Scheduling 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*

Abstract

Aperiodic requests with unknown *execution times and* unknown *arrival patterns are dynamically mapped to rate-based execution (RBE) tasks with variable rates and scheduled such that the real-time tasks are guaranteed to meet their deadlines. The aperiodic requests dynamically share the available processor capacity without reserving a fixed processor capacity for any one aperiodic request (or class of requests). This approach was selected over the traditional approach of using a static set of server tasks to process aperiodic requests so that the available processor capacity could be proportionally 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 [14]. The RBE model is a generalization of Mok's sporadic task model [22] 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 environments.

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. For this reason, the RBE model was selected to model the work load in a mobile robot, which interacts with peer robots and a base station. Each robot has a small set of hard-real-time tasks that execute with well-defined average rates. However, the robots also receive aperiodic requests from the base station to transmit information about its state or to initiate tasks such as sensor fusion and correlation, which execute for extended periods of time and transmit intermediate results to be used by the base station or other mobile robots. Frequently a set of these long-running requests will arrive at once and it is desirable for them to time-share available processor capacity with each other and aperiodic tasks already executing.

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 in this case).

The canonical approach to supporting aperiodic requests in a uniprocessor real-time system has been to add a server that processes the aperiodic (non-real-time) requests [17, 27, 25, 10, 26, 8, 9, 1, 5, 4, 6, 16]. The server is allocated a portion of the CPU bandwidth such that it progresses at a constant rate (or a fixed minimum rate). Within a server, the aperiodic requests are usually scheduled in FIFO order. An obvious drawback is that it neglects the aperiodic request's urgency, which can be represented by weight. Other drawbacks of existent server methods is that a fixed number of aperiodic request servers is assumed or they are allocated a "constant bandwidth." Even moderately complex systems have dynamic work loads in which it is desirable to support a variable number of servers, servers with variable bandwidth, or both.

In many proportional-share algorithms, the concept of virtual time was developed to solve the problem of dynamic work loads (e.g., [34, 24, 30]). Since virtual time maintains the order of finish times or deadlines (depending on the priority mechanism) with respect to the order they occur in real time, it avoids priority adjustment when the system workload changes. However, when hard real-time tasks (which have hard deadlines in real time) are combined with aperiodic requests (which are assigned soft deadlines in virtual time), the priority of real-time tasks must be mapped to vir-

[*]Supported, in part, by grants from NASA (grant NCC5-169) and the National Science Foundation (grant CCR-0208619).

tual time to satisfy their time constraints [31, 11]. 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 in a dynamic system.

The approach presented here combines elements of Earliest Eligible Virtual Deadline First (EEVDF) [30], Constant Bandwidth Server (CBS) [1], and GRUB [18]. Instead of creating a static set of servers for aperiodic requests, we dynamically map aperiodic requests to RBE tasks with variable rates (the rates change as the system workload changes). The primary limitation with the method presented is that the allocated bandwidth ratio between any two aperiodic servers must remain constant. This limitation is an artifact of the system requirement that aperiodic requests share available processor capacity in proportion to an associated weight. 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 [25]) or period (e.g., Constant Bandwidth Sever [1])—assuming the bandwidth ratio between any two servers remained constant. The bandwidth ratio limitation could also be removed by changing the deadline assignment function.

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. We conclude with a summary and discussion of future work in Section 6.

2. The Model

A uniprocessor system is assumed 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 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.

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 [14] 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. A request's weight, relative to the weight of other aperiodic requests, determines the rate at which the request is executed. This is equivalent to the approach taken by many proportional-share resource allocation models to ensure fairness in resource sharing (e.g., [2, 21, 23, 30, 32, 33]). The term variable rate is used rather than proportional share to be more consistent with the concept of the RBE model.

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 variable rate of execution based on a proportional sharing of the CPU capacity allocated for aperiodic requests. In 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 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 [30], 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 request quantum used to execute aperiodic requests. In fact, we will show that, when aperiodic requests are mapped to variable-rate 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

One obvious method for supporting aperiodic requests in the RBE model 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 [26], but the execution cost of some of the aperiodic requests is unknown a priori.

A better approach than using a TBS would be to use a CBS [1], or a set of such servers, 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 q is a system specified parameter that defines the request 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 [4] could be used to share unused capacity with another CBS server. However, the CASH algorithm allocates the unused capacity of one server to the next server that needs it, independent of the classification of the server. In our case, it was more desirable to share the available capacity between active aperiodic requests in proportion to their weight, which results in a variable number of aperiodic request servers with dynamic execution rates.

Most research in proportional share resource allocation (e.g., [21, 23, 32, 33, 3, 15, 30, 28, 29]) is based on the seminal work in bandwidth allocation for packet-switched networks by Demers et al. [7], Golestani [13], and Parekh and Gallager [24]. Weighted Fair-Queueing (WFQ) (also known as packet-by-packet generalized processor sharing) allocates a proportional share of a network's 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 [3]. Although originally proposed as a non-preemptive scheduling algorithm (for network packets), WFQ can be easily modified to support preemptive task scheduling [19].

Rather than employing the two-level WFQ hierarchy, the EEVDF algorithm [30] schedules tasks according to their eligible times and deadlines in the virtual-time time domain (as proposed by Zhang [34] and independently by Parekh and Gallager [24]) 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 [30] 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. Eligible time was introduced to prevent a task from being executed earlier than when it should in the generalized processor share model, which is similar to WF^2Q [3].

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 hard real-time tasks are combined 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 [31, 11]. 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, CBS and GRUB. 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 variable-rate RBE tasks appears to be equivalent to maintaining a CBS server with a variable rate 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.

4. Scheduling Aperiodic Requests

Rather than creating a dedicated server process to schedule a class of aperiodic requests, each aperiodic request in $\mathcal{A}$ is dynamically mapped to a variable-rate RBE task when the request arrives 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 variable-rate 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 [25], 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 [25] 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 variable-rate 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.[1]

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 variable-rate 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 variable-rate 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

[1] 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.

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 rate at which each aperiodic request is executed and, consequently, 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 variable-rate 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

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

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

$$\begin{aligned} lag_x(t_x^f) &= lag_x(D_x(l)) - (D_x(l) - t_x^f) \cdot f_x(t_x^f) \\ &= lag_x(D_x(l)) + (t_x^f - D_x(l)) \cdot f_x(t_x^f). \end{aligned} \tag{10}$$

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.

$D_x(l)$ 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.

$$\begin{aligned} D'_i(j) &= D_x(l) + (D_i(j) - D_x(l)) \cdot \frac{f_i}{f'_i} \\ &= (t_x^f - \frac{lag_x(t_x^f)}{f_x}) + (D_i(j) - (t_x^f - \frac{lag_x(t_x^f)}{f_x})) \cdot \frac{f_i}{f'_i} \\ &= t_x^f + (D_i(j) - t_x^f) \cdot \frac{f_i}{f'_i} - \frac{lag_x(t_x^f)}{f_x}(1 - \frac{f_i}{f'_i}) \\ &= t_x^f + (D_i(j) - t_x^f) \cdot \frac{f_i}{f'_i} - \frac{lag_x(t_x^f)}{f_x}\left(1 - \frac{\frac{w_i}{W}\hat{F}}{\frac{w_i}{W'}\hat{F}}\right) \\ &= t_x^f + (D_i(j) - t_x^f) \cdot \frac{f_i}{f'_i} - \frac{lag_x(t_x^f)}{f_x}(1 - \frac{W'}{W}) \\ &= t_x^f + (D_i(j) - t_x^f) \cdot \frac{f_i}{f'_i} - \frac{lag_x(t_x^f)}{f_x}(\frac{w_x}{W}) \\ &= t_x^f + (D_i(j) - t_x^f) \cdot \frac{f_i}{f'_i} - \frac{lag_x(t_x^f)}{\frac{w_x}{W}\hat{F}}(\frac{w_x}{W}) \\ &= t_x^f + (D_i(j) - t_x^f) \cdot \frac{f_i}{f'_i} - \frac{lag_x(t_x^f)}{\hat{F}} \end{aligned} \tag{11}$$

Observe that if aperiodic request A_x terminates with $lag_x(t_x^f) = 0$, then $D_x(l) = t_x^f$ and Equation (11) reduces to Equation (8), just as one would expect to occur under this condition.

The effect of Equation (11) 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 [30]. 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 [30] 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 proportional 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 (11).

Thus, using Equations (8) and (11) the deadlines of all existing aperiodic jobs can be updated whenever an aperiodic request enters or leaves the system (respectively). Moreover, Equation (11) 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 (11).

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

4.2. Deadline Assignments

The previous section presented a method for dynamically adjusting the deadline of pending aperiodic requests when the membership of $\mathcal{A}$ changes. We now address the issue of initial deadline assignment and then summarize the computation of deadlines for all aperiodic requests with a single deadline assignment function, Equation (15).

When a new aperiodic request A_i enters the system with existing aperiodic requests, it is assigned a deadline of $t_i + d_i(t_i) = t_i + \frac{q}{f_i(t_i)}$. However, 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}, the deadline of job J_{n1} will be set too early unless the lag of request A_x is tracked and transferred to the new request.

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. To simplify notation, let $d_n = d_n(t_n)$. 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 $t_n + d_n$. Observe that

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

Thus, 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.

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

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).

$$D_i(1) = \max(\theta, t_i) + d_i \qquad (14)$$

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

To summarize, Equations (1), (8), (11), and (14) for computing deadlines of aperiodic requests are combined in Equation (15) to form a single expression for computing deadlines of $\hat{T}_i = \psi(A_i)$.

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

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 request 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.

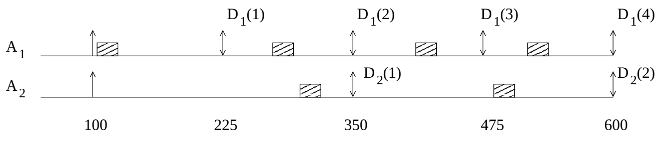

Figure 1. Execution pattern when no change in share allocations occur.

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 variable-rate 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:

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

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: $D_1(2) = 250 + (350 - 250) \cdot \frac{200}{100} = 450$, $D_2(1) = 250 + (350 - 250) \cdot \frac{200}{100} = 450$.

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.

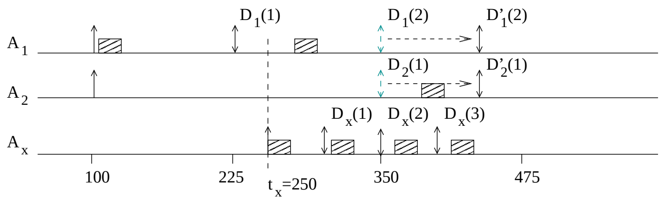

Figure 2. Deadline adjustments when a new aperiodic request arrives.

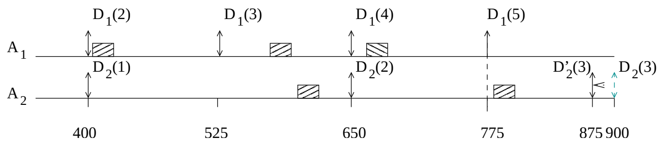

Figure 3. Deadline adjustment when an aperiodic request terminates.

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

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

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.

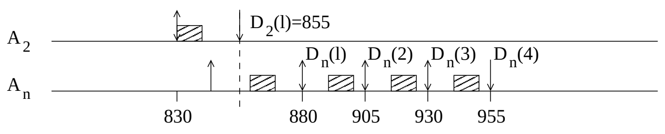

Figure 4. Deadline adjustment when an aperiodic request joins the system with no aperiodic requests pending.

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 [14] 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 a 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 [14] 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 $T_i = (x_i, y_i, d_i, c_i)$,*

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

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: For space considerations, the proofs of this section are contained in [12]. □

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 [12]. □

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 [12]. □

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 execute each request with a variable-rate such that it makes progress relative to other aperiodic requests 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 [12] 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 execute at faster rates than specified, receiving 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 [30].

As the following theorem from [12] shows, when the task set is schedulable, our approach to scheduling aperiodic requests provides a least upper bound of $lag_i(t) \le q(1 - f_i)$ 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*

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

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 [12]. □

6. Summary

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 static set aperiodic request servers. Instead each aperiodic request is dynamically mapped to a variable-rate 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 contributions of this work are to introduce the concept of variable-rate RBE tasks and to generalize the theory of aperiodic request scheduling in hard-real-time systems when resources are not shared. Since deadline driven scheduling of periodic or sporadic task 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 and the bandwidth ratio between any two aperiodic servers remained constant. The server bandwidth-ratio limitation is an artifact of the system requirement that aperiodic requests share available processor capacity in proportion to an associated weight. Removal of this requirement requires new deadline adjustment equations that are beyond the scope of this paper.

Instead of associating either a weight or bandwidth fraction, 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—as long as the resulting bandwidth ratio between any two servers remained constant.

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 [25].

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] 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.

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

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

[5] 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.

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

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

[8] 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.

[9] 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.

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

[11] 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.

[12] Goddard, S., Liu, X. “Proportional-Share Scheduling of Aperiodic Requests under the Rate-Based Execution Model,” TR02-050101, Dept. of CSE, UNL, May 2002.

[13] Golestani, S.J., “A Self-Clocked Fair Queueing Scheme for Broadband Applications,” *Proc. of IEEE INFOCOM’94*, pp. 636-646, April 1994.

[14] 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.

[15] Goyal, P., Guo, X., Vin, H.M., “A Hierarchical CPU Scheduler for Multimedia Operating Systems,” *Proc. of the 2nd OSDI Symp.*, October 1996.

[16] 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.

[17] 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.

[18] G. Lipari, S. Baruah, “Greedy reclamation of unused bandwidth in constant-bandwidth servers”, *Proceedings of the EuroMicro Conferences on Real-Time Systems*, pp. 193-200, Stockholm, Sweden. June 2000.

[19] Liu, J.W.S., *Real-Time Systems*, Prentice Hall, 2000.

[20] 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.

[21] Maheshwari, U., “Charged-based Proportional Scheduling,” Technical Memorandum, MIT/LCS/TM-529, Laboratory for CS, MIT, July 1995.

[22] 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.

[23] 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.

[24] 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.

[25] Spuri, M., Buttazzo, G., “Efficient Aperiodic Service Under the Earliest Deadline Scheduling,” *Proc. of the IEEE Symposium on Real-Time Systems*, December 1994.

[26] Spuri, M., Buttazzo, G., Sensini, F., “Robust Aperiodic Scheduling Under Dynamic Priority Systems,” *Proc. of the IEEE Symposium on Real-Time Systems*, December 1995.

[27] 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.

[28] 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.

[29] Stiladis, D., Varma, A., “Efficient Fair Queuing Algorithms for Packet-Switched Networks,” *IEEE/ACM Transactions on Networking*, vol 6, no 2, April 1998.

[30] 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.

[31] 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.

[32] 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.

[33] Waldspurger, C. A., *Lottery and Stride Scheduling: Flexible Proportional-Share Resource Management*,” Ph.D. Thesis, MIT, Laboratory for CS, September 1995.

[34] 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.