Proportional-Share Scheduling of Aperiodic Requests under the Rate-Based Execution Model*

Steve Goddard *Xin Liu*
Computer Science & Engineering
University of Nebraska—Lincoln
Lincoln, NE 68588-0115
{*goddard,lxin*}*@cse.unl.edu*

Technical Report TR02-050101

May 2002

Abstract

The rate-based execution (RBE) task model was developed to support the real-time execution of event-driven tasks in which no *a priori* characterization of the *actual* arrival rates of events is known; only the *expected* arrival rates of events is known. The RBE model is well suited for systems that must execute in environments that are not well-behaved (i.e., when the arrival rate of events is neither periodic nor sporadic).

Aperiodic requests with *unknown* execution times and *unknown* arrival patterns are mapped to RBE tasks and scheduled such that the real-time tasks are guaranteed to meet their deadlines while aperiodic requests share the available processor capacity without reserving a fixed processor capacity for any one aperiodic request. This approach was selected over the traditional approach of using a server task to process aperiodic requests so that the available processor capacity could be dynamically shared between active aperiodic requests.

1 Introduction

The rate-based execution (RBE) task model was developed to support the real-time execution of event-driven tasks in which no *a priori* characterization of the *actual* arrival rates of events is known; only the *expected* arrival rates of events is known [19]. The RBE model is a generalization of Mok's sporadic task model [27] in which tasks are expected to execute with an average execution rate of x times every y time units, and was motivated, in part, by distributed multimedia applications. A strength of the RBE task model is that it supports the *bursty* packet arrival pattern common in networked multimedia environments.

*Supported, in part, by grants from NASA (grant NCC5-169) and the National Science Foundation (grant CCR-0208619). This is the Technical Report version of "Scheduling Aperiodic Requests under the Rate-Based Execution Model," *Proceedings of the IEEE Real-Time Systems Symposium*, Austin, TX, December 2002.

The RBE model is an attractive execution model for systems that execute in unpredictable environments where the arrival pattern of events is neither periodic nor sporadic. However, many such systems also receive aperiodic command and control messages as well as other aperiodic requests. In many cases, the aperiodic requests do not have hard deadlines, but there is a clear difference in the urgency between aperiodic requests. That is, some aperiodic requests are much more urgent than other aperiodic requests and must be done sooner.

This work addresses the theory of integrating RBE tasks with aperiodic requests on a uniprocessor. The only known scheduling algorithm for RBE tasks is based on the earliest-deadline-first (EDF) scheduling algorithm, which requires the specification of task parameters that are generally unknown for aperiodic requests (or too pessimistic to be useful). Moreover, even the concept of an aperiodic request executing with a rate seems, on the surface, to be at odds with usual execution semantics of such requests. We usually think of an aperiodic request executing once and then terminating. In contrast, RBE tasks are expected to execute indefinitely, making x request every y time units.

The canonical approach to supporting aperiodic requests in a uniprocessor real-time system has been to add a server that processes aperiodic (non-real-time) requests [23, 32, 30, 15, 31, 13, 14, 1, 10, 9, 11, 22]. The server is allocated a portion of the CPU bandwidth and aperiodic requests are executed by the server such that no real-time job misses a deadline. The approach taken here differs in that we do not create a separate server for aperiodic requests. Instead, we dynamically map aperiodic requests with *unknown* execution times and *unknown* arrival rates to RBE tasks and schedule all tasks in the system with a simple EDF scheduling algorithm.

The primary contribution of this work is to generalize the theory of aperiodic request scheduling in hard-real-time systems. A portion of the processor capacity is allocated to aperiodic processing and each aperiodic request shares this capacity in proportion to its urgency, represented by a weight. Deadlines of hard real-time tasks are guaranteed to be met while aperiodic requests dynamically and proportionally share their allocation of processor capacity. The theory and approach presented in this work can be applied, with minor modifications, to deadline-driven scheduling of periodic and sporadic task models since RBE is a generalization of these models. Rather than using a weight to share processor capacity, a (variable) fraction of the processor capacity could be specified for each aperiodic request as long as an admission control algorithm ensured the sum of the fractions did not exceed the portion of processor capacity allocated to aperiodic requests. In this sense, the proportional sharing mechanism presented here could be applied to a set of aperiodic server tasks that dynamically change their size (e.g., a Total Bandwidth Server [30]) or period (e.g., Constant Bandwidth Sever [1]). The theory presented here can also be applied to adaptive real-time systems in which tasks dynamically change their resource requirements.

The rest of this paper is organized as follows. Section 2 introduces the processing model assumed in this work. Section 3 discusses related work in proportional share scheduling and canonical approaches to scheduling aperiodic requests in uniprocessor, real-time system. Section 4 presents the mapping of aperiodic

requests to RBE specified tasks. Section 5 discusses the feasibility of scheduling the integrated RBE task set using a simple extension of the EDF scheduling algorithm. The issue of fairness for aperiodic requests is also discussed in Section 5. Section 6 discusses some of the issues encountered in applying the theory to actual systems. We conclude with a summary and discussion of future work in Section 7.

2 The Model

The genesis for this work is a uniprocessor system for which the RBE task model was a good choice for the real-time processing requirements, but the system also had to support aperiodic command and control requests as well as other aperiodic processing that was not rate based.

The model presented assumes a uniprocessor system that consists of a set of two distinct classes of tasks: real-time tasks with hard deadlines and tasks representing aperiodic requests without deadlines. All tasks are independent of each other (i.e., they do not share resources) and are preemptable at arbitrary points. Real-time tasks make a sequence of requests that can be described with a RBE rate specification, as described in Section 2.1. Aperiodic requests consist of a single request with *unknown* duration that terminates (and leaves the system) after its processing requirement has been fulfilled. No *a priori* characterization of the arrival rates of aperiodic requests is known.

Real-time tasks are modeled as a set of RBE tasks whose membership is static during the life of the system. Aperiodic requests are mapped to a set of tasks whose membership changes over time. Thus, from a scheduling theory perspective, the system consists of two distinct classes of tasks: *RBE* tasks and *aperiodic* tasks. Formally, the task system $\mathcal{T}(t)$ at time t consists of the set $\mathcal{A}(t)$ of aperiodic tasks at time t and the set $\mathcal{R}$ of RBE tasks, which is independent of t: $\mathcal{T}(t) = \mathcal{A}(t) \cup \mathcal{R}$. The set of aperiodic requests over an interval of time $[t_1, t_2]$ is specified as $\mathcal{A}([t_1, t_2]) = \bigcup_{t=t_1}^{t_2} \mathcal{A}(t)$. Thus, over the interval $[t_1, t_2]$, the task system is specified as $\mathcal{T}([t_1, t_2]) = \mathcal{A}([t_1, t_2]) \cup \mathcal{R}$. When the context is clear the temporal parameter will be dropped from the notation: $\mathcal{T} = \mathcal{A} \cup \mathcal{R}$.

The rest of this section provides a more detailed description of the model assumed for real-time and (non-real-time) aperiodic tasks. Section 2.1 provides an overview of the RBE task model and the execution semantics of RBE tasks. Section 2.2 describes the execution semantics assumed for aperiodic tasks.

2.1 RBE Tasks

A task is a sequential program that is executed repeatedly in response to the occurrence of events. Each instance of the execution of the task is called a *job* or a *task instance*. Jobs are made ready for execution, or *released*, by the occurrence of an event. An event may be externally generated, e.g., a device interrupt, or internally generated, e.g., a message arrival. In all cases, once released, a job must execute to completion before a well-defined deadline. We assume instances of an event type are indistinguishable and occur infinitely often. Thus over the life of a real-time system an infinite number of jobs of each RBE task will be released.

The RBE task model is a generalization of the real-time task model developed by Mok [27], and later extended by Baruah *et al.* [5], and Jeffay *et al.* [21]. RBE provides two fundamental extensions to the sporadic task model. First, it makes no assumptions about the points in time at which events occur. It is assumed that events are generated at a precise average rate (e.g., 30 events per second) but that the actual distribution of events in time is arbitrary. Second, tasks specify a desired rate of progress in terms of the number of events to be processed in an interval of specified length. This allows a task to process a "burst" of simultaneous events as a single event.

A RBE task is specified by a four-tuple (x, y, d, c) of integer constants. The pair (x, y) is referred to as the *rate specification* of a RBE task; x is the maximum number of executions expected to be requested in any interval of length y. Parameter d is a response time parameter that specifies the maximum desired time between the release of a task instance and the completion of its execution (i.e., d is the relative deadline of the task). Parameter c is the maximum amount of processor time required for any job of task T to execute to completion on a dedicated processor. It is assumed that time is discrete and clock ticks are indexed by the natural numbers. Task parameters, x, y, d, and c are expressed as integer multiples of the interval between successive clock ticks.

A RBE task set is schedulable if there exists a schedule such that the j^{th} release of task T_i at time $t_{i,j}$ is guaranteed to complete execution by time $D_i(j)$, where

$$D_i(j) = \begin{cases} t_{ij} + d_i & \text{if } 1 \leq j \leq x_i \\ \max(t_{ij} + d_i, D_i(j - x_i) + y_i) & \text{if } j > x_i \end{cases} \quad (1)$$

Thus the deadline of a job is the larger of the release time of the job plus its desired deadline or the deadline of the x^{th} previous job plus the y parameter of the task. Therefore, up to x jobs of a task may contend for the processor with the same deadline. Note that for all j, deadlines of jobs J_{ij} and $J_{i,j+x_i}$ of task T_i are separated by at least y time units. Without this restriction, if a set of jobs of a task were released simultaneously it would be possible to saturate the processor. With the restriction, the time at which a task must complete its execution is not wholly dependent on its release time. This is done to bound processor demand. See [19] for a more detailed discussion on the RBE task model.

2.2. Aperiodic Requests

Neither the arrival rate nor the execution cost of aperiodic requests is assumed a priori. However, it is assumed that each aperiodic requests is associated with a weight that represents its relative urgency with respect to other aperiodic requests.[1] A request's weight, relative to the weight of other aperiodic requests, determines the share of the CPU capacity allocated to aperiodic request processing that the request will receive. This is the approach taken by many proportional-share resource allocation models to ensure fairness in resource sharing (e.g., [3, 26, 28, 35, 37, 38]).

[1] A default value of one can always be used. If all aperiodic requests have the same weight they also have the same level of urgency.

More formally, a weight $w_i > 0$ is associated with each aperiodic request $A_i \in \mathcal{A}$. Let $\hat{F}$ denote the fraction of the CPU capacity allocated to processing aperiodic requests. This fraction will be shared by the aperiodic tasks in proportion to their respective weights. Thus, if $\mathcal{A}(t)$ denotes the set of aperiodic requests at time t, the fraction $f_i(t)$ of the CPU each aperiodic request $A_i \in \mathcal{A}(t)$ should receive can be computed as

$$f_i(t) = \begin{cases} 0 & \text{if } A_i \notin \mathcal{A}(t) \\ \frac{w_i}{\sum_{j \in \mathcal{A}(t)} w_j} \hat{F} & \text{otherwise.} \end{cases} \tag{2}$$

The goal in scheduling aperiodic requests is to achieve a proportional sharing of the CPU capacity allocated for aperiodic requests. Thus, for any interval of time L, aperiodic task A_i would receive $f_i(t)L$ time units in a *perfectly fair system*. However, the model presented here only approximates a perfectly fair system in that the CPU will be allocated to aperiodic requests in discrete quanta less than or equal to a maximum system specified quantum q. (Real-time tasks are not so restricted.)

Generally following the terminology and notation introduced by Stoica *et al.* in [35], the CPU time aperiodic request A_i would receive in a perfectly fair system during the time interval $[t_1, t_2]$ is

$$S_i(t_1, t_2) = \int_{t_1}^{t_2} f_i(t)dt \tag{3}$$

time units. Let $s_i(t_1, t_2)$ be the actual number of time units allocated to aperiodic request A_i in the same interval. The difference between the amount of time the request would receive in a perfectly fair system and the time it actually receives in a given interval is called *lag*. The lag of A_i at time t is

$$lag_i(t) = S_i(t_i, t) - s_i(t_i, t) \tag{4}$$

where A_i first becomes eligible for execution at time t_i. Since a perfectly fair system cannot be implemented with discrete allocation quanta, the goal in scheduling will be to bound the lag for all aperiodic requests such that $\forall t \geq 0, i \in \mathcal{A}(t) : lag_i(t) < q$ where q is a system specified parameter that defines the scheduling quantum used to execute aperiodic requests. In fact, we will show that, when aperiodic requests are mapped to RBE tasks and scheduled as described in Section 4, the lag of aperiodic requests is bounded such that

$$\forall t \geq 0, i \in \mathcal{A}(t) : lag_i(t) \leq q(1 - f_i)$$

where f_i is the minimum non-zero fraction of the processor allocated to aperiodic request A_i.

The next section relates the work presented here to prior research results found in the literature.

3 Related Work

The RBE task model was formally presented in [19] and summarized in Section 2.1. It is a generalization of the model of sporadic tasks developed by Mok [27], and later extended by Baruah *et al.* [5], and Jeffay *et al.* [21]. The sporadic task model is a simple variant of the Liu and Layland periodic task model [25]. Whereas

periodic tasks recur at constant intervals, sporadic tasks (as defined by Mok) have a lower bound on their inter-invocation time, which creates an upper bound on their rate of occurrence. As described earlier, RBE tasks have an expected rate of execution, (x, y), rather than an exact or lower bound on inter-invocation times. The advantage of the the RBE task model is that correct execution of real-time tasks is not dependent on a well-behaved environment. This is the primary reason the RBE model was selected for this work.

A significant drawback to selecting the RBE model is that there has been no work showing how to support aperiodic processing (i.e, non-rate-based processing) within the model. One obvious method for supporting aperiodic requests is to extend the theory of aperiodic servers to the RBE model. However, we did not want the aperiodic requests to execute in a FIFO manner with respect to each other. A preemptive aperiodic server could have been implemented, as described for the Total Bandwidth Server (TBS) in [31], but the execution cost of some of the aperiodic requests is unknown a priori (for the system being supported).

A better approach than using a TBS would be to use a Constant Bandwidth Server (CBS) [1], or a set of CBSs, with each CBS representing a class of aperiodic requests. Each CBS could be modeled as a RBE task with a server budget $Q_s = q$ and a period $T_s = \frac{q}{f_i}$ where where q is a system specified parameter that defines the scheduling quantum used to execute aperiodic requests and f_i is the fraction of the processor capacity allocated to CBS i. The RBE parameters would then be $(1, T_S, T_S, Q_s)$. Whenever the CBS budget was exhausted, the server would be preempted and a new deadline set with Equation (1) as though one RBE job had terminated and another was released. Doing so results in the same deadline assignments described in [1] as long as only one aperiodic request was ever processed by a CBS at a time. However, this requires reserving a fixed fraction of CPU capacity for each CBS, which would go unclaimed if there was no pending aperiodic request for that server. The unused capacity would then be shared by *all* tasks in the system, including real-time tasks. The CASH algorithm presented in [9] could be used to share unused capacity with another CBS server. However, this creates a form of priority inversion with respect to urgency of aperiodic requests since the CASH algorithm allocates the unused capacity of one server to the next server that needs it, independent of the classification of the server.

Thus, it was decided to use weights rather than fixed classifications to prioritize the aperiodic requests and to share the available processor capacity for aperiodic requests in proportion to the requests' weight. The advantage of this approach is that it separates urgency from execution duration or the server's replenishment period in prioritizing the requests.

Most research in proportional share resource allocation (e.g., [26, 28, 37, 38, 8, 20, 35, 33, 34]) is based on the seminal work in bandwidth allocation for packet-switched networks by Demers et al. [12], Golestani [18], and Parekh and Gallager [29]. Weighted Fair-Queueing (WFQ) (also known as packet-by-packet generalized processor sharing) allocates a proportional share of a networks bandwidth to a session by employing a two-level hierarchical scheduler. The WFQ scheduler creates a queue for each session. Each queue is parameterized by a weight and an expected finish time for its first packet. When the first packet in queue i departs,

the expected finish time ft_i is recomputed for the next packet as $ft_i = max(r_i, ft_i) + \frac{l}{B_i}$, where B_i is the bandwidth reserved for session i, l is the size of the next packet, r_i is the arrival time of the next packet, and ft_i is the finish time of the packet. Packets within a queue are scheduled under the FIFO principle, which can be substituted with other scheduling policies as described in [8]. Although originally proposed as a non-preemptive scheduling algorithm (for network packets), WFQ can be easily modified to support preemptive task scheduling [24] and is the basis for BERT [7] and SMART [28].

Rather than employing the two-level WFQ hierarchy, the Earliest Eligible Virtual Deadline First (EEVDF) algorithm [35] schedules tasks according to their eligible times and deadlines in the virtual-time time domain (as proposed by Zhang [39] and independently by Parekh and Gallager [29]) using a simple EDF algorithm. Based on the weights of the tasks in the system, virtual time is computed; virtual time may progress faster, slower or at the same rate as real time. According to task weights, release time and execution time, the virtual eligible time ve and virtual deadline vd of a task is computed using equations presented in [35] and summarized as follows:

$$ve^1 = V(t_0^i); \qquad vd^k = ve^k + \frac{r^{(k)}}{w_i}; \qquad ve^{k+1} = vd^{(k)}.$$

Tasks are scheduled by observing the Earliest Eligible Virtual Deadline First rule to ensure that no real-time task is ever late by more than $(q-1)$ time units (in the real-time time domain), where q is the length of the scheduling quantum. Eligible time was introduced to prevent a task from being executed earlier than when it should in the perfect generalized processor share model, which is similar to WF^2Q [8].

Virtual time is widely used in proportional-share algorithms to cancel the affect of dynamic work loads. Since virtual time maintains the order of deadlines with respect to the order they occur in real time, it avoids deadline adjustment when system workload changes. However, when we combine hard real-time tasks with aperiodic requests (which do not have hard deadlines), deadlines of real-time tasks must be recomputed to preserve the share they require with respect to the aperiodic requests [36, 16]. Thus, the primary advantage of using virtual time is lost when the number of real-time tasks is greater than the number of aperiodic requests.

The work presented here combines elements from WFQ, EEVDF, and CBS. In some sense, it is a generalization of the CBS to support variable execution periods and a variable number of servers in the system, but the extension does not yet support resource sharing. The mapping of aperiodic requests to RBE tasks appears to be equivalent to maintaining a CBS server with a variable share for each aperiodic request, though this has not yet been verified. The total processor share of all aperiodic servers is fixed, equal to the share allocated to aperiodic requests. If aperiodic requests were mapped to periodic or sporadic tasks, rather than RBE tasks, the model would reflect a generalization of the Constant Utilization Server (CUS) first presented in [13] and extended as part of an open system in [14].

4. Scheduling Aperiodic Requests

Rather than creating a server process to schedule aperiodic requests, each aperiodic request in $\mathcal{A}$ is mapped to a RBE task and scheduled with the RBE tasks of $\mathcal{R}$ using a simple EDF algorithm. Since the actual computation time of an aperiodic request is not known a priori we model the aperiodic request as a RBE task with each job requiring q time units until the request terminates. A timer will be used to enforce a maximum request duration of q time units for each release of an aperiodic request.

The mapping is achieved by setting the RBE x parameter to 1 and the RBE c parameter to q. Using the same concept proposed by Spuri and Buttazzo in [30], the response time parameter d is set to $\frac{q}{f_i(t)}$. To complete the RBE specification, the y parameter is set to the same value, $\frac{q}{f_i(t)}$. In any interval between aperiodic requests arriving or terminating, $f_i(t)$ is equal to some constant f_i and these parameters are equal to the more familiar looking constant $\frac{q}{f_i}$ from [30] where q is the duration of the aperiodic request.

More formally, the function $\psi(A_i) : A_i \rightarrow \hat{T}_i$ maps aperiodic request $A_i \in \mathcal{A}(t)$ to RBE task $\hat{T}_i$ as follows:

$$\begin{aligned} \psi(A_i) : A_i \rightarrow \hat{T}_i &= (x_i, y_i(t), d_i(t), c_i) \\ &= (1, \frac{q}{f_i(t)}, \frac{q}{f_i(t)}, q) \end{aligned} \tag{5}$$

where $f_i(t)$, defined by Equation (2) in Section 2.2, is the fraction of the CPU allocated to aperiodic task $A_i \in \mathcal{A}(t)$ and q is the maximum allocation quantum for aperiodic requests. Since $d_i(t) = y_i(t)$, the fraction of the processor reserved for task $\hat{T}_i$ is $\frac{x_i c_i}{y_i(t)}$. This is the same share of the processor that needs to be allocated to aperiodic request A_i with weight w_i:

$$\frac{x_i c_i}{y_i(t)} = \frac{q}{\frac{q}{f_i(t)}} = f_i(t) = \frac{w_i}{\sum_{j \in \mathcal{A}(t)} w_j} \hat{F}.$$

See Section 4.3.1 for an example of two aperiodic requests being executed as RBE tasks.[2]

The observant reader may notice two potential problems with modeling aperiodic requests as RBE tasks defined using the mapping function $\psi(A_i)$. The first is that q may not be a multiple of the request's share $f_i(t)$, which violates the assumption of integral RBE parameters. This problem also occurs in the models presented in [30, 31, 13, 14, 35] and many other related models. Fortunately, this situation can be easily handled, as described in Section 6. The second potential problem with our approach is that the actual execution time of an aperiodic request may not be a multiple of q. This situation also occurs in the model assumed by Stoica *et al.* in [35] when the actual execution time does not match the expected execution time. It also occurs in the CBS proposed by Abeni and Buttazzo in [1] when the execution time is not a multiple of the server's budget. Our approach to this situation is very similar to the approach taken in [35] and is described in Section 6. To simplify the presentation, the remainder of this section and Section 5 assumes q is a multiple of $f_i(t)$ and the execution of times of all aperiodic requests are multiples of q.

[2] Rather than inserting examples after each new concept, Section 4.3 provides an extended example composed of subsections that illustrate each concept separately but with a common context.

In its simplest form, the scheduling of an aperiodic request proceeds as follows. When aperiodic request A_i arrives at time t_i, it is mapped to a RBE task and assigned a deadline using Equation (1). That is, $\psi(A_i) : A_i \rightarrow \hat{T}_i$ maps aperiodic request A_i to RBE task $\hat{T}_i$ and the first job of $\hat{T}_i$ is assigned a deadline of $t_i + d_i(t_i) = t_i + \frac{q}{f_i(t_i)}$. Since the processor share allocated to aperiodic request A_i does not change until the membership of $\mathcal{A}$ changes, $f_i = f_i(t_i)$ and $D_i(1) = t_i + \frac{q}{f_i}$ until an existing aperiodic request terminates or a new aperiodic request arrives.

The task $\hat{T}_i$ is inserted into the ready queue with other RBE tasks and scheduled with the EDF scheduling algorithm. When job J_{ij} of task $\hat{T}_i$ is dispatched (i.e., begins to execute), an execution timer is set to preempt the execution of job J_{ij} after q time units. If task $\hat{T}_i$ is preempted by another task, the execution timer state is saved with the context of task $\hat{T}_i$ and restored when job J_{ij} resumes execution. When the timer set for job J_{ij} expires, task $\hat{T}_i$ is preempted and, as though one job had completed and a new job released, a new deadline is set for job J_{ij+1} using Equation (1) and the RBE parameters of $\hat{T}_i$, which is similar to the method used by a CBS in [1] when a request overruns the server's budget.

The actual scheduling of aperiodic requests is a little more complicated in practice than described above, and illustrated in the simple example of Section 4.3.1, because the set of aperiodic requests is dynamic. The next section addresses the complexities of scheduling dynamic sets of aperiodic requests with a deadline driven algorithm, such as EDF.

4.1. Dynamic Deadline Adjustment

When a new aperiodic request arrives, the processor share of existing aperiodic requests decreases. When the processing required for an aperiodic request represented by task $\hat{T}_k$ completes and the task leaves the system, the processor share of other aperiodic requests increases. In both cases, the fraction f_i of the processor allocated to each existing aperiodic request must be recomputed using Equation (2). The change in processor share results in a change in the deadline for all pending aperiodic jobs. (Note that the deadlines for jobs of real-time applications remain unchanged.)

We show in Section 5 that if the task set was schedulable before the deadline changes, it will be schedulable after the deadline change and no task will miss its deadline.

There are two cases to be considered. The first is when an aperiodic request joins the system, which moves the deadlines of pending aperiodic jobs back (i.e., their deadlines occur later). The second is when an aperiodic request terminates and leaves the system, which moves the deadlines of pending aperiodic jobs up (i.e., their deadlines occur earlier).

Case 1: Aperiodic request A_x joins the system at time t_x. Let f_i' be the new fraction computed for $A_i \neq A_x \in \mathcal{A}(t_x)$ using Equation (2) at time t_x. Pending deadlines at time t_x are re-computed by dividing the expected remaining service time required to complete pending job J_{ij} by its new fraction f_i' and adding this to time t_x. Let r_i be the expected remaining service time required to complete job J_{ij}. That is, r_i denotes

the amount of remaining service time job J_{ij} would have in a perfectly fair system. Since aperiodic request A_i is modeled as RBE task $\hat{T}_i$ with $x_i = 1$ and $y_i(t) = d_i(t)$, the new deadline for the current job J_{ij} of task $\hat{T}_i$ is computed using Equation (6).

$$D'_i(j) = t_x + \frac{r_i}{f'_i} \tag{6}$$

In a perfectly fair system, the remaining service time r_i for job J_{ij} is computed as

$$r_i = \bar{S}_i(t, D_i(j)) = \int_{t_x}^{D_i(j)} f_i(t)dt = (D_i(j) - t_x) \cdot f_i \tag{7}$$

where $\bar{S}_i(t_1, t_2)$ denotes the service time task $\hat{T}_i$ would receive in a perfectly fair system if none of the weights were changed at time t_x (and f_i is the fraction of the processor that would have allocated to $\hat{T}_i$ in the interval).

By combining Equations (6) and (7), the deadline for pending aperiodic requests can be rewritten using Equation (8).

$$\begin{aligned} D'_i(j) &= t_x + \frac{\bar{S}_i(t_x, D_i(j))}{f'_i} \\ &= t_x + \frac{(D_i(j) - t_x) \cdot f_i}{f'_i} \\ &= t_x + (D_i(j) - t_x) \cdot \frac{f_i}{f'_i} \end{aligned} \tag{8}$$

See Section 4.3.2 for an example of deadline adjustments made when a new aperiodic request joins the system.

Case 2: Aperiodic request A_x terminates at time t_x^f. After A_x terminates at time t_x^f, the processor share allocated to each pending aperiodic request should increase since the total weight of all aperiodic requests decreases. In a perfectly fair system, the change in processor shares would happen immediately and the deadlines of pending aperiodic jobs would be updated using Equation (8) by substituting t_x with t_x^f. However, Equation (8) can only be used to update deadlines when A_x terminates with $lag_x(t_x^f) = 0$.

Aperiodic request A_x may terminate with non-zero lag since a perfectly fair system can only be approximated. To accommodate this approximation, the termination of aperiodic request A_x is treated as though it occurred at an expected finish time t_x^e such that $lag_x(t_x^e) = 0$. Deadlines of pending aperiodic requests can then be adjusted by substituting t_x with t_x^e in Equation (8). The deadline updates are made at time t_x^f and request A_x is allowed to leave the system immediately. However, the change in processor shares for the remaining aperiodic requests does not take effect until the expected finish time t_x^e of request A_x. In what follows, we show from a proportional share perspective that the deadlines of pending aperiodic jobs are changed to the same value whether we wait until time t_x^e to make the updates or if we update the deadlines immediately at time t_x^f.

The request is expected to terminate at its deadline. That is, $t_x^e = D_x(l)$ where $D_x(l)$ is the deadline when A_x terminates. Note: $D_x(l) \geq t_x^f$ always holds if all deadlines are met, and a sufficient condition for

determining the schedulability of the task set is presented and proven in Section 5. Since the actual service time is the same and only the expected service times differ, the lag of A_x at time $D_x(l)$ can be expressed as

$$lag_x(D_x(l)) = lag_x(t_x^f) + S_x(t_x^f, D_x(l)) = lag_x(t_x^f) + \int_{t_x^f}^{D_x(l)} f_x(t)dt = lag_x(t_x^f) + (D_x(l) - t_x^f) \cdot f_x(t_x^f).$$

Therefore, the lag of A_x at time t_x^f can be expressed as

$$\begin{aligned} lag_x(t_x^f) &= lag_x(D_x(l)) - (D_x(l) - t_x^f) \cdot f_x(t_x^f) \\ &= lag_x(D_x(l)) + (t_x^f - D_x(l)) \cdot f_x(t_x^f). \end{aligned} \tag{9}$$

Thus $D_x(l) = t_x^f - \frac{lag_x(t_x^f)}{f_x(t_x^f)}$ because $lag_x(D_x(l)) = 0$ when the task set is schedulable (by Theorem 5.9 in [17], which is Theorem A.8 in the Appendix).

$D_x(l)$ can now be substituted for t_x in Equation (8) to compute the new deadlines for pending aperiodic requests. Let W represent the weight summation of aperiodic requests, including w_x of request A_x, and W' represent the weight summation excluding w_x. The new deadlines for pending aperiodic requests are computed as follows.

$$\begin{aligned} D'_i(j) &= D_x(l) + (D_i(j) - D_x(l)) \cdot \frac{f_i}{f'_i} \\ &= (t_x^f - \frac{lag_x(t_x^f)}{f_x}) + (D_i(j) - (t_x^f - \frac{lag_x(t_x^f)}{f_x})) \cdot \frac{f_i}{f'_i} \\ &= t_x^f + (D_i(j) - t_x^f) \cdot \frac{f_i}{f'_i} - \frac{lag_x(t_x^f)}{f_x}(1 - \frac{f_i}{f'_i}) \\ &= t_x^f + (D_i(j) - t_x^f) \cdot \frac{f_i}{f'_i} - \frac{lag_x(t_x^f)}{f_x}\left(1 - \frac{\frac{w_i}{W}\hat{F}}{\frac{w_i}{W'}\hat{F}}\right) \\ &= t_x^f + (D_i(j) - t_x^f) \cdot \frac{f_i}{f'_i} - \frac{lag_x(t_x^f)}{f_x}(1 - \frac{W'}{W}) \\ &= t_x^f + (D_i(j) - t_x^f) \cdot \frac{f_i}{f'_i} - \frac{lag_x(t_x^f)}{f_x}(\frac{w_x}{W}) \\ &= t_x^f + (D_i(j) - t_x^f) \cdot \frac{f_i}{f'_i} - \frac{lag_x(t_x^f)}{\frac{w_x}{W}\hat{F}}(\frac{w_x}{W}) \\ &= t_x^f + (D_i(j) - t_x^f) \cdot \frac{f_i}{f'_i} - \frac{lag_x(t_x^f)}{\hat{F}} \end{aligned} \tag{10}$$

Observe that if aperiodic request A_x terminates with $lag_x(t_x^f) = 0$, then $D_x(l) = t_x^f$ and Equation (10) reduces to Equation (8), just as one would expect to occur under this condition.

The effect of Equation (10) is to distribute non-zero lag to the remaining aperiodic requests and allow requests to leave the system as soon as they terminate even though changes in processor share do not take effect until the deadline of the completed request. The same concept was used by Stoica *et al* in [35]. However, in this work the lag is distributed proportionally to the remaining aperiodic requests through deadline

adjustments. The main difference between our approach and that used in [35] is that our method operates in real time and not in virtual time. The approaches are similar in that each pending aperiodic request A_i will have its lag adjusted by $\overline{lag}_i = lag_x(t_x^f) \cdot \frac{w_i}{W'}$. In real-time this is accomplished by subtracting $\frac{\overline{lag}_i}{f_i'}$ from the updated deadline computed by Equation (8) for each pending aperiodic request represented by job J_{ij}. Since $f_i' = \frac{w_i}{W'}\hat{F}$, a proportionate distribution of the remaining lag of request A_x to pending aperiodic requests by modifying Equation (8) (with $t_x = t_x^f$) reduces to Equation (10):

$$\begin{aligned} D'_i(j) &= t_x^f + (D_i(j) - t_x^f) \cdot \frac{f_i}{f_i'} - \frac{\overline{lag}_i}{f_i'} \\ &= t_x^f + (D_i(j) - t_x^f) \cdot \frac{f_i}{f_i'} - lag_x(t_x^f) \cdot \frac{w_i}{W' \cdot \frac{w_i}{W'}\hat{F}} \\ &= t_x^f + (D_i(j) - t_x^f) \cdot \frac{f_i}{f_i'} - \frac{lag_x(t_x^f)}{\hat{F}} \\ &= \text{ Equation (10).} \end{aligned} \tag{11}$$

Thus, using Equations (8) and (10) the deadlines of all existing aperiodic jobs can be updated whenever an aperiodic request enters or leaves the system (respectively). Moreover, Equation (10) shows that, in an implementation, one can distribute lag proportionally by updating pending deadlines without actually tracking the lag; the new deadlines can be computed using the deadline of the leaving request, as shown in the first form of the equality expressed by Equation (10).

See Section 4.3.3 for an example of deadline adjustments made when an aperiodic request terminates and leaves the system.

4.2. Auxiliary Variable θ

What if the last aperiodic job J_{xl} in the system finishes and then another aperiodic request A_n arrives before the deadline of job J_{xl}? As currently defined, the deadline of job J_{n1} would be set using Equation (1) as $t_n + d_n$. However, unless the lag of request A_x is tracked and transferred to the new request, the deadline of job J_{n1} will be set too early, which will create more processor demand than aperiodic requests are allocated. As before, let $D_x(l)$ be the deadline of job J_{xl} (recall that $lag_x(D_x(l)) = 0$), t_x^f be the actual finish time, and t_n be the arrival time of request A_n. Intuitively, if request A_n arrives at time t_n such that $t_x^f < t_n \leq D_x(l)$, the deadline of job J_{n1} should be set to $D_x(l) + d_n$ rather than $t_n + d_n$, as specified by Equation (1). Observe that

$$\begin{aligned} \forall t \in [t_x^f, D_x(l)] : lag_x(t) &= S_x(t_x, t) - s_x(t_x, t) \\ &= S_x(t_x, t_x^f) + (t - t_x^f) f_x - s_x(t_x, t_x^f) \\ &= S_x(t_x, t_x^f) - s_x(t_x, t_x^f) + (t - t_x^f) f_x \\ &= lag_x(t_x^f) + (t - t_x^f) f_x \end{aligned} \tag{12}$$

The intuitive deadline assignment equation $D_n(1) = D_x(l) + d_n$ can be derived from Equation (1) such that the remaining lag of request A_x at time t_n is transferred to request A_n as follows.

$$\begin{aligned}
D_n(1) &= t_n + d_n - \frac{lag_x(t_n)}{\hat{F}} \\
&= t_n + d_n - \frac{lag_x(t_x^f) + (t_n - t_x^f)f_x}{\hat{F}} \quad \text{by Equation (12)} \\
&= t_n + d_n - \frac{lag_x(D_x(l)) + (t_x^f - D_x(l))f_x + (t_n - t_x^f)f_x}{\hat{F}} \quad \text{by Equation (9)} \\
&= t_n + d_n - \frac{lag_x(D_x(l)) + (t_n - D_x(l))f_x}{\hat{F}} \\
&= t_n + d_n - (t_n - D_x(l)) \quad \text{since } f_x = \hat{F} \text{ and } lag_x(D_x(l)) = 0 \\
&= D_x(l) + d_n
\end{aligned} \tag{13}$$

If request A_n arrives after time $D_x(l)$ (i.e., $t_n > D_x(l)$), then Equation (1) should be used to assign a deadline to job J_{n1} since $lag_x(t_n) = 0$ (and hence, the system lag is also zero).

Thus, the auxiliary variable θ is introduced to record the point in time at which the system lag reaches zero. Initially $\theta = 0$. Each time the last aperiodic job in the system terminates, the expected finish time of that job, $D_x(l)$, is recorded as $\theta = D_x(l)$. Using the auxiliary variable θ, the deadline of job J_{i1} for each newly arriving aperiodic request A_i at time t_i is computed using Equation (14).

$$D_i(1) = \max(\theta, t_i) + d_i \tag{14}$$

See Section 4.3.4 for an example using Equation (14) to set the deadline of an aperiodic request.

To summarize, Equations (1), (8), (10), and (14) for computing deadlines of aperiodic requests are combined in Equation (15) to form a single expression for computing deadlines of $\hat{T}_i = \psi(A_i)$.

$$D_i(j) = \begin{cases} \max(\theta, t_i) + d_i(t_i) & \text{if } j = 1 \\ \max(t_{ij} + d_i(t_{ij}), D_i(j-1) + y_i(t_{ij})) & \text{if } j > 1 \\ t_x + (D_i(j) - t_x)\frac{f_i}{f_i'} & \text{if } A_x \text{ arrives at } t_x \\ D_x(l) + (D_i(j) - D_x(l))\frac{f_i}{f_i'} & \text{if } A_x \text{ terminates at } t_x^f \end{cases} \tag{15}$$

When the task set is schedulable, the second line of Equation (15) can be reduced to $D_i(j-1) + y_i(t_{ij})$ since job J_{ij} of $\hat{T}_i$ is released as soon as job J_{ij-1} has executed for q time units.

4.3. Examples

This section provides an extended example composed of subsections that illustrate each concept separately with a common context. The fraction of the CPU allocated to aperiodic request processing is $\hat{F} = 0.4$ and the system assigned quantum for aperiodic requests is 10 (i.e., $q = 10$). Neither values will change during the life of the system. Initially, the weight summation of all aperiodic requests in $\mathcal{A}$ is 70, which will change over time.

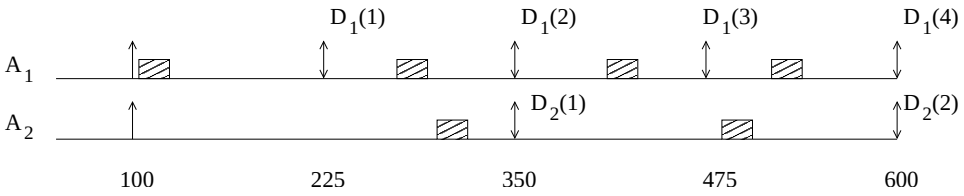

Figure 1: Execution pattern when no change in share allocations occur.

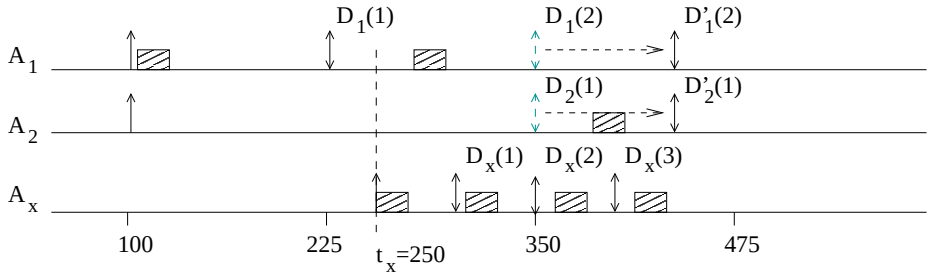

Figure 2: Deadline adjustments when a new aperiodic request arrives.

4.3.1. Nominal execution of A_1 and A_2

At time 100 A_1 and A_2 join the system with $w_1 = 20$ and $w_2 = 10$. The summation of weights in the system, W, now changes from 70 to 100. By Equation (2), the fraction of the processor allocated to each request is $f_1 = \frac{20}{100}0.4 = 0.08$ and $f_2 = \frac{10}{100}0.4 = 0.04$ respectively. Using Equation (5), A_1 and A_2 are mapped to RBE tasks $\hat{T}_1 = (1, 125, 10, 125)$ and $\hat{T}_2 = (1, 250, 10, 250)$. If no request enters or leaves the system after time 100, A_1 and A_2 will follow the execution pattern shown in Figure 1.

4.3.2. A New Aperiodic Request Arrives

To illustrate deadline adjustment when a new aperiodic request arrives, assume A_x arrives at time 250 with $w_x = 100$. W now changes from 100 to 200. Consequently the fractions of the CPU capacity allocated to A_1, A_2, and A_x at time 250 are set using Equation (2) as follows:

$$f_1 = \frac{w_1}{W}\hat{F} = \frac{20}{200}0.4 = 0.04,$$
$$f_2 = \frac{w_2}{W}\hat{F} = \frac{10}{200}0.4 = 0.02,$$
$$f_x = \frac{w_x}{W}\hat{F} = \frac{100}{200}0.40 = 0.2.$$

The RBE specifications are then changed using $\psi()$, defined by Equation (5): $\hat{T}_1 = (1, 250, 10, 250)$, $\hat{T}_2 = (1, 500, 10, 500)$, $\hat{T}_x = (1, 50, 10, 50)$. Finally, the deadlines of pending aperiodic requests are modified. Deadlines $D_1(2)$ and $D_2(()1$ are modified as follows and illustrated in Figure 2: $D_1(2) = 250 + (350 - 250) \cdot \frac{200}{100} = 450$, $D_2(()1 = 250 + (350 - 250) \cdot \frac{200}{100} = 450$.

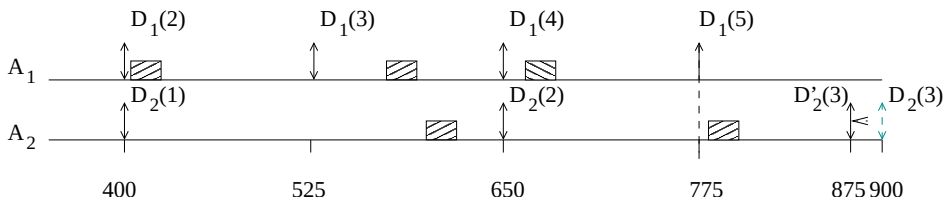

Figure 3: Deadline adjustment when an aperiodic request terminates.

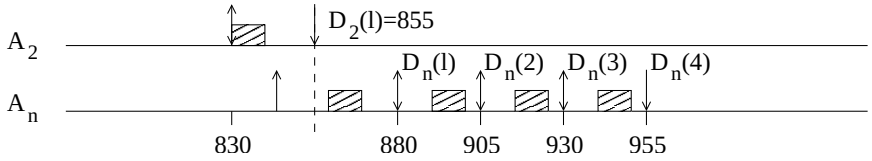

Figure 4: Deadline adjustment when an aperiodic request joins the system with no aperiodic requests pending.

4.3.3. An Aperiodic Request Terminates

Assume aperiodic request A_x terminated at some point between time 350 and time 475, with no other changes in the set of aperiodic requests, and deadlines were adjusted accordingly with $W = 100$. Then at time 670, assume aperiodic request A_1 terminates. At this point, the summation of weights in the system, W, changes from 100 to 80. Let $W' = 80$ represent the new weight summation. The fraction of the CPU allocated to request A_2 is changed to $f_2' = \frac{w_2}{W'}\hat{F} = \frac{10}{80}0.4 = 0.05$ and the deadline of A_2 needs to be changed. Using Equation (15), the new deadline is

$$\begin{aligned} D_2(2) &= D_1(5) + (D_2(2) - D_1(5))\frac{f_2}{f_2'} \\ &= 775 + (900 - 775)\frac{0.04}{0.05} = 875. \end{aligned}$$

Figure 3 illustrates this change.

4.3.4. Deadline Assignment with Variable θ

Assume the last aperiodic request A_2 terminates and leaves the system at time 840. The deadline of A_2 is recorded by $\theta = D_2(l) = 855$. If a new aperiodic request A_n with weight $w_n = 50$ arrives at time 843, it will take over the $\hat{F} = 0.4$ fraction of the CPU allocated to aperiodic processing. The RBE specification of A_n will be $\hat{T}_n = (1, 25, 10, 25)$ and its first deadline is $D_n(1) = max(843, 855) + 25 = 880$ as shown in Figure 4.

5 Schedulability and Bounding Lag

A task set is schedulable if there exists a schedule such that no task instance misses its deadline. Thus, if $Demand(L)$ represents the total processor demand in an interval of length L, a task set is schedulable if

$L \geq Demand(L)$ for all $L > 0$. Section 5.1 summarizes a prior result from [19] that bounds the processor demand of RBE tasks in an interval. Section 5.2 bounds the processor demand created by aperiodic requests. Section 5.3 combines the results of the first two subsections to create a sufficient schedulability condition for a task set $\mathcal{T} = \mathcal{A} \cup \mathcal{R}$ where $\mathcal{A}$ is the set of aperiodic tasks at any time $t \geq 0$ and $\mathcal{R} = \{(x_1, y_1, d_1, c_1), \ldots (x_n, y_n, d_n, c_n)\}$ is the set of real-time RBE tasks. Section 5.4 presents an least upper bound on the lag of any aperiodic request that holds when the task set is schedulable.

5.1. Bounding Demand for RBE Tasks

Lemma 5.1 was presented as Lemma 4.1 in [19] to bound the processor demand of a RBE task T_i in an interval. It is reproduced here (in a slightly different form) since it is used in the sufficient condition of Theorem 5.6 for the set of tasks $\mathcal{T}$ considered in this work.

Lemma 5.1. *For a RBE task $T_i = (x_i, y_i, d_i, c_i)$,*

$$\forall t > 0, \;\; dbf_i(t) = \begin{cases} 0 & \text{if } t \in [0, d_i) \\ \lfloor \frac{t - d_i + y_i}{y_i} \rfloor x_i c_i & \text{if } t \in [d_i, \infty] \end{cases} \tag{16}$$

is a least upper bound on the number of units of processor time required to be available in the interval $[0, L]$ to ensure that no job of T_i misses a deadline in $[0, L]$.

5.2. Bounding Demand for Aperiodic Requests

The demand bound function defined by Equation (16) assumes that the task may begin executing at time 0 and will continue to execute for the life of the system with fixed RBE parameters. Aperiodic requests enter and leave the system dynamically, which results in changing RBE parameters during the life of an aperiodic request A_i.

Let t_i denote the arrival time of aperiodic request A_i, $\hat{T}_i = \psi(A_i)$, and $D_i(l)$ be the deadline time of the last job J_{ij} of $\hat{T}_i$ representing aperiodic request A_i. Under these assumptions, the processor demand for $\hat{T}_i$ in the intervals $[0, t_i)$ and $(D_i(l), \infty]$ is 0 since the first job is not released until time t_i and the last job J_{il} of $\hat{T}_i$ completes by time $D_i(l)$. It should be the case, since we are trying to give each aperiodic request A_i a portion of the CPU capacity equal to $f_i(t)$ that the processor demand created by $\hat{T}_i$ is never greater than $\int_{t_i}^{l} f_i(t)dt$ for all $l \in [t_i, D_i(l)]$. Lemma 5.2 shows that this is indeed the case. Observe that when $f_i = f_i(t)$ is constant over the interval $[t_i, l]$, then $\int_{t_i}^{l} f_i(t)dt = (l - t_i)f_i$, which yields the expected demand for a fixed interval and processor share.

Lemma 5.2. *Let $\hat{T}_i = \psi(A_i)$ represent the aperiodic request $A_i \in \mathcal{A}(t)$. If no job of $\hat{T}_i$ released before time $t_0 \geq 0$ requires processor time in the interval $[t_0, l]$ to meet a deadline in the interval $[t_0, l]$, then*

$$\forall l > t_0, \;\; \widehat{dbf}_i([t_0, l]) = \int_{t_0}^{l} f_i(t)dt \tag{17}$$

is an upper bound on the processor demand in the interval $[t_0, l]$ *created by* $\hat{T}_i$ *where* $\psi(A_i)$ *is defined by Equation* (5) *and* $f_i(t)$ *is defined by Equation* (2)*.*

Proof: See the appendix of proofs, Section A. □

Clearly $t_0 = 0$ satisfies the requirement specified for t_0 in Lemma 5.2. Thus, with the simple substitution of $t_0 = 0$ and $l = L$, Corollary 5.3 follows immediately from Lemma 5.2.

Corollary 5.3. *Let* $\hat{T}_i = \psi(A_i)$ *represent the aperiodic request* $A_i \in \mathcal{A}(t)$*. The processor demand created by* $\hat{T}_i$ *will never exceed its processor share. That is,*

$$\forall L > 0, \;\; \widehat{dbf}_i([0, L]) = \int_0^L f_i(t)dt$$

is an upper bound on the processor demand in the interval $[0, L]$ *where* $\psi(A_i)$ *is defined by Equation* (5) *and* $f_i(t)$ *is defined by Equation* (2)*.*

Lemma 5.2 bounds the processor demand created by a single aperiodic requests in an interval. The following lemma extends this result to bound the processor demand created by all aperiodic requests in an interval.

Lemma 5.4. *If no job of an aperiodic request released before time* $t_0 \geq 0$ *requires processor time in the interval* $[t_0, l]$ *to meet a deadline in the interval* $[t_0, l]$*, then*

$$\forall l > t_0, \;\; (l - t_0)\hat{F} \tag{18}$$

is an upper bound on the processor demand in the interval $[t_0, l]$ *created by the set of aperiodic requests* $\mathcal{A}([t_0, l])$*.*

Proof: See the appendix of proofs, Section A. □

With the simple substitution of $t_0 = 0$ and $l = L$, Corollary 5.5 follows immediately from Lemma 5.4.

Corollary 5.5. *The processor demand created by the set of aperiodic requests* $\mathcal{A}$ *will never exceed its processor share,* $\hat{F}$*. That is,*

$$\forall L > 0, \quad L\hat{F}$$

is an upper bound on the processor demand in the interval $[0, L]$ *created by the set of aperiodic requests* $\mathcal{A}([0, L])$*.*

5.3. A Sufficient Schedulability Condition

The following Theorem presents a sufficient condition for determining the schedulability of the task set $\mathcal{T} = \mathcal{A} \cup \mathcal{R}$ where $\mathcal{A}$ is the set of aperiodic tasks at any time $t \geq 0$ and $\mathcal{R} = \{(x_1, y_1, d_1, c_1), \ldots (x_n, y_n, d_n, c_n)\}$ is the set of real-time RBE tasks. Corollary 5.7 shows that the schedulability of the task set $\mathcal{T}$ can be evaluated efficiently in polynomial time when all RBE d parameters are equal to their respective y parameters.

Theorem 5.6. *Let the task set $\mathcal{T} = \mathcal{A} \cup \mathcal{R}$ be the set $\mathcal{A} = \bigcup_{t=0}^{\infty} \mathcal{A}(t)$ of aperiodic tasks and the set $\mathcal{R} = \{(x_1, y_1, d_1, c_1), \ldots (x_n, y_n, d_n, c_n)\}$ of RBE tasks. Preemptive EDF will succeed in scheduling $\mathcal{T}$ if*

$$\forall L > 0, \;\; L \geq \sum_{i=1}^{n} dbf_i(L) + L\hat{F} \tag{19}$$

where $\hat{F}$ is the portion of the CPU capacity allocated to aperiodic requests $\mathcal{A}$ and $dbf_i(L)$ is as defined in Lemma 5.1.

Proof: See the appendix of proofs, Section A. □

Corollary 5.7. *Let the task set $\mathcal{T} = \mathcal{A} \cup \mathcal{R}$ be the set $\mathcal{A} = \bigcup_{t=0}^{\infty} \mathcal{A}(t)$ of aperiodic tasks and the set $\mathcal{R} = \{(x_1, y_1, d_1, c_1), \ldots (x_n, y_n, d_n, c_n)\}$ of RBE tasks with $d_i = y_i, 1 \leq i \leq n$. Preemptive EDF will succeed in scheduling $\mathcal{T}$ if Equation* (20) *holds where $\hat{F}$ is the portion of the CPU capacity allocated to aperiodic requests $\mathcal{A}$.*

$$\sum_{i=1}^{n} \frac{x_i \cdot c_i}{y_i} + \hat{F} \leq 1 \tag{20}$$

5.4. Bounding Lag

The goal in scheduling aperiodic requests is to approximate fair scheduling wherein each request receives time in proportion to its associated weight. By breaking the request into a sequence of request, each of duration q time units, we are able to identify exact points in time at which the request will have received its processor share. It is shown in [17] that if the task set is schedulable, the lag of aperiodic request A_i is guaranteed to be less than or equal to zero at the deadline of each job of $\hat{T}_i = \psi(A_i)$. The lag may be less than zero when, for example, real-time RBE tasks execute at lower rates than specified or for less than their worst-case execution times. When that happens, the aperiodic requests get more than their "expected share" of the processor. Without using eligible times to control the rate of execution of an aperiodic request, its lag can become negative because it receives more processor time than would otherwise be possible.

In this work, we are not interested in completely bounding fairness; we are only interested ensuring aperiodic requests receive a minimum processor share while real-time tasks meet all deadlines. Only an upper bound on the maximum lag that can accumulate for any aperiodic request can be derived when it is scheduled under the RBE model since tasks are allowed to execute faster than their rate specification if processor capacity is available. (This is a desirable feature for the application with which we are working.) One way to provide

a lower bound on processor lag (should one be needed), is to map aperiodic requests to sporadic tasks, track eligible times, and only release jobs of aperiodic requests when they are eligible—as was done by Stoica *et al.* in [35].

As the following theorem from [17] shows, when the task set is schedulable, our approach to scheduling aperiodic requests provides a least upper bound of $lag_i(t) \leq q(1 - f_i)$ on the maximum lag for aperiodic request A_i.

Theorem 5.8. *Let the task set $\mathcal{T} = \mathcal{A} \cup \mathcal{R}$ be the set $\mathcal{A} = \bigcup_{t=0}^{\infty} \mathcal{A}(t)$ of aperiodic tasks and the set $\mathcal{R} = \{(x_1, y_1, d_1, c_1), \ldots\ (x_n, y_n, d_n, c_n)\}$ of RBE tasks. If the task set is schedulable under preemptive EDF when deadlines are assigned using Equation* (15)*, the lag of aperiodic requests is bounded such that*

$$\forall t \geq 0, i \in \mathcal{A}(t) : lag_i(t) \leq q(1 - f_i) \tag{21}$$

where f_i is the minimum non-zero fraction of the processor allocated to aperiodic request (i.e., $f_i = \min\{f_i(t) | t \in [t_i, t_i^f]\}$).

Proof: See the appendix of proofs, Section A. □

6 Discussion

We now address the two potential problems with the model that were identified in Section 4: (*i*) when q is not a multiple of $f_i(t)$, and (*ii*) when the execution time of an aperiodic request is not a multiple of q.

6.1. What if q is not a multiple of $f_i(t)$?

In practice, mapping $\psi(A_i)$ defined by Equation (5) seems to works fine since the aperiodic jobs never execute for more than q time units and the real-time jobs seldom use all of the processor capacity they reserve. However, strictly speaking, the resulting RBE task specification may be invalid. The problem with $\psi(A_i)$ is that it does not ensure the resulting task parameters are positive integers, as assumed by the RBE model.

The obvious correction to this problem is to round the y and d parameters to the next higher integer values when necessary. Let $\psi'(A_i) : A_i \rightarrow T_i$ map A_i to task T_i as follows:

$$\begin{aligned} \psi'(A_i) : A_i \rightarrow T_i &= (x_i, y_i, d_i, c_i) \\ &= (1, \lceil \frac{q}{f_i} \rceil, \lceil \frac{q}{f_i} \rceil, q) \end{aligned} \tag{22}$$

Using $\psi'(A_i)$ for the mapping, the resulting task is guaranteed a slightly smaller fraction, f_i' of the processor:

$$f_i' = \frac{x_i c_i}{y_i} = \frac{q}{\lceil \frac{q}{f_i} \rceil} \leq \frac{q}{\frac{q}{f_i}} = f_i = \frac{w_i}{\sum_{j \in \mathcal{A}(t)} w_j}(1 - F).$$

However, this approach may change the *effective* relative weight with which each aperiodic request shares the CPU. That is, using $\psi'()$ may result in $f_i' = f_j'$ when $w_i \neq w_j$. This may or may not be a problem in practice, depending on the system requirements.

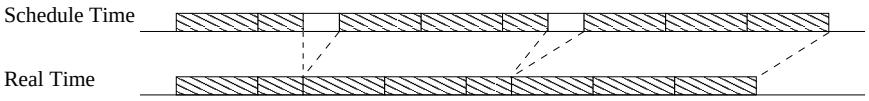

Figure 5: Mapping between schedule time and real time.

6.2. What if the execution time of A_x is not a multiple of q?

In the mapping from an aperiodic request to a RBE task, the execution time of the request is assumed to be a multiple of q. The actual execution time e_x is unknown. If e_x is not a multiple of q, then the deadline assigned to the last job of the aperiodic request will be a little later than the deadline would be if e_x were known: $(\lceil \frac{e_x}{q} \rceil \cdot q)/f_i > \frac{e_x}{f_x}$. There is no way to compute the "correct" deadline in advance since we do not know e_x, which is only known after request A_x terminates.

The impact of this problem is that the request requires less processor share in the interval $[D_x(l-1), D_x(l)]$ than it is allocated and terminates with more (positive) lag than accounted for by Equation (10). We cannot adjust the deadline after it is finished because doing so may violate the EDF principle. One way to solve this "problem" is to insert $\lceil \frac{e_x}{q} \rceil \cdot q - e_x$ units of idle time into the schedule at the end of the execution of A_x (when e_x is determined).

Of course we do not want the processor to be idle just so the theory of proportional share scheduling holds! Rather than idling the processor, the system can skip the idle interval by adjusting the *schedule time*– i.e., making a jump in schedule time. Schedule time, as it is introduced here, is similar to the concept of virtual time used in proportional share scheduling algorithms. (Making a jump in virtual time was used in [35] to solve the same problem.) The main difference between schedule time and virtual is that schedule time progresses at the same rate as real time, except when an aperiodic request uses less execution time than anticipated and schedule time advances $\lceil \frac{e_x}{q} \rceil \cdot q - e_x$ units instantaneously. For example, consider an aperiodic request that terminates at time t_x^f with 3 left in its timer counter. Instead of keeping the processor idle for 3 time units, we adjust the schedule time from t_x^f to $t_x^f + 3$. Then the deadlines of all pending aperiodic jobs are updated using Equation (10). The mapping between schedule time and real time is shown in Figure 5. Schedule time always progresses at or ahead of real time. Clearly, if all deadlines are met in schedule time, they are also met in real time.

An alternative approach to advancing schedule time is to adopt a method similar to that used by the resource reclaiming TBS presented in [31] or to the CBS CASH mechanism proposed in [9]. In this case, the unused processor allocation (still available) could be transferred to the next aperiodic request (if it arrived soon enough). However, this would result in violations of the proportional sharing principle.

7 Summary and Future Work

We have presented a task model and scheduling algorithm capable of executing RBE tasks and aperiodic requests using a simple EDF scheduler. Neither the arrival rate nor the execution duration of aperiodic requests must be known a priori. Our approach differs from the canonical approach in that we do not create a separate server for aperiodic requests. Instead each aperiodic request is dynamically mapped to a RBE task that shares the allocated processor capacity in proportion to its weight. If the sufficient schedulability condition of Theorem 5.6 is met, the hard deadlines of all real-time tasks are guaranteed to be met while aperiodic requests dynamically and proportionally share their allocation of processor capacity.

The primary contribution of this work is to generalize the theory of aperiodic request scheduling in hard-real-time systems when resources are not shared. Since deadline driven scheduling of periodic and sporadic tasks sets is a special case of scheduling RBE tasks, the theory and approach presented can be applied to periodic and sporadic task models by enforcing an inter-release time for jobs of an aperiodic request.

Rather than a weight, a fraction of the processor capacity could be specified for each aperiodic request as long as an admission control algorithm ensured the sum of the fractions did not exceed the portion of processor capacity allocated to aperiodic requests. Similarly, a specific quantum and period could be associated with each request. Thus, the proportional sharing mechanism presented could be applied to a set of Total Bandwidth Servers that dynamically change their size or a set of Constant Bandwidth Servers that dynamically change their period or budget. The theory presented here can also be applied to adaptive real-time systems in which tasks dynamically change their resource requirements.

Even as presented, our approach for scheduling aperiodic requests represents a generalization of the CBS first proposed in [1]. Each task $\hat{T}_i$ represents an instance of a CBS with a server budget $Q_s = q$ and a period $T_s = \frac{q}{f_i}$ that serves jobs for that task until the request terminates. When there exists only one aperiodic request in the system at a time, the execution schedule created by our approach is identical to one created by a CBS. Similarly, if each request requires at most q time units, the execution schedule created by our approach is identical to one created by the original TBS presented in [30].

Recently researchers have developed mechanisms for sharing resources among real-time periodic tasks and aperiodic servers [10, 11, 22]. Similar approaches may prove useful in relaxing the assumption made in this work that all tasks are independent. Moreover, we hope to combine the schedulability conditions presented in [19] for resource sharing among RBE tasks with deadlines less than their y parameter with a utilization test for aperiodic request processing similar to those presented in [10, 11, 22], depending on the synchronization protocol assumed. This should result in tighter sufficient schedulability conditions than those presented in [10, 11, 22].

A least upper bound on the difference between the ideal processor allocation in a perfectly fair processor allocation scheme and the actual allocation for an aperiodic request was also presented. This difference is called lag.

In general, a lower bound on lag cannot be determined without modeling the aperiodic requests as either periodic or sporadic tasks and setting release times for jobs of an aperiodic request. That is, the current mapping to RBE tasks does not result in a completely fair processor scheduling algorithm. Forcing a minimal inter-release time on jobs of an aperiodic request will result in tight bounds on fairness and the resulting model will change from resembling a generalization of CBS to a generalization of the Constant Utilization Server [13]. To achieve both fairness and good response times for aperiodic requests, minimum inter-release times could be relaxed when the processor is idle, using an approach similar to that described by Liu in [24] for a Starvation-Free Constant Utilization/Background Server.

We are investigating an alternative approach that breaks up the request of real-time jobs into multiple requests of length q. The advantages of this approach are that a lower bound on lag greater than $-q$ should be attainable, all requests will be able to make proportional progress, and job release times need not be enforced. The modified task set represents an instance of the Generalized Multiframe Task model introduced by Baruah *et al.* [6] and a new (more complicated) scheduling condition must be developed.

References

[1] Abeni, L., Buttazzo, G., "Integrating Multimedia Appplications in Hard Real-Time Systems," *Proc. IEEE Real-Time Systems Symp.*, Madrid, Spain, Dec. 1998.

[2] Anderson, D.P., Tzou, S.Y., Wahbe, R., Govindan, R., Andrews, M., "Support for Live Digital Audio and Video", *Proc. of the Tenth International Conference on Distributed Computing Systems*, Paris, France, May 1990, pp. 54-61.

[3] Baruah, S., Gehrke, J. E., Plaxton, C. G.., "Fast Scheduling of Periodic Tasks on Multiple Resources," *Proc. of the* 9^{th} *International Parallel Processing Symposium*, April 1995, pp. 280-288.

[4] Baruah, S., Howell, R., Rosier, L., "Algorithms and Complexity Concerning the Preemptively Scheduling of Periodic, Real-Time Tasks on One Processor," *Real-Time Systems Journal*, Vol. 2, 1990, pp. 301-324.

[5] Baruah, S., Mok, A., Rosier, L., "Preemptively Scheduling Hard-Real-Time Sporadic Tasks With One Processor," *Proc. 11th IEEE Real-Time Systems Symp.*, Lake Buena Vista, FL, Dec. 1990, pp. 182-190.

[6] Baruah, S., Chen, D., Gorinsky, S., Mok, A., "Generalized Multiframe Tasks," Real-Time Systems 17 (1), pp. 5-22. July 1999.

[7] Bavier, A., Peterson, L., Mosberger, D., "BERT: A scheduler for best effort and realtime tasks," Technical Report TR602 -99, Department of Computer Science, Princeton University, Mar. 1999.

[8] Bennett, J., Zhang, H., "WF2Q : Worst-case Fair Queueing," *Proc. of IEEE INFOCOM'96*, San-Francisco, March 1996.

[9] Caccamo, M., Buttazzo, G., Sha, L., "Capacity Sharing for Overrun Control," *Proc. IEEE Real-Time Systems Symp.*, Orlando, FL, Dec. 2000.

[10] Caccamo, M., Lipari, G., Buttazzo, G., "Sharing Resource among Periodic and Aperiodic Tasks with Dynamic Deadlines," *Proc. IEEE Real-Time Systems Symp.*, Phoenix, AZ, Dec. 1999.

[11] Caccamo, M., Sha, L., "Aperiodic Servers with Resource Constraints," *Proc. IEEE Real-Time Systems Symp.*, London, England, Dec. 2001.

[12] Demers, A., Keschav, S., Shenkar, S., "Analysis and Simulation of a Fair Queueing Algorithm," *Journal of Internetworking Research & Experiences*, Oct. 1990, pp. 3-12.

[13] Z. Deng, Liu, J.W.S., Sun, J., "A Scheme For Scheduling Hard Real-Time Applications in Open System Environment," In *Proceedings of the Ninth Euromicro Workshop on Real-Time System*, Toledo, Spain, June 1997, pp. 191-199.

[14] Deng, Z., Liu, J.W.S., "Scheduling Real-Time Applications in an Open Environment," *Real-Time Systems Journal*, vol. 16, no. 2/3, pp.155-186, May 1999.

[15] Ghazalie, T. M., Baker, T. P., Aperiodic Servers in Deadline Scheduling Environment, *Real-Time Systems Journal*, vol. 9, no. 1, pp. 31-68, 1995.

[16] Goddard, S., Tang, J., "EEVDF Proportional Share Resource Allocation Revisited," *Proceedings of the 21st IEEE Real-Time Systems Symposium Work in Progress*, Orlando, Florida, December 2000, pp. 21-24.

[17] Goddard, S., Liu, X. "Scheduling Aperiodic Requests under the Rate-Based Execution Model," TR02-050101, Dept. of CSE, UNL, May 2002.
[18] Golestani, S.J., "A Self-Clocked Fair Queueing Scheme for Broadband Applications," *Proc. of IEEE INFOCOM'94*, pp. 636-646, April 1994.
[19] Jeffay, K., Goddard, S., "A Theory of Rate-Based Execution," *Proceedings of the 20th IEEE Real-Time Systems Symposium*, Phoenix, Arizona, December 1999, pp. 304-314.
[20] Goyal, P., Guo, X., Vin, H.M., "A Hierarchical CPU Scheduler for Multimedia Operating Systems," *Proc. of the 2nd OSDI Symp.*, October 1996.
[21] Jeffay, K., Stanat, D.F., Martel, C.U., "On Non-Preemptive Scheduling of Periodic and Sporadic Tasks," *Proc. of the* 12^{th} *IEEE Real-Time Systems Symposium*, San Antonio, TX, 1991, pp. 129-139.
[22] Lamastra, G., Lipari, G., Abeni, L., "A Bandwidth Inheritance Algorithm for Real-Time Task Synchronization in Open Systems," *Proc. IEEE Real-Time Systems Symp.*, London, England, Dec. 2001.
[23] Lehoczky, J.P., Sha, L., and Strosnider, J.K., "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments," *Proceedings of IEEE Real-Time Systems Symposium*, pp. 261-270, Dec. 1987.
[24] Liu, J.W.S., *Real-Time Systems*, Prentice Hall, 2000.
[25] Liu, C., Layland, J., "Scheduling Algorithms for multiprogramming in a Hard-Real-Time Environment," *Journal of the ACM*, Vol 30., Jan. 1973, pp. 46-61.
[26] Maheshwari, U., "Charged-based Proportional Scheduling," Technical Memorandum, MIT/LCS/TM-529, Laboratory for CS, MIT, July 1995.
[27] Mok, A.K.-L., *Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment*," Ph.D. Thesis, MIT, Department of EE and CS, MIT/LCS/TR-297, May 1983.
[28] Nieh, J., Lam, M. S., "Integrated Processor Scheduling for Multimedia," *Proc. of the Fifth International Workshop on Network and Operating System Support for Digital Audio and Video*, Durham, N.H., April 1995.
[29] Parekh, A.K., Gallager, R.G. , "A Generalized Process Sharing Approach to Flow Control in Integrated Services networks-The Single Node Case," *IEEE/ACM Transactions on Networking*, Vol. 1, No. 3, pp. 334-357, 1992.
[30] Spuri, M., Buttazzo, G., "Efficient Aperiodic Service Under the Earliest Deadline Scheduling," *Proc. of the IEEE Symposium on Real-Time Systems*, December 1994.
[31] Spuri, M., Buttazzo, G., Sensini, F., "Robust Aperiodic Scheduling Under Dynamic Priority Systems," *Proc. of the IEEE Symposium on Real-Time Systems*, December 1995.
[32] Sprunt, B., Sha, L., Lehoczky, J.P., "Aperiodic Task Scheduling for Hard Real-time Systems," *Real-Time Systems Journal*, vol 1, no. 1, pp. 27-60, 1989.
[33] Stiladis, D., Varma, A., "Rate-Proportional Servers: A Design Methodology for Fair Queuing Algorithms," *IEEE/ACM Transactions on Networking*, vol 6, no 2, April 1998.
[34] Stiladis, D., Varma, A., "Efficient Fair Queuing Algorithms for Packet-Switched Networks," *IEEE/ACM Transactions on Networking*, vol 6, no 2, April 1998.
[35] Stoica, I., Abdel-Wahab, H., Jeffay, K., Baruah, S. K., Gehrke, J. E., Plaxton, C. G., "A Proportional Share Resource Allocation Algorithm for Real-Time, Time-Shared Systems," *Proc. of the IEEE Symposium on Real-Time Systems*, December 1996.
[36] Stoica, I., Abdel-Wahab, H., Jeffay, K., "On the Duality between Resource Reservation and Proportional Share Resource Allocation," *Multimedia Computing and Networking 1997*, SPIE Proceedings Series, Volume 3020, February 1997, pp. 207-214.
[37] Waldspurger, C. A., Weihl, W. E., "Lottery Scheduling: Flexible Proportional-Share Resource Management," *Proc. of the First Symposium on Operating System Design and Implementation*, Nov. 1994, pp. 1-12.
[38] Waldspurger, C. A., *Lottery and Stride Scheduling: Flexible Proportional-Share Resource Management*," Ph.D. Thesis, MIT, Laboratory for CS, September 1995.
[39] Zhang, L., "Virtual Clock: A New Traffic Control Algorithm for Packet-Switched Networks," *ACM Trans. On Comp. Systems*, Vol 9, No. 2, pp. 101-124, May 1991.

A Appendix of Proofs

This appendix provides the restatement and proof of all corollaries, lemmas, and theorems presented, but not proven, in the body of the paper.

Lemma A.1. *Let $\hat{T}_i = \psi(A_i)$ represent the aperiodic request $A_i \in \mathcal{A}(t)$. If no job of $\hat{T}_i$ released before time $t_0 \geq 0$ requires processor time in the interval $[t_0, l]$ to meet a deadline in the interval $[t_0, l]$, then*

$$\forall l > t_0, \;\; \widehat{dbf}_i([t_0, l]) = \int_{t_0}^{l} f_i(t)dt \tag{23}$$

is an upper bound on the processor demand in the interval $[t_0, l]$ created by $\hat{T}_i$ where $\psi(A_i)$ is defined by Equation (5) *and $f_i(t)$ is defined by Equation* (2)*.*

Proof: The proof of this lemma is separated into two parts. First, we show that Equation (23) is an upper bound on the processor demand created by task $\hat{T}_i$ when its share $f_i(t)$ never changes in the interval $[t_0, l]$. Second, we show the lemma also holds when $f_i(t)$ changes at any time $t_x \in [t_0, l]$, which is done by mapping segments of the interval to the first case.

Case 1: $f_i(t)$ never changes through the interval $[t_0, l]$. This is a straightforward reduction from Lemma 5.1, which states that the tight upper bound on processor demand created by a RBE task is $dbf_i(L) = \lfloor \frac{L - d_i + y_i}{y_i} \rfloor x_i c_i$. Let $f_i = f_i(t)$ be the constant share in the interval and $L = l - t_0$. Based on the mapping $\hat{T}_i = \psi(A_i)$: $x_i = 1, c_i = q, d_i = y_i = \frac{q}{\frac{w_i}{W}\hat{F}} = \frac{q}{f_i}$. Since $f_i(t)$ remains constant in the interval $[t_0, l]$ and no job of $\hat{T}_i$ released before time t_0 requires processor time in the interval, the demand created by $\hat{T}_i$ is

$$\begin{aligned} dbf_i(L) &= \lfloor \frac{L - d_i + y_i}{y_i} \rfloor x_i c_i = \lfloor \frac{L}{y_i} \rfloor q \\ &\leq \frac{L}{y_i} q = \frac{L}{\frac{q}{f_i}} q \\ &= L f_i = (l - t_0) f_i \\ &= \int_{t_0}^{l} f_i(t)dt = \widehat{dbf}_i([t_0, l]). \end{aligned}$$

Thus, Equation (23) is an upper bound on the processor demand in the interval $[t_0, l]$ created by $\hat{T}_i$ and the lemma holds for this case.

Case 2: $f_i(t)$ changes at time $t_x \in [t_0, l]$. Without loss of generality, assume t_x is the first time $f_i(t)$ changes in the interval $[t_0, l]$. The remaining portion of this proof will assume that $f_i(t)$ remains constant in the interval $[t_x, l]$. If $f_i(t)$ changes in the interval $[t_x, l]$, then this proof can applied recursively to that interval. There are two sub-cases to consider: when t_x is a deadline for a job of $\hat{T}_i$, and when it is not.

Case 2a: t_x is a deadline for a job of $\hat{T}_i$ in $[t_0, l]$, as shown in Figure 6.

In this case, the execution of $\hat{T}_i$ in the interval $[t_0, l]$ can be treated as two separate portions: $[t_0, t_x]$ and $[t_x, l]$. From the result of Case 1, the demand created by $\hat{T}_i$ in the subinterval $[t_0, t_x]$ is bounded from above by $\widehat{dbf}_i([t_0, t_x])$ since $f_i(t)$ does not change in the subinterval. Similarly, the demand created by $\hat{T}_i$ in the

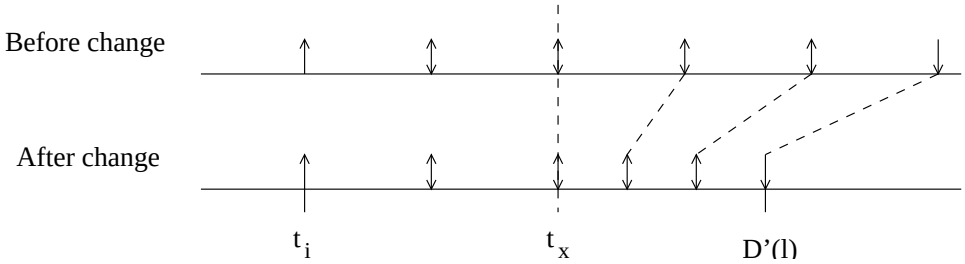

Figure 6: Dynamic deadline adjustment when t_x coincides with a deadline of $\hat{T}_i$.

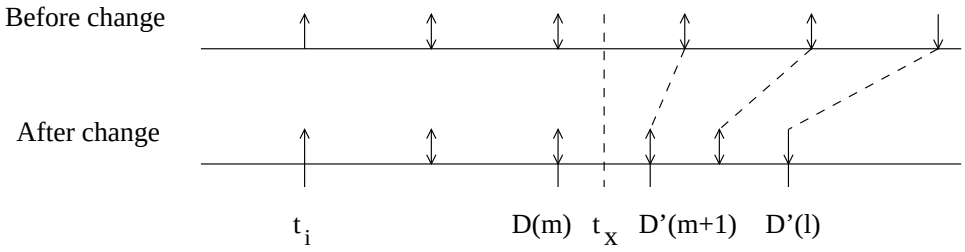

Figure 7: Dynamic deadline adjustment when t_x does not coincide with a deadline of $\hat{T}_i$.

subinterval $[t_x, l]$ is bounded from above by $\widehat{dbf}_i([t_x, l])$. Thus, the processor demand created by $\hat{T}_i$ in the interval $[t_0, l]$ less than or equal to

$$\begin{aligned}\widehat{dbf}_i([t_0, t_x]) + \widehat{dbf}_i([t_x, l]) &= \int_{t_0}^{t_x} f_i(t)dt + \int_{t_x}^{l} f_i(t)dt \\ &= \int_{t_0}^{l} f_i(t)dt = \widehat{dbf}_i([t_0, l])\end{aligned}$$

and the lemma holds for this case.

Case 2b: t_x is not a deadline for a job of $\hat{T}_i$ in $[t_0, L]$, as shown in Figure 7.

Let $D_i(m+1) > t_x$ be the (initial) deadline of job J_{im+1} at time t_x, and $D_i(m) < t_x$ be the deadline of job J_{im}. (If $t_x \in (t_i, D_i(1))$, let $D_i(m) = t_i$.) At time t_x, the share $f_i(t_x)$ changes and deadline $D_i(m+1)$ is recomputed using Equation (15). Let $D'_i(m+1)$ represent this recomputed deadline. Observe that $D'_i(m+1) > t_x$ when computed using Equation (15).

Thus, for this case, the interval $[t_0, l]$ is divided into three subintervals: $[t_0, D_i(m)]$, $[D_i(m), D'_i(m+1)]$ and $[D'_i(m+1), l]$. Let $f_i = f_i(t)$ for $t \in [t_0, t_x]$ and $f'_i = f_i(t)$ for $t \in [t_x, l]$.

From Case 2a, $\widehat{dbf}_i([t_0, D_i(m)])$ is an upper bound on the processor demand in the interval $[t_0, D_i(m)]$.

The demand created by $\hat{T}_i$ in the interval $[D_i(m), D_i(m+1)]$ is, by definition, q time units. Thus, since deadline $D_i(m+1)$ is recomputed to $D'_i(m+1)$, the processor demand created by $\hat{T}_i$ in the interval [$D_i(m)$,$D'_i(m+1)$] is q time units. Thus, the proof obligation is to show that $\widehat{dbf}_i([D_i(m), D'_i(m+1)]) \geq q$.

If t_x represents the time $f_i(t)$ changes due to an aperiodic request joining the system then

$$\begin{aligned}
\widehat{dbf}_i([D_i(m), D'_i(m+1)]) &= \int_{D_i(m)}^{D'_i(m+1)} f_i(t)dt \\
&= \int_{D_i(m)}^{t_x} f_i(t)dt + \int_{t_x}^{D'_i(m+1)} f_i(t)dt \\
&= (t_x - D_i(m))f_i + (D'_i(m+1) - t_x)f'_i \\
&= (t_x - D_i(m))f_i + (t_x + (D_i(m+1) - t_x) \cdot \frac{f_i}{f'_i} - t_x)f'_i \text{ by Equation (8)} \\
&= (t_x - D_i(m))f_i + (D_i(m+1) - t_x)f_i \\
&= (D_i(m+1) - D_i(m))f_i \\
&= y_i(D_i(m)) \cdot f_i(D_i(m)) = \frac{q}{f_i(D_i(m))} f_i(D_i(m)) = q
\end{aligned}$$

If t_x represents the time $f_i(t)$ changes due to aperiodic request A_x leaving the system then, $t_x = D_x(l)$ (the deadline of the last job of A_x) since that is when changes in processor shares take effect.

$$\begin{aligned}
\widehat{dbf}_i([D_i(m), D'_i(m+1)]) &= \int_{D_i(m)}^{D'_i(m+1)} f_i(t)dt \\
&= \int_{D_i(m)}^{t_x = D_x(l)} f_i(t)dt + \int_{t_x = D_x(l)}^{D'_i(m+1)} f_i(t)dt \\
&= (D_x(l) - D_i(m))f_i + (D'_i(m+1) - D_x(l))f'_i \\
&= (D_x(l) - D_i(m))f_i + (D_x(l) + (D_i(m+1) - D_x(l)) \cdot \frac{f_i}{f'_i} - D_x(l))f'_i \\
&\qquad \text{by Equation (15)} \\
&= (D_x(l) - D_i(m))f_i + (D_i(m+1) - D_x(l))f_i \\
&= (D_i(m+1) - D_i(m))f_i \\
&= y_i(D_i(m)) \cdot f_i(D_i(m)) \\
&= \frac{q}{f_i(D_i(m))} f_i(D_i(m)) \\
&= q
\end{aligned}$$

It follows that $\widehat{dbf}_i([D_i(m), D'_i(m+1)])$ is an upper bound on the processor demand in the interval $[D_i(m), D'_i(m+1)]$.

From Case 2a, $\widehat{dbf}_i([D'_i(m+1), l])$ is an upper bound on the processor demand in the interval $[D'_i(m+1), l]$ since $D'_i(m+1)$ meets the requirement for t_0 (in Case 2a) and $f'_i = f_i(t)$ for $t \in [D'_i(m+1), l]$.

Thus, the processor demand in the interval $[t_0, l]$ created by $\hat{T}_i$ is less than or equal to

$$\int_{t_0}^{D_i(m)} f_i(t)dt + \int_{D_i(m)}^{D'_i(m+1)} f_i(t)dt + \int_{D'_i(m+1)}^{l} f_i(t)dt = \widehat{dbf}_i([t_0, l])$$

and the lemma holds for this and each of the other cases. □

Corollary A.2. *Let $\hat{T}_i = \psi(A_i)$ represent the aperiodic request $A_i \in \mathcal{A}(t)$. The processor demand created by $\hat{T}_i$ will never exceed its processor share. That is,*

$$\forall L > 0, \;\; \widehat{dbf}_i([0, L]) = \int_0^L f_i(t)dt$$

is an upper bound on the processor demand in the interval $[0, L]$ where $\psi(A_i)$ is defined by Equation (5) *and $f_i(t)$ is defined by Equation* (2)*.*

Proof: Clearly $t_0 = 0$ satisfies the requirement specified for t_0 in Lemma A.1. Thus, with the simple substitution of $t_0 = 0$ and $l = L$, Corollary A.2 follows immediately from Lemma A.1. □

Lemma A.3. *If no job of an aperiodic request released before time $t_0 \geq 0$ requires processor time in the interval $[t_0, l]$ to meet a deadline in the interval $[t_0, l]$, then*

$$\forall l > t_0, \;\; (l - t_0)\hat{F} \tag{24}$$

is an upper bound on the processor demand in the interval $[t_0, l]$ created by the set of aperiodic requests $\mathcal{A}([t_0, l])$.

Proof: By Lemma A.1, Equation (23) provides an upper bound on the processor demand created by any request $A_i \in \mathcal{A}([t_0, l])$. Thus, demand created by all aperiodic requests in the interval $[t_0, l]$ is less than or equal to

$$\begin{aligned}
\sum_{i \in \mathcal{A}([t_0, l])} \widehat{dbf}_i([t_0, l]) &= \sum_{i \in \mathcal{A}([t_0, l])} \int_{t_0}^{l} f_i(t)dt \\
&= \int_{t_0}^{l} \sum_{i \in \mathcal{A}(t)} f_i(t)dt \\
&= (l - t_0)\hat{F}
\end{aligned}$$

□

Corollary A.4. *The processor demand created by the set of aperiodic requests $\mathcal{A}$ will never exceed its processor share, $\hat{F}$. That is,*

$$\forall L > 0, \quad L\hat{F}$$

is an upper bound on the processor demand in the interval $[0, L]$ created by the set of aperiodic requests $\mathcal{A}([0, L])$.

Proof: Clearly $t_0 = 0$ satisfies the requirement specified for t_0 in Lemma A.3. Thus, with the simple substitution of $t_0 = 0$ and $l = L$, Corollary A.4 follows immediately from Lemma A.3. □

Theorem A.5. *Let the task set $\mathcal{T} = \mathcal{A} \cup \mathcal{R}$ be the set $\mathcal{A} = \bigcup_{t=0}^{\infty} \mathcal{A}(t)$ of aperiodic tasks and the set $\mathcal{R} = \{(x_1, y_1, d_1, c_1), \ldots (x_n, y_n, d_n, c_n)\}$ of RBE tasks. Preemptive EDF will succeed in scheduling $\mathcal{T}$ if*

$$\forall L > 0, \;\; L \geq \sum_{i=1}^{n} dbf_i(L) + L\hat{F} \tag{25}$$

where $\hat{F}$ is the portion of the CPU capacity allocated to aperiodic requests $\mathcal{A}$ and $dbf_i(L)$ is as defined in Lemma 5.1.

Proof: To show the sufficiency of Equation (25), it is shown that the preemptive EDF scheduling algorithm can schedule all releases of tasks in $\mathcal{T}$ without any job missing a deadline if the tasks satisfy Equation (25). This is shown by contradiction.

Assume that $\mathcal{T}$ satisfies Equation (25) and yet there exists a release of a task in $\mathcal{T}$ that misses a deadline at some point in time when $\mathcal{T}$ is scheduled by the EDF algorithm. Let t_d be the earliest point in time at which a deadline is missed and let t_0 be the later of:

- the end of the last interval prior to t_d in which the processor has been idle (or 0 if the processor has never been idle), or
- the latest time prior to t_d at which a task instance with deadline after t_d stops executing prior to t_d (or time 0 if such an instance does not execute prior to t_d).

By the choice of t_0, *(i)* only releases with deadlines less than time t_d execute in the interval $[t_0, t_d]$, *(ii)* any task instances released before t_0 will have completed executing by t_0 or have deadlines after t_0, and *(iii)* the processor is fully utilized in $[t_0, t_d]$.

Thus, by a result due to Baruah *et al.* (Lemma 3.5 in reference [4]), at most

$$\sum_{i=1}^{n} \left\lfloor \frac{t_d - t_0 - d_i + y_i}{y_i} \right\rfloor x_i$$

instances of tasks in $\mathcal{R}$ can have deadlines in the interval $[t_0, t_d]$, and

$$\sum_{i=1}^{n} \left\lfloor \frac{t_d - t_0 - d_i + y_i}{y_i} \right\rfloor x_i c_i = \sum_{i=1}^{n} dbf_i(t_d - t_0)$$

is the least upper bound on the units of processor time required to be available in the interval $[t_0, t_d]$ to ensure that no task release misses a deadline in $[t_0, t_d]$.

By Lemma A.3, at most $(t_d - t_0)\hat{F}$ units of processing time are are needed to processes aperiodic requests in the interval under deadline scheduling and, by Lemma A.1, at most $\widehat{dbf}_j([t_0, t_d])$ time units are needed to process any single aperiodic request in the interval.

If the tasks in $\mathcal{T}$ are scheduled with a deadline driven scheduling algorithm, such as EDF, then

$$\sum_{i \in \mathcal{R}} dbf_i([t_0, t_d]) + \sum_{j \in \mathcal{A}([t_0, t_d])} \widehat{dbf}_j([t_0, t_d]) = \sum_{i=1}^{n} dbf_i([t_0, t_d]) + (t_d - t_0)\hat{F}$$

is an upper bound on the processor demand in the interval $[t_0, t_d]$. Thus, since the processor is fully used in the interval $[t_0, t_d]$ and since a deadline is missed at time t_d, it follows that

$$\sum_{i=1}^{n} dbf_i([t_0, t_d]) + (t_d - t_0)\hat{F} > (t_d - t_0).$$

However, this contradicts our assumption that $\mathcal{T}$ satisfies Equation (25).

Hence if $\mathcal{T}$ satisfies Equation (25), then no release of a task in $\mathcal{T}$ misses a deadline when $\mathcal{T}$ is scheduled by a deadline driven algorithm such as EDF. It follows that satisfying Equation (25) is a sufficient condition for schedulability under preemptive EDF. □

Corollary A.6. *Let the task set $\mathcal{T} = \mathcal{A} \cup \mathcal{R}$ be the set $\mathcal{A} = \bigcup_{t=0}^{\infty} \mathcal{A}(t)$ of aperiodic tasks and the set $\mathcal{R} = \{(x_1, y_1, d_1, c_1), \ldots (x_n, y_n, d_n, c_n)\}$ of RBE tasks with $d_i = y_i, 1 \leq i \leq n$. Preemptive EDF will succeed in scheduling $\mathcal{T}$ if Equation* (26) *holds where $\hat{F}$ is the portion of the CPU capacity allocated to aperiodic requests $\mathcal{A}$.*

$$\sum_{i=1}^{n} \frac{x_i \cdot c_i}{y_i} + \hat{F} \leq 1 \tag{26}$$

Proof:

$$\begin{aligned}
\sum_{i=1}^{n} \frac{x_i c_i}{y_i} + \hat{F} \leq 1 \implies \forall L > 0,\ L &\geq \sum_{i=1}^{n} L \cdot \frac{x_i c_i}{y_i} + L\hat{F} \\
&= \sum_{i=1}^{n} \frac{L}{y_i} \cdot x_i c_i + L\hat{F} \\
&= \sum_{i=1}^{n} \frac{L - y_i + y_i}{y_i} \cdot x_i c_i + L\hat{F} \\
&= \sum_{i=1}^{n} \frac{L - d_i + y_i}{y_i} \cdot x_i c_i + L\hat{F} \quad \text{since } d_i = y_i \\
&\geq \sum_{i=1}^{n} dbf_i(L) + L\hat{F}.
\end{aligned}$$

□

Theorem A.7. *Let the task set $\mathcal{T} = \mathcal{A} \cup \mathcal{R}$ be the set $\mathcal{A} = \bigcup_{t=0}^{\infty} \mathcal{A}(t)$ of aperiodic tasks and the set $\mathcal{R} = \{(x_1, y_1, d_1, c_1), \ldots (x_n, y_n, d_n, c_n)\}$ of RBE tasks. If the task set is schedulable under preemptive EDF when deadlines are assigned using Equation* (15)*, the lag of aperiodic request A_i is less than or equal to zero at the deadline of each of its jobs. That is,*

$$\forall k, i : 1 \leq k \leq l, i \in \mathcal{A}(t) :: lag_i(D_i(k)) \leq 0 \tag{27}$$

where l is the number of jobs executed for aperiodic request A_i.

Proof: By Equation (4), the lag of A_i at time $D_i(k)$ is

$$lag_i(D_i(k)) = S_i(t_i, D_i(k)) - s_i(t_i, D_i(k))$$

where A_i first becomes eligible for execution at time t_i. By Equation (3), $S_i(t_i, D_i(k)) = \int_{t_i}^{D_i(k)} f_i(t)dt$, and it was shown in the proof of Lemma A.1 that $\int_{D_i(k-1)}^{D_i(k)} f_i(t)dt = q$. Therefore

$$S_i(t_i, D_i(k)) = \int_{t_i}^{D_i(k)} f_i(t)dt = kq.$$

Once a job is released, its deadline is computed and a timer set to ensure that it never executes for more than q time units. Once the timer expires, the job is suspended (terminated) and the next job is released. Thus, if the task set is schedulable, *at least* k jobs, each of duration q time units, will have completed execution by time $D_i(k)$. Therefore, $s_i(t_i, D_i(k)) \geq kq$.

It immediately follows that

$$\begin{aligned} lag_i(D_i(k)) &= S_i(t_i, D_i(k)) - s_i(t_i, D_i(k)) \\ &= kq - s_i(t_i, D_i(k)) \\ &\leq 0 \end{aligned}$$

and the theorem holds. □

Theorem A.8. *Let the task set $\mathcal{T} = \mathcal{A} \cup \mathcal{R}$ be the set $\mathcal{A} = \bigcup_{t=0}^{\infty} \mathcal{A}(t)$ of aperiodic tasks and the set $\mathcal{R} = \{(x_1, y_1, d_1, c_1), \ldots (x_n, y_n, d_n, c_n)\}$ of RBE tasks. Let $D_i(l)$ be the deadline of the last job of aperiodic request A_i when it terminates. If the task set is schedulable under preemptive EDF when deadlines are assigned using Equation* (15) *and the execution time of aperiodic request A_i is a multiple of q, the lag of aperiodic request A_i is zero at time $D_i(l)$, the deadline of its last job. That is,*

$$\forall i \in \mathcal{A}(t) : lag_i(D_i(l)) = 0 \tag{28}$$

where l is the number of jobs of length q executed for aperiodic request A_i.

Proof: As shown in the proof of Theorem A.7, $S_i(t_i, D_i(l)) = \int_{t_i}^{D_i(l)} f_i(t)dt = lq$.

Once a job is released, its deadline is computed and a timer set to ensure that it never executes for more than q time units. Once the timer expires, the job is suspended (terminated) and the next job is released. Thus, if the task set is schedulable, *exactly* l jobs, each of duration q time units, will have completed execution by time $D_i(l)$. Therefore, $s_i(t_i, D_i(l)) = lq$.

It immediately follows that

$$\begin{aligned} lag_i(D_i(l)) &= S_i(t_i, D_i(l)) - s_i(t_i, D_i(l)) \\ &= lq - lq \\ &= 0 \end{aligned}$$

and the theorem holds. □

Theorem A.9. *Let the task set $\mathcal{T} = \mathcal{A} \cup \mathcal{R}$ be the set $\mathcal{A} = \bigcup_{t=0}^{\infty} \mathcal{A}(t)$ of aperiodic tasks and the set $\mathcal{R} = \{(x_1, y_1, d_1, c_1), \ldots (x_n, y_n, d_n, c_n)\}$ of RBE tasks. If the task set is schedulable under preemptive EDF when deadlines are assigned using Equation* (15)*, the lag of aperiodic requests is bounded such that*

$$\forall t \geq 0, i \in \mathcal{A}(t) : lag_i(t) \leq q(1 - f_i) \tag{29}$$

where f_i is the minimum non-zero fraction of the processor allocated to aperiodic request (i.e., $f_i = \min\{f_i(t) | t \in [t_i, t_i^f]\}$).

Proof: By Theorem A.7, $lag_i(D_i(k)) \leq 0, \forall k : 1 \leq k \leq l$ where l is total number of jobs of aperiodic request A_i. Thus, the maximum (positive) lag for request A_i must occur somewhere between two deadlines. This occurs when job J_{ij} does not begin to execute until time $D_i(j+1) - q$. If the job began to execute any later, it would miss its deadline at time $D_i(j+1)$ since each job requires q time units of execution.

Thus, let time $D_i(j+1) - q$ be the point in time at which the lag of request A_i is greatest. Since job J_{ij} does not execute in the interval $(D_i(j), D_i(j+1)-q)$, the lag of request A_i at time $D_i(j+1)-q$ is bounded such that

$$\begin{aligned}
lag_i(D_i(j+1)-q) &= S_i(t_i, D_i(j+1)-q) - s_i(t_i, D_i(j+1)-q) \\
&= S_i(t_i, D_i(j)) + S_i(D_i(j), D_i(j+1)-q) \\
&\qquad - (s_i(t_i, D_i(j)) + s_i(D_i(j), D_i(j+1)-q)) \\
&= S_i(t_i, D_i(j)) - s_i(t_i, D_i(j)) + S_i(D_i(j), D_i(j+1)-q) \\
&\qquad - s_i(D_i(j), D_i(j+1)-q) \\
&= lag_i(D_i(j)) + S_i(D_i(j), D_i(j+1)-q) - s_i(D_i(j), D_i(j+1)-q) \\
&= S_i(D_i(j), D_i(j+1)-q) - s_i(D_i(j), D_i(j+1)-q) \\
&= S_i(D_i(j), D_i(j+1)-q) \\
&= \int_{D_i(j)}^{D_i(j+1)-q} f_i(t)dt \\
&\leq ((D_i(j)-q) - D_i(j+1))f_i \\
&= (y_i(j)-q)f_i \\
&\leq (\frac{q}{f_i} - q)f_i \\
&= q(1-f_i).
\end{aligned}$$

When the processor share remains constant during the life of request A_i, $f_i = f_i(t)$ and $lag_i(t) = q(1-f_i)$, $\forall t \in [t_i, D_i(l)]$. Therefore (29) is a least upper bound on any request's lag when the task set is schedulable. □