Real-Rate Scheduling

Ashvin Goel
University of Toronto

Jonathan Walpole
OGI@OHSU, Portland

Molly Shor
Oregon State University, Corvallis

***Abstract*— Traditionally, real-time scheduling mechanisms have been used to provide predictable scheduling latency but these mechanisms are difficult to use in general-purpose operating systems (OSs) because they require precise specification of thread requirements in terms of low-level resources such as CPU cycles. In a general-purpose environment such a specification may not be statically available. In this paper, we present the design, implementation and evaluation of a novel feedback-based *real-rate* scheduler that automatically infers thread requirements and thus makes it easier to use real-time scheduling mechanisms in general-purpose OSs. The real-rate controller uses thread-specified *time-stamps* that indicate a thread's progress to estimate resource requirements. The goal of the controller is to regulate the overallocation of resources and the delay experienced by a thread. It meets these goals by using gain compensation and by choosing an appropriate sampling period for the controller that depends only on the granularity of thread time-stamps. A key benefit of the real-rate approach is that it can be easily applied in a general-purpose environment across different applications because the controller does not require any tuning.**

I. Introduction

Traditionally, real-time scheduling mechanisms such as priority schemes [7], [11] and reservation-based schemes [10] have been used to provide predictable and low scheduling latency. Priority schemes assume that the CPU needs of each thread are known and the highest priority thread voluntarily yields the CPU. If the highest-priority thread does not voluntarily yield the CPU such schemes can cause starvation. Reservation-based schemes such as proportion-period scheduling (PPS) avoid the problem of starvation. For example, PPS provides temporal protection to threads by allocating a fixed proportion of the CPU at each thread period. However, similar to priority-based schemes, PPS assumes that the CPU requirements of threads are known ahead of time. Hence threads must specify their proportion and their period to the scheduler before the scheduler accepts them.

Our goal is to support time-sensitive applications, such as interactive media and video surveillance applications, in a general-purpose environment. In such an environment, there are several reasons why the resource requirements of threads may not be known statically. First, these requirements depend on the processor speed. For example, a real-time thread will generally require a smaller proportion of the CPU on a faster processor. Second, the resource needs are often data dependent. For example, video encoding and decoding times of variable-bit rate (VBR) streams such as MPEG depend on the size and the type of frames. Larger size frames take longer encoding and decoding times because more data is accessed. Also, decoding times depend on whether frames are encoded independently (I frames in MPEG) or differentially (B and P frames). For example, Figure 1 shows the decoding times of frames for two different MPEG movies (the first is a clip from a full feature Hollywood movie while the second is a short dance video) played for approximately 200 and 20 seconds respectively. The decoding times vary considerably based on the complexity of the scene and the type of frames. Finally, given that memory access times are significantly larger than cache access times, a thread's resource needs change dramatically based on the mix of applications running on the system, which affects cache pollution. With all these effects, resource specification becomes a complex problem. Hence, although proportion-period scheduling provides temporal protection and fine-grained control over resource allocation, it has not been widely accepted for general purpose OSs.

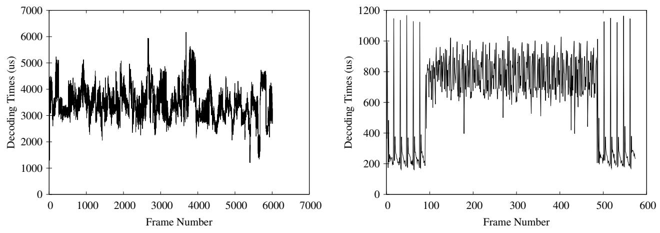


Fig. 1. Decoding times of MPEG frames can vary dramatically over short intervals of time.

To use proportion-period scheduling in such an environment, a method of dynamically estimating the resource needs of time-sensitive threads is needed, so that their timing requirements can be met. The key insight in this work is that from a thread's point of view, there is a basic difference in its timing needs and its resource needs. Threads are aware of their timing needs. For example, a video decoding thread knows how often it needs to process data. However, as explained above, this thread does not know its resource needs. Consequently, a method for automatically determining a mapping between a thread's timing needs and its resource needs is required. Such a mapping changes over time and hence has to be determined dynamically.

In this paper, we present the design and implementation of a novel CPU scheduler, called a *real-rate* scheduler, that uses feedback to dynamically determine a thread's resource needs from its "specified" timing needs and its observed timing behavior, and then automatically specifies these resource needs to a proportion-period scheduler. In our application model, a thread specifies its timing needs to the real-rate scheduler with *time-stamps* that indicate a thread's desired rate of *progress*. Ideally, the scheduler should ensure that every thread maintains its desired rate of progress towards completing its work. Allocating more CPU than is needed will be wasted, whereas allocating less than is needed will delay the thread. In essence, the real-rate solution monitors thread time-stamps to measure thread progress and increases or decreases the allocation of CPU to those threads as needed.

This paper shows that our software system can be modeled and the real-rate controls designed using gain compensation in a linear controller. The goal of the controller is to regulate the overallocation of resources and the delay experienced by a thread. Overallocation wastes resources, while delay slows down a thread's progress. These two goals are conflicting since precise allocation can delay a thread when its resource requirements increase suddenly. We use simulations to tune the controller and select the "best" parameters to make delay small while keeping overallocation within a certain bound. To do so, we show that the controller sampling period must be tuned based on the granularity of thread time-stamps that indicate progress.

The novelty of this work is in the application of control-based scheduling to meet the timing requirements of systems software without requiring resource specification. The real-rate controller is a reasonably simple linear control design but it allows satisfying requirements such as limiting overallocation to a certain range and reducing worst-case delay for threads whose resource needs can be very different and whose needs can vary dramatically over time. This simple real-rate control design not only reduces controller overhead but a key benefit of our approach is that it can be easily applied in a general-purpose environment because the controller does not require any tuning.

The next section provides an overview of the real-rate scheduler. Then Section III describes the design of the real-rate scheduler and Section IV presents our evaluation. Section V describes related work in scheduling. Finally, Section VI concludes by justifying our claims about the benefits of our feedback scheduling approach.

II. Scheduler Overview

Our feedback-based real-rate scheduling solution assigns proportions to threads dynamically and automatically as the resource requirements of threads change over time based on monitoring each thread's progress. The desired behavior is that a thread progress at the same rate as real time. We measure thread progress by observing the input/output (I/O) and the inter-process communication (IPC) events performed by a thread because such events capture the timing requirements imposed by the external world. In particular, the scheduler requires that the data items transferred as part of I/O and IPC events are *time-stamped.* These time-stamps capture the thread's progress, i.e., the difference between time stamps on two successive I/O items represents the time that *should* expire between their successive consumption (input) or production (output) if the system is running at the desired or specified rate. For example, a thread can time-stamp packets in a flow to indicate the progress rate desired by the thread. Time-stamps not only specify the desired progress rate but they also allow us to monitor the actual timing behavior of the running system and to compare it with the desired behavior. Note that time-stamps are assigned by the thread, either prior to run time, for example, in the case of a video stream that has been prepared and stored for later playout, or at run time, for example, in the case of a video stream that is being captured from a live source. Details about time-stamp assignment are presented in Section III-A.

Each time-stamp on a data item is a static value. The real-rate scheduler reads these values periodically and observes them "progressing" in real-world time. The basic feedback goal of the *real-rate* scheduler is to use these time-stamps to assign proportions to threads so that the rate of progress of time-stamps matches real-time. Given this goal, an accurate and responsive feedback controller will limit the instantaneous rate-mismatch between the thread's notion of time (difference in time-stamp values) and real-time (difference in clock values). In addition, the accumulated mismatch over time between the two quantities is a measure of delay introduced in a real-rate thread and the controller can be tuned to reduce this value.

A. Notation and Definitions

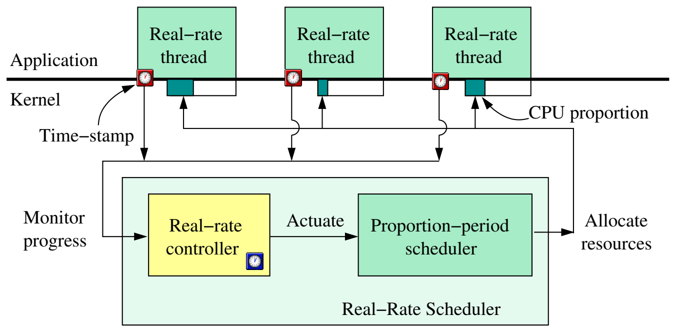
Similar to real-time systems, a real-rate system consists of

- Threads that are a stream of jobs.
- A job processes an I/O event, which we will refer to as a (data) packet.
- Temporal constraints are associated with packets and not with jobs. A job "inherits" the constraints of the packet that it is serving.
- The temporal constraints are in the form of time-stamps and not deadlines. The time-stamps are relative and only meaningful when compared with other time-stamps.

The *real rate* of a thread r_i is defined as the interval between time-stamps monitored unit sampling time interval apart. Based on this definition, when the real-rate is less than unity, the thread progresses slower than real-time and vice-versa. The goal of the real-rate mechanism is to maintain a *constant unit* real rate of progress. When this goal is met, the time-stamps of a thread will progress at the same rate as real-time, which reduces delay at a thread due to a rate mismatch. The key notation used in this paper is shown below:

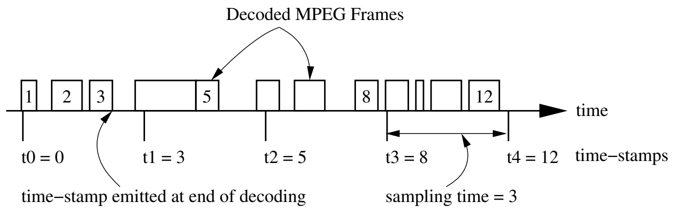
T_s	Sampling time interval
t_i	Thread time-stamp monitored at sampling time i
$r_i = (t_{i+1} - t_i)/T_s$	Real rate of the thread between sampling time i and $i+1$
$d_i = iT_s - t_i$	Thread delay at sampling time i
T_g	Granularity of time-stamps
$Q = T_g/T_S$	Quantization of time-stamps

Real-rate control tuning has two sub-goals. First, it keeps the controller's allocation overshoot (described in more detail in Section III-F), which leads to wasted resources, within a fixed bound. Second, the parameters are tuned to ensure that the



The real-rate controller monitors the rate of progress of real-rate threads using time-stamps, and calculates new proportions based on the results. It actuates these values using a standard proportion-period scheduler.

Fig. 2. Block diagram of the real-rate scheduler.



The decoder indicates its progress to the controller by attaching time-stamp to frames after decoding them. The controller reads these time-stamps periodically even though time-stamps can be issued aperiodically.

Fig. 3. A multimedia decoder time-stamps frames to indicate progress.

delay d_i at each thread converges to zero and maximum delay is kept small. These goals are conflicting and the controller tuning process attempts to find an appropriate balance.

There are two main differences between real-rate scheduling and other feedback-based real-time scheduling schemes such as FC-EDF [9]. First, real-rate scheduling does not *explicitly* have a notion of deadlines or missed deadlines and, hence, does not aim to reduce the number of missed deadlines. Instead, the goal of real-rate scheduling is to ensure that time-stamps progress in real time (to control the difference between consecutive finishing times) and the controller's performance is judged by the maximum time-stamp delay that can be introduced (roughly, the maximum number of instantaneously missed deadlines). Second, previous feedback approaches [9], [2] measure the processing time of each job. With real-rate scheduling, time-stamps are monitored a fixed sampling period apart, immaterial of the number of time-stamps (on packets or jobs) that occur in between. With this approach, monitoring is periodic while jobs may or may not be periodic. A constant sampling-period controller limits monitoring and control overhead when the job rate is high.

B. System Architecture

Figure 2 shows the high-level architecture of the real-rate scheduler. The scheduler consists of two main components: a proportion-period scheduler and a real-rate controller. The proportion-period scheduler is a hard reservation scheduler that ensures that threads receive their assigned proportion of the CPU during their period. The controller periodically monitors the progress made by these threads, which we call *real-rate threads*, and adjusts each thread's proportion automatically. Note that the diagram resembles a classic closed-loop, or a feedback controlled system.

The *proportion-period scheduler* (PPS) in our architecture allocates CPU to both real-rate threads that have a visible metric of progress but do not have a known proportion and *reserved* threads that have a known proportion and period. PPS is a specific hard reservation scheduler based on EDF. However, any other reservation scheduler will suffice for the real-rate approach. The next section describes the main component in our architecture, the real-rate controller.

III. Real-Rate Controller

The real-rate controller uses a monitoring component to measure progress and assigns proportion to real-rate threads to ensure that they make their desired progress. The following sections describe the design of the real-rate controller in detail, including the monitoring component, the proportion control mechanism, the issue of time-stamp granularity, the choice of sampling period, the method for handling resource overloading and the method for tuning the parameters of the real-rate controller.

A. Monitoring Progress

The novelty of the real-rate approach lies in the estimation of thread progress as a means of controlling CPU allocation. The real-rate controller requires that a thread specify its timing requirements to the controller using time-stamps that indicates its desired rate of progress.

Consider a multimedia decoder that must decode frames in real-time so that the frames can be displayed with a small amount of buffering and low delay. Figure 3 shows that the decoder time-stamps each frame and the controller uses these time-stamps to monitor the progress of the decoder. Time-stamps can be assigned after each frame or at a finer granularity than a frame such as an MPEG macroblock or a coarser granularity such as a group of pictures.

B. Proportion Control Mechanism

To design a control law for real-rate threads, we first need to understand the model of the system or the system's dynamics. Figure 4 shows the system and control variables on a time-line. This figure shows sampling instants i and $i+1$ that are T_s time apart. At these sampling instants, the time-stamps that are monitored have values t_i and t_{i+1}. The proportion assigned to the thread between the two sampling instants is p_i. With these definitions, $t_{i+1} - t_i$ is the progress made by the thread T_s real-time interval apart. Hence, the real rate of the thread r_i between the two sampling instants is $(t_{i+1} - t_i)/T_s$.

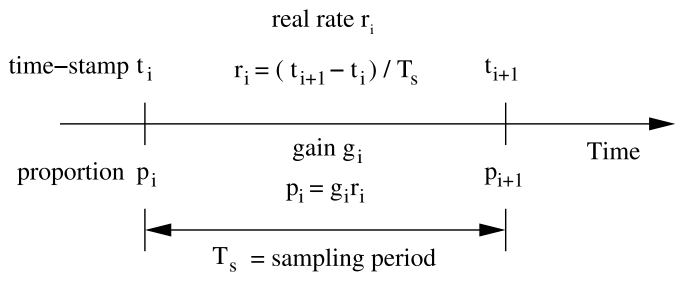


Fig. 4. Time-line showing system and control variables.

1) System Model: The key assumption in our feedback-based real-rate model is that a real-rate thread makes progress that is approximately proportional to the CPU allocated to it. We define this relationship between the real-rate of a thread and the proportion of CPU allocated to it as $r_i = (1/g_i)p_i$. We assume that g_i can vary instantaneously over time but within some range and its average value varies slowly over time. This assumption is motivated by our experience with multimedia applications where decoding times can vary instantaneously but within a range of 2-5 times its current value as shown in Figure 1.

We will call the factor g_i the *proportional gain* of the thread. From the definitions above, the system model can be defined as follows: The real-rate $r_i = p_i/g_i$ but it is also $r_i = (t_{i+1} - t_i)/T_s$. Rearranging the terms and replacing r_i in the two equations produces the system model shown in Equation 1 below.

$$t_{i+1} = t_i + T_s p_i/g_i \tag{1}$$

This system model equation defines the value of the thread time-stamp at sampling instant $i + 1$, given the time-stamp at sampling instant i and the proportion p_i assigned at that time. This system model is a time-variant equation because the proportional gain g_i is not a constant, but an unknown variable whose value depends on time i. Next, this equation is used to derive the real-rate control law.

2) Control Law: The control law should be designed so that the system goals can be met and the disturbance in the system model can be rejected. The disturbance in our system model is g_i and, ideally, it can be canceled by selecting our control law to be $p_i = g_i$. In this case, time-stamps would increase by T_s every sampling time in Equation 1, which would ensure that the real-rate is 1. However, we will see shortly that g_i is not known at sampling instance i. To design the control law, we will first assume that g_i is available, then design an estimator for g_i to replace its exact value in the control law.

The *closed-loop* model of a control system is the dynamics of the combination of the system model and the control law that drives the system. To derive a *linear, time-invariant* closed-loop model, we use *gain compensation*, where the control law inverts the disturbance g_i. Here, we use this standard technique to the system model in Equation 1 to derive the control law for the real-rate controller as shown in Equation 2.

$$p_i = (g_i/T_s)[t_{i+1}^{des} - t_i - (\alpha - 1)(t_i - t_i^{des})] \tag{2}$$

The proportion p_i is the proportion assigned to the thread at sampling instant i until sampling instant $i+1$. The variable t_i is the time-stamp of the thread, monitored at sampling instant i and t_i^{des} is the desired value of the time-stamp. The term $t_i - t_i^{des}$ is an error term since it is the difference between the actual output and the desired output of the system. To obtain t_i^{des} and t_{i+1}^{des}, note that the goal of the controller is to reduce the build up of delay d_i at each real-rate thread. Delay d_i will not build up at sampling instant i if the time-stamp value t_i is iT_s since the thread will have made real-rate progress equal to real-time. Hence, $t_i^{des} = iT_s$ (if t_0 is non-zero, then it should be subtracted from all time-stamps). Having derived the desired values of the time-stamps, control Equation 2 can be simplified to finally yield the real-rate control Equation 3.

$$p_i \quad = \quad g_i[1 + \alpha(i - t_i/T_s)] \tag{3}$$

This control equation shows that the proportion p_i is increased (or decreased) when the observed time-stamp t_i is less (or greater) than iT_s. When it is less, delay d_i, which is equal to $iT_s - t_i$, is being accumulated and hence the allocation of the thread should be increased to speed up the thread. Note that due to gain compensation, the time-variant proportional gain term g_i is present in the control equation also. Below, we show how it can be estimated.

Analysis of Equation 2 shows that the gain parameter α should lie between 0 and 2 for stable controller behavior. The parameter α determines the responsiveness of this control equation. For example, larger values of α make the control equation more responsive because the proportion changes in larger steps for the same error term $t_i - t_i^{des}$. Section IV discusses this issue further.

3) System Parameter Estimation: The control law in Equation 3 assumes that the proportional gain g_i is known. Recall from the previous section that we assumed that the relationship between the proportion of CPU allocated to a thread and its real-rate progress is linear, and defined as $r_i = (1/g_i)p_i$. Hence g_i is equal to p_i/r_i. Unfortunately, p_i is assigned at time i and hence r_i is only known at $i+1$ (see Figure 4) and hence g_i can only be known at time $i+1$ (and not at time i). In this sense, Equation 3 is non-causal because it depends on values in the future, and hence cannot be directly solved.

To get around non-causality, the real-rate control model assumes that the average value of the proportional gain g_i changes slowly over time, or $g_i \approx g_{i-1}$. Hence, g_i can be approximated as p_{i-1}/r_{i-1}, or $p_{i-1}T_s/(t_i - t_{i-1})$. Both these values are known at time i. In the absence of a precise model of the dynamic nature of g_i, we use a simple low-pass filter to estimate g_i based on past values of p and r. This approach reduces noise in the estimation at the expense of being less responsive to changes in g_i. In particular, $\hat{g}_i$ the estimate of g_i is derived as: $\widehat{g_i} = (1-\beta)\widehat{g_{i-1}} + \beta T_s p_{i-1}/(t_i - t_{i-1})$. With this parameter estimation technique, the real-rate estimation and control laws can be expressed with Equations 4 and 5 shown below. These laws together constitute the real-rate controller.

$$\widehat{g_i} = (1-\beta)\widehat{g_{i-1}} + \beta T_s p_{i-1}/(t_i - t_{i-1}) \quad (4)$$
$$p_i = \widehat{g_i}[1 + \alpha(i - t_i/T_s)] \quad (5)$$

We choose the initial proportion gain $\widehat{g_0}$ at the start time to be 0. However, the proportion p_0 at the start time has to be a non-zero value to get the estimation started. Currently, we choose p_0 to be a small system-defined allocation value, 0.1% of total CPU. This choice can increase startup delay since the proportion allocated will be smaller than needed to make unit real-rate progress. However, our experiments show that this is not a serious problem because the proportion tends to increase rapidly (almost exponentially) in the beginning and hence the estimate converges quickly. The parameter β is chosen based on the amount of noise in the sampling of the time-stamp t_i. We describe how the α and β parameters are chosen in more detail in Section IV.

There are two boundary conditions in the controller. First, if the time-stamp t_i does not increase, then the thread has made no visible progress in the last sampling period and hence $\widehat{g_i}$ the control state and p_i the output state are not changed. Second, when p_i becomes very small, the thread will not be able to run and its progress cannot be measured correctly. Currently, the minimum proportion a thread is assigned at any time is the same as the system-defined starting proportion value, or 0.1% of the CPU.

C. Granularity of Time-stamps

The granularity of time-stamps generated by a real-rate thread has a significant impact on the performance of the real-rate controller. In particular, the controller cannot sample progress effectively at a sampling period finer than the time-stamp granularity. In general, the time-stamp granularity depends on application semantics. For example, a video encoding thread may time-stamp every frame, in which case the time-stamp granularity is 33.3 ms for a 30 frames per second video. On the other hand, a CD quality audio encoder that generates 44100 samples per second may time-stamp every 100th sample (time-stamps on every sample would have high overhead) so the time-stamp granularity would be 2.27 ms.

To simplify the job of the controller and to tune the estimator and control law parameters, the real-rate controller requires real-rate threads to specify the approximate granularity of the time-stamps they will be generating. This thread parameter is specified to the controller as part of the monitoring interface described in Section III-A. In the future, we plan to determine time-stamp granularity based on observing the progress of time-stamps.

D. Choice of Sampling Period

The sampling period of the controller determines its responsiveness. Finer-grained sampling allows the controller to be more responsive although it increases system overhead. However, the controller should sample no faster than the granularity of the time-stamps that indicate thread progress or else the progress signal gets heavily quantized, which introduces error and potential instability in the feedback system. We define the term time-stamp *quantization* Q as T_g/T_s, where T_g denotes the time-stamp granularity and T_s is the sampling period. If time-stamps are large as compared to the sampling period, then they have a large quantization and vice-versa. We use experiments, described in Section IV, to determine the optimal sampling period given the time-stamp granularity so that the real-rate controller can achieve one of its goals of reducing delay. Since the time-stamp granularity T_g is specified by threads, the optimal sampling period can be determined using the optimal quantization value. In addition to time-stamp granularity, the thread period also helps determine the sampling period [3].

E. Control Mechanism During Overload

When enough CPU resources are not available the proportion p_i that is calculated at step i will be higher than the amount of available resources. The real-rate controller uses a simple saturation model for the actuator where it assigns as much resources as are available. If the resource requirements are temporarily high then the thread will be delayed temporarily after which the controller will allocate resources to speed up the thread appropriately. To reduce unnecessary saturation where a large control effort leads to saturation even though enough resources are available, we limit control effort by tuning the value of the α parameter in Equation 5 to a small value as described in more detail in Section IV-A.

F. Tuning the Real-Rate Controller

This section describes qualitatively how the real-rate feedback controller is tuned to achieve its goal. The goal of the real-rate controller is to reduce the maximum *delay* introduced at each thread while keeping the proportion allocation *overshoot* within a fixed range (the overshoot for a step input is the maximum amount the output exceeds its final goal as a percentage of the step size). The delay d_i experienced by the thread is $iT_s - t_i$ in Equation 5. This value should be minimized or else adaptive applications on realizing that they are being delayed may unnecessarily reduce their rate requirements. Second, larger overshoot will cause faster resource overloading than needed, which will again lead to adaptive applications unnecessarily reducing their rate requirements.

The choice of the α and β parameters in Equations 4 and 5 affects the delay and the overshoot experienced by the thread. Increasing α and β parameters decreases the additional delay but also increases the overshoot. In this paper, we use simulation to understand the trade-off between these goals and tune α and β so that delay is kept small while at the same time overshoot is limited to a reasonable value such as 15-30 percent, used in standard control. Our simulation also provides intuition and a systematic methodology for tuning the controller parameters.

IV. Evaluation

We have implemented the real-rate scheduler as a combination of the proportion-period CPU scheduler and the real-rate controller in the Linux 2.4.20 kernel [4], [3]. In this section, we use simulation to evaluate and characterize the performance of our prototype controller. We examine the responsiveness of the controller as a real-rate thread's resource requirements change over time and explain how the controller parameters should be tuned. Next we use the decoding times of real MPEG files to evaluate how well the controller can meet its goals.

We analyze the real-rate mechanism in Equations 4 and 5 with a step input, where the proportional gain g_i or the processing needs of the thread are increased instantaneously from g_{min} to g_{max} to simulate instantaneous changes in the mapping between resource requirements and real-rate progress. The simulation measures the delay and allocation overshoot introduced by the controller, and hence indicates how well the control law together with the estimator tracks g_i. We call the ratio $G = g_{max}/g_{min}$ the step ratio. As the step ratio G increases, the delay and overshoot is likely to increase because the controller takes longer to stabilize to the larger change in the proportional gain. For tracking the proportional gain, the monitor provides time-stamps whose granularity introduces a "monitor" disturbance which makes tracking harder. To understand the effect of this disturbance, the experiments below vary time-stamp granularity T_g while inducing the proportional gain step.

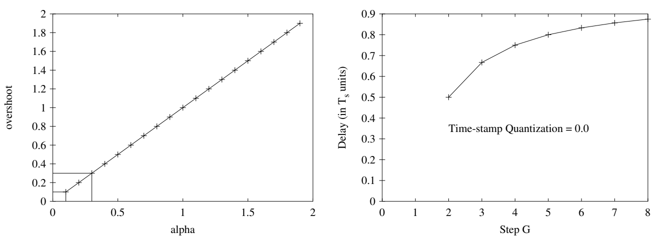
To summarize, the real-rate mechanism has three parameters, α, β and p_0 that need to be tuned to meet the controller's goals of low delay and overshoot. The variables in the experiment are the step ratio G (system disturbance) and the time-stamp granularity T_g (monitor disturbance). The following sections explain how the parameters α and β should be chosen. Due to space limitation, the effect of the choice of p_0 on delay is described elsewhere [3].

A. Choice of α

The first set of simulation experiments show how the α parameter of the control law in Equation 5 should be chosen. To do so, we first show how the choice of α values affects delay and overshoot. Here, time-stamps are assumed to be continuous, i.e., quantization Q is zero or there are no quantization issues due to coarse granularity time-stamps. The next section shows that the optimal value of the estimator parameter β is 1 under this situation. Hence, β is chosen to be 1 in these experiments. The next section relaxes this assumption.

Figure 5(a) shows the overshoot as α is changed from 0 to 2, where it must lie for stability, when $G = 2$. As expected, larger values of α increase the proportion overshoot because the controller becomes more responsive to variations in current delay. A large overshoot can result in overload or saturation so from now on we will choose α values so that overshoot is limited to within a certain maximum range from 10-30% as shown with the lines at the bottom-left of Figure 5(a).

Figure 5(b) shows the maximum delay introduced by the controller for different values of the step ratio G. Increasing



(a) Relation between overshoot and gain parameter α.

(b) Relation between delay and G when $\alpha = 0.15$.

Fig. 5. Choice of control parameter α.

the value of G implies that the controller has to adapt to larger changes in progress needs, hence the increased delay. Note that the absolute value of the proportion requirements does not affect delay. For example, the increased delay will be the same if the proportion requirements increase by a factor of 2 from say 10% to 20% versus from 20% to 40%. The real-rate controller possesses this desirable property because it explicitly estimates the value of g_i in Equation 4, which it then uses as a parameter for the control law in Equation 5. A linear control law that does not estimate g_i but uses a constant parameter would not possess this property.

In Figure 5(b), the largest possible value of α was chosen to reduce the worst-case delay, while keeping overshoot to less than 15%. In this case, when there is no quantization and β is chosen to be 1, the α value was 0.15 in all cases. Hence, from now on, we will use an α value close to 0.15 because it limits overshoot. In our implementation, α is chosen to be a negative power of 2 for integer arithmetic in the kernel and thus is 1/8 or 0.125.

B. Choice of β

The previous section made the unrealistic assumption that the time-stamp quantization Q is zero. In practice, real-rate threads perform work and hence progress at a certain granularity. This granularity is captured in the value of Q, which is the ratio of the time-stamp granularity and the sampling period. Quantization affects the behavior of the estimator in Equation 4. Generally, with increasing quantization, the estimator parameter β must be made smaller to reduce disturbance due to quantization. However, reducing β also reduces the response of the estimator to real changes in the value the proportional gain g_i which is required by the control law.

To understand how β should be chosen for different time-stamp granularities, we performed the previous experiments for measuring delay again. This time α was fixed to $1/8$ while β was varied. In addition, the quantization Q was varied. We performed exhaustive experiments with different values of β and Q. The details of the experiments and the intuition behind the choice of β are not presented here due to space constraints. However, the final results are shown in in Figure 6. Here the x-axis shows the quantization in time-stamps and the y-axis shows the delay in terms of any arbitrary time-stamp

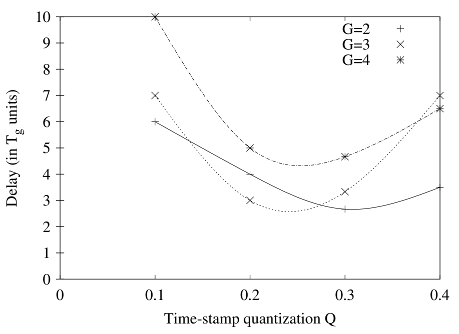


Fig. 6. For minimizing delay, optimal quantization value lies between 0.2 and 0.3.

granularity. The different lines show the results for different values of the step ratio G. The values of β are not shown in the graph. They are roughly 0.75 when $Q = 0.1$, 0.5 when Q lies between 0.2 and 0.3 and 0.25 when $Q = 0.4$. These graphs show that when quantization Q is small and equal to 0.1 (sampling is done once every 10 samples) the delay is high because the controller is unresponsive. However when Q is large and equal to 0.4 then the delay is high because time-stamp granularity introduces disturbance. This figure shows that the optimal quantization $Q_{optimal}$ lies between 0.2 and 0.3 since then delay in minimal. Based on this figure, the sampling period is optimal when $T_s = T_g/Q_{optimal}$, or the sampling period T_s should be 3 to 5 times the time-stamp granularity. At this operating point, the β parameter should be chosen to be a fixed value of 0.5. The next section discusses the implications of these numbers.

C. Discussion

The previous sections have shown that the real-rate controller can be easily tuned for reducing the delay of real-rate threads. The sampling period of the controller should be 3-5 times the time-stamp granularity specified by the thread. The parameters α and β in Equations 4 and 5 can be fixed at 1/8 and 1/2. To start the equations, $\widehat{g_0}$ is set to 0 and p_0 can be chosen arbitrarily but a correct choice reduces startup delay.

Figure 6 summarizes the behavior of a real-rate controller. If the instantaneous change in g_i in an application is a factor of 2 ($G = 2$), then the worst-case delay lies between 2-3 times the time-stamp granularity. We have made measurements of the value of G for the video streams shown in Figure 1 and for other videos [13] and its value was found to lie between 2 and 5. Assuming $G = 2$, the controller can introduce 66-100 ms of delay for a video application, if the video data is time-stamped 33.3 ms apart. To reduce this delay, data has to be time-stamped at a finer granularity. For example, sub-frames of each frame could be time-stamped at a finer-granularity. A CD audio application with time-stamps 2.27 ms apart (time-stamps every 100 samples in a 44KHz signal) will experience 5-6 ms of worst-case delay if the maximum variation in G is 2. In the future, we plan to measure the value of G rigorously for other types of real-rate applications.

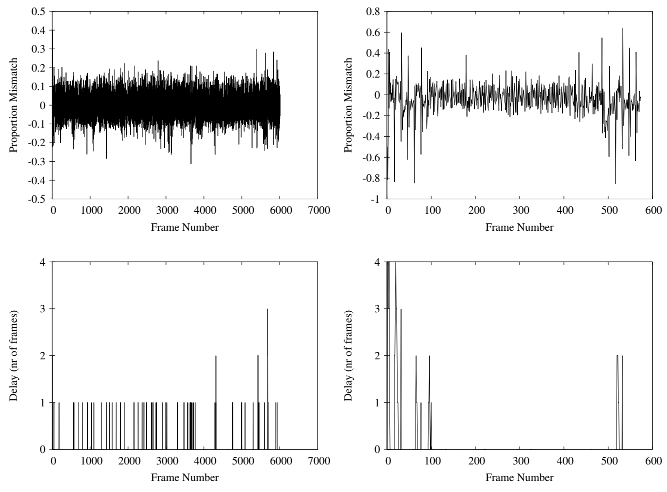


Fig. 7. Controller simulation with decoding times of two MPEG streams.

D. Controller Evaluation for MPEG Decoding

In this section, we evaluate how well the controller meets its goals by simulating it with different MPEG video streams. Here we describe our results for two variable bit-rate videos: the first is a 200 second action clip from the movie Charlie's Angels while the second is a short dance video. The actual decoding times of each frame are used to simulate the proportional gain in the controller. The output of the controller is the proportion assignment. We measure how well the controller meets its goals of 1) keeping the proportion overshoot within a bound and 2) and keeping the thread delay to a small value. Based, on Figure 6, the controller was run with quantization of 0.25 or the sampling time was 4 times the granularity of the time-stamps where the time-stamp granularity was a frame duration.

Figure 7 shows the results of our evaluation. The graphs on the left show the results for the Angels video while those on the left are for the dance video. The graphs on the top show the mismatch in the proportion and the decoding times when the decoding times are as shown in Figure 1. These graphs show that as per our goal, the controller is able to limit the mismatch on the positive side or the overshoot to less than 15-30%. Occasionally, the overshoot rises above 30% in the second video because G, the instantaneous change in the proportional gain, rises above 5 at these times while our controller design assumed a worst case of 4. Note that the average proportion allocation mismatch is close to zero so our controller does *not* over-allocate resources in the long term.

The graphs at the bottom of Figure 7 show the delay introduced by the controller. Figure 6 shows that the worst-case delay of the controller should be no more than 5 times the time-stamp granularity or 5 frames with the optimal sampling period. The bottom graphs confirm this result and show that if a multimedia decoder keeps a buffer of 4 frames then our controller will deliver all frames on time. Even with one or two frames, the missed "deadlines" are reduced significantly.

V. Related Work

Our feedback scheduling solution is similar to rate-based scheduling proposed by Jeffay [5]. However, with rate-based scheduling, applications must specify WCET and an upper bound on response time. In contrast, our system provides dynamic estimation and adjustment of rate parameters, and only requires that the progress metric be specified.

Stankovic [9], [8] uses a feedback controlled earliest-deadline first (FC-EDF) algorithm to adjust allocations of threads in order to reduce the thread's missed deadlines. While this approach uses missed deadlines as an input to the feedback controller, our feedback approach uses time-stamps and does not need missed deadlines to monitor progress. Abeni [1] uses feedback in a reservation-based scheduler to remove the need for specifying WCET in the task parameters. Later, Abeni [2] developed an accurate mathematical model of a CPU reservation, and formally analyzed the performance of their closed loop scheduler using control theoretic tools.

There are several differences between these approaches and real-rate scheduling. Real-rate scheduling does not *explicitly* have a notion of deadlines or missed deadlines. The goal of real-rate scheduling is to ensure that time-stamps as specified by a thread, for example in the packets of a flow, progress in real time. As a result, instead of reducing the number of missed deadlines, the controller's performance is judged by the worst-case time-stamp delay that can be introduced. Second, the previous approaches measure the processing time of each job. With real-rate scheduling, time-stamps are monitored a fixed sampling period apart, immaterial of the number of time-stamps (on packets or jobs) that occur in between. Finally, given our focus on scheduling time-sensitive applications in a general-purpose environment, a key goal of this work is to simplify the choice of feedback parameters as much as possible.

Li [6] uses a standard PID feedback controller for resource allocation. In the past, we had also used a PID controller for real-rate scheduling [12]. The problem we noticed with the PID controller is that it could be tuned to perform well around a given fixed point (say 50% allocation) but the same controller did not perform well when the resource needs were vastly different (say 5%). The reason is that it does not allow explicit estimation of resource requirements as we do with the combination of Equations 4 and 5.

VI. Conclusions

The key benefit of the real-rate controller is that applications do not have to specify their scheduling requirements in resource specific terms such as CPU cycles. Instead, applications use an application-specific notion of progress expressed as timing information. The controller uses feedback control to automatically derive the resource requirements based on this timing information.

This paper shows that software systems can be modeled and controllers designed for them using a standard gain compensation technique. The goal of the real-rate controller is to limit proportion overshoot and reduce the worst-case delay experienced by a thread. Our analysis based on simulations has shown how such delay can be reduced and quantified by using an appropriate sampling period for the controller given the granularity of time-stamps generated by the thread. In general, finer-grained time-stamps help to reduce scheduling delays.

While several researchers have used such control analysis for scheduling, its applicability in generic OS environments has been an open issue since the behavior of software systems is harder to characterize under a variable mix of applications compared to a dedicated control system. Our real-rate approach shows that software systems can be modeled and analyzed, and further, controllers using standard control techniques can be designed for them using very generic assumptions about the operating environment.

References

[1] Luca Abeni and Giorgio Buttazzo. Adaptive bandwidth reservation for multimedia computing. In *Proceedings of the IEEE Real-Time Computing Systems and Applications*, Hong Kong, December 1999.

[2] Luca Abeni, Luigi Palopoli, Giuseppe Lipari, and Jonathan Walpole. Analysis of a reservation-based feedback scheduler. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 71–80, December 2002.

[3] Ashvin Goel. *Operating System Support for Low-Latency Streaming*. PhD thesis, OGI School of Science and Engineering, OHSU, Portland, July 2003.

[4] Ashvin Goel, Luca Abeni, Charles Krasic, Jim Snow, and Jonathan Walpole. Supporting time-sensitive applications on a commodity OS. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, pages 165–180, December 2002.

[5] Kevin Jeffay and David Bennett. A rate-based execution abstraction for multimedia computing. In *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pages 67–77, April 1995.

[6] Baochun Li and Klara Nahrstedt. A control-based middleware framework for quality of service adaptations. *IEEE Journal on Selected Areas in Communications, Special Issue on Service Enabling Platforms*, June 1999.

[7] C. L. Liu and J. Layland. Scheduling algorithm for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, Jan 1973.

[8] C. Lu, J. A. Stankovic, T. F. Abdelzaher, G. Tao, S. H. Son, and M. Marley. Performance specifications and metrics for adaptive real-time systems. In *Proceedings of the 21th IEEE Real-Time Systems Symposium*, Orlando, FL, December 2000.

[9] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son. Design and evaluation of a feedback control edf scheduling algorithm. In *Proceedings of the 20th IEEE Real-Time Systems Symposium*, Phoenix, AZ, December 1999.

[10] C. W. Mercer, S. Savage, and H. Tokuda. Processor capacity reserves: Operating system support for multimedia applications. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, pages 90–99, May 1994.

[11] Lui Sha, Raghunathan Rajkumar, and John Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1184, September 1990.

[12] David Steere, Ashvin Goel, Joshua Gruenberg, Dylan McNamee, Calton Pu, and Jonathan Walpole. A feedback-driven proportion allocator for real-rate scheduling. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, pages 145–158, February 1999.

[13] David Steere, Molly H. Shor, Ashvin Goel, Jonathan Walpole, and Calton Pu. Control and modeling issues in computer operating systems: Resource management for real-rate computer applications. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*, December 2000.