2). The additional operation – $(\Delta W \leftarrow \Delta W + (t_{\text{cur}} - V_i) \cdot U_i)$ – resulting from transition 2 (c) is due to the following reason. Recall that the value of $V_i$ is maintained in such a manner that, at each instant in time, task $\tau_i$ has received the same amount of service that it would have received by time $V_i$ if executing on a dedicated processor of capacity $U_i$. Hence if a server $\tau_i$ undergoes transition 2 (c) (and thus becomes inactive) at time-instant $t_{\text{cur}}$ with $V_i < t_{\text{cur}}$, then it is the case that server $\tau_i$ was actually inactive over time-interval $[V_i, t_{\text{cur}}]$; only, we did not know this. Since we now do know it, we update the value of $\Delta W$ by the amount of work we had incorrectly attributed to $\tau_i$ over this interval.

By using the value of $\Delta W$, MAS can compute more accurate response-time bounds (equivalently, assign tighter deadlines) to aperiodic jobs. The improved bounds are presented in Table 1.

5 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
Response time of aperiodic jobs: Inequality 4 is refined to
\[
f_i \leq \left( \frac{mE_i + \sum_{r \in \mathcal{E}} P_rU_i(1 - U_i) + E_R(A_i) - \Delta W(A_i)}{m - U_{\text{sum}}(\tau)} \right)
\]
Deadline assigned: Inequality 5 is refined to
\[
D_i \leftarrow \max\left( D_{i-1}, A_i + \frac{mE_i + \sum_{r \in \mathcal{E}} P_rU_i(1 - U_i) + E_R(A_i) - \Delta W(A_i)}{m - U_{\text{sum}}(\tau)} + P_{\text{max}}(\tau) \right)
\]

Table 1. By storing some additional information during run-time, Algorithm MAS can provide tighter response times (and deadlines). These tighter results are presented here; the procedure for obtaining these tighter bounds is described in Section 4.

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. [9, 11, 10], 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 MAS actually uses processor capacity that is not available to periodic tasks.

References


