4-1-1999

The Rate-Based Execution Model

Kevin Jeffay
University of North Carolina at Chapel Hill

Steve Goddard
University of Nebraska – Lincoln, goddard@cse.unl.edu

In submission to: *The 20th IEEE Real-Time Systems Symposium*, December 1999.

University of Nebraska-Lincoln, Computer Science and Engineering
Technical Report # TR-UNL-CSE-1999-001; issued 4/1/99

The Rate-Based Execution Model

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 new 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. We call this new task model rate-based execution (RBE), and it is a generalization of the common sporadic task model. The RBE model is motivated naturally by distributed multimedia and digital signal processing applications.

We identify necessary and sufficient conditions for determining the feasibility of an RBE task set, and an optimal scheduling algorithm (based on preemptive earliest-deadline-first (EDF) scheduling) for scheduling the execution of an RBE task set. With respect to the class of work-conserving scheduling algorithms (i.e., the class of scheduling algorithms that schedule without inserting idle time in the schedule), we present necessary and sufficient feasibility conditions and optimal algorithms for non-preemptive scheduling and preemptive scheduling with shared resources.

1 Introduction

Real-time applications frequently interact with external devices in an event-driven manner. The delivery of a message, or a hardware interrupt, signals the event arrival. A task is then dispatched to process the event. In real-time environments, the system provides some form of guarantee that the task will finish processing the event within d time units of the event occurrence, where d_i is called the relative deadline of event e_i. 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 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. Thus, we have created the rate-based execution (RBE) task model. 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 with distributed multimedia and distributed signal processing applications demonstrates that this task model more naturally models the actual implementation of distributed, event-driven, real-time systems [5, 6, 10]. Moreover, it has been argued that the actual execution of applications based on general processing graphs, such as Processing Graph Method (PGM) [16], is neither periodic nor sporadic unless nodes of the processing graph are forced to execute as such, which increases latency [5, 6]. RBE provides a more natural model of the real-time execution of nodes in a PGM graph.

While RBE has been used to model the execution of applications ranging from multimedia computing to digital signal processing [5, 6, 10], this is the first formal presentation of the RBE task model. We present necessary and sufficient conditions for determining the feasibility of scheduling an RBE task set such that no task misses its deadline. We also present an optimal scheduling algorithm (based on preemptive earliest-deadline-first (EDF) scheduling) for scheduling the execution of an RBE task set. By optimal, we mean that if a feasible schedule exists, our scheduling algorithm will find one. We then consider non-preemptive scheduling and preemptive scheduling with shared resources. In both cases, we present necessary and sufficient feasibility conditions and optimal scheduling algorithms with respect to the class of work-conserving scheduling algorithms (i.e., the class of scheduling algorithms that schedule without inserting idle time in the schedule). A corollary of our results is that RBE tasks cannot be scheduled with static priority schedulers.

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 presents the RBE task model. Section 4 presents necessary and sufficient conditions for preemptive scheduling, non-preemptive scheduling, and preemptive scheduling with shared resources. Optimal scheduling algorithms, based on EDF scheduling, for each case are also presented in Section 4. Section 5 discusses the separation of the execution semantics of an RBE task from a specific scheduling algorithm, and compares RBE to *proportional share resource allocation* [2, 13, 15, 19, 21, 22] and the Total Bandwidth server [17, 18]. We conclude our 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.* [7]. In [4], Baruah *et al.* developed the seminal complexity analysis for determining the feasibility of a sporadic task set. 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. At present, 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 an optimal algorithm for scheduling tasks, based on EDF scheduling, has 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 with minimum inter-invocation time (or "period") p are separated by at lest p time units. Tasks that are invoked in response to events generated by devices may not satisfy this property. For example, consider a simple video conferencing application. When video frames are transmitted across a network, they may be delayed for arbitrary intervals at nodes in the network. Therefore, even though video frames are, in theory, generated periodically, their arrival pattern at a conference receiver may be highly irregular. In this case, the transmission rate is precise and the average receive rate is precise, but the instantaneous receive rate is potentially unbounded (depending on buffering in the network). 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 it will increase 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 extend the sporadic task to be a task that executes at *an average rate*. We have the same characterization of a sporadic task as before, however, we make no assumptions about the spacing in time of invocations of a sporadic task. If a task with "period" p is invoked at time t the task is scheduled (placed into the run queue) with a deadline for processing that is sufficient to ensure that the task actually makes progress at its specified rate. This model, called the Rate-Based Execution (RBE) model, is described in greater detail in Sections 3 and 4.

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 "bursts" [5, 6]. Moreover, source data often

arrives in bursts in distributed systems, which precludes the efficient modeling of node execution with either periodic or sporadic task models. We explain this further in Section 3 after the RBE task model has been formally presented.

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.

Comparison of the RBE model to *proportional share resource allocation* [2, 13, 15, 19, 21, 22] and the Total Bandwidth Server [17, 18] are provided in Section 5.

3 RBE Task Model

Here we introduce the concept of rate-based execution and formally 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.

In the real-time systems literature, 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 points in time at which events occur. 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. As shown below, this allows a task to process a "burst" of simultaneous events as a single event.

Formally, RBE is a general task model consisting of a collection of independent processes specified a four-tuple (x, y, d, c) of constants where:

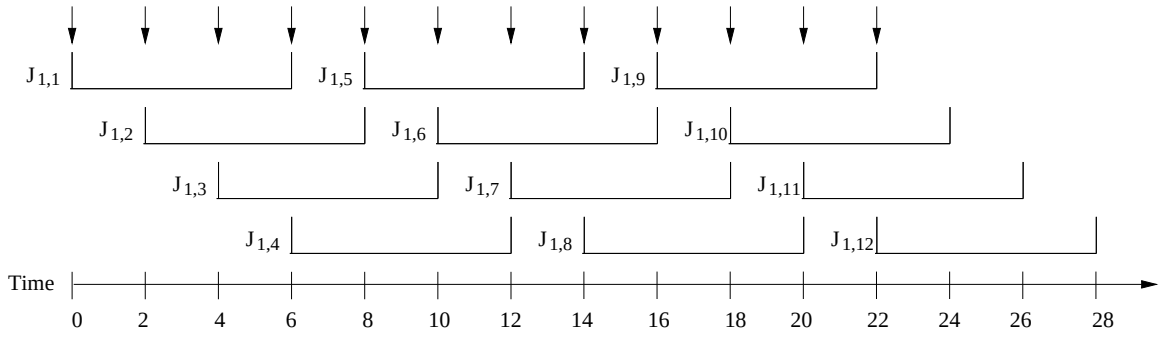

Figure 1: Release times and deadlines for jobs of $T_1 = (x = 1, y = 2, d = 6, c)$.

- The pair (x, y) is referred to as the *rate specification* of an 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).
- Parameter c, called the *cost*, is the maximum amount of processor time required for any job of task T to execute to completion on a dedicated processor.

We assume throughout that time is discrete and clock ticks are indexed by the natural numbers. Task parameters, x, y, d, and c are expressed as integer multiples of the interval between successive clock ticks.

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, $t_{ij} \leq t_{ij+1}$). Once released, job J_{ij} must complete execution before a deadline $D_i(j)$ given by the following recurrence relation

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

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.

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,

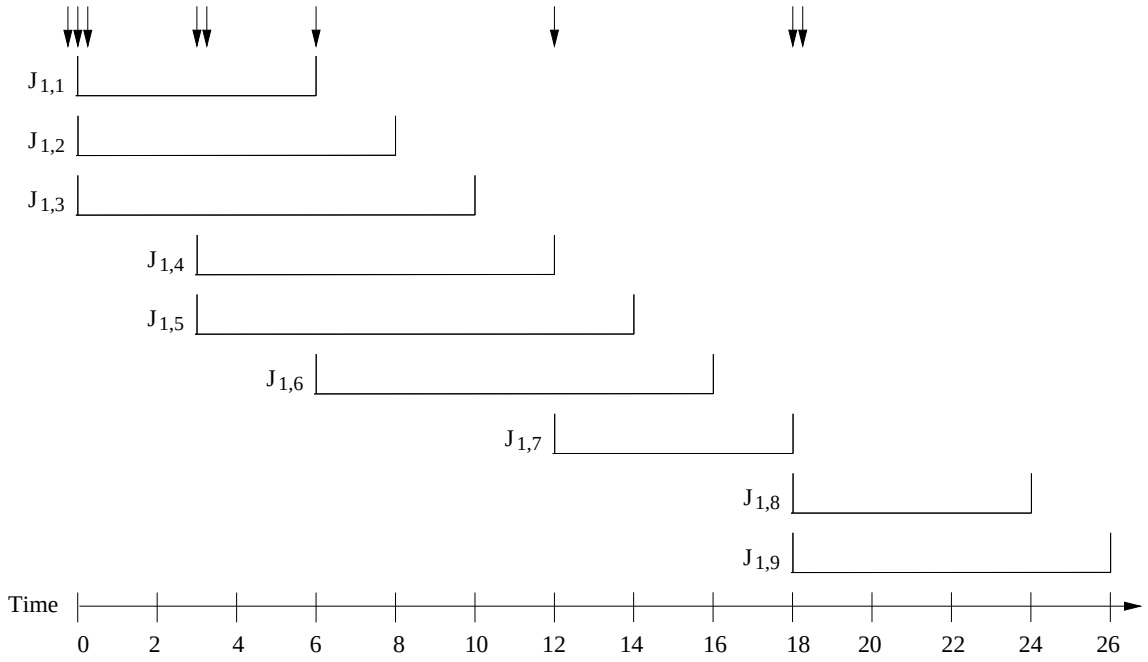

Figure 2: Bursty release times and deadlines for jobs of $T_1 = (x = 1, y = 2, d = 6, c)$.

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 very 2 time units in this case, then T_1 will execute as a periodic task (albeit 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 $T_2 = (x = 3, y = 6, d = 6, c)$ 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

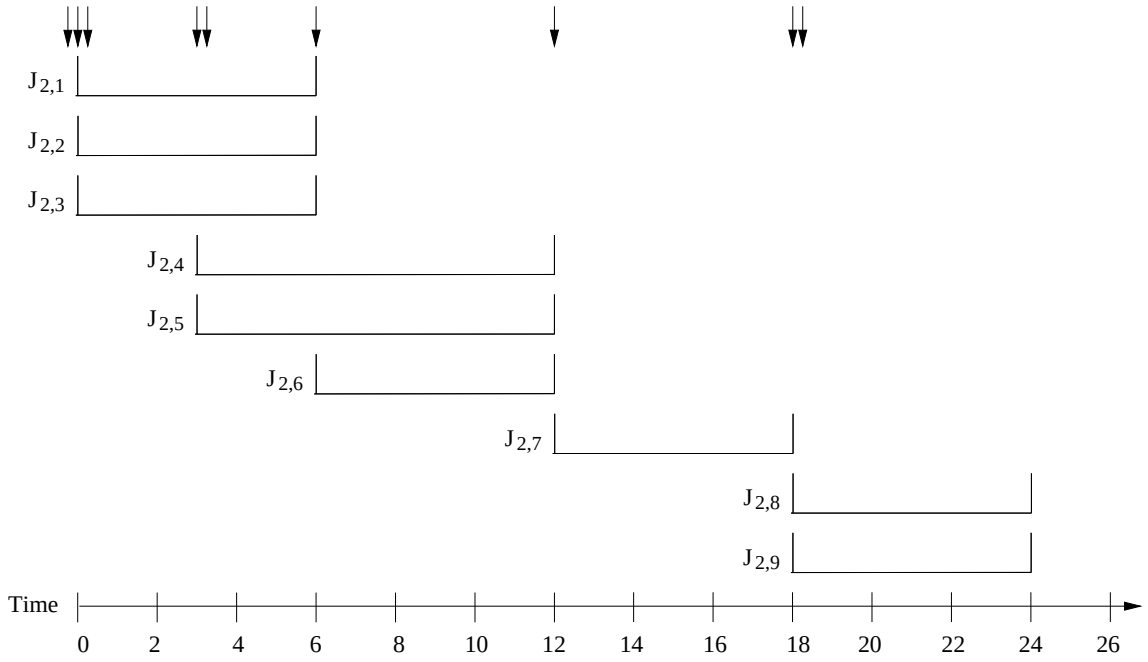

Figure 3: Bursty release times and deadlines for jobs of $T_2 = (x = 3, y = 6, d = 6, c)$.

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 ensures 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 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 task T_1 and T_2 is that if jobs are not released periodically, jobs of T_2 will have a lower 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 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 second 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.

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

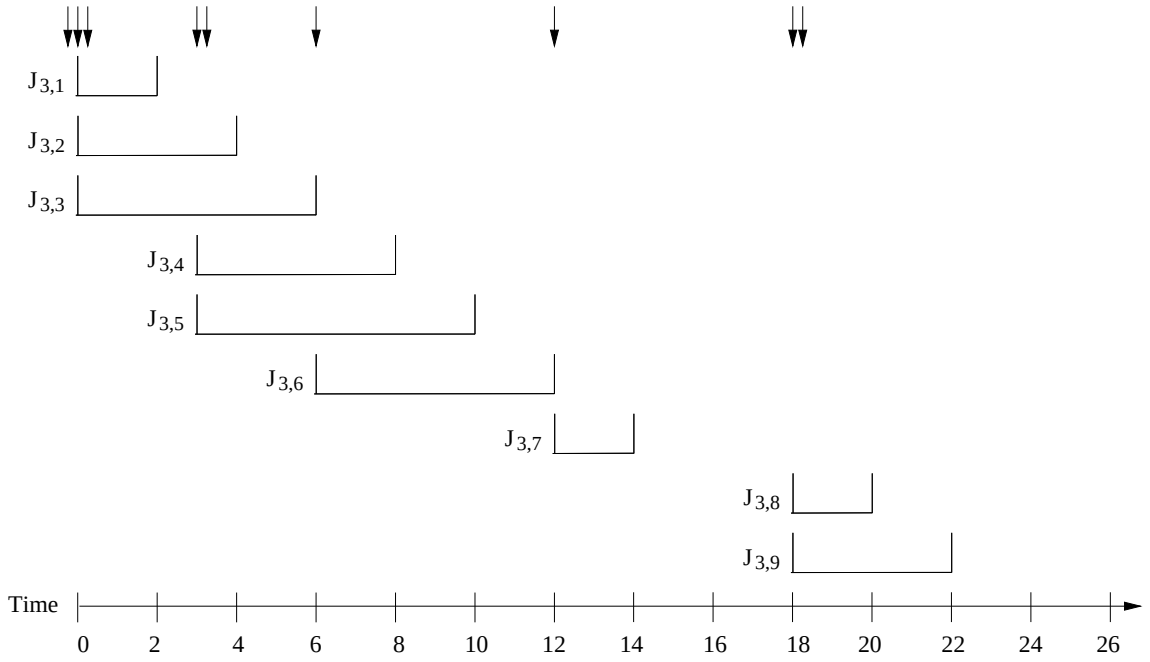

Figure 4: Bursty release times and deadlines for jobs of $T_3 = (x = 1, y = 2, d = 2, c)$.

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 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.

It is natural to consider modeling RBE task T_1 as a sporadic task with a minimum period of $\frac{y_1}{x_1}$ time units. However, this doesn't work because there is no minimum inter-arrival time that can be defined for the media samples processed by task T_1 ($\frac{y_1}{x_1}$ is the average inter-arrival time, not the minimum). One might also consider modeling RBE task T_1 as x_1 sporadic tasks with a minimum period of y_1 time units. This method also fails because there is no minimum inter-arrival time that can be defined for the media samples processed by task T_1. The "simultaneous" arrival of $x_1 + 1$ media samples would result in more processor demand than the sporadic task model anticipates unless the $x_1 + 1^{th}$ media sample is assigned a deadline that is y_1 time units later than the other x_1 media samples. This, of course, is exactly what the RBE task model does.

4 Feasibility of RBE Tasks

Returning to the formal definition of the task model, our goal is to determine relations on 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 $D_i(j)$.

That is, for all t_{ij}, J_{ij} must execute within the closed interval $[t_{ij}, D_i(j)]$.

Deadline-assignment Function $D_i(j)$ prevents release jitter from creating more processor demand in an interval by a task than that which is specified by the rate parameters. Release jitter is the phenomenon that occurs when release times vary such that in one interval fewer releases occur than expected and in another interval more releases occur than expected. 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 the interval $[a, b]$.

If deadlines for jobs of RBE task T_i were assigned by simply adding d_i to the job's release time, then more than x_i releases in an interval of length y_i may create more processor demand than the processor can support and deadlines may be missed. To ensure that no task instance misses a deadline, we must bound the maximum processor demand for all tasks in all intervals and verify that the processor has enough capacity to support the processor demand created by the RBE task set. We begin by bounding the maximum processor demand for a task in the interval $[0, L]$.

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

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

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 task instance of* T *misses a deadline in* $[0, L]$, *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 processor demand 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 instances of T have deadlines in $[0, d]$. After d time units have elapsed, x instances of T have deadlines every y time units. Thus the number of instances with deadlines in the interval $[d, L]$ is $\left\lfloor \frac{L-d}{y} \right\rfloor \cdot x$. Therefore, $\forall L \geq d$, the number of instances of T with deadlines in the interval $[0, L]$ is

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

For all $L < d$, no instances of T have deadlines in $[0, L]$, hence the right-hand side of Equation (3) gives the maximum number of instances 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, Expression (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 instance of T misses a deadline in $[0, L]$. □

Note that there are many sets of task release times that maximize the processor demand of a task in the interval $[0, L]$. For example, given the recurrence relation for deadlines defined by $D_i(j)$, it is straightforward to show that the less pathological set of task 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 $Demand(L)$ represents the total processor demand in an interval of length L, a task set is feasible if and only if $L \geq Demand(L)$ for all $L > 0$. The following gives a necessary condition for scheduling a set of RBE tasks.

Lemma 4.2. *Let $\mathcal{T} = \{(x_1, y_1, d_1, c_1), \ldots (x_n, y_n, d_n, c_n)\}$ be a set of RBE tasks. If $\mathcal{T}$ is feasible, then*

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

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

Proof: The necessity of Equation (4) is shown by establishing the contrapositive, i.e., a negative result from Equation (4) implies that $\mathcal{T}$ is not feasible. To show that $\mathcal{T}$ is not feasible it suffices to

demonstrate the existence of a set of task release times for which at least one release of a task in $\mathcal{T}$ misses a deadline.

Assume a negative result from Equation (4), that is,

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

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

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

an instance of a task in $\mathcal{T}$ must miss a deadline in $[0, l]$. Thus there exists a set of release times such that a deadline is missed when Equation (4) does not hold. This proves the contrapositive. Thus, if the task set $\mathcal{T}$ is feasible, Equation (4) must hold. □

Lemma 4.3. *Let $\mathcal{T} = \{(x_1, y_1, d_1, c_1), \ldots (x_n, y_n, d_n, c_n)\}$ be a set of RBE tasks. Preemptive EDF will succeed in scheduling $\mathcal{T}$ if Equation (4) holds.*

Proof: To show the sufficiency of Equation (4), it is shown that the preemptive EDF scheduling algorithm can schedule all releases of tasks in $\mathcal{T}$ without any job missing a deadline if the tasks satisfy Equation (4). This is shown by contradiction.

Assume that $\mathcal{T}$ satisfies Equation (4) and yet there exists a release of a task in $\mathcal{T}$ that misses a deadline at some point in time when $\mathcal{T}$ is scheduled by the EDF algorithm. Let t_d be the earliest point in time at which a deadline is missed and let t_0 be the later of:

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

By the choice of t_0, *(i)* only releases with deadlines less than time t_d execute in the interval $[t_0, t_d]$, and *(ii)* the processor is fully used in $[t_0, t_d]$. Only releases with deadlines less than time t_d execute in the interval $[t_0, t_d]$ and, by the choice of t_0, any task instances released before t_0 will have completed executing by t_0. Thus, by a result due to Baruah *et al.* (Lemma 3.5 in reference [3]), at most

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

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

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

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 task release misses a deadline in $[t_0, t_d]$. The problem of scheduling the RBE task set in the interval $[t_o, t_d]$ is equivalent to scheduling a periodic task set where each of the x_i instances of task T_i are represented by a separate periodic task since we have assumed worst-case task releases in which all x_i instances of task T_i are released at the same time. It is a well known fact that EDF is an optimal scheduling algorithm for independent periodic task sets [12]. By optimal, we mean that if a valid schedule exists, the EDF scheduling algorithm will create one. Let $\mathcal{E}$ be the amount of processor time consumed by tasks in $\mathcal{T}$ in the interval $[t_0, t_d]$ when scheduled by the EDF algorithm. Since $\sum_{i=1}^{n} f\left(\frac{t_d - t_0 - d_i + y_i}{y_i}\right) \cdot x_i \cdot c_i$ is a least upper bound on the processor time required in the interval $[t_0, t_d]$ and $\mathcal{E}$ is processor time consumed using the EDF algorithm, it must be the case that

$$\sum_{i=1}^{n} f\left(\frac{t_d - t_0 - d_i + y_i}{y_i}\right) \cdot x_i \cdot c_i \geq \mathcal{E}.$$

Thus, since the processor is fully used in the interval $[t_0, t_d]$ and since a deadline is missed at time t_d, it follows that $\mathcal{E}$ is greater than the processor time available in the interval $[t_0, t_d]$, namely $t_d - t_0$. Hence,

$$\sum_{i=1}^{n} f\left(\frac{t_d - t_0 - d_i + y_i}{y_i}\right) \cdot x_i \cdot c_i \geq \mathcal{E} > t_d - t_0.$$

However this contradicts our assumption that $\mathcal{T}$ satisfies Equation (4). Hence if $\mathcal{T}$ satisfies Equation (4), then no release of a task in $\mathcal{T}$ misses a deadline when $\mathcal{T}$ is scheduled by the EDF algorithm. It follows that satisfying Equation (4) is a sufficient condition for feasibility. □

Thus, Equation (4) is a necessary and sufficient condition for the feasibility of an RBE task set.

Theorem 4.4. *Let $\mathcal{T} = \{(x_1, y_1, d_1, c_1), \ldots (x_n, y_n, d_n, c_n)\}$ be a set of RBE tasks. $\mathcal{T}$ will be feasible if and only if Equation (4) holds.*

Proof: The proof follows from Lemmas 4.2 and 4.3, which establish Equation (4) to be both necessary (Lemma 4.2) and sufficient (Lemma 4.3). □

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

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

since

$$\begin{aligned}
\sum_{i=1}^{n} \frac{x_i \cdot c_i}{y_i} \leq 1 \implies \forall L > 0, \quad L &\geq \sum_{i=1}^{n} L \cdot \frac{x_i \cdot c_i}{y_i} \\
&= \sum_{i=1}^{n} \frac{L}{y_i} \cdot x_i \cdot c_i \\
&= \sum_{i=1}^{n} \frac{L - y_i + y_i}{y_i} \cdot x_i \cdot c_i \\
&= \sum_{i=1}^{n} \frac{L - d_i + y_i}{y_i} \cdot x_i \cdot c_i \quad \text{since } d_i = y_i \\
&\geq \sum_{i=1}^{n} f\left(\frac{L - d_i + y_i}{y_i}\right) \cdot x_i \cdot c_i.
\end{aligned} \tag{6}$$

Equation (5) computes processor utilization for the task set $\mathcal{T}$ and is a generalization of the EDF feasibility condition $\sum_{i=1}^{n} \frac{c_i}{y_i} \leq 1$ for independent tasks with deadlines equal to their period given by Liu & Layland [12].

Let the *rate-based-execution earliest-deadline-first (RBE-EDF)* scheduling algorithm be the EDF scheduling algorithm with deadlines assigned using Equation (1). A scheduling algorithm is said to be *optimal* if it can schedule a set of tasks such that no task instance misses its deadline.

Corollary 4.5. *Preemptive RBE-EDF is an optimal scheduling algorithm for an RBE task set.*

Proof: The proof follows from Lemmas 4.2 and 4.3. □

Corollary 4.6. *Static Priority scheduling cannot be used to schedule RBE task sets preemptively.*

Proof: Consider the simple RBE task set $\mathcal{T} = \{(x_1 = 1, y_1 = 4, d_1 = 4, c_1 = 1), (x_2 = 1, y_2 = 4, d_2 = 4, c_2 = 1)\}$. The task set is feasible since $\sum_{i=1}^{2} \frac{x_i \cdot c_i}{y_i} \leq 1$ and $d_i = y_i$ for $i = 1, 2$. Assume that, at time 0, eight jobs of each task are released simultaneously. If task T_1 has priority over task T_2, then the first two jobs of task T_2 will miss their deadlines. Likewise, if task T_2 has priority over task T_1, then the first two jobs of task T_1 will miss their deadlines. Thus, no static priority scheduler can be guaranteed to schedule a feasible RBE task set such that no job misses its deadline. □

4.2 Feasibility under Non-Preemptive Scheduling

We now present a necessary and sufficient condition 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 deriving necessary and sufficient conditions for determining the feasibility of non-work-conserving, non-preemptive scheduling.

Theorem 4.7. *Let $\mathcal{T} = \{(x_1, y_1, d_1, c_1), \ldots (x_n, y_n, d_n, c_n)\}$ 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 $d_i \geq d_j$). $\mathcal{T}$ will be feasible under non-preemptive scheduling if and only if*

$$\forall L > 0, \quad L \geq \sum_{i=1}^{n} f\left(\frac{L - d_i + y_i}{y_i}\right) \cdot x_i \cdot c_i \tag{7}$$

and

$$\forall i, 1 < i \leq n; \forall L, d_1 < L < d_i : \quad L \geq c_i + \sum_{j=1}^{i-1} f\left(\frac{L - 1 - d_j + y_j}{y_j}\right) \cdot x_j \cdot c_j \tag{8}$$

where $f(a)$ 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.

Let the *non-preemptive rate-based-execution earliest-deadline-first (NP-RBE-EDF)* scheduling algorithm be the non-preemptive EDF scheduling algorithm with deadlines assigned using Equation (1).

Corollary 4.8. *With respect to the class of non-preemptive work-conserving schedulers, NP-RBE-EDF is an optimal scheduling algorithm for an RBE task set.*

Corollary 4.9. *Static Priority scheduling cannot be used to schedule RBE task sets non-preemptively.*

Proof: The task set created for the proof of Corollary 4.6 is also feasible under non-preemptive scheduling. However, the same release pattern used in the proof of Corollary 4.6 results in deadlines being missed under any static priority assignment. □

4.3 Feasibility under Preemptive Scheduling with Shared Resources

We now consider the case when tasks perform operations on shared resources. Resources are serially reusable and must be accessed in a mutually exclusive manner. To model the access of a set of m shared resources $\{R_1, R_2, \ldots R_m\}$, we specify the computation requirement c_i of T_i as a set of n_i phases $\{(c_{ij}, C_{ij}, r_{ij}) | 1 \leq j \leq n_i\}$ 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.

Thus, 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 require exclusive access to a resource. A job may have multiple phases that require the same resource. The resource required by T_i during the j^{th} phase of its computation is represented by an integer r_{ij}, $0 \leq r_{ij} \leq m$. If $r_{ij} = k$, $k \neq 0$, then the j^{th} phase of T_i's computation requires no resources. In this case the j^{th} phase of T_i imposes no mutual exclusion constraints on the execution of other tasks. 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. 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. A minimum cost of zero indicates that a phase of a task is optional. A fundamental restriction is that each phase of each task will require access to at most one resource at a time.

We assume that in principle tasks are preemptable at arbitrary points. However, 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 $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$, where

$$T_i = (x_i, y_i, d_i, \{(c_{ij}, C_{ij}, r_{ij}) | 1 \leq j \leq n_i\}),$$

that share a set of serially reusable, single unit 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, $\delta_i = \min_{1 \leq j \leq n} (d_j | r_j = i)$. The following theorem establishes necessary and sufficient conditions for feasibility.

Theorem 4.10. *Let $\mathcal{T} = \{T_i = (x_i, y_i, d_i, \{(c_{ij}, C_{ij}, r_{ij}) | 1 \leq j \leq n_i\}) | 1 \leq i \leq n\}$ be a set of RBE tasks sorted in non-decreasing order by d parameter that share a set of serially reusable, single unit resources $\{R_1, R_2, \ldots, R_m\}$. $\mathcal{T}$ will be feasible under work-conserving scheduling (i.e., without inserted idle time) if and only if*

$$\forall L > 0, \quad L \geq \sum_{i=1}^{n} f\left(\frac{L - d_i + y_i}{y_i}\right) \cdot x_i \cdot E_i \tag{9}$$

and

$$\forall i, 1 < i \leq n, \forall k, 1 \leq k \leq n_i \wedge r_{ik} \neq 0, \forall L, \delta_{r_{ik}} < L < d_i - S_{ik} :$$
$$L \geq C_{ik} + \sum_{j=1}^{i-1} f\left(\frac{L - 1 - d_j + y_j}{y_j}\right) \cdot x_j \cdot E_j \tag{10}$$

where:

- $f(a)$ *is as defined in Lemma 4.1,*
- $\delta_{r_{ik}} = \min_{1 \le j \le n} (d_j | \exists l, 1 \le l \le n_j : r_{jl} = r_{ik})$,
- $E_j = \sum_{l=1}^{n_j} C_{jl}$, *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 those for non-preemptive scheduling. The parameter E_i represents the maximum cost of an invocation of task T_i and replaces the c_i term in Condition (7) of Theorem 4.7. Condition (10) 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 Condition (10) 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 have terminated, and thus the earliest time phase k can be scheduled is S_{ik} time units after the start of an invocation of T_i. 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 $k-1$. Also note that no demand due to phases of T_i other than k appear in Condition (10). In the event that each task in $\mathcal{T}$ consists of only a single phase (i.e. non-preemptive scheduling), Conditions (9) and (10) reduce to the feasibility conditions of Theorem 4.7.

The proofs of Theorem 4.10 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 4.1, 4.3, and 4.4 in [8] for scheduling sporadic tasks that share a set of serially reusable, single unit resources.

A *generalized EDF* scheduling algorithm was introduced in [8] to schedule sporadic tasks that share a set of serially reusable, single unit resources $\{R_1, R_2, \ldots, R_m\}$. Let the *generalized RBE-EDF* scheduling algorithm be the *generalized EDF* scheduling algorithm of [8] with deadlines assigned using Equation (1).

Corollary 4.11. *With respect to the class of work-conserving schedulers, generalized RBE-EDF is an optimal scheduling algorithm for an RBE task set.*

5 Discussion

The RBE task model specifies the real-time execution of tasks such that no more than x deadlines expire in any y time units. It does not specify how the jobs must be scheduled to guarantee that deadlines are met, just as the periodic and sporadic task models do not specify a scheduling algorithm. In this paper, we have presented optimal scheduling algorithms, based on EDF scheduling, for scheduling

preemptively, non-preemptively, and preemptively with shared resources. While we have shown that no static priority scheduling algorithm can successfully schedule an RBE task set such that no job misses its deadline, there may be other dynamic priority scheduling algorithms (besides EDF) that can schedule an RBE task set in accordance with the model. We leave open the problem of identifying other scheduling algorithms that support the semantics of RBE.

The RBE task model has (in the past) been compared to other task models and scheduling algorithms that incorporate some notion of a rate. We now make our own comparison of the RBE task model to some of these models to (hopefully) clear up confusion and misunderstandings. Specifically we compare the RBE task model to *proportional share resource allocation* [2, 13, 15, 19, 21, 22] and the Total Bandwidth Server [17, 18].

Proportional share resource allocation is usually used to ensure fairness in resource sharing. It can, however, be used in the scheduling of real-time tasks [19]. In proportional share resource allocation, a *weight* is associated with each task that specifies the relative *share* of a 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. For example, if w is the weight associated with task T and W is the sum of all weights associated with tasks in the task set $\mathcal{T}$, the fraction of the resource allocated to task T is $f = \frac{w}{W}$. Thus, as competition for a resource increases, the fraction of the resource 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 (a resource) defined by $\frac{x \cdot c}{y}$ when $d = y$, no matter how much competition there is for the CPU (assuming the task set is feasible). One can, of course, fix the share of a resource allocated to a task in proportional share resource allocation by varying the task's weight relative to the other task weights [20]. Note that to schedule an RBE task T with $d < y$ using a proportional share resource allocation, the 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 proportional share resource allocation both support task execution rates, the systems differ markedly in the flexibility allowed in task scheduling. Proportional share resource allocation allows variable execution rates while RBE defines a maximum execution rate (tasks are allowed to execute at a slower rate but not a rate greater than that specified). The relative deadline of a task executed under proportional share 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 Total Bandwidth server (TB) was first proposed by Spuri and Buttazzo in [17], and later extended by Spuri, Buttazzo and Sensini in [18]. The original TB server is allocated a portion of the processor utilization, denoted U_S, to process aperiodic requests. The rest of the processor utilization,

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 $d_k = \max(r_k, d_{k-1}) + \frac{C_k}{U_S}$ where C_k is the worst case execution time of the k^{th} aperiodic request and U_S is the processor utilization allocated to the TB server. Thus, deadlines are assigned to aperiodic requests based on the rate at which the TB server can serve them, not at the rate which they are expected to arrive. Moreover, the aperiodic deadlines are assigned such that the k^{th} requests completes before the $k + 1^{th}$ 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 TB server is able to process them.

The TB server only serves aperiodic requests whereas the RBE task model assumes tasks execute with an expected rate. The RBE task model does not directly support aperiodic requests. However, the TB server can be combined with an RBE task set in the same way it was combined with a periodic task set and scheduled non-preemptively using a variation of EDF. It is not immediately clear that the semantics of a TB server can be preserved if it is combined with an RBE task set that is scheduled non-preemptively or with shared resources.

We have shown that static priority schedulers, such as rate monotonic (RM), cannot directly schedule RBE tasks because more than x jobs of task T may be released simultaneously. A static priority scheduler is unable to differentiate between "normal" jobs and "early" jobs. Thus, low priority jobs may miss deadlines while the static priority scheduler executes "early" jobs of high priority tasks. One possible remedy for this problem is to use a periodic (or sporadic) server for each task. The server for task T is scheduled with a static priority scheduler, and executes at most x jobs of T during each invocation. Thus, "early" jobs cannot execute until the next period of the server. This raises many scheduling issues that we do not address here since we are primarily interested in dynamic priority scheduling algorithms that support the natural execution of RBE tasks.

6 Summary and Conclusions

We have presented a new 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. We call this new task model *rate-based execution* (RBE), and it is a generalization of the common sporadic task model developed by Mok [14].

In the RBE model, tasks are expected to execute with an average execution rate of x times every y time units. We believe this task model more naturally models the actual implementation of event-driven, real-time systems. While RBE has been used to model the execution of applications ranging from multimedia computing to digital signal processing [5, 6, 10], this is the first formal presentation of the RBE task model.

We have identified necessary and sufficient conditions for determining the feasibility of scheduling an RBE task set such that no task misses its deadline. We also presented an optimal scheduling algorithm, called *rate-based-execution earliest-deadline-first* (RBE-EDF). RBE-EDF is the EDF scheduling algorithm with deadlines assigned using Function $D_i(j)$, Equation (1) on 5. By optimal, we mean that if a feasible schedule exists, RBE-EDF will find one.

We then considered non-preemptive scheduling and preemptive scheduling with shared resources. In both cases, we presented necessary and sufficient feasibility conditions and optimal scheduling algorithms with respect to the class of work-conserving scheduling algorithms (i.e., the class of scheduling algorithms that schedule without inserted idle time). The *non-preemptive rate-based-execution earliest-deadline-first* (NP-RBE-EDF) scheduling algorithm is a non-preemptive EDF scheduling algorithm with deadlines assigned using Function $D_i(j)$. NP-RBE-EDF is an optimal work-conserving scheduling algorithm for the non-preemptive execution of RBE tasks. The *generalized RBE-EDF* scheduling algorithm is the *generalized EDF* scheduling algorithm of [8] with deadlines assigned using Function $D_i(j)$, and it is an optimal work-conserving scheduling algorithm for the preemptive execution of RBE tasks with shared resources. A corollary of our results is that RBE tasks cannot be scheduled with static priority schedulers.

References

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

[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] 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 Dataflow Execution Paradigm using a Synthetic Aperture Radar Application," *Proc. IEEE Real-Time Technology and Applications Symposium*, 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 IEEE Real-Time Technology and Applications Symposium*, June 1998, pp. 59-70. *IEEE Real-Time Technology and Applications Symposium*, 1998

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

[8] Jeffay, K., "Scheduling Sporadic Tasks with Shared Resources in Hard-Real-Time Systems", *Proc. of the* 13^{th} *IEEE Real-Time Systems Symposium*, Phoenix, AZ, 1992, pp. 89-99.

[9] Jeffay, K., Stone, D., "Accounting for Interrupt Handling Costs in Dynamic Priority Task Systems", *Proc. of the* 14^{th} *IEEE Real-Time Systems Symposium* , Durham, NC, 1993, pp. 212-221.

[10] Jeffay, K., Bennett, D. "A Rate-Based Execution Abstraction For Multimedia Computing," *Lecture Notes in Computer Science*, T.D.C. Little and R. Gusella eds., Vol. 1018, Springer-Verlag, Heidelberg, 1995, pp 65-75.

[11] Jeffay, K., Goddard, S., "The Rate-Based Execution Model," Technical Report UNL-CSE-98-003, Computer Science and Engineering, University of Nebraska — Lincoln, May 1999.

[12] 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.

[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, Department of EE and CS, MIT/LCS/TR-297, May 1983.

[15] 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.

[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. of the IEEE Symposium on Real-Time Systems*, December 1994.

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

[19] 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.

[20] 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.

[21] 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.

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