Supporting Rate-Based Processes in an Integrated System

Anna Povzner Caixue Lin
Scott A. Brandt
Computer Science Department, University of California, Santa Cruz
{apovzner,lcx,sbrandt}@soe.ucsc.edu

Abstract

As real-time applications are becoming common in general-purpose computing systems, scheduling solutions have been developed to support processes with a variety of different timeliness constraints in an integrated way. In this paper, we identify a separate category of soft real-time processes, called rate-based processes, and investigate the ways to add the support of this class of processes into an integrated system. We consider rate-based processes as a separate class of soft real-time processes because they do not have timing constraints in the form of the deadline. Instead, they have continuous processing requirements in terms of the constant rate. The goal of the efficient scheduling mechanism for rate-based processes as part of the integrated system is to provide good overall system performance while maintaining the required throughput of rate-based processes.

1 Introduction

We identify rate-based processes as a separate class of soft real-time processes, because they have different timeliness constraints. They do not have constraints in the form of deadlines, but they have continuous processing requirements. We currently consider rate-based processes as producer/consumer types of processes, where it is necessary for the producer to produce data objects at the rate at which they are consumed. Multimedia audio and DVD burner are examples of popular rate-based applications in general-purpose computer environments. For example, in the audio player application, audio frames should be generated continuously in the time it takes to play the same number of frames. Otherwise, the user may experience audio dropouts.

Some of the previous work identified rate-based processes as a separate class of processes and addressed the problem of their scheduling. However, the previous work does not address the problem of scheduling rate-based processes in an integrated system. In such systems, the scheduling solution should not only ensure the continuous processing requirements of rate-based processes, but also ensure good performance of other classes of processes while they are interacting with rate-based processes.

In this work, we investigate possible approaches to support rate-based processes in an integrated systems. In order to guarantee rate-based performance, we used worst-case reservations. Since actual resource usage may vary, we studied the ways to efficiently distribute unused resources to applications which occasionally need more resources than they have reserved. We defined several principles which we will use in our future work to to provide an efficient scheduling mechanism that ensures the required throughput of rate-based processes as well as good overall system performance.

2 Related work

A Feedback-driven Proportion Allocator [7] dynamically estimates the proportion and period needed by a particular job based on observations of its progress. Producer/consumer types of applications expose the buffer's fill level, size and the role of each thread (producer or consumer) to the system. Using this information, the scheduler estimates the progress of the process by monitoring the buffer's fill level, and increases or decreases the allocation of CPU to applications accordingly. The drawback of this approach is the incurred overhead of constant monitoring of buffer's fill level. In addition to incurring the lower overhead, our work aims to minimize the information exposed to the scheduler,

The producer/consumer process model is also used in the YARTOS kernel [1], which models process interactions as producer/consumer systems with a timing constraint on the rate at which the consumer must service the producer. This model, however, tries to solve a different problem: the i-th output of the producer must be consumed by the consumer before the $i+1$-th output is

produced. Our work focuses on the case when several data objects can be buffered.

In the rate-based execution model [2, 3] processes execute according to a general rate specification of x process executions every y time units. Processes specify their rate of progress in terms of the number of events they desire to process in an interval of specified duration. This is a different definition of rate-based processes from what we use.

SMART [5] determines the overall resource allocation for each task based on importance, and when each task is given its allocation based on urgency. To schedule audio player applications, deadlines are changed dynamically to the display time of the last audio sample in the buffer. Changing deadlines dynamically does not generate slack, and we aim to improve the overall system performance by re-distributing slack generated by rate-based processes.

3 Integrated system

The system we are using to extend with rate-based execution support is an integrated system that simultaneously supports hard real-time, soft real-time, and best-effort processes. The system is described in our previous work BACKSLASH [4] and is based on the Resource Allocation/Dispatching model and the Rate-Based Earliest Deadline (RBED) scheduler [6]. The system consists of a resource allocator and an EDF-based dispatcher. The resource allocator assigns the target rate of progress and period for each process in the system, based on the specific processing requirements of each process. The EDF-based dispatcher dispatches processes in EDF order but interrupts them via a programmable one-shot timer when they consume their worst-case execution time. In this system, hard real-time processes are guaranteed to receive their desired resource rate. Soft real-time processes receive their desired rate or less, depending on the availablility of resources. A minimum resource utilization is reserved to guarantee that no best-effort process is starved.

Critical applications typically use worst case resource reservation to guarantee their timeliness constraints. Because actual application execution time may vary, applications often use less resources than they have reserved, creating dynamic slack—reserved but unused resources. BACKSLASH provides efficient reclamation and redistribution of dynamic slack to processes that need more resources than they have reserved, which significantly improves the performance of both soft real-time and best-effort processes.

4 Scheduling as periodic tasks

4.1 Algorithm description

Our goal is to provide efficient scheduling that provides good overall system performance and guarantees continuous processing requirements for rate-based processes. We first present a straightforward approach that schedules rate-based processes as periodic tasks with worst-case resource reservations. This approach can be implemented with different choices of periods, allowing us to investigate how different periods affect the overall system performance.

In order to implement this approach, the scheduler does not need to be changed; rate-based tasks can be scheduled as a hard-real time tasks. Consider an audio application, where a rate-based task is a thread that reads audio frames from local storage, decodes and writes them to the buffer in the audio device. This thread is scheduled as follows. We choose any block size of n bytes which is less than half of the buffer. After the application decodes and writes n bytes to the buffer, it schedules the decoder thread as a hard real-time task, providing period p, assigned the time to play n bytes, and worst-case execution time $wcet$, assigned the time to decode n bytes. When the first period ends, the initially filled n bytes are consumed, but the scheduler guarantees that next n bytes are produced by this time. Thus, the algorithm ensures that the buffer always has the data for the audio device to consume. Note that the half of the buffer is the maximum value of n we can choose, because we need $2 * n$ bytes of unused space in the buffer in the worst case to pre-fill it with n bytes and then produce n bytes during the first period.

With this approach, the performance of rate-based processes is guaranteed by worst-case resource reservation. Since actual execution time for each period may vary, one would expect a rate-based task to generate dynamic slack which could be distributed to other tasks.

4.2 Performance evaluation

We used mpg123, an open source MPEG Audio Player, to experiment with the initial algorithm. This application uses a buffer size of 32 KB. We first investigated how the choice of different periods for scheduling a rate-based task affects the performance of a fixed set of soft real-time tasks. The workload consisted of two periodic soft real-time tasks SRT1 and SRT2, one CPU bound task, and one rate-based task, mpg123. In this workload, soft real-time tasks have normally distributed execution times with their server budget set to their average case execution times. SRT1 has a period of 23 ms and CPU utilization of 50%,

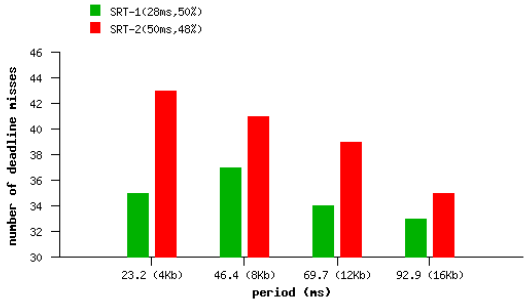

Figure 1: Number of deadline misses vs. decoder period

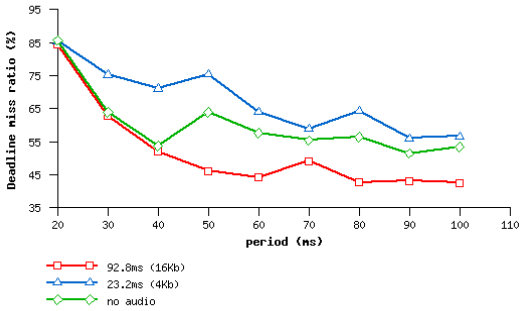

Figure 2: Deadline miss ratio

and SRT2 has a period of 50 ms and CPU utilization of 48%. The periods of a rate-based task are 23.2 ms (time to play 4 KB), 46.4 ms, 69.7 ms, and 92.9 ms (time to play 16KB).

Figure 1 shows the number of missed deadlines of soft real-time tasks for different periods of an audio application. The total number of missed deadlines of soft real-time tasks decreases with the growth of the rate-based task period. This is the opposite of what we would expect from a traditional hard real-time task. Traditional hard real-time processes with smaller periods donate slack earlier, more often, and with finer granularity, providing better soft real-time performance than from later, less frequently available, and larger granularity slack.

The behavior of rate-based processes is different because they have potentially unused buffer space which is bigger for smaller periods. If the period is assigned the time to consume half of the buffer, a rate-based application behaves similar to conventional hard real-time processes. The algorithm initially fills the half of the buffer before scheduling a rate-based process as a hard real-time task. If the other half of the buffer is filled earlier than predicted by worst case reservation, then the buffer becomes full and the task generates slack. Because the buffer is approximately half full in the beginning of each period, the slack is likely to be generated during each period when the required buffer size is filled earlier than predicted.

However, the behavior of rate-based tasks with smaller periods is different. Suppose the period p is assigned the time to play $1/8$ of the buffer, and the actual execution times are close to worst-case times. The algorithm initially fills $1/8$ of the buffer before scheduling a rate-based process as a hard real-time task, and then tries to fill $1/8$ of the buffer during each period. In this case, less than $1/4$ of a buffer is filled and the remaining part of the buffer is unused. If the actual execution time is less than worst-case time during some periods, $1/8$ of the buffer is filled earlier, but the slack is not generated—the remaining time is used to fill some of the unused space in the buffer.

We also investigated the effect of the smallest and the biggest period of a rate-based process on performance of soft real-time tasks with different periods. The workload consists one CPU bound task and one periodic soft real-time task, the period of which changes from 20 ms to 100 ms, with CPU utilization of 97%. We also used this workload with an audio application, with period 92.2 ms and 23.2 ms. Figure 2 shows the deadline miss ratio of the soft real-time task. The results show that a rate-based task with 92.2ms period, the biggest possible period for this audio application, improves the performance of soft real-time processes. As we explained earlier, this is due to the generated slack by the rate-based process. In contrast, the rate-based task with 23.2 ms period degrades the performance of the soft real-time processes, which suggests that the rate-based task does not generate slack in this case.

To summarize, the smaller the period of a rate-based task the less slack is generated. Thus, the choice of bigger periods improves the overall system performance when scheduling rate-based tasks as periodic tasks with worst-case reservations.

5 Natively supporting rate-based processing

During the experiments with scheduling rate-based processes as periodic tasks, we learned that rate-based processes behave different from conventional periodic tasks. Rate-based processes provide good overall system performance only in the case when the buffer is almost full. The problem with this solution is that it does not capture the important differences between rate-based applications and conventional periodic soft real-time applications. The main property of rate-based processes is that they can be very flexible in terms of how much of the resource to give to such a process and when to give the allocated resource. If more space is filled in the buffer a longer time may

elapse before the buffer is again filled, and if less space is filled in the buffer a shorter time may elapse before the buffer is again filled. Scheduling with a fixed period and keeping the buffer almost full does not make use of this flexibility.

Our integrated system is based on the Resource Allocation/Dispatching model, which explicitly separates resource allocation and dispatching management and allows them to be independently adjusted at run-time. Therefore, we can make use of the above properties and tailor the resource allocation and dispatching management precisely to the needs of rate-based processes.

We identified two approaches for scheduling rate-based processes based on their properties. The first is to set deadlines of rate-based processes in a rate-specific way. The main idea of this approach is to set a new deadline each time the current job of a rate-based task finishes executing. We set a new deadline to the time the buffer is going to be empty and set the execution time to the worst case time to produce amount of data currently in the buffer. To do this, the system should monitor the buffer's fill level using worst case estimation of how fast the buffer is filled by the producer.

This approach takes advantage of the flexibility of rate-based processes. If more space is filled in the buffer the further the current deadlines is, and if less space is filled in the buffer the closer the current deadline is.

The second approach is a feedback-based slack releasing mechanism. In our experiments with scheduling rate-based processes as periodic tasks we learned that slack is mostly generated when the buffer is almost full. Our goal is to make a rate-based process release slack whenever it is generated and, in particular, when less of the buffer is filled. This can be accomplished by comparing the execution time to the state of the buffer. If the buffer is more full than expected, indicated a less-than-worst-case execution time, slack from the rate-based process can immediately be made available to other processes.

We are going to investigate these two approaches further and combine them in order to provide an efficient solution for scheduling rate-based processes in an integrated system.

5.1 Conclusions

We investigated the behavior of rate-based processes in an integrated system and how they interact with other processes in the system. We found that handling rate-based processes as periodic tasks is feasible, but not very efficient. We identified two approaches that make use of specific properties of rate-based processes. We will investigate these approaches further and provide an efficient scheduling algorithm that provides good overall system performance and guarantees continuous processing requirements for rate-based processes.

References

[1] Kevin Jeffay. The real-time producer/consumer paradigm: a paradigm for the construction of efficient, predictable real-time systems. In *SAC '93: Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing*, pages 796–804, New York, NY, USA, 1993. ACM Press.

[2] Kevin Jeffay and David Bennett. A rate-based execution abstraction for multimedia computing. In *NOSSDAV '95: Proceedings of the 5th International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 64–75, London, UK, 1995. Springer-Verlag.

[3] Kevin Jeffay and Steve Goddard. A theory of rate-based execution. In *RTSS '99: Proceedings of the 20th IEEE Real-Time Systems Symposium*, page 304, Washington, DC, USA, 1999. IEEE Computer Society.

[4] Caixue Lin and Scott A. Brandt. Improving soft real-time performance through better slack reclaiming. In *2005 IEEE Real-Time Systems Symposium, Miami USA*, December 2005.

[5] Jason Nieh and Monica S. Lam. The design, implementation and evaluation of SMART: a scheduler for multimedia applications. In *SOSP '97: Proceedings of the sixteenth ACM symposium on Operating systems principles*, pages 184–197, New York, NY, USA, 1997. ACM Press.

[6] Brandt SA, Banachowski S, Caixue Lin, and Bisson T. Dynamic integrated scheduling of hard real-time, soft real-time, and non-real-time processes. In *Proceedings. 24th IEEE International Real-Time Systems Symposium. IEEE Comput. Soc. 2003, pp.396-407. Los Alamitos, CA, USA*, 2003.

[7] David C. Steere, Ashvin Goel, Joshua Gruenberg, Dylan McNamee, Calton Pu, and Jonathan Walpole. A feedback-driven proportion allocator for real-rate scheduling. In *OSDI '99: Proceedings of the third symposium on Operating systems design and implementation*, pages 145–158, Berkeley, CA, USA, 1999. USENIX Association.