A Theory of Rate-Based Execution*

Kevin Jeffay
Department of Computer Science
University of North Carolina at Chapel Hill
Chapel Hill, NC 27599-3175
jeffay@cs.unc.edu

Steve Goddard
Computer Science & Engineering
University of Nebraska — Lincoln
Lincoln, NE 68588-0115
goddard@cse.unl.edu

Abstract

We present a task model for 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 model, called *rate-based execution* (RBE), is a generalization of Mok's sporadic task model [14]. The RBE model is motivated naturally by distributed multimedia and digital signal processing applications. We derive necessary and sufficient conditions for determining the feasibility of an RBE task set and demonstrate that *earliest deadline first* (EDF) scheduling is an optimal scheduling algorithm for both preemptive and non-preemptive execution environments, as well as hybrid environments wherein RBE tasks access shared resources.

Our analysis of RBE tasks demonstrates a fundamental distinction between deadline based scheduling methods and static priority based methods. We show that for deadline-based scheduling methods, feasibility is solely a function of the distribution of task deadlines in time. This is contrasted with static priority schedulers where feasibility is a function of the actual arrival rates of work for tasks. Thus whereas the feasibility of static priority schedulers is a function of the periodicity of tasks, the feasibility of deadline schedulers is independent of task arrival processes and hence deadline schedulers are more suitable for use in distributed, event-driven, real-time systems.

1. Introduction

Real-time applications frequently interact with external devices in an event-driven manner. The delivery of a message, or the generation of a hardware interrupt is an event that causes the operating system to schedule a task to respond to the event. In real-time environments, one must provide some form of guarantee that the processing corresponding to an event will complete within d time units of the event's occurrence. Hard-real-time systems guarantee that *every* event e_i will be processed within d_i time units of its occurrence. Soft-real-time and firm-real-time systems provide weaker guarantees of timeliness.

Most real-time models of execution are based on the Liu and Layland periodic task model [12] or Mok's sporadic task model [14]. Periodic tasks are real-time programs that service events at precise, periodic intervals. Events serviced by sporadic tasks have a lower bound on their inter-arrival time, but no upper bound on inter-arrival time.

We have found in practice, especially in distributed real-time systems, that the inter-arrival of events is neither periodic nor sporadic. There is, however, usually an *expected* or *average* event arrival rate that can be specified. For example, in an Internet video conferencing system, media samples are typically generated precisely periodically. However, after they are transmitted over the network, samples can arrive at the receiver at nearly arbitrary rates. The transmission rate is precise and the *average* reception rate is precise, but the instantaneous reception rate is potentially unbounded (depending on the amount of buffering in the network).

Our goal here is to understand the complexity of directly modeling the rate-based nature of systems such as distributed multimedia systems. We have created a simple model of real-time tasks that execute at well-defined average rates but have no constraints on their instantaneous rate of invocation. Our model of rate-based execution, called RBE, is a generalization of Mok's sporadic task model in which tasks are expected to execute with an average execution rate of x times every y time units. Our experience designing distributed, event-driven, real-time systems, such as multimedia systems and classes of military signal processing systems, demonstrates that this task model more naturally models the actual implementation and run-time behaviors of these systems [5, 6, 10].

In this work we present necessary and sufficient conditions for determining the feasibility of scheduling an RBE task set on a single processor such that no task misses its deadline. The analysis holds for *earliest deadline first* (EDF) scheduling which is also shown to be an optimal scheduling algorithm for both preemptive and non-preemptive execution environments, as well as hybrid environments wherein tasks access shared memory resources.

The analysis of EDF scheduling demonstrates a fundamental distinction between deadline based scheduling methods and static priority based methods. We show that for deadline-based scheduling methods, feasibility is solely a function of the distribution of task deadlines in time and is independent

* Work supported by grants from the National Science Foundation (grants CCR-9510156, CDA-9624662, & CCR-9732916) and the IBM Corporation.

Published in: *Proceedings of the 20th IEEE Real-Time Systems Symposium*,
Phoenix, AZ, December 1999, pages 304-314.

of the rate at which tasks are invoked. In contrast, the opposite is true of static priority schedulers. For *any* static priority scheduler, feasibility is a function of the rate at which tasks are invoked and is independent of the deadlines of the tasks. Said more simply, the feasibility of static priority schedulers is solely a function of the periodicity of tasks, while the feasibility of deadline schedulers is solely a function of the periodicity of the occurrence of a task's deadlines. Given that it is often the operating system that assigns deadlines to tasks, this means that the feasibility of a static priority scheduler is a function of the behavior of the external environment (*i.e.* arrival processes) while the feasibility of a deadline driven scheduler is a function of the implementation of the operating system. We believe this is a significant observation as one typically has more control over the implementation of the operating system than they do over the processes external to the system that generate work for the system. Therefore, we conclude that deadline based scheduling methods have a significant and fundamental advantage over priority based methods when there is uncertainty in the rates at which work is generated for a real-time system, such as is the case in virtually all distributed real-time systems.

The rest of this paper is organized as follows. Section 2 provides the motivation for considering the RBE task model and describes related work. Section 3 formally presents the RBE task model. Section 4 presents necessary and sufficient conditions for preemptive scheduling, non-preemptive scheduling, and preemptive scheduling with shared resources and demonstrates the optimality of EDF scheduling in each case. Section 5 discusses these results and demonstrates the infeasibility of static priority scheduling. In addition, Section 5 compares RBE to other models of rate-based execution and scheduling such as *proportional share resource allocation* [2, 13, 15, 19, 21, 22] and server algorithms such as the *total bandwidth server* [17, 18]. We conclude our presentation of the RBE model with a summary in Section 6.

2. Motivation and Related Work

The starting point for this work is the model of sporadic tasks developed by Mok [14], and later extended by Baruah *et al*. [4], and Jeffay *et al*. [9]. A sporadic task is a simple variant of a periodic task. 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. The fact that sporadic tasks may execute at a variable (but bounded) rate makes them well-suited for supporting event-driven applications.

Baruah *et al*. developed the seminal complexity analysis for determining the feasibility of a sporadic task set [4]. Today, the theory of sporadic tasks is general enough to accommodate a model of computation wherein tasks may communicate via shared memory (*i.e*., tasks may have critical sections) [8], and tasks may be preempted by interrupt handlers (*i.e*., realistic device interactions can be modeled) [9]. A set of relations on model parameters that are necessary and sufficient for tasks to execute in real-time are known, and optimal algorithms for scheduling tasks, based on EDF scheduling, have been developed.

One practical complexity that arises in applying the existing models of sporadic tasks to actual systems is the fact that the real world does not always meet the assumptions of the model. Consider a task's minimum inter-invocation time parameter. The formal model assumes that consecutive invocations of a sporadic task are separated by at least p time units for some constant p. Tasks that are invoked in response to events generated by devices such as network interfaces may not satisfy this property. For example, for the simple video conferencing application described in the introduction, when video frames are periodically transmitted across an internetwork, they may be delayed for arbitrary intervals at intermediate nodes and arrive at a conference receiver at a highly irregular rate. One solution to this problem is to simply buffer video frames at the receiver and release them at regular intervals to the application (although this begs the question of how one implements and models the real-time tasks that perform this buffering process). This approach is undesirable because it is difficult and tedious to implement correctly and because buffering inherently increases the acquisition-to-display latency of each video frame (and latency is the primary measure of conference quality).

Our approach is to alter the formal model to account for the fact that there may be significant "jitter" (deviation) in the inter-invocation time of real-time tasks. We develop a characterization of a task that is similar to that of a sporadic task, however, we make no assumptions about the spacing in time of invocations of an RBE task. Instead, we allow one to specify an *average execution rate* that is desired for a task. In the RBE model, if a task is invoked at time t, the task is scheduled with a deadline for processing that is sufficient to ensure that the task actually makes progress at its specified rate.

Digital signal processing is another domain in which the RBE task model naturally describes the execution of applications. Processing graphs are a standard design aid in the development of complex digital signal processing systems. We have found that, even on a single-CPU system with periodic input devices, processing graph nodes naturally execute in highly aperiodic "spurts" [5, 6]. Moreover, source data often arrives in bursts in distributed implementations of processing graphs. As discussed in Section 5, this fact precludes the efficient modeling of node execution with either periodic or sporadic task models.

With respect to previous attempts to explicitly specify a task's progress in terms of an execution rate, the RBE task

model is most similar to the *linear bounded arrival process* (LBAP) model as defined and used in the DASH system [1]. In the LBAP model, processes specify a desired execution rate as the number of messages to be processed per second, and the size of a buffer pool used to store bursts of messages that arrive for the process. Our task model generalizes the LBAP model to include a more generic specification of rate and adds an independent response time (relative deadline) parameter to enable more precise real-time control of task executions. Moreover, we analyze the model in more complex environments such as those wherein tasks communicate via shared memory and thus have preemption constraints. A more detailed comparison to other models of rate-based execution is deferred until Section 5.

3. RBE Task Model

Here we formally define the concept of rate-based execution and present the RBE task model.

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 task will be released.

For a given real-time task, two commonly studied paradigms of event occurrences are *periodic*, in which events are generated every p time units for some constant p, and *sporadic*, in which events are generated no sooner than every p time units for some constant p. We consider two fundamental extensions to these models. First, we make no assumptions about the relationships between the points in time at which events occur for a task. We assume 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, we allow tasks to specify a desired rate of progress in terms of the number of events to be processed in an interval of specified length.

Formally, we consider a real-time system to be composed of a set of RBE tasks. An RBE task is uniquely characterized by a four-tuple (x, y, d, c) of integer constants where:

- y is an interval in time,
- x is the maximum number of executions expected to be requested in any interval of length y,
- d is a response time parameter that specifies the maximum time that is desired to elapse between the release of a task instance and the completion of its execution (*i.e.*, d is the relative deadline of the task), and
- c is the maximum amount of processor time required for any job of task T to execute to completion on a dedicated processor.

The pair (x, y) is referred to as the *rate specification* of an RBE task. A task with rate specification (x, y) expects to receive and process, on average, x events in every interval of length y. More precisely, jobs of a task are constrained to execute as follows. Let t_{ij} be the release of J_{ij}, the j^{th} job of the i^{th} task. We assume throughout that the order of jobs of a task corresponds to the order of event occurrences for the task (*i.e.*, for all i and j, $t_{ij} \le t_{i,j+1}$). Once released, job J_{ij} must complete execution before a deadline $D_i(j)$ given by the following recurrence relation:

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

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 (the averaging interval) of the task. This deadline assignment function confers two important properties on RBE tasks. First, up to x consecutive jobs of a task may contend for the processor with the same deadline and second, 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 the latter restriction, if a set of jobs of a task were released simultaneously it would be possible to saturate the processor. However, 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.

For example, Figure 1 shows the job release times and deadlines for a task T_1 = (x=1, y=2, d=6, c). The downward arrows in the figure indicate release times for jobs of T_1. For each job, the interval represented by the open box indicates the interval of time in which the job must execute to completion. (The actual times at which jobs execute are not shown.) Figure 1 shows that if jobs of T_1 are released periodically, once every 2 time units in this case, then T_1 will execute as a periodic task with a desired deadline that is different from its period. In particular, if jobs are released periodically then the rate specification of T_1 does not come into play in the computation of deadlines.

Figures 2 and 3 show the effect of job releases that occur at the same average rate as before, but where jobs are not released periodically. In these figures, three jobs are released simultaneously at time 0, two jobs are released simultaneously at time 3, one job is released at time 6, *etc*. Figure 2 shows the job release times and deadlines for task T_1 = (x=1, y=2, d=6, c). For comparison, Figure 3 shows the effect of the same pattern of job releases on a task T_2 = (x=3, y=6,

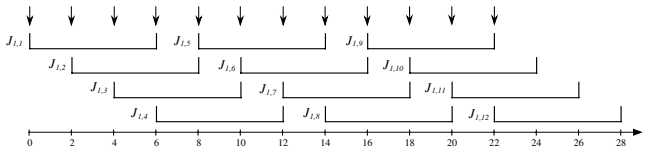

Figure 1: Release times and deadlines for jobs of $T_1 = (x=1, y=2, d=6, c)$.

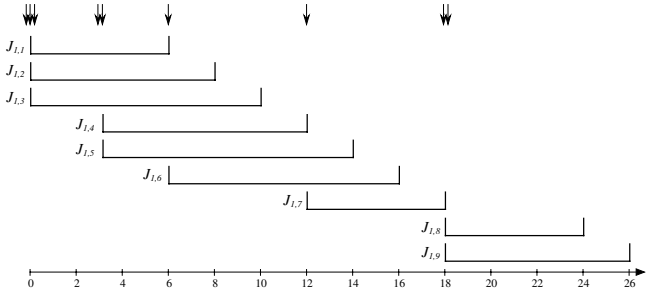

Figure 2: Bursty release times and deadlines for jobs of $T_1 = (x=1, y=2, d=6, c)$.

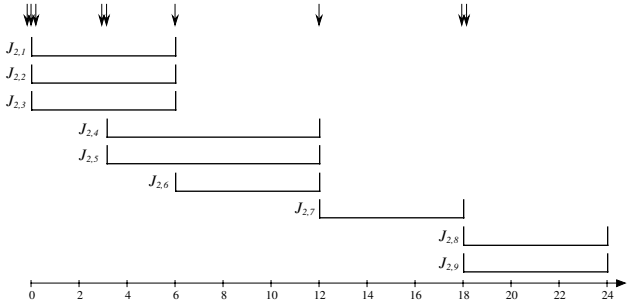

Figure 3: Bursty release times and deadlines for jobs of $T_2 = (x=3, y=6, d=6, c)$.

d=6, c) with the same desired deadline but a different rate specification. Since job releases are not periodic, the actual deadlines of jobs are a function of the rate specification of the task. Note that tasks T_1 and T_2 will consume the same fraction of the processor and both will complete, on average, one job every two time units.

The effect of the different rate specification is two-fold. First, when bursts of events occur, up to three jobs of task T_2 may execute with the same deadline. Thus, for example, task T_2 might be used to implement the media play-out process in a distributed multimedia system wherein (1) media samples are generated at the precise rate of one sample every six time units at a sender, and (2) each sample is too large to fit into a single network packet and thus is fragmented at the sender into three network packets, which are transmitted one immediately following the other to the receiver. At the receiver, media samples arrive, on average, one sample every two time-units. However, since the sender fragments media samples and transmits the fragments one after the other, it is likely that bursts of three simultaneous (or nearly simultaneous) packet arrivals at the receiver will be common. Moreover, at the receiver, while there is a deadline to complete the processing of each complete media sample, there is no obvious deadline for processing individual fragments of the media sample (other than the deadline for the processing of the complete media sample).

The fundamental problem here is that the arrival rate of inputs at the receiver (3 network packets received every 6 time units), is not the same as the output rate at the receiver (1 media sample displayed every 6 time units). By giving a rate specification of (x=3, y=6), the receiver can effectively process groups of up to three network packets with the same deadline — the deadline for completion of the processing of a media sample. Thus by specifying an execution rate, we avoid the artificial problem of having to assign deadlines to intermediate processing steps.

Note that this example is overly simplistic as in practice packet arrivals are discrete events, and hence fundamentally cannot occur "at the same time." Thus in practice, packets arriving as described above will have deadlines that are separated by at least the minimum inter-arrival time of a pair of packets on the given network transmission medium (*e.g.*, 5 microseconds on a 100BaseT Ethernet). However, the fact that the deadlines for packets arriving in a burst would have slightly offset deadlines has the positive side-effect of ensuring that the operating system will process the packets in arrival order (assuming a deadline-driven scheduler).

A task with a rate specification such as T_1 in Figure 2, might be used to implement the play-out process in a different multimedia system wherein media samples (such as audio samples) are small enough to fit into a single network packet and thus the packet arrival rate is the same as the sample play-out rate. Here all network packets should have the same relative deadline for completion of processing (*e.g.*, the expected inter-arrival time of packets). The pattern of deadlines in Figure 2 ensures that the play-out application is guaranteed (assuming the workload is feasible) that in the worst case a media sample will be ready for play-out every y time units starting at time 6.

The second effect of having different rate specifications for tasks T_1 and T_2 is that if jobs are not released periodically, jobs of T_2 will have a lower guaranteed response time than jobs of T_1.

Note that there are times at which it is possible for both tasks to have more than x_i jobs active simultaneously (*e.g.*, in the interval [0,16] for task T_1 and in the interval [3,16] for T_2). This is because the rate specification for a task only specifies the rate at which jobs are *expected* to be released. The actual release rate is completely determined by the environment in which the tasks execute. (In fact, over the entire interval shown in Figures 2 and 3, jobs are released at a *slower* rate than expected.) Also note that the times when individual jobs complete (and hence whether or not there ever are actually

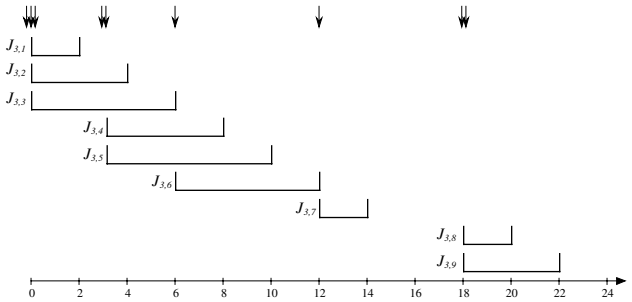

Figure 4: Bursty release times and deadlines for jobs of $T_3 = (x=1, y=2, d=2, c)$.

multiple jobs of a task eligible for execution simultaneously) will depend on the scheduling policy employed. Figures 1-3 should be interpreted as describing a realm of possible execution patterns of tasks.

For a final comparison, Figure 4 shows the effect of the job release times illustrated in Figures 2 and 3 on the task $T_3 = (x=1, y=2, d=2, c)$. Task T_3 is identical to T_1 except with a smaller desired deadline. Figures 2 and 4 can be used to illustrate one benefit of decoupling a task's deadline from its arrival rate and, in particular, the benefit of having a deadline that is greater than the expected inter-job release time. Consider the case where task T_1 is used to implement the media play-out process in a distributed multimedia system wherein media samples are generated at the precise rate of one sample every two time units at the sender. Assume each media sample fits into a network packet and media samples are buffered for up to six time units at the receiver prior to play-out to smooth delay-jitter in the network.

Since samples are expected to be buffered at the receiver, there is little utility to the system in processing samples with a deadline that is less than the expected buffer residence time. That is, if a job of T_3 completes the processing of a media sample within two units of the sample's arrival (which is guaranteed to happen if the arrival of media samples is not bursty), then the media sample will reside in a buffer for at least four time units after this processing completes. In contrast, since T_1 has a larger desired deadline, one would expect that samples processed by T_1 would spend more time waiting to be processed and less time being buffered prior to play-out. Thus the distinction between jobs of T_1 and T_3 is that the media samples processed by jobs of the former task will likely spend more time waiting to be processed (*i.e.*, "buffered in the run-queue") and less time in play-out buffers than when processed by jobs of T_3. The time between media arrival and play-out will be the same in both cases, however. Thus the desired deadline for task T_1 is more appealing in practice as its use will improve the response time for the processing of aperiodic and non-real-time events.

4. Feasibility of RBE Tasks

Our goal is to determine relations on RBE task parameters that are necessary and sufficient for a set of tasks to be *feasible*. A set of RBE tasks is feasible if and only if for all job release times t_{ij}, and for all J_{ij}, it is possible to execute J_{ij} such that:

1. J_{ij} commences execution at or after time t_{ij}, and
2. J_{ij} completes execution at or before time $D_i(j)$.

Our analysis proceeds by analyzing the demand for the processor created by a set of RBE tasks in an interval of length L. In general the demand for the processor created by any set of real-time tasks is a function of the scheduling discipline in use. Here we limit our consideration to *earliest deadline first* (EDF) scheduling. We justify this restriction by showing that EDF is an optimal scheduling discipline for RBE tasks. Optimality here means that an EDF scheduler can guarantee a correct execution to any feasible RBE task set. In Section 5 we discuss alternate scheduling disciplines.

Under EDF scheduling, the demand for the processor in an interval is a function of the number of jobs of tasks that have deadlines in the interval. The deadline assignment function $D_i(j)$ decouples the processor demand from the arrival rate of events and bounds the number of jobs that can have a deadline in any given interval. This in turn bounds the processor demand in any interval.

More precisely, the processor demand in an interval $[a, b]$ is the amount of processor time required to be available in $[a, b]$ to ensure that all tasks released prior to time b with deadlines in $[a, b]$ complete in $[a, b]$. The maximum processor demand in an interval $[a, b]$ occurs when

1. a marks the end of an interval in which the processor was idle (or 0 if the processor is never idle),
2. the processor is never idle in the interval $[a, b]$, and
3. as many deadlines as possible occur in $[a, b]$.

To ensure that no job misses a deadline, we must bound the maximum cumulative processor demand of all tasks in all intervals, and verify that the processor has sufficient capacity to satisfy this demand. To begin, we bound the maximum processor demand for an RBE task in the interval $[0, L]$.

Lemma 4.1: For an RBE task $T = (x, y, d, c)$,

$$\forall L > 0,\ f\left(\frac{L - d + y}{y}\right) \cdot xc \tag{2}$$

is a least upper bound on the processor demand in the interval $[0, L]$, where

$$f(a) = \begin{cases} \lfloor a \rfloor & \text{if } a \geq 0 \\ 0 & \text{if } a < 0 \end{cases}$$

Proof: To derive a least upper bound on the amount of processor time required to be available in the interval [0, L], it suffices to consider a set of release times of T that results in the maximum demand for the processor in [0, L]. If t_j is the time of the j^{th} release of task T, then clearly the set of release times $t_j = 0$, $\forall j > 0$, is one such set. Under these release times, x jobs of T have deadlines in [0, d]. After d time units have elapsed, x jobs of T have deadlines every y time units thereafter. Thus the number of jobs with deadlines in the interval [d, L] is $\left\lfloor \frac{L-d}{y} \right\rfloor \cdot x$. Therefore, for all $L > d$, the number of jobs of T with deadlines in the interval [0, L] is

$$\begin{aligned} x + \left\lfloor \frac{L-d}{y} \right\rfloor \cdot x &= \left(1 + \left\lfloor \frac{L-d}{y} \right\rfloor\right) \cdot x \\ &= \left\lfloor \frac{L-d}{y} + 1 \right\rfloor \cdot x \\ &= \left\lfloor \frac{L-d+y}{y} \right\rfloor \cdot x \end{aligned} \quad (3)$$

For all $L < d$, no jobs of T have deadlines in [0, L], hence the right-hand side of (3) gives the maximum number of jobs of T with deadlines in the interval [0, L], for all $L > 0$. Finally, as each instance of T requires c units of processor time to execute to completion, (2) 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 misses a deadline in [0, L]. ■

Note that there are an infinite number of sets of job release times that maximize the processor demand of an RBE task in the interval [0, L]. For example, it is straightforward to show that the less pathological set of job release times $t_j = \left\lfloor \frac{j-1}{x} \right\rfloor \cdot y$, $\forall j > 0$, also maximizes the processor demand of task T in the interval [0, L].

4.1 Feasibility under preemptive scheduling

A task set is feasible if and only if there exists a schedule such that no task instance misses its deadline. Thus, if L_D represents the total processor demand in an interval of length L, a task set is feasible if and only if $L \geq L_D$ for all $L > 0$. The following gives a necessary and sufficient condition for scheduling a set of RBE tasks when preemption is allowed at arbitrary points.

Theorem 4.2: Let $\tau = \{(x_1, y_1, d_1, c_1), \ldots, (x_n, y_n, d_n, c_n)\}$ be a set of RBE tasks. τ will be feasible if and only if:

$$\forall L > 0, \quad L \geq \sum_{i=1}^{n} f\left(\frac{L - d_i + y_i}{y_i}\right) \cdot x_i c_i \quad (4)$$

where $f()$ is as defined in Lemma 4.1.

Proof: The necessity of (4) is shown by establishing the contrapositive, *i.e.*, a negative result from evaluating (4) implies that τ is not feasible. To show that τ is not feasible it suffices to demonstrate the existence of a set of task release times for which at least one job of a task in τ misses a deadline.

Assume

$$\exists l > 0: \quad l < \sum_{i=1}^{n} f\left(\frac{l - d_i + y_i}{y_i}\right) \cdot x_i c_i .$$

Let t_{ij} be the release time of the j^{th} job of task T_i. Consider the set of release times $t_{ij} = 0$, for all i, $1 \leq i \leq n$, and $j > 0$. By Lemma 4.1, the least upper bound for the processor demand created by task T_i is $f\left(\frac{l - d_i + y_i}{y_i}\right) \cdot x_i c_i$ units of processor time in the interval of [0, l]. Moreover, from the proof of Lemma 4.1, the set of release times $t_{ij} = 0$, $1 \leq i \leq n$ and $j > 0$, creates the maximum processor demand possible in the interval [0, l]. Therefore, for τ to be feasible, it is required that $\sum_{i=1}^{n} f\left(\frac{l - d_i + y_i}{y_i}\right) \cdot x_i c_i$ units of work be available in [0, l]. However, since

$$l < \sum_{i=1}^{n} f\left(\frac{l - d_i + y_i}{y_i}\right) \cdot x_i c_i ,$$

a job of a task in τ must miss a deadline in [0, l]. Thus there exists a set of release times such that a deadline is missed when (4) does not hold. This proves the contrapositive. Thus, if the task set τ is feasible, (4) must hold.

To show the sufficiency of (4), it is shown that the preemptive EDF scheduling algorithm can schedule all jobs of tasks in τ without any missing a deadline if the tasks satisfy (4). This is shown by contradiction.

Assume that τ satisfies (4) and yet there exists a job of a task in τ that misses a deadline at some point in time when τ 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 job with deadline after t_d stops executing prior to t_d (or time 0 if no such job executes prior to t_d).

By the choice of t_0, (*i*) only jobs with deadlines earlier than time t_d execute in the interval [t_0, t_d], (*ii*) all jobs released prior to time t_0 will have completed executing by t_0, and (*iii*) the processor is fully used in [t_0, t_d]. It is straightforward to show that at most

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

job of tasks in τ can have deadlines in the interval $[t_0, t_d]$ [3], and hence

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

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 job misses a deadline in $[t_0, t_d]$ when τ is scheduled under the EDF algorithm.

Let ε be the amount of processor time consumed by tasks in τ in the interval $[t_0, t_d]$ when scheduled by the EDF algorithm. It follows that

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

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 the amount of processor time consumed by τ in $[t_0, t_d]$ when scheduled by the EDF algorithm is greater than the processor time available in $[t_0, t_d]$. Since the processor time available in $[t_0, t_d]$ is $t_d - t_0$, we have

$$\sum_{i=1}^{n} \left\lfloor \frac{t_d - t_0 - d_i + y_i}{y_i} \right\rfloor \cdot x_i c_i \geq \varepsilon > t_d - t_0.$$

However this contradicts our assumption that τ satisfies equation (4). Hence if τ satisfies equation (4) then no release of a task in τ misses a deadline when τ is scheduled by the EDF algorithm. It follows that satisfying (4) is a sufficient condition for feasibility. Thus, (4) is a necessary and sufficient condition for the feasibility of an RBE task set. ■

If the cumulative processor utilization for an RBE task set is strictly less than one (*i.e.*, $\sum_{i=1}^{n} \frac{x_i c_i}{y_i} < 1$) then condition (4) can be evaluated efficiently (in pseudo-polynomial time) using techniques developed by Baruah *et al*. [3]. Moreover, for the special case of $d_i = y_i$, for all T_i in τ, the evaluation of (4) reduces to the polynomial-time feasibility condition [9]

$$\sum_{i=1}^{n} \frac{x_i \cdot c_i}{y_i} \leq 1. \tag{5}$$

Equation (5) computes processor utilization for the task set τ and is a generalization of the EDF feasibility condition for periodic tasks with deadlines equal to their period given by Liu and Layland [12].

Finally, note that the proof of the necessity of (4) in Theorem 4.2 is independent of the scheduling policy in use. This allows us to conclude that EDF is an optimal scheduling algorithm for RBE tasks.

Corollary 4.3: EDF is an optimal preemptive scheduling algorithm for sets of RBE tasks.

Proof: Theorem 4.2 has established that independent of the scheduling policy in use, condition (4) is necessary for feasibility. Moreover, Theorem 4.2 also establishes that (4) is sufficient for ensuring that no job will ever miss a deadline when scheduled by the EDF algorithm. Since (4) is necessary for feasibility and sufficient for a correct execution under the EDF algorithm, the EDF algorithm EDF is an optimal scheduling algorithm for sets of RBE tasks ■

4.2 Feasibility under non-preemptive scheduling

We now present necessary and sufficient conditions for evaluating the feasibility of RBE task sets under non-preemptive, work-conserving scheduling algorithms (*i.e.*, the class of scheduling algorithms that schedule non-preemptively without inserting idle time in the schedule). We leave open the problem of deciding feasibility under non-work-conserving, non-preemptive scheduling.

Theorem 4.4: Let $\tau = \{(x_1, y_1, d_1, c_1), \ldots, (x_n, y_n, d_n, c_n)\}$ be a set of RBE tasks sorted in non-decreasing order by d parameter (*i.e.*, for any pair of tasks T_i and T_j, if $i > j$, then $d_i \geq d_j$). τ will be feasible under non-preemptive scheduling if and only if

$$\forall L > 0, \quad L \geq \sum_{i=1}^{n} f\left(\frac{L - d_i + y_i}{y_i}\right) \cdot x_i c_i \tag{6}$$

and

$$\forall i,\ 1 < i \leq n; \quad \forall L,\ d_1 < L < d_i: \quad L \geq c_i + \sum_{j=1}^{i-1} f\left(\frac{L - 1 - d_j + y_j}{y_j}\right) \cdot x_j c_j \tag{7}$$

where $f()$ is as defined in Lemma 4.1.

The proofs of this theorem and the following corollary are contained in the full version of this paper [11]. However, it should be noted that they are straightforward extensions of the proofs of Theorems 3.2, 3.4 and Corollary 3.4 in [8] for non-preemptive scheduling of sporadic tasks. In addition, the optimality of non-preemptive EDF scheduling demonstrated in [8] also holds for RBE tasks.

Corollary 4.5: With respect to the class of non-preemptive work-conserving schedulers, non-preemptive EDF is an optimal scheduling algorithm for RBE tasks [11].

4.3 Feasibility under preemptive Scheduling with shared resources

We now consider the case when tasks perform operations on shared memory resources. Shared memory resources are serially reusable but must be accessed in a mutually exclusive manner. Our model of computation is based on that presented in [8].

Access to a set of m shared resources $\{R_1, R_2, \ldots, R_m\}$, is modeled by specifying the computation requirement of task T_i as a set of n_i phases $\{(c_{ij}, C_{ij}, r_{ij}) \mid 1 \le j \le n_i\}$ where:

- c_{ij} is the minimum computational cost: the minimum amount of processor time required to execute the j^{th} phase of T_i to completion on a dedicated processor.
- C_{ij} is the maximum computational cost: the maximum amount of processor time required to execute the j^{th} phase of T_i to completion on a dedicated processor.
- r_{ij} is the resource requirement: the resource (if any) that is required during the j^{th} phase of T_i. If $r_{ij} = k$, $1 \le k \le m$, then the j^{th} phase of T_i requires exclusive access to resource R_k. If $r_{ij} = 0$ then the j^{th} phase of T_i requires no shared resources.

The execution of each job of task T_i is partitioned into a sequence of n_i disjoint phases. A phase is a contiguous sequence of statements that together either require exclusive access to a single shared resource, or require no shared resources. In the latter case, the j^{th} phase of T_i imposes no mutual exclusion constraints on the execution of other tasks. If a phase of a task requires a resource then the computational cost of the phase represents only the cost of using the required resource and not the cost (if any) of acquiring or releasing the resource. Note that since different tasks may perform different operations on a resource, it is reasonable to assume that phases of tasks that access the same resource have varying computational costs. A minimum cost of zero indicates that a phase of a task is optional.

We assume that in principal tasks are preemptable at arbitrary points. The requirement of exclusive access to resources places two restrictions on the preemption and execution of tasks. For all task i and k, if $r_{ij} = r_{kl}$ and r_{ij}, $r_{kl} \neq 0$, then (*i*) the j^{th} phase of a job of T_i may neither preempt the l^{th} phase of a job of T_k, nor (*ii*) execute while the l^{th} phase of T_k is preempted.

Consider a set of RBE tasks $\tau = \{T_1, T_2, \ldots, T_n\}$, where $T_i = (x_i, y_i, d_i, \{(c_{ij}, C_{ij}, r_{ij}) \mid 1 \le j \le n_i\})$, that share a set of resources $\{R_1, R_2, \ldots, R_m\}$. Let δ_i represent the deadline parameter of the "shortest" task that uses resource R_i. That is, $\delta_i = \min_{1 \le j \le n} (d_j \mid r_j = i)$. The following theorem establishes necessary and sufficient conditions for feasibility.

Theorem 4.6: Let τ be a set of RBE tasks, sorted in non-decreasing order by d parameter, that share a set of serially reusable resources $\{R_1, R_2, \ldots, R_m\}$. τ will be feasible under work-conserving scheduling if and only if

$$\forall L > 0, \quad L \ge \sum_{i=1}^{n} f\left(\frac{L - d_i + y_i}{y_i}\right) \cdot x_i \cdot E_i \tag{8}$$

and

$$\forall i,\ 1 < i \le n;\ \forall k,\ 1 \le k \le n_i \wedge r_{ik} \neq 0;\ \forall L,\ \delta_{r_{ik}} < L < d_i - S_{ik}:$$

$$L \ge C_{ik} + \sum_{j=1}^{i-1} f\left(\frac{L - 1 - d_j + y_j}{y_j}\right) \cdot x_j \cdot E_j \tag{9}$$

where:

- $f()$ is as defined in Lemma 4.1,
- $\delta_{r_{ik}} = \min_{1 \le j \le n} (d_j \mid \exists l,\ 1 \le l \le n_j,\ r_{jl} = r_{ik})$,
- $E_j = \sum_{l=1}^{n_j} C_{jl}$, and
- $S_{ik} = \begin{cases} 0 & \text{if } k = 1 \\ \sum_{j=1}^{k-1} c_{ij} & \text{if } 1 < k \le n_i \end{cases}$

The feasibility conditions are similar to (and in fact a generalization of) those for non-preemptive scheduling. The parameter E_i represents the maximum cost of a job of task T_i and replaces the c_i term in conditions (6) and (7) of Theorem 4.4. Condition (9) now applies to only a resource requesting phase of a job of task T_i rather than to the job as a whole. Because of this, the range of L in (9) is more restricted than in the single phase case of non-preemptive scheduling. The range is more restricted since the k^{th} phase of a job of task T_i cannot start until all previous phases of the job have terminated, and thus the earliest time phase k can be scheduled is S_{ik} time units after the start the job. For the k^{th} phase of a job, the range of intervals of length L in which one must compute the achievable processor demand will be shorter than in the single phase case by the sum of the minimum costs of phases 1 through $k - 1$. Moreover, no demand of phases of T_i other than k appear in (9). Finally, note that for the special case where each task in τ consists of only a single phase, the scheduling problem reduces to simple non-preemptive scheduling and conditions (8) and (9) reduce to the feasibility conditions of Theorem 4.4.

The proofs of Theorem 4.6 and the following corollary are contained in the full version of this paper [11]. It is again the case that they are straightforward extensions of the proofs in the original paper ([8]) for scheduling sporadic tasks that share a set of shared memory resources.

A generalized EDF scheduling algorithm was introduced in [8] to schedule sporadic tasks that share a set of resources. This algorithm was shown to be optimal for sporadic tasks. It is also optimal for RBE tasks that share a set of shared memory resources.

Corollary 4.7: With respect to the class of work-conserving schedulers, generalized EDF is an optimal scheduling algorithm for RBE task sets that share a set of serially reusable resources [11].

5. Discussion

5.1 On modeling RBE tasks as sporadic tasks

The RBE task model specifies the real-time execution of tasks such that no more than x deadlines expire in any inter-

val of length y. Given the similarity of this model to Mok's sporadic task model it is natural to consider modeling RBE tasks as a collection of sporadic tasks. For example, a common question is whether or not an RBE task T is equivalent to a sporadic task with a minimum period of y/x. The answer is that an RBE task cannot be so modeled because in order for a sporadic task to be feasible it is required that there exist a minimum separation time between releases of consecutive jobs of a task. There is no minimum inter-arrival time that can be defined in the environments that motivated the development of the RBE model. Returning to the multimedia examples from Sections 1 and 2, the media samples processed by an RBE task will arrive at an average rate of one sample every y/x time units but in fact multiple instantaneous arrivals are possible (and indeed often likely).

The possibility of instantaneous arrivals (or more generally, the lack of a guaranteed minimum, non-zero, inter-arrival time) implies that an RBE task cannot be equivalent to any single sporadic task. One might therefore consider modeling an RBE task as a collection of x sporadic tasks, each with a minimum period of y time units. This attempt also fails because again, there is no guaranteed minimum inter-arrival time that can be defined for any of the x sporadic tasks. While a collection of x sporadic tasks can respond to x simultaneous events, they can not can respond to $x+k$ simultaneous events for any $k > 1$. If $x+k$ events occur simultaneously, then the processor demand of the sporadic tasks would exceed the capacity of the processor. The only solution would be to defer the processing of work by extending the deadlines of some tasks and this is exactly what the RBE task model does.

5.2 On static priority scheduling of RBE tasks

Throughout we have considered only EDF task scheduling. This has been for good reason. As the following theorem demonstrates, it is not possible to schedule *any* RBE task set using *any* static priority scheduling algorithm. We show this by proving that there does not exist a feasible static priority assignment for RBE tasks. That is, for any static priority assignment to a set of RBE tasks, one can never guarantee that all deadlines of all RBE task jobs will be met. It will always be possible for a job to miss a deadline.

Theorem 5.1: There exists no feasible static priority assignment scheme for RBE tasks.

Proof: Let $\tau = \{(x_1, y_1, d_1, c_1), \ldots, (x_n, y_n, d_n, c_n)\}$ be a set of RBE tasks sorted in non-decreasing order by priority (*i.e.*, for any pair of tasks T_i and T_j, if $i < j$, then jobs of T_i executes with higher priority than those of T_j). Assume at time 0, $N = \left\lceil d_j / c_i \right\rceil$ jobs of task T_i, $i < n$, are released simultaneously along with a single job release of task T_j, $i < j \leq n$. Since $i < j$, jobs of task T_i have priority over jobs of T_j. Therefore, starting at time 0, at least $Nc_i \geq d_j$ units of processor time will be spent executing jobs with higher priority than task T_j's first job and hence the first job of T_j will miss its deadline. Therefore, there does not exist a feasible static priority assignment scheme for RBE tasks. ■

The essence of Theorem 5.1 is that unless one bounds the actual arrival rates of work, a static priority scheduler will never be feasible. This is because processor demand under a static priority scheduler is a function of the times at which jobs are released. If the release process is not constrained then, as illustrated in the proof of Theorem 5.1, high-priority tasks can fully consume the processor and starve lower priority tasks. (Note that Theorem 5.1 places no constraints on the execution of RBE tasks. Thus the result holds independent of whether preemption is allowed or whether resource sharing exists.)

In contrast, processor demand under an EDF scheduler is a function of the times at which deadlines occur and is independent of the rates at which jobs are actually released. This can be observed by noting that in each execution environment considered in Section 4, RBE tasks had the same feasibility conditions as sporadic tasks. Therefore, the feasibility of the sporadic tasks did not depend on the fact that there existed a minimum separation time between successive job releases of a task. The RBE analysis shows that the feasibility of the sporadic tasks depended only on the fact that there existed a minimum separation time between *deadlines* of successive jobs of a task.

This demonstrates a fundamental distinction between deadline-based scheduling methods and static priority based methods. Static priority scheduling methods *require* periodic (or periodic in the worst case) job release times. Deadline-based scheduling methods require periodic (or periodic in the worst case) job deadlines. Given that it is often the operating system or the application that assigns deadlines to tasks, this means that the feasibility of a static priority scheduler is a function of the behavior of processes in the external environment, while the feasibility of a deadline driven scheduler is a function of the implementation of the computer system.

Moreover, as one typically has more control over the implementation of the system software than they do over the operation of the environment external, from a theoretical standpoint, we conclude that deadline-based scheduling methods have a significant and fundamental advantage over priority based methods when there is uncertainty in the rates at which work is generated for a real-time system.

Although static priority scheduling methods cannot be used to implement event-driven systems when the arrival rates of events are unbounded, one can, of course, employ polling methods and thereby smooth any arrival process to conform to any rate specification. This can be a highly effective tech-

nique and indeed, as we discuss next, has been the subject of much research. From a practical standpoint, the issue therefore comes down to whether or not it is considered a more efficient or parsimonious solution to poll for event arrivals and use static priority scheduling or to defer deadlines and use a deadline-based scheduler. There clearly can be no one correct answer to this question.

5.3 Comparison with other models of rate-based execution

Beyond the LBAP model discussed in Section 2, there are additional models of rate-based execution that deserve a closer examination and comparison with our RBE model. Here we compare the RBE task model to *proportional share* (PS) *resource allocation* [2, 13, 15, 19, 21, 22] and the *total bandwidth server* (TBS) [17, 18].

Proportional share resource allocation is used to ensure fairness in resource sharing. It can also be used to schedule real-time tasks [19]. In PS resource allocation, a *weight* is associated with each task that specifies the relative *share* of a CPU (or any other resource) that the task should receive with respect to other tasks. A share represents a fraction of the resource's capacity that is allocated to a task. The actual fraction of the resource allocated to the task is dependent on the number of tasks competing for the resource and their relative weights. If w is the weight associated with task T and W is the sum of all weights associated with tasks in the task set τ, then the fraction of the CPU allocated to task T is $f = \frac{w}{W}$. Thus, as competition for the CPU increases, the fraction of the CPU allocated to any one task decreases. This is in contrast to the RBE task model in which each task is guaranteed a fixed share of the CPU equal to $\frac{x \cdot c}{y}$, no matter how much competition there is for the CPU (assuming the task set is feasible). One can, of course, fix the share of the CPU allocated to a task in PS resource allocation by varying the task's weight relative to the other task weights as tasks are created and destroyed [20]. However, note that to schedule an RBE task T with $d < y$ using PS resource allocation, a share of $\frac{x \cdot c}{d}$ must be allocated to the task, which reserves more resource capacity than is actually needed by the task.

Thus, while RBE and PS resource allocation both support task execution rates, the systems differ markedly in the flexibility allowed in task scheduling. PS resource allocation allows variable execution rates while RBE simply defines a maximum execution rate. The relative deadline of a task executed under PS resource allocation is dependent on the resource share allocated to the task, which is dependent on the task's relative weight. The relative deadline of an RBE task is independent of the task's execution rate; it may be larger or smaller than its y parameter.

The TBS server algorithm was first proposed by Spuri and Buttazzo in [17], and later extended by Spuri, Buttazzo and Sensini in [18]. The original TBS allocated a portion of the processor's capacity, denoted U_S, to process aperiodic requests. The remaining processor capacity, U_P, is allocated to periodic tasks with hard deadlines. Aperiodic requests are scheduled with periodic tasks using the EDF scheduling algorithm. When the k^{th} aperiodic request arrives at time r_k, it is assigned a deadline $d_k = \max(r_k, d_{k-1}) + C_k/U_S$ where C_k is the worst case execution time of the k^{th} aperiodic request and U_S is the processor capacity allocated to the TBS server. Thus, deadlines are assigned to aperiodic requests based on the rate at which the TBS server can serve them, not at the rate which they are expected to arrive and not on any application-specified requirements. Moreover, the aperiodic deadlines are assigned such that the k^{th} request completes before the $k+1^{st}$ request will begin executing when they are scheduled with the EDF algorithm. That is, aperiodic requests are processed in a FCFS manner (relative to other aperiodic requests) at the rate at which the TBS server is able to process them.

The TBS server only serves aperiodic requests and deadlines are derived rather than specified by applications. In contrast, the RBE task model assumes tasks execute at an average rate with arbitrary deadlines (modulo feasibility). The RBE task model does not directly support aperiodic requests. However, the TBS server can be combined with an RBE task set in the same way it was combined with a periodic task set and scheduled preemptively using a variation of EDF. It is not immediately clear that the semantics of a TBS server can be preserved if it is combined with an RBE task set that is scheduled non-preemptively or that shares resources.

6. Summary & Conclusions

We have presented a generalization of the sporadic task model developed by Mok [14] for 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. We call this new task model *rate-based execution* (RBE). In the RBE model, tasks are expected to execute with an average execution rate of x times every y time units. When deadlines of consecutive RBE jobs of the same task are given by the deadline assignment function $D_i(j)$ defined in Section 3, EDF scheduling has been shown to be an optimal discipline for preemptive scheduling, non-preemptive scheduling, and scheduling in the presence of shared resources. Moreover, one can decide feasibility efficiently in all cases.

We believe the RBE task model more naturally models the actual implementation of event-driven, real-time systems. RBE has been used to model the execution of applications ranging from multimedia computing to digital signal processing [5, 6, 10].

This work highlights an important distinction between deadline-driven scheduling methods and scheduling based on a static priority assignment to tasks. Under deadline-driven scheduling feasibility is not a function of the arrival rates of events; it is solely a function of the rates at which deadlines occur. As applications can typically control deadlines this gives deadline-driven scheduling an advantage in environments where the arrival rates of events cannot be bounded. In such environments static priority scheduling is inherently inferior as for any static priority assignment and any RBE task set, it will always be possible or a job of a task to miss a deadline.

7. References

[1] Anderson, D., Tzou, S., Wahbe, R., Govindan, R., Andrews, M., *Support for Live Digital Audio and Video*, Proc. 10th Intl. Conf. on Distributed Computing Systems, Paris, France, May 1990, pp. 54-61.

[2] Baruah, S., Gehrke, J., Plaxton, G., *Fast Scheduling of Periodic Tasks on Multiple Resources*, Proc. 9th Intl. Parallel Processing Symp., April 1995, pp. 280-288.

[3] 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.

[4] 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.

[5] Goddard, S., Jeffay, K., *Analyzing the Real-Time Properties of a Data flow Execution Paradigm using a Synthetic Aperture Radar Application*, Proc. 3rd IEEE Real-Time Technology & Applications Symp., Montreal, Canada, June 1997, pp. 60-71.

[6] Goddard, S., Jeffay, K., *Managing Memory Requirements in the Synthesis of Real-Time Systems from Processing Graphs*, Proc. of 4th IEEE Real-Time Technology and Applications Symp., Denver, CO, June 1998, pp. 59-70.

[7] Jeffay, K., Stanat, D.F., Martel, C.U., *On Non-Preemptive Scheduling of Periodic and Sporadic Tasks*, Proc. 12th IEEE Real-Time Systems Symp., San Antonio, TX, Dec. 1991, pp. 129-139.

[8] Jeffay, K., *Scheduling Sporadic Tasks with Shared Resources in Hard-Real-Time Systems*, Proc. 13th IEEE Real-Time Systems Symp., Phoenix, AZ, Dec 1992, pp. 89-99.

[9] Jeffay, K., Stone, D.L., *Accounting for Interrupt Handling Costs in Dynamic Priority Task Systems*, Proc. 14th IEEE Real-Time Systems Symp., Durham, NC, Dec. 1993, pp. 212-221.

[10] Jeffay, K., Bennett, D., *A Rate-Based Execution Abstraction For Multimedia Computing*, Proc. of the 5th Intl. Workshop on Network and Operating System Support for Digital Audio & Video, Durham, N.H., April 1995, Lecture Notes in Computer Science, T.D.C. Little & R. Gusella, eds., Vol. 1018, Springer-Verlag, Heidelberg, pp. 65-75.

[11] Jeffay, K., Goddard, S., *The Rate-Based Execution Model*, Technical Report UNL-CSE-99-003, Computer Science & Engineering, University of Nebraska, May 1999.

[12] Liu, C.L., Layland, J.W., *Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment*, Journal of the ACM, Vol. 20, No. 1, (January 1973), pp. 46-61.

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

[14] Mok, A.K.-L., *Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment*, Ph.D. Thesis, MIT, Dept. of EE and CS, MIT/LCS/TR-297, May 1983.

[15] Nieh, J., Lam, M.S., *Integrated Processor Scheduling for Multimedia*, Proc. 5th Intl. Workshop on Network and Operating System Support for Digital Audio & Video, Durham, N.H., April 1995, Lecture Notes in Computer Science, T.D.C. Little & R. Gusella, eds., Vol. 1018, Springer-Verlag, Heidelberg.

[16] *Processing Graph Method Specification*, prepared by NRL for use by the Navy Standard Signal Processing Program Office (PMS-412), Version 1.0, Dec. 1987.

[17] Spuri, M., Buttazzo, G., *Efficient Aperiodic Service Under the Earliest Deadline Scheduling*, Proc. 15th IEEE Real-Time Systems Symp., Dec. 1994, pp. 2-11.

[18] Spuri, M., Buttazzo, G., Sensini, F., *Robust Aperiodic Scheduling Under Dynamic Priority Systems*, Proc. 16th IEEE Real-Time Systems Symp., Dec. 1995, pp. 288-299.

[19] Stoica, I., Abdel-Wahab, H., Jeffay, K., Baruah, S., Gehrke, J., Plaxton, G., *A Proportional Share Resource Allocation Algorithm for Real-Time, Time-Shared Systems*, Proc. 17th IEEE Real-Time Systems Symp., Dec. 1996.

[20] Stoica, I., Abdel-Wahab, H., Jeffay, K., *On the Duality between Resource Reservation and Proportional Share Resource Allocation*, Multimedia Computing & Networking '97, SPIE Proceedings Series, Vol. 3020, Feb. 1997, pp. 207-214.

[21] Waldspurger, C.A., Weihl, W.E., *Lottery Scheduling: Flexible Proportional-Share Resource Management*, Proc. of the First Symp. on Operating System Design and Implementation, Nov. 1994, pp. 1-12.

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