A CPU Scheduling Algorithm for Continuous Media Applications*

Raj Yavatkar and K. Lakshman**

Department of Computer Science
University of Kentucky, Lexington, KY

Abstract. We provide an overview of a CPU management algorithm called RAP (Rate-based Adjustable Priority Scheduling) that provides *predictable* service and dynamic QOS control in the presence of varying compute times, arrival and departure of processes, and CPU overloads. A significant feature of RAP includes an application-level QOS manager that implements policies for graceful adaptation in the face of CPU overload.

1 Introduction

We are currently investigating operating system (OS) mechanisms and policies for managing end-system resources (such as CPU, network interface, memory, and bus bandwidth) so that an OS can provide *predictable service* to multimedia (MM) applications. In this paper, we provide an overview of a CPU management algorithm called RAP (Rate-based Adjustable Priority Scheduling). Our design goal is similar to the objectives of the dynamic QOS control schemes proposed earlier [1, 3, 4]. However, our aim is to provide dynamic QOS control and predictable service in the presence of varying compute times, arrival and departure of processes, and CPU overloads.

The design of RAP is based on the following assumptions. First, RAP does *not* assume a priori knowledge of compute times needed by MM applications. In particular, we have observed that the resource requirements of an MM application usually vary a lot during an application's lifetime. For example, the traces of execution times needed by the Berkeley MPEG player show that the amount of execution time needed to play back a single frame varies a lot within a GOP (Group of Pictures) and even the average execution time needed over a GOP shows considerable variations as a result of changes in scene or video contents. Second, RAP assumes that MM applications can tolerate occasional delays in execution and, therefore, does not try to schedule processes to meet their deadlines on per execution basis. Instead, RAP only ensures that each process will execute at an average rate within an acceptable range specified by the process. Third, RAP assumes that MM applications are adaptive in nature and

* This research was supported in part by the National Science Foundation Grant No. NCR-9111323 and Grant no. STI-9108764.

** Supported by the Center for Computational Sciences, University of Kentucky.

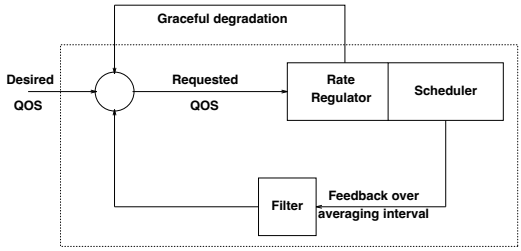

Fig. 1. The figure shows an abstract representation of the RAP algorithm.

can gracefully adapt to resource overloads by modifying their behavior to reduce their resource requirements. Examples of such adaptations include dynamically reducing spatial or temporal resolution in the case of a video application, selectively playing back portions of a hierarchically encoded video, and adjusting the audio sampling rate without compromising the ability to deliver consistently good playback quality.

2 Rate-based Adjustable Priority Scheduling

The RAP algorithm is based (figure 1) on the Phase-Locked-Loop (PLL) principle and consists of two components: a rate-based CPU scheduler which ensures that MM applications are serviced at a steady rate, and an application-level QOS manager which provides adaptive QOS management. The CPU scheduling algorithm is based on a service discipline called *Rate Controlled Static Priority* (RCSP) introduced in [5] for real-time packet scheduling. RAP borrows the ideas on admission control and scheduling from RCSP and extends them to the problem of CPU scheduling for providing predictable service to multimedia applications. In the following, we briefly outline a typical scenario for an application using RAP and describe the functions of each of the components that make up RAP (more details can be found in [2]).

- At the beginning, an application specifies a desired *average* rate of execution (e.g., 20 times a second) and an averaging interval over which the rate of execution is to be measured.
- RAP's admission control algorithm must first decide whether to accept a new process. RAP maintains an estimate of the average computing time needed by each admitted process. Based on these estimates, RAP's admission control algorithm calculates the available capacity for the new process, determines whether the new process can be accepted, and if so, allocates the new process a computing time based on the remaining available capacity.
- The new process is assigned a priority based on its requested rate and is inserted in a priority-based queue for execution.
- RAP schedules admitted processes using a *rate regulator* and a priority-based scheduler. The rate regulator ensures that a process with an accepted rate R

of execution does not execute more than R times a second and the scheduler ensures that a process roughly executes once every $T = \frac{1}{R}$ time interval.
- After the application starts executing, the average computing time it needs, and its rate of execution are monitored over the averaging interval. At the end of the averaging interval for a process, the RAP scheduler provides feedback to the application-level QOS manager about the observed rate of progress.
- The QOS manager compares the observed rate against previously requested rate and reacts either by reducing the process's computing time per execution or by asking for a reduced rate of execution.
- In addition, the RAP scheduler also provides feedback to application-level QOS managers when CPU capacity is over-utilized (under-utilized) and the scheduler wants each process to reduce (or increase) its individual demand by a fraction proportional to the process's current share of CPU capacity.

3 Preliminary Results

We have evaluated the performance of RAP using a trace-driven simulator that uses traces of execution times needed by the Berkeley MPEG player on a Sun Sparc-20. The experiments involved two classes of applications, namely, a CBR (Constant Bit Rate) class corresponding to audio or music player and a VBR (Variable Bit Rate) class corresponding to a compressed video player. CBR applications need almost constant amount of compute time per execution whereas computing times needed by VBR applications vary over time. In the following, we present results of a sample experiment designed to demonstrate the effectiveness of admission control and rate adaptation algorithms.

- Figure 2A shows an overload condition with admission control turned off. Without admission control, the overload causes missed deadlines and wide rate fluctuations for the processes.
- Figure 2B shows the effect of introducing admission control test in the previous case. Due to admission control, the CBR process that arrives at the 3 second mark is refused service. The VBR process that requests execution rate of 50 is admitted and allocated a maximum compute time of only 1 ms based on the available capacity. However, the VBR application needs more CPU and experiences significant rate fluctuations.
- Figure 2C shows the result of adding application level rate adaptation to case B. Based on the observed rate jitter, the QOS manager for the VBR application chooses to reduce its desired rate to 30 times/second. However, due to the limited available capacity, the admission control still accepts it with maximum compute time of only 1 ms and the VBR application still sees rate fluctuations. Its QOS manager then further reduces its rate and eventually stabilizes at 20 executions per second.
- Figure 2D shows the result of graceful degradation when the scheduler asks processes to reduce their share of CPU capacity. Admission of both *cbr_100_3* and the VBR application would cause overload and, therefore, the scheduler

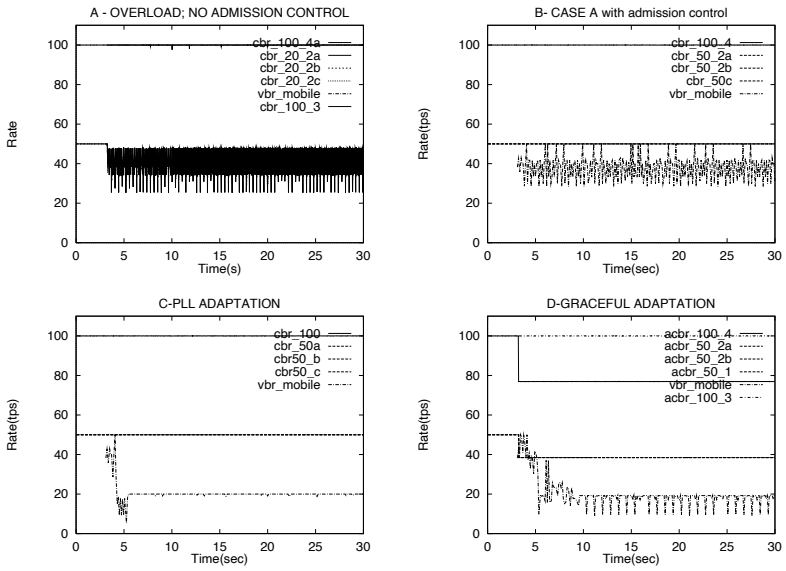

Fig. 2. Plots show average rates of execution for a combination of processes in different cases. The notation used is as follows: class_rate_compute-time; i.e., cbr_100_3 means a CBR applications running 100 times a second with an average compute time of 3 ms. VBR_mobile is based on a MPEG stream with varying compute times.

asks all processes to reduce their rates before admitting new processes. In the case of CBR applications, the rate reduction is sufficient to execute them at a steady rate, but the VBR application needs a lot more processor capacity and does not stabilize. In that case, the VBR application's QOS manager observes the rate jitter and reduces its own rate further until the jitter is within its acceptable range.

References

1. Andrew Campbell, Geoff Coulson, and David Hutchison. A multimedia enhanced transport service in a quality of service architecture. In *4th International NOSSDAV Workshop*, pages 124–137, 1993.
2. K. Lakshman and Raj Yavatkar. Adaptive CPU Management for Multimedia Applications. Technical report, University of Kentucky - Dept. of Computer Science, March 1995.
3. K.K. Ramakrishnan and et.al. Operating system support for a video-on-demand file service. In *4th International NOSSDAV Workshop*, pages 216–227, 1993.
4. Hide Tokuda and Takuro Kitayama. Dynamic QOS Control based on Real-Time threads. In *4th International NOSSDAV Workshop*, pages 114–123, 1993.
5. Hui Zhang and Domenico Ferrari. Rate-controlled static priority queueing. In *Proc. IEEE Infocom '93*, May 1993.