2004

Variable Rate Execution

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

Xin Liu
University of Nebraska, lxin@cse.unl.edu

University of Nebraska-Lincoln, Computer Science and Engineering
Technical Report # TR-UNL-CSE-2004-0008

Variable Rate Execution

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

Technical Report TR-UNL-CSE-2004-8
April 2004

Abstract

We present a task model for adaptive real-time tasks in which a task's execution rate requirements are allowed to change at any time. The model, variable rate execution (VRE), is an extension of the rate-based execution (RBE) model. We relax the constant execution rate assumption of canonical real-time task models by allowing both the worst case execution time (WCET) and the period to be variable. The VRE model also supports tasks joining and leaving the system at any time. Another advantage of the new task model is that the exact execution rate need not be known for soft real-time or non-realtime applications; instead, an approximate execution rate can be assigned to an application and then dynamically adjusted during runtime. A schedulability condition for the VRE task model is presented that can be used as an on-line admission control test for the acceptance of new tasks or rate changes. Finally, a VRE scheduler was implemented in Linux as a loadable module, and several experiments demonstrate its correctness and analyze the overhead.

1 Introduction

Quality of Service (QoS) can be viewed as a spectrum of execution rate guarantees: hard real-time tasks are assigned an execution rate to meet all deadlines; soft real-time tasks are assigned a rate that meets most deadlines; non-real-time tasks, without any deadline, are assigned a best-effort rate that will not affect the hard and soft real-time tasks. Most conventional real-time operating systems provide both time-sharing and static priority scheduling algorithms. The time-sharing algorithms are *best-effort*, making no guarantee of execution rate; the fixed priority schedulers attempt to make fixed execution rate guarantees with an *all-or-nothing* approach, resulting in either success or failure.

In practice, many applications need to change their QoS requirements during runtime. In a multiple-target, multiple-sensor radar tracking system, the tasks tracking fast-moving targets have tighter time constraints than tasks tracking slow-moving targets. In a multi-agent system, the agents might dynamically negotiate with each other and decide the execution rate for each agent. Multimedia applications are common applications with dynamic soft QoS requirements. For example, a video decoder decodes 30 frames per second and it changes its QoS requirements when it degrades its service quality by either reducing the

resolution or skipping frames. Even with constant service quality, the encoding and decoding time of a MPEG frame can vary, depending on many factors such as the frame type or frame length [3].

In support of such dynamic QoS requirements, we first introduced the *variable rate execution* (VRE) model in [22], which is essentially an extension of the rate-based execution (RBE) task model [15]. While [22] and [23] focus on the implementation of variable rate tasks in Linux, this work formally presents the theoretical model. The VRE task model extends the RBE model to address dynamic QoS requirements by allowing tasks to execute with variable execution rates and supporting a dynamic task set. It forms a foundation for feedback control or adaptive applications where task execution rates change during runtime.

In the VRE model, a variable rate task is denoted by a four-tuple $(x_i(t), y_i(t), d_i(t), c_i(t))$ where each parameter is represented as a function of time t. Similar to the RBE model, $y_i(t)$ is the interval (or period) in which $x_i(t)$ jobs are expected to be released; $d_i(t)$ is the relative deadline, which is typically equal to the period $y_i(t)$; and $c_i(t)$ is the worst-case execution time (WCET). By relaxing either or both the *WCET* and *period*, a task can change the size of its jobs and/or change the release frequency of the jobs. Similar execution patterns are also supported by the *rate-based earliest deadline* (RBED) scheduler [6], which was independently and simultaneously developed. The RBED scheduler, however, might delay the acceptance of new tasks, while the VRE model can immediately accept new tasks by changing pending deadlines (as described in Section 3.1). Moreover, the VRE model is a more general model than that assumed in [6].

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 presents the VRE model. Section 4 presents the theoretical correctness and a schedulability condition. Section 5 presents our evaluation results of a VRE scheduler implemented in Linux as a loadable module. We conclude with a summary in Section 6.

2 Related Work

A variable execution rate is not a new idea. It comes with the concept of multi-programming. In conventional time-sharing systems, tasks actually execute at variable rates, which depend on the total number of tasks in the system. In Linux, a process receives a time slice in a variable length period, which is the sum of the time slices of all running processes.

In proportional-share systems, a task essentially runs at a variable execution rate as well, depending on the sum of all weights. This is why most proportional-share systems have the concept of *virtual time*. In real time, a task might be running at a variable rate; in virtual time, the task is treated as executing at a constant rate.

Weighted Fair Queueing (WFQ) [9] (also known as packet-by-packet generalized processor sharing (GPS)) is a well-known proportional-share scheduling algorithm from the networking literature. The WFQ scheduler associates a weight to each connection session; all the connection sessions share the router's bandwidth in proportion to their weights. The transmission rate of each session depends on the combination

of its weight and the summation of all weights. The virtual time $v(t)$ is defined as follows:

$$V(t) = \int_0^t \frac{1}{\sum_{j \in A(\tau)} w_j} d_\tau \tag{1}$$

where w_j is the weight of task j and $A(\tau)$ is the set of active tasks at time τ. Thus, virtual time progresses at a rate inversely proportional to the summation of all weights. That is, the more sessions in the system, the slower transmission rate each session gets.

Two recent multimedia schedulers are built on WFQ, SMART [33] and BERT [4]. The SMART scheduler [33] prioritizes a task by two parameters: *priority* and *biased virtual finishing time (BVFT)*. The scheduler always chooses the task with the highest priority. When multiple tasks are at the same priority level, the scheduler tries to satisfy as many BVFTs as possible. BERT [4] is essentially an implementation of WF^2Q plus a cycle stealing mechanism. Worst-case Fair Weighted Fair Queuing (WF^2Q) [5] is an extension of WFQ that prevents a task from getting executed faster than expected in a perfect fair share scheduler. While WF^2Q provides proportional sharing, the cycle stealing mechanism provides a flexible way for urgent tasks to meet their deadlines when their demands exceed their shares.

The Earliest Eligible Virtual Deadline First (EEVDF) [41] algorithm is another proportional-share algorithm that employs *virtual time*. The EEVDF algorithm puts all aperiodic jobs into the same queue and assigns a deadline for each job. According to task weight w_i, release time t_0^i and execution time r^k, the virtual eligible time ve and virtual deadline vd of a task are computed using equations presented in [41] and summarized as follows:

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

The virtual time in EEVDF is identical to the definition in WFQ, shown in Equation (1).

Although time-sharing and proportional-share systems actually execute tasks at variable rates, they do not explicitly state the variable execution rate in real time units. Instead, tasks are viewed as running at a constant virtual rate on a virtual processor whose speed varies. Obviously, without explicit admission control algorithms neither time-sharing nor proportional-share systems can make any QoS guarantees—if the number of tasks in the system grew very large, the resulting execution rate for each task would be very low.

In the context of real-time systems, two canonical task models are the *periodic model* [21], where jobs are released every period, and the *sporadic model* [28], where jobs are released with a minimum separation time. Although the two models guarantee temporal correctness, both are too strict for many applications. Thus, many variations of these models have been developed over the years.

A common technique to extending these models to provide QoS guarantees to non-real-time and soft real-time applications is to add a server task (e.g., [19, 39, 12, 10, 11, 2]). The server methods, on the one hand, guarantee a constant execution rate for the server, which partially satisfies the QoS requirements of non-real-time (aperiodic) requests. On the other hand, these models do not support explicit and dynamic rate changes. Although some algorithms, such as GRUB (Greedy Reclamation of Unused Bandwidth) [20],

allow bandwidth reclamation, this is more like stealing spare time than adjusting the bandwidth (execution rate).

There have been many task models introduced over the years that relax the strict assumptions of the periodic and sporadic task models without adding a "server" task. For example, the *rate-based execution* (RBE) model [15] is a generalization of the sporadic model that was developed to support the real-time execution of event-driven tasks in which no *a priori* characterization of the *actual* arrival rates of events is known; only the *expected* arrival rates of events are known. A RBE task is parameterized by a four-tuple (x, y, d, c). The task is expected to process x events every y time units, where each event takes no longer than c time units and event processing must be completed within d time units of its arrival. Rate is achieved by deadline assignment. The jth job of a RBE task T_i, J_{ij}, is assigned a deadline as follows:

$$D_i(j) = \begin{cases} t_{ij} + d_i & \text{if } 1 \leq j \leq x_i \\ \max(t_{ij} + d_i, D_i(j - x_i) + y_i) & \text{if } j > x_i \end{cases} \tag{2}$$

where t_{ij} is the release time of job J_{ij}. The RBE model schedules tasks at average rates. It does not, however, allow any of the task parameters or the set of tasks to vary at runtime.

The *enhanced* RBE model [13] was designed to integrate non-real-time tasks into the RBE model without using a "server" task. In that model, RBE tasks reserve a constant computing bandwidth while all aperiodic tasks share the remaining computing capacity in proportion to their weight. Since the number of aperiodic requests is dynamic, changing at runtime, each aperiodic request actually runs at a variable rate.

Several researchers have developed techniques for supporting variable computation times and/or release patterns (e.g., [26, 27, 42, 43]). However, each of these provides relatively strict bounds on how much these parameters are allowed to vary, as compared to the VRE model presented here. Researchers have also proposed methods for reducing task execution rates or computation times in overload conditions (e.g., [1, 17, 18, 29, 30, 31, 38]).

The first work to provide explicit increasing and decreasing hard QoS guarantees on a task-by-task basis appears to be the *elastic* task model created by Buttazzo, Lipari, and Abeni [8]. In the elastic task model, a task is parameterized by a five-tuple $(C, T_0, T_{min}, T_{max}, e)$ where C is the tasks's WCET, T_0 is the nominal period for the task, T_{min} and T_{max} denote minimum and maximum periods for the task, and e is an elastic coefficient. The elastic coefficient e "specifies the flexibility of the task to vary its utilization" [8]. In this case, the utilization is varied by changing the length of the period, which is allowed to "shrink" to T_{min} or "stretch" to T_{max}, depending on the system load. The VRE model presented here also allows the period of a task to shrink or stretch. In the VRE model, however, no bounds on the length of the period are defined a priori. Moreover, the VRE model also supports increasing and decreasing the WCET, which is not supported by the elastic task model.

Other researchers have taken a system-level approach to support adaptive real-time computing (e.g., [7, 24, 25, 34, 36, 40, 32, 35, 16, 37]). Most of these systems focus on over-load conditions and use various combinations of value-based scheduling, mode changes, and/or feedback mechanisms to shed or reduce

load in an attempt to meet the most critical deadlines. While the VRE model is designed to support adaptive real-time computing systems, the VRE model differs from these systems in that the VRE model does not rely on any of these techniques to handle over-load.

The work most similar to the VRE model is the *rate-based earliest deadline* (RBED) scheduler presented by Brandt et al. in [6]. In that work, the authors try to "flatten the scheduling hierarchy" by supporting hard real-time, soft real-time, and non-real-time tasks with a single scheduler. Their algorithm allows periodic tasks to dynamically change utilizations and periods. While the RBED scheduler delays the acceptance of new tasks until some tasks terminate and there is enough bandwidth available, a VRE scheduler releases the required bandwidth by adjusting existing deadlines. The details are discussed in Section 3.1. The underlying task model assumed by Brandt et al. in [6] is a generalization of the Liu and Layland periodic task model [21]. The VRE model is a generalization of the RBE task model, which is a generalization of Mok's sporadic task model [28]. The VRE task model reduces to the task model in [6] when $x_i(t) = 1, \forall i, t$, and jobs are released with a strictly (variable) periodic pattern rather than a (variable) sporadic pattern.

3 VRE Task Model

In conventional real-time terms, a task is a sequential program that executes repeatedly in response to the occurrence of events. Each instance of the task is called a job or a task instance. Each job of a task is assumed to execute no longer than a constant bound called the worst-case execution time (WCET). Classic real-time task models include the *periodic* task model [21], in which jobs are generated every *p* time units for some constant *p*, and the *sporadic* task model [28], in which jobs are generated no sooner than every *p* time units for some constant *p*. The *rate-based execution* (RBE) task model [15] is a generalization of sporadic tasks that allows early release patterns. It makes no assumptions about the relationships between the points at which jobs are released for a task; it assumes jobs are generated at a precise average rate but that the actual arrivals of jobs in time is arbitrary. The *variable-rate execution* (VRE) task model provides two primary extensions to these models: (i) variable WCET and periods, which may change at any time t, and (ii) a dynamic task set in which tasks are allowed to enter and leave the system at arbitrary times.

3.1 Variable Rate Execution

In contrast to a RBE task, a VRE task reserves an initial execution rate and then may dynamically adjust its execution rate by changing either its WECT or its period. If the execution rate of a task does not change, VRE task execution is identical to RBE task exectuion. Moreover, if a VRE task never generates more than one job simultaneously and never changes it execution rate, it reduces to a sporadic task.

Following the notation of the RBE model, a VRE task is described by four parameters $(x_i(t), y_i(t), d_i(t), c_i(t))$. Similar to the RBE model, $y_i(t)$ is the interval in which $x_i(t)$ jobs are expected to be released; $d_i(t)$ is the relative deadline, which is typically equal to the period $y_i(t)$; and $c_i(t)$ is the WCET. (We assume $d_i(t) = y_i(t)$ in this work.) Rather than the constant rate of the RBE model, each parameter is a variable, which may change during runtime. To effect a rate change, a VRE task can change either its execution time, $c_i(t)$, or

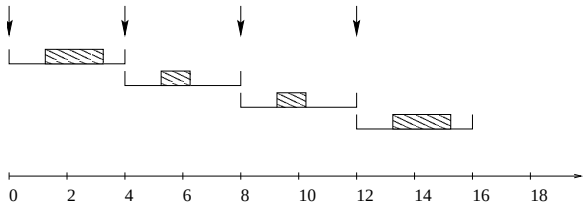

Figure 1: The initial execution rate is $(1, 4, 4, 2)$. At time 4, the execution rate changes to $(1, 4, 4, 1)$, and the execution rate changes back to $(1, 4, 4, 2)$ at time 12.

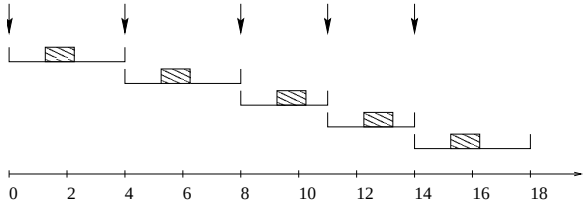

Figure 2: The initial execution rate is $(1, 4, 4, 1)$. At time 8, the execution rate changes to $(1, 3, 3, 1)$, and the execution rate changes back to $(1, 4, 4, 1)$ at time 14.

its job release rate, $(x_i(t), y_i(t))$. To reflect the ability of a task to change its execution rate, the deadline assignment function of Equation (2) is extended to Equation (3) as follows.

$$D_i(j) = \begin{cases} t_{ij} + d_i(t) & \text{if } 1 \leq j \leq x_i(t) \\ \max(t_{ij} + d_i(t), D_i(j - x_i(t)) + y_i(t)) & \text{if } j > x_i(t) \end{cases} \tag{3}$$

where t_{ij} is the release time of job J_{ij}.

Figures 1 and 2 are two simple examples that illustrate how the variable rate execution model works. For simplicity, the rate changes in these examples are made at task deadlines, but this is not required. In Figure 1, the initial execution rate is $(1, 4, 4, 2)$, and the $c_i(t)$ parameter is adjusted during runtime. At time $t = 4$, the WCET is changed from 2 to 1. Thus, execution rate changes to $(1, 4, 4, 1)$, and the next two execution intervals each require at most 1 time unit. At time $t = 12$, the task's $c_i(t)$ parameter is changed to 2, and the execution rate changes back to its initial specification: $(1, 4, 4, 2)$. This example might represent a scenario in which a video player changes its resolution and needs more or less execution time in an interval of $y_i(t)$ time units.

A scenario in which a video player skips frames is shown in Figure 2. In this case, the $y_i(t)$ parameter is adjusted during runtime. The initial execution rate specification is $(1, 4, 4, 1)$, and at time $t = 8$ the execution rate changes to $(1, 3, 3, 1)$. The execution rate changes back to $(1, 4, 4, 1)$ at time $t = 14$.

Adjusting Pending Deadlines. Equation (3) defines the deadline assignment rule for newly released jobs. However, a task may have pending jobs when its rate changes since "early releases" are allowed. The simplest approach to handling pending jobs is to keep the rate unchanged until all of a task's pending jobs are completed. Thus, the deadlines of the real-time jobs are not modified once they are released. The

RBED scheduler [6] uses this method. Since the utilization is always 100 percent in a mixed system if there exists any best-effort task, the RBED scheduler actually has to delay the acceptance of new tasks until some running tasks terminate and enough bandwidth is released.

In some cases, however, the new tasks might have tighter time constraints than some running tasks, and we want to immediately change the execution rates of the low-priority running tasks. In these cases, the deadlines of a low-priority task's pending jobs are modified such that demand is bounded and the bandwidth is released immediately.

As previously stated, we assume in this work that $d_i(t) = y_i(t)$, which allows an efficient on-line admission and rate-change control function. Observe that the fraction of the CPU allocated to any one job of a VRE task V_i at time t is $f_i(t) = \frac{c_i(t)}{y_i(t)}$. Ideally, if either $c_i(t)$ or $y_i(t)$ change at some time t_x, then each of the pending deadlines of task V_i can be re-computed by dividing the expected remaining service time required to complete the pending job by its new fraction $f_i(t_x)$ and adding this to time t_x. Let $D_i(j)$ be a pending deadline and r be the expected remaining service time, which is the amount of service time that would remain in a perfectly fair system. The new deadline is computed using Equation (4).

$$D_i'(j) = t_x + \frac{r}{f_i(t_x)} \tag{4}$$

In a perfectly fair system, the remaining service time r is computed as

$$r = \bar{S}_i(t_x, D_i(j)) = \int_{t_x}^{D_i(j)} f_i(t)dt = (D_i(j) - t_x) \cdot f_i(t_x - 1) \tag{5}$$

where $\bar{S}_i(t_1, t_2)$ denotes the service time the job of task V_i would receive in a perfectly fair system during the interval $[t_1, t_2]$ and $f_i(t_x - 1)$ is the fraction of the processor that would have been allocated to the job of V_i in the interval.

By combining Equations (4) and (5), the pending deadline can be rewritten using Equation (6).

$$\begin{aligned} D_i'(j) &= t_x + \frac{\bar{S}_i(t_x, D_i(j))}{f_i(t_x)} = t_x + \frac{(D_i(j) - t_x) \cdot f_i(t_x - 1)}{f_i(t_x)} \\ &= t_x + (D_i(j) - t_x) \cdot \frac{f_i(t_x - 1)}{f_i(t_x)} \end{aligned} \tag{6}$$

Equation (6) actually assumes that the lag of the job is zero. The lag is defined as the difference between the ideal and actual service times. That is, $lag_i(t_x) = S_i(0, t_x) - s_i(0, t_x)$, where $s_i(0, t_x)$ is the actual service time received in the interval. The lag may be zero, strictly negative, or strictly positive. If the lag is strictly negative, the job is executing ahead of its ideal execution rate. This case never creates a problem because the rate change function of Equation (6) moves the deadline based on an ideal execution rate, and simply not executing the job for a period of time equal to $\frac{lag_i(t_x)}{f_i(t_x)}$ would eliminate the lag, which was assumed to be zero.

If the lag is strictly positive, the task is proceeding behind its ideal service time. If a rate change at time t_x results in $f_i(t_x - 1) > f_i(t_x)$, then the deadline will be postponed, which gives time for the lag to return to zero as the job's actual execution rate "catches up" to its ideal rate. The only possible problem arises when $f_i(t_x - 1) < f_i(t_x)$. In this case, the new deadline is moved to an earlier time, but it must be large enough to allow the actual service time to "catch up" with the ideal execution rate. Thus, with $s_i(t_x)$ equal to the actual service time at time t_x, we rewrite Equation (6) as follows:

$$D_i'(j) = t_x + max((D_i(j) - t_x) \cdot \frac{f_i(t_x - 1)}{f_i(t_x)}, c_i(t_x - 1) - s_i(t_x)) \tag{7}$$

There are three parameters that can be used to adjust the rate of a VRE task, $c_i(t)$, $y_i(t)$, and $x_i(t)$. In the following, we respectively describe the three cases when only one parameter changes at a time. Simultaneous changes to more than one parameter can be achieved by combining the corresponding rules. In the special case of simultaneous changes to $c_i(t)$, and $y_i(t)$, the combination of Rules 1 and 2 reduce to Equation (7).

- **Rule 1:** $c_i(t)$ *changes at time* t_x. The pending deadlines of jobs of task V_i are changed to accommodate the change in WCET, as long as the new $c_i(t_x)$ parameter is greater than the amount of time already consumed. Since, in this case,

$$\frac{f_i(t_x-1)}{f_i(t_x)} = \frac{\frac{c_i(t_x-1)}{y_i(t_x)}}{\frac{c_i(t_x)}{y_i(t_x)}} = \frac{c_i(t_x-1)}{c_i(t_x)},$$

  substituting $\frac{f_i(t_x-1)}{f_i(t_x)}$ with $\frac{c_i(t_x-1)}{c_i(t_x)}$ in Equation (7), we get the following equation:

$$D_i'(j) = t_x + max((D_i(j)-t_x)\cdot\frac{c_i(t_x-1)}{c_i(t_x)}, c_i(t_x-1)-s_i(t_x)).$$

  If the new $c_i(t_x)$ parameter is less than or equal to the actual amount of execution time already consumed by the job, $c_i(t_x) \leq s_i(t_x)$, the rate change takes place at the end of the current execution period—or at the next earliest point at which the job's lag reduces to zero, and the new deadline is assigned using Equation (3).

- **Rule 2:** $y_i(t)$ *changes at time* t_x. Pending deadlines of the task are adjusted by substituting $\frac{f_i(t_x-1)}{f_i(t_x)}$ with $\frac{y_i(t_x)}{y_i(t_x-1)}$ in Equation (7).

$$D_i(j) = t_x + ((D_i(j)-t_x)\cdot\frac{y_i(t_x)}{y_i(t_x-1)}, c_i(t_x-1)-s_i(t_x)).$$

- **Rule 3:** $x_i(t)$ *changes at time* t_x. This case is different from the other two since the change in the $x_i(t)$ parameter affects the total fraction of the CPU allocated to the task, but not the fraction of the CPU allocated to any one job of the task. Let $(D_i(j), D_i(j+1), D_i(j+2), ..., D_i(j+k-1), D_i(j+k))$ be the set of pending deadlines ordered by time. We treat all pending jobs to be released at time t_x. Thus, the pending deadlines are modified as follows:

$$D_i'(j+m) = t_x + y_i(t_x)\cdot(\lfloor\frac{m}{x_i(t_x)}\rfloor + 1),\ 0 \leq m \leq k$$

We show in Section 4 that these deadline adjustments will not affect temporal correctness of the task set.

3.2 Supporting a Dynamic Task Set

As stated previously, the VRE model supports a dynamic task set, allowing tasks either to enter or to leave the system at any time. When a new VRE task V_{new} arrives at time t, the task is tentatively added to the set of

tasks $V(t)$ and the schedulability condition $\sum_{i \in V(t)} f_i(t) \leq 1$, which is presented in Section 4, is evaluated. An affirmative result means that the task is accepted and deadlines are assigned using Equation (3).

Theoretically, a task leaves the system when its lag reaches zero. At this point in time, the fraction of the processor allocated to that task can be allocated to another task, and the task is removed from the task set. Usually when a job finishes before its deadline, however, it has negative lag. Thus, if the last job of task V_i executed for $c_i(t)$ time units, the fraction of the processor allocated to task V_i cannot be re-allocated until the deadline of that job is reached.

In an implementation of the task model, there are two simple options for tracking the system utilization when jobs enter and leave. The first method is to set a timer to expire at the deadline of the last job of a terminating task V_i. When the timer expires at time t_f, the lag has reached zero and the task is removed from the task set. In practice this simply means subtracting $x_i(t_f) \cdot f_i(t_f)$ from the total allocated processor utilization.

Alternatively, when a task finishes with non-zero lag, the deadline of the last job can be inserted in a queue sorted by non-decreasing finish times. Using this method, the task is not removed from the task set until its processor utilization is needed by another task. This only happens when the schedulability condition yields a negative result. At this point, all entries in the queue with finish times less than or equal the current time are dequeued. For each of these dequeued tasks V_i, $x_i(t_f) \cdot f_i(t_f)$ is subtracted from the total allocated processor utilization. If the schedulability condition still yields a negative result, subsequent jobs in the queue with the next earliest finish times are tentatively removed from the queue and their processor utilizations subtracted from the total allocated processor utilization. If there is still insufficient processor bandwidth, the new task is not allowed to join the system. On the other hand, if this results in sufficient processor bandwidth being made available, the new task is allowed to join the system, but lag of the jobs with future finish times must be transferred to the new task. Let Q_f denote this set of jobs. The simplest way to transfer the lag of the jobs in Q_f to the new task V_{new} is to set the deadline of the first job of V_{new} at time t is as follows:

$$D_{new}(1) = t + d_{new}(t) - \sum_{i \in Q_f} \frac{lag_i(t)}{f_i(t)} \tag{8}$$

Of course, there are many other ways of transferring the negative lag of jobs in the set Q_f. The advantages of this approach is its simplicity and the fact that the processor utilization is updated only when necessary to accept a new task (or an increase in the execution rate of a current task), which reduces overhead that might occur in a very dynamic task set.

4 Feasibility

This section presents a schedulability condition when $d_i(t) = y_i(t)$. We first define the processor demand bound for variable rate tasks. Then, we give a sufficient schedulability condition for the variable rate task set. We leave open the question of necessary and sufficient conditions since they cannot be computed without a priori knowledge rate changes, which precludes their use as on-line admission and rate-change controller

functions.

Given a variable rate task $V_i = (x_i(t), y_i(t), y_i(t)c_i(t),)$ and a specific time interval $[t_{x0}, t_{xk}]$, assume that the rate changes at time $t_{x1}, t_{x2}, ...t_{xk-1}, t_{x0} < t_{x1} < t_{x2} < ... < t_{xk}$. For the case in which we change the rate after all pending deadlines are finished, the execution of V_i can be viewed as a sequence of intervals such that each interval is a RBE task exectuion. According to [15], the least upper bound on the processor demand in the interval $[t_{xj}, t_{xj+1}]$ is $\lfloor \frac{t_{xj+1}-t_{xj}}{y_i(t_{xj})} \rfloor \cdot x_i(t_{xj}) \cdot c_i(t_{xj})$. Thus, the least upper bound on demand is a sum of the demand in each of these intervals

$$\sum_{j=0}^{k-1} \lfloor \frac{t_{xj+1} - t_{xj}}{y_i(t_{xj})} \rfloor \cdot x_i(t_{xj}) \cdot c_i(t_{xj})$$

which depends on the actual values of $t_{x1}, t_{x2}, ...t_{xk-1}$.

If the rate change takes effect immediately (as opposed to the end of an execution interval for the task), the least upper bound on demand is an even more complicated step function that can only be computed in advance if exact future times of rate changes are known. The following demand bound, however, is easy to compute and is tight at the deadline of each job.

Lemma 4.1. *Let V_i be a variable rate task $(x_i(t), y_i(t), y_i(t), c_i(t))$. If no job of V_i released before time $t_0 \geq 0$ requires processor time in the interval $[t_0, l]$ to meet a deadline in the interval $[t_0, l]$, then*

$$\forall l > t_0, \ \ \widehat{dbf}([t_0, l]) = \int_{t_0}^{l} f_i(t)dt \tag{9}$$

is an upper bound on the processor demand in the interval $[t_0, l]$ created by V_i where $f_i(t)$ is the fraction function of V_i computed by $f_i(t) = \frac{x_i(t) \cdot c_i(t)}{y_i(t)}$.

Proof: The proof of this lemma is built upon the RBE task demand bound in [15], and separated into two parts. First, we show that Equation (9) is an upper bound on the processor demand created by task V_i when its share $f_i(t)$ never changes in the interval $[t_0, l]$. Second, we show the lemma also holds when $f_i(t)$ changes at any time $t_x \in [t_0, l]$, which is done by mapping segments of the interval to the first case.

Case 1: Execution rate never changes through the interval $[t_0, l]$. This is a straightforward reduction from Lemma 4.1 in [15], which states that the tight upper bound on processor demand created by a RBE task is $dbf_i(L) = \lfloor \frac{L-d_i+y_i}{y_i} \rfloor x_i c_i$. Since we assume $d_i = y_i$, let (x_i, y_i, c_i, y_i) be the constant execution rate in the interval and $L = l - t_0$. Based on Lemma 4.1 in [15] and the fraction definition ($f_i(t) = \frac{x_i(t) \cdot c_i(t)}{y_i(t)}$), the demand created by V_i is

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

Thus, Equation (9) is an upper bound on the processor demand in the interval $[t_0, l]$ created by V_i and the lemma holds for this case.

Case 2: Execution rate changes at time $t_x \in [t_0, l]$. Without loss of generality, assume t_x is the first time $f_i(t)$ changes in the interval $[t_0, l]$. The remaining portion of this proof will assume that $f_i(t)$ remains constant in the interval $[t_x, l]$. If $f_i(t)$ changes in the interval $[t_x, l]$, then this proof can be applied recursively to that interval. There are two sub-cases to consider: when there is no pending deadlines at time t_x, and when there exist some pending deadlines.

Case 2a: there is no pending deadlines at time t_x.

In this case, the execution of V_i in the interval $[t_0, l]$ can be treated as two separate portions: $[t_0, t_x]$ and $[t_x, l]$. From the result of Case 1, the demand created by V_i in the subinterval $[t_0, t_x]$ is bounded from above by $\widehat{dbf}_i([t_0, t_x])$ since the rate does not change in the subinterval. Similarly, the demand created by V_i in the subinterval $[t_x, l]$ is bounded from above by $\widehat{dbf}_i([t_x, l])$. Thus, the processor demand created by V_i in the interval $[t_0, l]$ is less than or equal to

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

and the lemma holds for this case.

Case 2b: There exist pending deadlines of V_i at time t_x.

Let $D_p = \{D_i(j), D_i(j+1), D_i(j+2), ..., D_i(j+k)\}$ be the set of all pending deadlines ordered by time, D_p are modified with the three rules in Section 3 when the execution rate changes. We know the demand bound function (9) holds before the change. Thus, $\forall D \in D_p, Demand([t_0, D]) \leq \int_{t_0}^{t_x} f_i(t)dt + f_i(t_x - 1) \cdot (D - t_x)$.

Suppose a deadline $D \in D_p$ is changed to D', we show that $\int_{t_0}^{D'} f_i(t)dt$ is still a demand bound for the change by each rule.

For *Rules 1 and 2*, the order of the modified deadlines is preserved. Thus, the demand created in the interval $[t_0, t_x]$ after the rate change is the same as the demand in the interval$[t_0, t_x]$ before the rate change. Before the change, the demand δ is less than or equal to $\int_{t_0}^{t_x} f_i(t)dt + f_i(t_x - 1) \cdot (D - t_x)$.

Based on *Rule 1* and *Rule 2*, $(D' - t_x) \cdot f_i(t_x) = (D - t_x) \cdot f_i(t_x - 1)$. Thus,

$$\int_{t_0}^{D'} f_i(t)_{dt} = \int_{t_0}^{t_x} f_i(t)_{dt} + (D' - t_x) \cdot f_i(t_x) = \int_{t_0}^{t_x} f_i(t)d_t + f_i(t_x - 1) \cdot (D - t_x)$$

still holds as a demand bound.

Rule 3 is even simpler. Let $D_i'(j+m)$ be a modified pending deadline ($t_x < D_i'(j+m)$). We have $Demand([t_0, t_x]) \leq \int_{t_0}^{t_x} f_i(t)d_t$ and $Demand([t_x, D_i'(j+m)]) \leq \int_{t_x}^{D_i'(j+m)} f_i(t)d_t$ by *Rule 3*. Thus, $\int_{t_0}^{D_i'(j+m)} f_i(t)d_t$ still holds as a demand bound.

This completes the proof. □

Theorem 4.2. *Let the task set $\mathcal{V} = \bigcup_{t=0}^{\infty} V(t)$ be a set of variable rate tasks with $d_i(t) = y_i(t), 1 \leq i \leq n$. Preemptive EDF will succeed in scheduling $\mathcal{V}$ if*

$$\forall L > 0, L \geq \sum_{j \in \mathcal{V}} \widehat{dbf}_j(L) \tag{10}$$

Proof: To show the sufficiency of Equation (10), it is shown that the preemptive EDF scheduling algorithm can schedule all releases of tasks in $\mathcal{V}$ without any job missing a deadline if the tasks satisfy Equation (10). This is shown by contradiction.

Assume that $\mathcal{V}$ satisfies Equation (10) and yet there exists a release of a task in $\mathcal{V}$ that misses a deadline at some point in time when $\mathcal{V}$ is scheduled by the EDF algorithm. Let t_d be the earliest point in time at which a deadline is missed and let t_0 be the later of:

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

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

By Lemma 4.1, at most $\widehat{dbf}_i([t_0, t_d])$ units of processing time are needed to process requests of variable rate task V_i in the interval under a deadline driven scheduling algorithm, such as EDF. Thus,

$$\sum_{j \in V([t_0, t_d])} \widehat{dbf}_j([t_0, t_d])$$

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

$$\sum_{j \in \mathcal{V}} \widehat{dbf}_j(t_d, t_0) \geq t_d - t_0.$$

However, this contradicts our assumption that $\mathcal{V}$ satisfies Equation (10).

Hence if $\mathcal{V}$ satisfies Equation (10), then no release of a task in $\mathcal{V}$ misses a deadline when $\mathcal{V}$ is scheduled by a deadline driven algorithm such as EDF. It follows that satisfying Equation (10) is a sufficient condition for schedulability under preemptive EDF. □

Corollary 4.3. *Let the task set $\mathcal{V} = \bigcup_{t=0}^{\infty} V(t)$ be a set of variable rate tasks with $d_i(t) = y_i(t), 1 \leq i \leq n$. Preemptive EDF will succeed in scheduling $\mathcal{V}$ if Equation* (11) *holds where $f_i(t) = \frac{x_i(t) \cdot c_i(t)}{y_i(t)}$ is the portion of the CPU capacity allocated to variable rate task V_i at time t.*

$$\forall t, \sum_{i \in V(t)} f_i(t) \leq 1 \tag{11}$$

Proof:

$$\sum_{i \in V(t)} f_i(t) \leq 1 \Rightarrow t - t_0 \geq \int_{t_0}^{t} (\sum_{i \in V(t)} f_i(t) dt$$

$$t - t_0 \geq \sum_{i \in V(t)} \int_{t_0}^{t} f_i(t)) dt$$

$$t - t_0 \geq \sum_{i \in V(t)} \widehat{dbf}_i([t_0, t]) \quad \text{by Lemma 4.1}$$

□

Equation (11) looks like the necessary and sufficient condition of EDF in [21], but it is actually different. The VRE model supports a dynamic task set in which tasks are allowed to release jobs early. This means we can have intervals of time in which the utilization function is greater than 1 adjacent to intervals of time in which the utilization function less than 1, and the task set may still be schedulable. Thus, Equation (11) is only sufficient, and not necessary. To develop a tighter condition, which is both sufficient and necessary, the actual times of rate changes must be known a priori. Since we do not make this assumption, it is infeasible to evaluate such a condition.

Corollary 4.3 can be used as the condition for admission and rate-change control. When a new variable rate task arrives or an existing variable rate task requests to change its rate, the system will recompute the sum of the fractions. If the sum is less than or equal to 1, accept the request; otherwise, reject the request. (See Section 3.2 for more details on the use of such a condition for admission and rate-change control.)

5 Evaluation

This section introduces our experimental results and overhead measurements. We first present an experiment which focused on adjusting the execution rate and correctness. Following that, we discuss the overhead.

The scheduler was implemented as a loadable Linux module on Redhat 8.0 (kernel version 2.4.18). Only a small modification was made to the Linux kernel. Thus, users can load or unload our scheduler without reboot. The experiments were done on an IBM Thinkpad T30 with a 2.0G Hz P4 processor and 256M DDR memory. We set the *time tick* to be $1ms$ and recompiled the Linux kernel. When we refer to an execution rate, we use a *tick* as the time unit. For example, rate $(1, 20, 20, 2)$ means 2 ticks (2ms) every 20 ticks (20ms).

The first experiment was on adjusting the execution rates. We simulated a multi-agent system, where three agents negotiate with each other to decide their execution rates during runtime. The three agents did nothing but execute a null loop and change execution rates at specific times, shown in Table 1. Figure 3 is the actual execution time of the three agents. It is clear that the actual execution rate changes are consistent with the assigned rates in Table 1.

We also sampled the deadlines of 180 jobs of the multi-agent task system and their corresponding finish times, which is shown in Figure 4. All the jobs finished before their deadlines. We can see a gap between

Time	0	19	37
Agent 1	(1,20,20,2)	(1,20,20,2)	(1,20,20,6)
Agent 2	(1,20,20,10)	(1,20,20,2)	(1,20,20,6)
Agent 3	(1,20,20,4)	(1,20,20,12)	(1,20,20,4)

Table 1: Rate adjustment of the three agents.

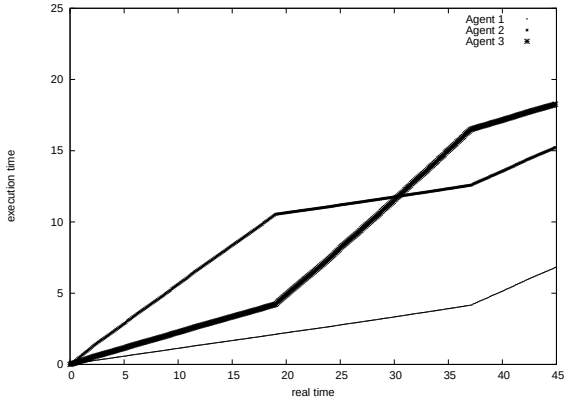

Figure 3: The actual execution time of the three agents.

the deadlines and the finish times. Actually, the gap enlarged as time went by. Since the three agents did not occupy all the processor capacity, 20% of the processor time is reserved for non-real-time tasks, such as the shell. When we compute the execution rate of the non-real-time tasks, we round off a *float* variable to an integer variable. Thus, the non-real-time tasks run slower than expected, and the VRE tasks run faster than their assigned rate.

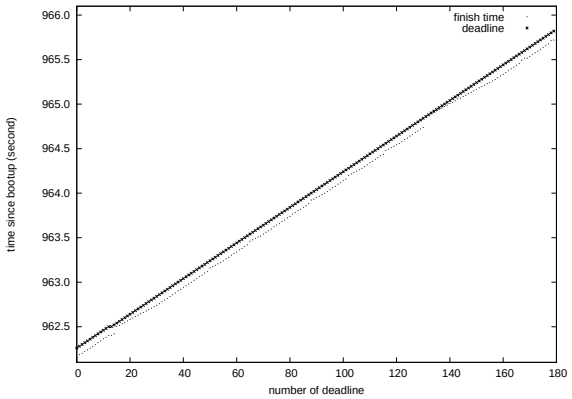

Figure 4: Deadlines and finish times.

In the Linux kernel, all running processes are put in a list called *runqueue*. The Linux scheduler scans the entire list and selects the process with the highest priority. Our implementation also follows this pattern though another implementation might be more efficient. Thus, the overhead shall be a linear function of the number of running processes. We measured the overhead of our scheduler, and compared it with the

overhead of the original Linux scheduler. The overhead was measured in CPU cycles, which was retrieved by the "*rdtsc*" instruction (*read timestamp counter*). Figure 5 shows that this implementation results in an at most 2.25% overhead for scheduling and context switching. More importantly, the VRE scheduler provides assured and dynamic QoS to processes, which the native Linux scheduler cannot provide, with only a slight increase in overhead. These results are consistent with Brandt et al. in [6] where a slightly simpler variable rate task model was implemented in Linux, with the change made to the kernel rather that as a loadable module. See [23] for a more detailed analysis of the implementation and performance of the VRE scheduler.

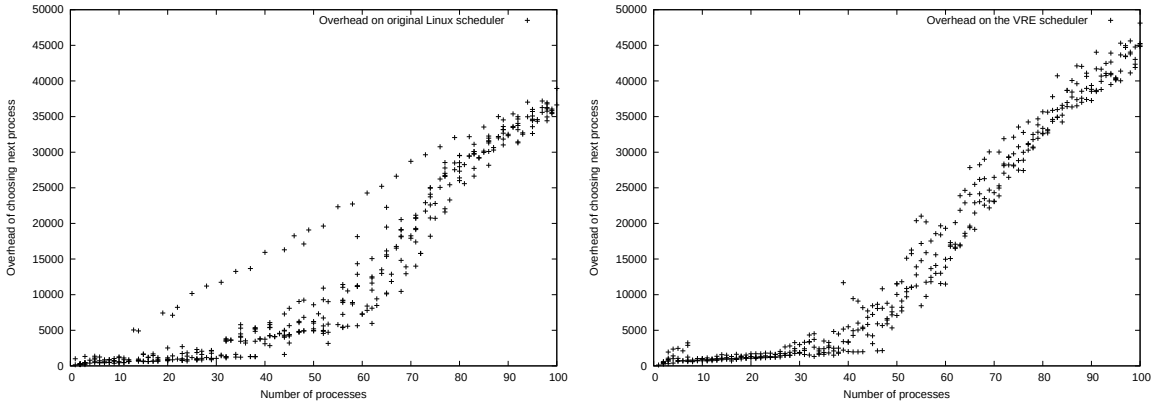

Figure 5: The overhead is a linear function of the number of processes under bother the original Linux scheduler and our scheduler. The overhead of our scheduler is a little bit higher than the original Linux scheduler.

6 Conclusion

model for real-time tasks in which execution rate requirements might change during runtime. We called the new task model *variable rate execution* (VRE). In the VRE model, we relax the assumptions made by canonical real-time task models by allowing both the worst case execution time (WCET) and the period to be variable. For soft or non-real-time tasks, an advantage of the new task model is that the exact execution rates need not be known when the task begins to execute; instead, we can assign an approximate execution rate to an application and dynamically adjust the rate during runtime. An efficient schedulability condition was also presented that can be used as an admission and rate-change control function. A scheduler supporting the VRE task model was implemented in Linux as a loadable module, and several experiments demonstrated its correctness and analyzed the overhead.

References

[1] Abdelzaher, T.F., Atkins, E.M., and Shin, K.G. "QoS Negotiation in Real-Time Systems and Its Applica-tions to Automated Flight Control," *Proceedings of the IEEE Real-Time Technology and Ap-*

plications Symposium, Montreal, Canada, June 1997.

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

[3] Bavier, A., Montz, A., and Peterson, L., "Predicting MPEG execution times," *Proc. of the Joint International Conf. on Measurement and Modelling of Computer Systems*, Madison, WI, Jun. 1998, pp. 131-140.

[4] Bavier, A., Peterson, L. and Mosberger D., "BERT: A scheduler for best effort and real-time tasks," Technical Report TR-602-99, Department of Computer Science, Princeton University, Mar, 1999.

[5] Bennett, J., Zhang, H., "WF2Q: Worst-case Fair Weighted Fair Queueing," IEEE INFOCOM '96, San Francisco, CA, Mar. 1996, pp. 120-128.

[6] Brandt, S. A., Banachowski, S., Lin, C., and Bisson, T., "Dynamic Integrated Scheduling of Hard Real-Time, Soft Real-Time and Non-Real-Time Processes," *Proceedings of IEEE Real-Time System Symposium*, December 2003, pp. 396-407.

[7] Burns, A., D. Prasad, A. Bondavalli, F. Di Giandomenico, K. Ramamritham, J. Stankovic, and L. Stringini, "The Meaning and Role of Value in Scheduling Flexible Real-Time Systems," *Journal of Systems Architecture*, 2000.

[8] Buttazzo, G. C., Lipari, G., and Abeni, L., "Elastic Task Model for Adaptive Rate Control," *Proceedings of IEEE Real-Time System Symposium*, December 1998, pp. 286-295.

[9] Demers, A., Keshav, S., and Shenker, S., "Analysis and simulation of a fair queuing algorithm," *Proceedings of the SIGCOMM '89 Symposium*, Sep. 1989, pp. 112.

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

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

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

[13] Goddard, S., Liu, X., "Scheduling Aperiodic Requests under Rate-Based Execution model" *Proceedings of IEEE Real-Time System Symposium*, December 2002, pp. 15-25.

[14] Goddard, S., Liu, X., "Variable Rate Execution," *Technical Report*, University of Nebraska-Lincoln, Department of Computer Science and Engineering, TR-UNL-CSE-2004-8, April 2004.

[15] Jeffay, K., Goddard, S., “A Theory of Rate-Based Execution,” *Proceedings of the 20th IEEE Real-Time Systems Symposium*, Phoenix, Arizona, December 1999, pp. 304-314.

[16] Jehuda, J. and A. Israeli, “Automated Meta-Control for Adaptable Real-Time Software,” *Real-Time Systems Journal*, 14(2), pp. 107-134, Mar. 1998.

[17] Kuo, T.-W., and Mok, A. K, “Load Adjustment in Adaptive Real-Time Systems,” *Proceedings of the 12th IEEE Real-Time Systems Symposium*, December 1991.

[18] Lee, C., Rajkumar, R., and Mercer, C., “Experiences with Processor Reservation and Dynamic QOSin Real-Time Mach,” *Proceedings of Multimedia Japan 96*, April 1996.

[19] Lehoczky, J.P., Sha, L., and Strosnider, J.K., “Enhanced Aperiodic Responsiveness in Hard Real-Time Environments,” *Proceedings of IEEE Real-Time Systems Symposium*, pp. 261-270, Dec. 1987.

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

[21] 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.

[22] Liu, X., Goddard, S., “A Loadable Variable Rate Execution Scheduler,” *Proceedings of the Real-Time Linux Workshop*, Valencia, Spain, Nov. 2003, pp. 187-196.

[23] Liu, X., Goddard, S., “Supporting Dynamic QoS in Linux,” *in Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium*, Toronto, Canada, May 2004.

[24] Lu, C., J. Stankovic, T. Abdelzaher, G. Tao, S. Son, and M. Marley, “Performance Specifications and Metrics for Adaptive Real-Time systems,” *Proceedings of IEEE Real-Time Systems Symposium*, pp. 13-22, Nov. 2000.

[25] McElhone, C., and A. Burns, “Scheduling Optional Computations for Adaptive Real-Time Systems,” *Journal of Systems Architectures*, 2000.

[26] Mercer, C. W., Savage, S., and Tokuda, H., “Processor Capacity Reserves for Multimedia Operating Systems,” *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, May 1994.

[27] Mok, A. K., and Chen, D., “A multiframe model for real-time tasks,” *Proceedings of IEEE Real-Time System Symposium*, Washington, December 1996.

[28] 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.

[29] Nakajima, T., “Dynamic QOS Control and Resource Reservation,” *IEICE, RTP’98*, 1998.

[30] Nakajima, T., “Resource Reservation for Adaptive QOS Mapping in Real-Time Mach,” *Sixth International Workshop on Parallel and Distributed Real-Time Systems*, April 1998.

[31] Nakajima, T., and Tezuka, H., “A Continuous Media Ap-plication supporting Dynamic QOS Control on Real-Time Mach,” *Proceedings of the ACM Multimedia ’94*, 1994.

[32] Nett, E., M. Gergeleit, and M. Mock, “An Adaptive Approach to Object-Oriented Real-Time Computing,” *Proceedings of ISORC*, April 1998.

[33] Nieh, J., Lam, M., “The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications,” *Proc. of the 16th ACM Symposium on Operating Systems Principles*, Saint-Malô, France, Oct. 1997, pp. 184-197.

[34] Prasad, D., and A. Burns, “A Value-Based Scheduling Approach for Real-Time Autonomous Vehicle Control,” *Robotica*, 2000.

[35] Rajkumar, R., K. Juvva, A. Molano, and S. Oikawa, “Resource Kernels: A Resource-Centric Approach to Real-Time Systems,” *Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking*, Jan. 1998.

[36] Rosu, D., *Dynamic Resource Allocation for Adaptive Real-Time Applications*, PhD thesis, Dept. of Computer Science, Georgia Institute of Technology, 1999.

[37] Rosu, D., K. Schwan, S. Yalamanchili, and R. Jha, “On Adaptive Resource Allocation for Complex Real-Time Applications,” *Proceedings of the 18th IEEE Real-Time Systems Symposium*, pp. 320-329, Dec. 1997.

[38] Seto, D., Lehoczky, J.P., Sha, L., and Shin, K.G., “On Task Schedulability in Real-Time Control Systems,” *Proceedings of the IEEE Real-Time Systems Symposium*, December 1997.

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

[40] Steere, D., A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole, “A Feedback-driven Proportion Allocator for Real-Rate Scheduling,” *Proceedings of the Symposium of Operating Systems Design and Implementation*, 1999.

[41] 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*, Dec. 1996, pp. 288-299.

[42] Sun, J., and Liu, J.W.S., “Bounding Completion Times of Jobs with Arbitrary Release Times and Variable Execution Times,” *Proceedings of IEEE Real-Time System Symposium*, December 1996.

[43] Tia, T.-S., Deng, Z., Shankar, M., Storch, M., Sun, J., Wu, L.-C., and Liu, J.W.-S., “Probabilistic Performance Guarantee for Real-Time Tasks with Varying Computation Times,” *Proceedings of IEEE Real-Time Technology and Applications Symposium*, Chicago, Illinois, January 1995.