May 1996

CPU management for UNIX-based MPEG video Applications

Veronica Baiceanu

CPU Management for UNIX-based MPEG Video Applications

Veronica Baiceanu

May 6, 1996

Abstract

While continuous media applications are becoming increasingly popular, poor quality resulting from resource scarcity is a common problem. Resource management for distributed multimedia systems is an active topic of research, addressing both performance issues and quality guarantees. Insufficient operating systems support for multimedia computing is often mentioned as a major problem, even for real-time platforms. In particular, continuous media applications require specific scheduling mechanisms, appropriate for periodic tasks with deadline and jitter constraints. While many scheduling algorithms have been proposed and implemented, we notice that most of the studies assume operating systems with real-time capabilities. Most research ignores widespread time-sharing systems such as UNIX, which is known to perform poorly for multimedia, offering no support for hard guarantees.

We evaluate the performance and quality of service guarantees enabled by the real-time features of recent versions of UNIX. While hard real-time guarantees cannot be provided, we argue that statistical guarantees are satisfactory, as a certain quality loss is usually tolerable. We propose a scheduling mechanism that approximates the rate-monotonic (RM) scheduling algorithm, and a test for granting admission to new applications. We have implemented our method on tasks simulating workloads derived from a real MPEG video player. Our experiments show that our mechanism provides better guarantees for computation deadlines and better performance than the UNIX standard scheduler. We discuss the reasons for guarantee violations and issues related to applying our method in real systems.

1 Introduction

While user demand for high quality multimedia presentations is increasing, multiple problems arise because of insufficient support at network and end-point system levels. The continuous transfer of massive amounts of data in real-time requires specific multimedia system design issues and forces tradeoffs between real-time performance and data accuracy. Thus, while many research efforts aim to enable satisfactory multimedia presentations, by providing adequate application and system support [2, 9, 3], other efforts are directed towards characterizing and guaranteeing a certain *quality of service* (QoS) [17, 13].

A possible definition of QoS is [20]: "QoS represents the set of those quantitative and qualitative characteristics of a distributed multimedia system necessary to achieve the required functionality of an application. Functionality includes both the presentation of multimedia data to the user and general user satisfaction."

Depending on whether there are or not QoS guarantees for an application, we can distinguish two main approaches for multimedia applications: the *best-effort* approach and the *resource-reservation* approach. The best-effort approach strives to deliver all data in real-time, by consuming as many resources as possible. The resource-reservation approach attempts to reserve resources in order to guarantee quality of service.

Resource reservation is performed through an *admission tester*, a system component to which applications will request guarantees for a certain amount of a given resource or resources. If accepted, an application will have that amount of resources guaranteed, and if rejected it can renegotiate with the admission tester for a smaller amount of resources. A resource reservation system should support *monitoring* – keeping track of the amount of resources consumed by applications, and *policing* – a mechanism for preventing applications from using more resources than the amount reserved.

For both the adaptive and resource-reservation approaches, we notice that a certain amount of resource consumption has to be associated to a user-perceived quality. While user perception of quality is insufficiently described in litterature, Staehli et al. [17] have specified an architecture for describing QoS at the application level. Tradeoffs among different quality dimensions should be specified by the application through a QoS interface. For example, a loss of spatial resolution for digital video could be prefered to having a lower playback frame rate.

In order to perform resource management, Staehli et al. [17] propose starting from application-level QoS specifications. This will require the translation from application-level parameters into low level resource requirements. The necessity of solving this "mapping problem" is widely recognized [10, 13] as the best means of allocating the correct amount of resources in the guaranteed approach.

Most of the effort for providing multimedia support, including guarantees, has focused on the network level [1]. However, many researchers notice the need for more operating system support for multimedia [19, 12]. In particular, multimedia applications require large disk and memory bandwidth, and impose scheduling constraints related to periodic arrival of data. Guaranteeing that tasks are going to meet their periodic deadlines requires real-time operating system features – operating system features that insure that the end time of a computation can be predetermined. A major problem encountered in real world is that most of the existing platforms are based on time-sharing systems such as UNIX, that do not offer support for real-time guarantees. Thus, most of the research has focused on providing scheduling mechanisms and guarantees for real-time operating systems . We argue that even though hard real-time guarantees (that is absolute guarantees, for instance no deadline missed), cannot be supported on a UNIX-like operating system, statistical guarantees can be offered. Some quality degradation from the level requested by a human user is usually tolerable, and thus giving statistical guarantees is useful, as long as the statistical bound for error tolerated by these guarantees is reasonably high.

Our work will focus on CPU management study, as we believe that the CPU is often the bottleneck resource for data delivery in end-to-end systems. We perform a simulation for estimating the statistical guarantees that can be offered on a UNIX platform for tasks with characteristics derived from a real MPEG player. Our approach for doing CPU management is QoS-driven: we will map application-level QoS parameters (resolution and frame rate for MPEG video) into task characteristics. We are adopting the resource-reservation approach, since resource scarcity is likely to remain a problem in the next years, and guarantees will be desirable. Thus, we will describe an appropriate scheduling mechanism and an admission tester for our particular kind of tasks.

The paper is organized as follows: Section 2 is an overview of CPU scheduling in the context of multimedia requirements; Section 3 introduces the scheduling mechanism we propose; Section 4 presents our admission tester's architecture; Sections 5, 6, and 7 describe our experiments; and finally, Sections 8 and 9 discuss conclusions, and mention related and future work.

2 CPU Management Context

2.1 Operating System Support for Real-Time Computing

Time-sharing systems such as UNIX aim to insure good responsiveness for interactive tasks. In contrast, the main goal of real-time systems is to offer support for predictable service. In order to guarantee completion time of tasks, the system should have a deterministic behavior. The existence of virtual memory or secondary storage makes real-time computing difficult, since page faults and disk accesses induce an unpredictable variation in servicing task requests. A nondeterministic behavior is also caused by an unpredictable dispatch latency, if tasks are non-preemptable in system calls or when blocking on I/O [16].

We should distinguish between hard real-time guarantees, when the completion time guaranteed has to be always satisfied, and soft real-time guarantees, when guarantee violations are acceptable, within certain statistical limits. As mentioned, UNIX does not have the features required for a real-time operating system, implying that hard guarantees are not possible. However, multi-threaded versions of UNIX like SOLARIS allow preemption within system calls, at so-called preemption points, where tasks are not within critical regions. In addition, recent versions of UNIX support real-time features that enable better performance of multimedia applications. Real-time operating systems are based on schedulers that are priority-based, and are known not to provide adequate support for background computation. The recent versions of UNIX that support the facility of assigning real-time priorities to tasks will also exhibit the undesirable features of real-time systems. Nieh et al. [14] have shown that the time-sharing properties of the entire system can be compromised by using these real-time facilities for continuous media applications. As expected, the response time of background or interactive tasks can become extremely high, and even starvation can occur. Our work will not address this particular aspect; we are rather interested to estimate the guarantees that can be given for continuous media applications. However, we believe that this kind of performance degradation can be prevented through a more sophisticated scheduling mechanism, together with CPU reservation and admission testing.

We notice that increase of CPU speed will benefit overall system performance, as more CPU will

be available for general-purpose and interactive tasks. Faster computing will never insure good real-time performance [18], since a task will still miss its close deadline, if other unurgent computations occupy the CPU. However, fast computing allows more real-time tasks to be scheduled, higher flexibility in their scheduling, and increases CPU availability for other kinds of tasks.

2.2 Scheduling Algorithms for Continuous Media

2.2.1 The Rate-Monotonic (RM) Scheduling Mechanism

As mentioned, continuous media are periodic media, requiring scheduling mechanisms for periodic tasks. In the following, we will focus on the description of the RM scheduling algorithm, since we consider that it is the algorithm most likely to be chosen for implementations on time-sharing platforms. In addition, it is one of the algorithms usually chosen for the study of continuous media scheduling.

The RM scheduling algorithm was first characterized by Liu and Layland [8] under the following assumptions:

- Tasks have periodic requests.
- The only constraint on the deadlines is that the task must complete before its next request.
- Tasks are independent – the request of one task does not depend on the initiation or completion of another task.
- The time a task runs within each of its periods is constant.
- Nonperiodic tasks in the system are special; they will displace periodic tasks and do not have themselves critical deadlines.

All these constraints can be relaxed, under certain conditions. For example, the second requirement can be violated, given a certain amount of buffering for incoming data. Also, the fourth condition can be relaxed by considering the maximum computation time of the task for all the periods. This

relaxation will result in wasting CPU time, but it can be necessary in cases where computation times are highly variable, like the case of MPEG streams.

RM is a *priority-driven* scheduling algorithm – tasks are assigned priorities and the task with the highest priority is given the CPU. It is a *static* scheduling algorithm, in the sense that priorities assigned to tasks do not change over time. Higher priorities are assigned to tasks having shorter period. RM is *preemptive*, meaning that a task with higher priority will preempt a task with lower priority that occupies the CPU. Preemption is assumed to occur at once and with no context switch overhead.

While Liu and Layland have provided a schedulability test for the RM scheduling algorithm, their test is somewhat pessimistic – a set of tasks might still be schedulable under RM even if it fails this test. Lehoczky et al. [6] have later provided the exact characterization of the RM scheduling algorithm. The term "exact" is used in the sense that if the Lehoczky et al. admission test fails, the set of tasks is guaranteed not to be schedulable under the RM mechanism.

Let us consider a set of tasks τ_1, τ_2,...,τ_n with corresponding periods T_1, T_2,...,T_n, sorted in increasing order. Then, the admission test described by Lehoczky et al. insures that τ_i will meet all its deadlines if and only if

$$\min_{0<t\leq T_i} \sum_{j=1}^{i} \frac{C_j}{t} \left\lceil \frac{t}{T_j} \right\rceil \leq 1.$$

Since this criterion is piecewise linear and decreasing, it is enough to test for values of t which are multiples of the T_j . If the test accepts all the tasks in the set, the whole set is schedulable under RM. We can also notice that a task with longer period has less chances to meet all its deadlines than a task with shorter period, since the value in the left hand side of the expression strictly increases with the value of T_i. We will consider this aspect in Section 6.

2.2.2 Discussion on Continuous Media Scheduling

For fixed priorities, RM is proven to be the optimum scheduling mechanism, in the sense that if a set of tasks is not schedulable under RM, it will not be schedulable under any fixed priority scheduling mechanism.

However, the set of tasks rejected by RM might still be scheduled under another scheduling mechanism. Liu and Layland have also described the *earliest deadline first scheduling algorithm* (EDF), which is optimal, in the sense that if a set of periodic tasks cannot be scheduled by EDF, it cannot be scheduled by any scheduling algorithm. EDF uses dynamic priority assignment – it dynamically assigns the highest priority to the task having the closest deadline. The five conditions required for RM and the assumption of having preemptive tasks hold for EDF as well.

A measure of performance for these algorithms is the *processor utilization factor* – the percentage of processor time spent in the execution of a task set [8]. Under RM, tasks are usually unschedulable for a processor utilization factor lower than for EDF. On the other hand, RM requires more context switching than EDF [19]. However, the priority recalculation required for EDF makes it much harder to implement than RM, and makes context switching more expensive.

As mentioned, both the RM and EDF scheduling algorithms require preemptive scheduling. However, this condition is not satisfied on most of the existing platforms. Nagarajan and Vogt [11] have described an admission tester for the RM scheduling mechanism for nonpreemptable tasks. While using their test results in drastically reducing CPU utilization, we believe it is worth studying RM under nonpreemptive scheduling and our work addresses this aspect.

Finally, we notice that continuous media are characterized not only by periodic arrival of data, but also by the requirement of periodic completion time of computations. Both RM and EDF address the problem of having a task completed before its new request time. However, this does not imply periodic completion of computations – a variance in completion times will be allowed, as long as the task finishes before its new run request. The human eye is however sensitive to the non-uniformity in continuous data delivery, known as *jitter*. It has been shown that jitter is lower for tasks scheduled under RM than for tasks scheduled under EDF [7]. In addition, jitter for RM can be ameliorated through a technique that achieves jitter reduction, but usually decreases CPU utilization [7].

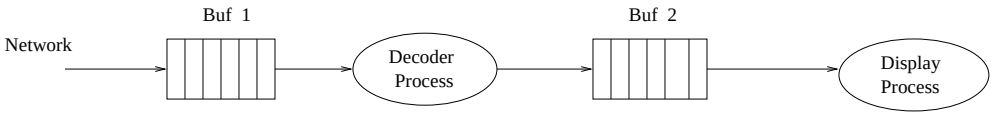

Figure 1: Architecture of MPEG player client for one stream

2.3 Client-Site Architecture of an MPEG Player

Our scheduling mechanism and admission tester will be analyzed by using tasks that simulate video processes from the client site of our distributed video player [2]. We will base our experiments on this player's architecture, as we found that it common for a distributed player, using compressed video streams. The MPEG compression standard [4] used in our player is also widely used.

Figure 2 1 represents the architecture of one MPEG video stream pipeline at the client part of the player. The processes relevant for our considerations are the *decoding process*, and the *display process*. Buf1 holds the frames assembled from network packets. The decoding process decodes the frames from Buf1, and stores the decoded frames in Buf2, and the display process displays the frames after reading them from Buf2.

This architecture is relevant for the resource interplay that generally occurs in such applications. For this particular case, the resource trade-offs are among buffer dimensions, network bandwidth and CPU scheduling restrictions. (For instance, if Buf2 is larger, we can relax the time constraints on the decoder – it can be late for one or more frames, the exact number being determined by the size of Buf2.) For the analysis of the scheduling algorithm and admission tester we propose, we will consider the case of having no buffering. This implies that the time constraints on the decoding process are the same as on the displaying process – the periods of the two processes have to be the same.

3 Runtime Scheduling

3.1 UNIX Facilities for Real-Time Computing

Time-sharing systems such as UNIX insure fairness by dynamically assigning priorities to processes, according to the length of their bursts. Once scheduled, a process is allowed to run for a time interval called time-slice. Meanwhile, no scheduler invocation can occur, unless the process voluntarily relinquishes the CPU. Thus, a higher priority process cannot preempt a lower priority process before the latter finishes its time-slice or yields the CPU – this phenomenon is called *priority inversion*.

As mentioned in Section 2.2, the new real-time facilities implemented on recent versions of UNIX allow the user to set the priority of a process so that it does not get degraded through the mechanism of priority aging. Real-time priorities do not degrade in time and all processes with real-time priorities will be prefered over the non-real-time ones. Processes with equal real-time priorities will be scheduled in round-robin fashion.

It is often mentioned that using the real-time priority facility presents many risks [14]. A process with real-time priority can prevent all other lower priority or non-real-time processes from running, if it occupies 100% percent of the CPU. This implies that even the console will be frozen until that process will voluntarily give up the CPU. Presently, for multimedia applications it is common that one or many of their processes occupy the entire CPU. For instance, the MPEG player we used for our experiments [Cen] becomes CPU bound when running a movie at 320x240 pixels resolution and 20 frames/second. Even if there is CPU available for non-real-time background or interactive tasks, their waiting time can be untolerably long [14].

We believe that an admission tester can prevent catastrophic or undesirable situations. By admitting only the video applications that can receive adequate service, our admission tester insures that admitted tasks do not impede on each other's performance.

To prevent malicious applications from taking over the CPU, the admission tester should run with root privileges and kill tasks that do not conform to their reservation level. Measuring the run time of tasks in UNIX is not a problem, since it is simply the sum of the times the process

has spent in kernel and user spaces. Thus, monitoring and policing can be supported for the tasks admitted by the admission tester.

Of course, this mechanism offers no guarantees on the behavior of other tasks in the system. Unless all real-time tasks in the system are characterized and tested for admission, a malicious real-time application can occupy the CPU. Note, however that setting real-time priorities requires special privileges, so only trusted users can run real-time tasks. This policy is a result of the risks described for the use of real-time priorities. Thus, the user should run only real-time tasks whose behavior is known and that have been tested for admission by using an adequate admission test.

3.2 Approximating the RM Scheduling Mechanism

Our goal is to adopt a scheduling mechanism that generates better performance for continuous media tasks than the native UNIX scheduler. In addition, an admission testing mechanism adequate to the scheduling mechanism adopted is required.

Our scheduling mechanism approximates the RM scheduling mechanism described in Subsection 3.3.– we assign higher priorities to tasks with shorter periods. We use the term "approximation" in the sense that in our case preemption is not possible at granularity lower than time-slice length (on our system, 10 ms), while RM requires preemptive scheduling. Additional imprecision is introduced through context switch overhead, timer interrupt imprecision, non-preemptability in system calls and during I/O blocks.

The admission tester we are studying uses the RM admission test described by Lehoczky et al. [6], which assumes perfect preemptability of tasks. In this sense, our admission tester is optimistic: Some of the tasks it will accept should be rejected, even if we ignore the influence of non-real-time UNIX characteristics. Notice that non-preemption at granularity lower than the time-slice does not mean that the operating system in not real-time. For instance, we could use the more restrictive admission test described by Nagarajan and Vogt [11] for nonpreemptable tasks.

We will show that statistical guarantees can be offered under our scheduling mechanism, using the admission tester mentioned. We will analyze this for sets of tasks simulating tasks characteristic

to an MPEG video player client. Furthermore, we will show that our scheduling mechanism leads to much better performance in terms of deadline misses and guarantees than the native UNIX scheduler. The jitter level will also be reduced.

4 Admission Tester Architecture

This section describes the admission tester we propose in the context of a real multimedia system. So far, we have only simulated the behavior of our admission tester for performance analysis.

As mentioned, the admission tester runs the Lehoczky admission test. The admission tester can run at user level, as a separate process in the application, if it is designed for one application only. In this case, the application will have multiple video streams, and it will be the only multimedia application running on the system. Or, the admission tester can run as a server process, if we assume multiple similar cooperative applications running on the system. In either cases, the admission tester has to be given highest priority, so that it becomes the running process when an application demands admission or stops running.

We have mentioned in Subsection 2.1 that we intend to perform QoS-driven admission testing. Thus, the relevant quality parameters for our applications – resolution and frame rate – will be mapped onto task parameters – computation time and period of tasks, respectively. When a new application demands admission or when a user demands a quality change, the resolution and frame rate desired are communicated to the admission tester, whether it is part of the application or in a separate module. Within the admission tester, resolutions and frame rates are first translated into computation times and periods. Based on the Lehoczky admission test, the one or many new video streams are granted or denied admission. The admission tester must also be invoked when a quality change is requested, and a new admission test occurs if this change results in more resource consumption. In case of rejection, the application can renegotiate with the admission tester, and lower its quality demands. An application will communicate to the admission tester that it stops running or that it is lowering its quality demands, so that the admission tester can do the necessary parameter updates.

We notice the difficulty of solving the mapping problem for the case of MPEG streams. MPEG streams consist of three kinds of encoded pictures, which vary significantly in size and required decoding time. Much irregularity can be noticed in the stream even for the same type of picture. Even though pictures form groups with the same picture pattern, the overall size and decoding times for groups still exhibit much variation.

For the average values of the decode times, we notice an approximatively linear variation of the decode time with the picture size (that is, with number of pixels of both decoded and encoded frames). The display time also varies approximately linearly with the picture size.

Note that we will consider the cummulative value decoding and dithering computation times for the admission test. (Dithering is an image processing algorithm that sacrifies resolution in order to get better user perception.) Both the decoding time and the cummulative value (average times) vary almost linearly with the size of the image, and we attempt to provide to the admission tester values as close as possible from our real MPEG player. For simplicity, in the description of simulation below, we will understand by decoding time the time corresponding to both decoding and dithering tasks.

There are many possible design issues related to the irregularity noticed for MPEG decoding times. A pessimistic admission tester will test using the maximum possible value of the decode time for all the frames. However, this will result in dramatic CPU overreservation. Rather, as mentioned in Section 2.3, the period for the decoding process should be relaxed, according to the amount of buffering available between the decoding and the displaying processes, and between the video buffering and decoding processes. These buffers are present in our particular client architecture, but they are usually a good design decision for a software decoded MPEG video stream.

5 Simulation Environment

As mentioned, the goal of our study is to characterize the effectiveness of the "approximate" RM scheduling mechanism and of the Lehoczky admission test for typical video tasks. We have simulated MPEG player video decoder and video display processes, by implementing indepen-

dent periodic processes with periods corresponding to discrete frame rates in the range 10 – 30 frames/second. We anticipate that these are the values likely to be chosen for video applications. The computation times of the simulation processes are derived from the computation times of the decoding and displaying tasks in the player.

We have considered a range of resolutions between 128 x 96 pixels, corresponding to the minimum resolution for our movies, and 640 x 480 pixels, corresponding to VGA resolution. We have interpolated one hundred resolution values between half the minimum mentioned and VGA resolution. We are starting with half the minimum value, to leave a provision for the situation when CPU speed will double.

We have chosen the computation times of the decoding process by taking the average values for each resolution. Thus, our admission tester will optimistically accept tasks for which individual frames will miss their deadlines. In practice, buffering allows relaxation of the constraints on individual frames.

We are simulating the video decoding and displaying tasks in our player using tasks that count. In order to reduce the time for admission testing, we are using a *feasibility* test: we reject the tasks with computation time longer than their period, since they are guaranteed to be unschedulable. Then, we are running the Lehoczky admission test. After the admission test is passed, we translate computation times into number of countings. Note that we run the admission tester for sets of tasks that approximate different sets of video streams from players. We are experimenting for sets of tasks corresponding to one, two, three or four players (two, four, six and eight tasks, respectively).

While we have not performed studies of human behavior, we appreciate that a random distribution of resolutions and frame rates within the range mentioned is adequate for a user selecting video applications on a general-purpose machine. Each resolution and frame rate correspond to a pair of decoding and displaying tasks. Since we are giving priorities to tasks according to RM, and the tasks corresponding to the decoding and displaying tasks of a same player have equal frame rates, they will be given the same priority. The fact of using pairs of tasks also determines a specific kind of relationship between the computation times: A pair will have a task with a long

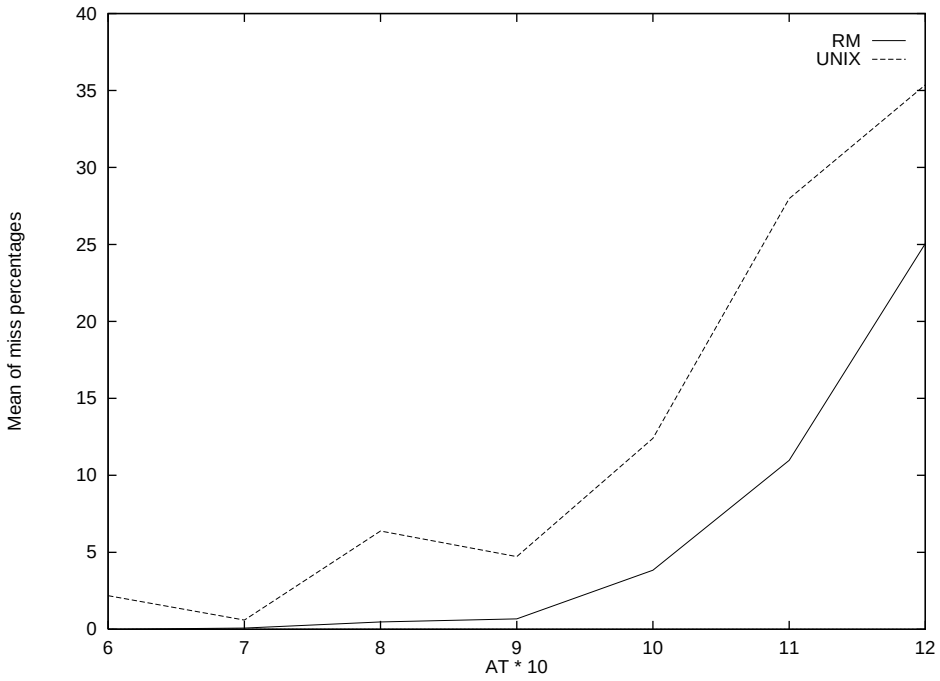

Figure 2: Arithmetic mean of miss percentages for three players

computation time (the decoding process) and a task with short computation time (the displaying process). Thus, our experiments target MPEG streams only, and we do not attempt to generalize the results for arbitrary periodic tasks.

6 Experiments

We run our experiments on a dedicated HP-UX machine, with clock frequency 80MHz. We are estimating the statistical guarantees that can be offered on a UNIX platform for MPEG streams, and we compare the "approximate" RM scheduling mechanism with the native UNIX scheduler.

For each set of tasks that is presented to the Lehoczky admission tester, the admission tester will return the value AT, given by:

$$AT = \min_{0<t\leq T_n} \sum_{j=1}^{n} \frac{C_j}{t} \left\lceil \frac{t}{T_j} \right\rceil \leq 1,$$

where T_n is the longest period for an n-task set. Recall from Section 2.2.1 that a task with

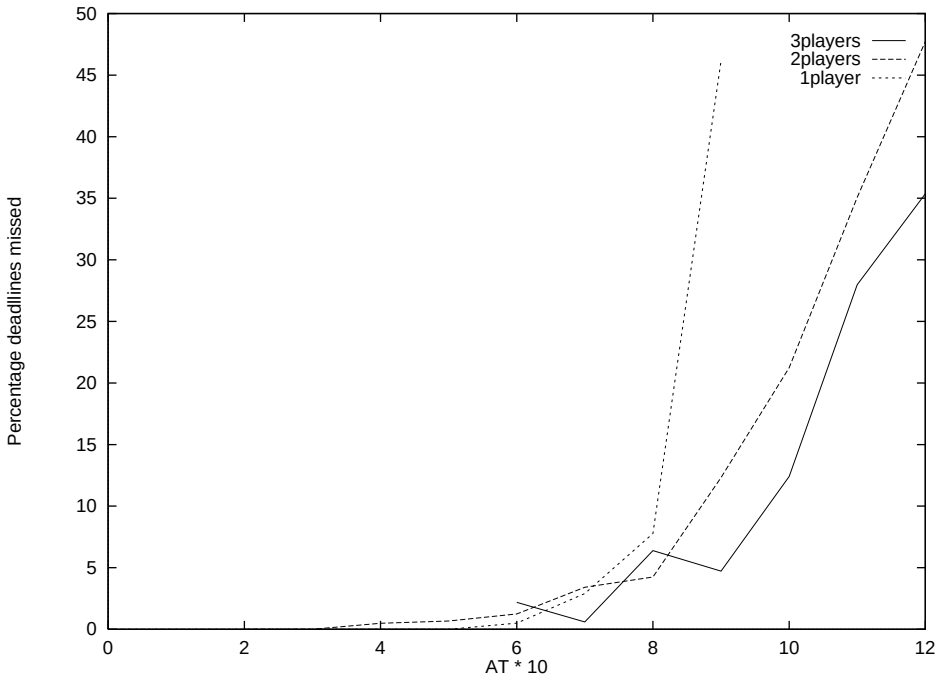

Figure 3: Arithmetic mean of miss percentages for native UNIX scheduler

longer period is less likely to meet all its deadlines than a task with shorter period. Thus, if T_n is admitted by the Lehoczky test, all the tasks in the task set are guaranteed to meet their deadlines.

In order to estimate the guarantees that can be offered on UNIX, we measure the percentages of deadlines missed for each task in a task set, for values of AT between 0 and 1.2. The reason why we are testing beyond the admission limit given by the Lehoczky test is that we can tolerate a certain guarantee violation and thus the percentage of deadlines missed when hard guarantees are impossible is still of interest. In addition, theoretically, we cannot ignore the possibility that a set of tasks rejected by the Lehoczky test is schedulable under the "approximate" RM mechanism or by the UNIX scheduler. It is however very unlikely that one of last two mechanisms can outperform RM, which for statically assigned priorities is the optimum scheduling mechanism.

As mentioned, our study aims to estimate the guarantees in the case of MPEG streams. Thus, we are deriving the percentage of deadlines missed per stream from the percentages of deadlines

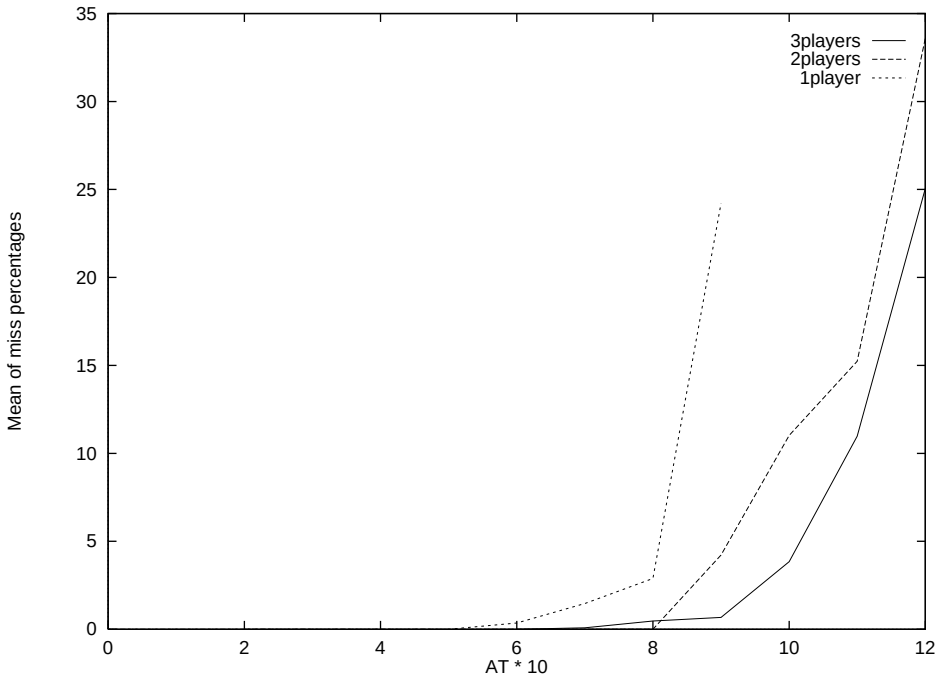

Figure 4: Arithmetic mean of miss percentages for approximate RM mechanism

missed per task. A stream is consider to miss its deadline if either of its decoder or display processes miss their deadline. This corresponds to the case of having no buffering between the decoder and the display processes.

We are running our admission testing mechanism on the input data 400000 times, 100000 times for each number of streams considered – one, two, three, and four streams. In order to estimate the guarantees offered for each value of AT we will record the average of the percentages of deadlines missed per set of streams, and the maximum of the percentages missed per set of streams.

Figure 2 presents the average of the percentages of deadlines missed per set of streams, for three players, depending on the value of AT x 10. (Because of our mechanism for data collection, the values of AT are discrete, while in fact they vary in a continuous way for our random input data set.) This measurement indicates that for a set of streams that have passed the feasibility test, we can expect the percentage of deadlines missed indicated in figure 2. The "approximate" RM

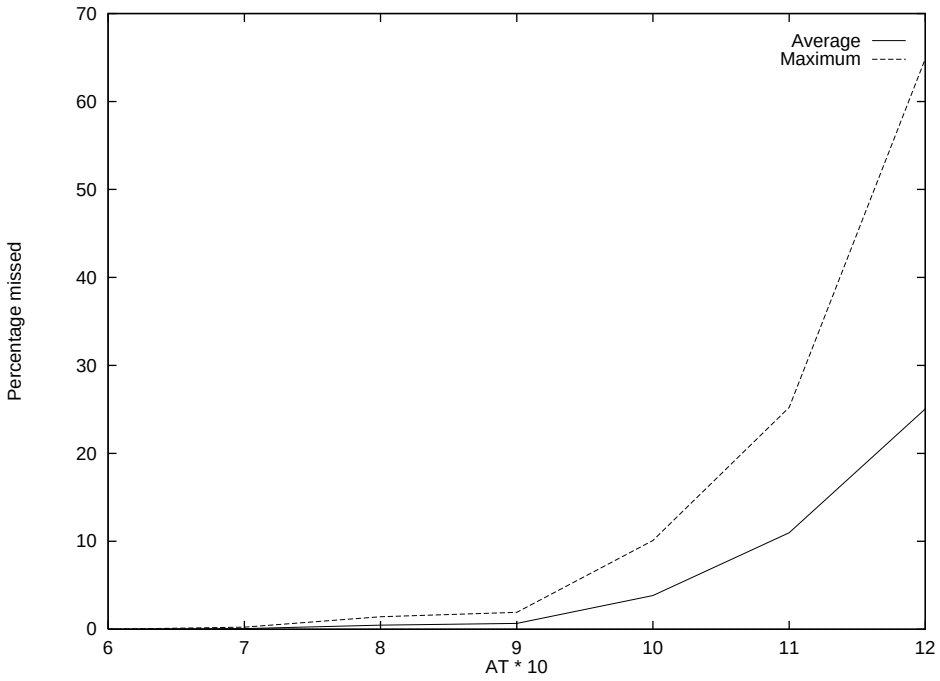

Figure 5: Maximum miss percentages for UNIX and RM

scheduling mechanism clearly outperforms the native UNIX scheduler, and starting from a value of AT = 0.9 (which usually corresponds to 80 – 90% CPU utilization), the percentage of deadlines missed becomes neglijible. However, we have to emphasize that we have represented the *mean* value for a number of three players with a certain AT. The variance for all the results we collect is high. We will discuss this aspect in the next section.

In order to estimate the maximum possible guarantee violations, in figure 5 we have represented the mean of the maxima of the percentages missed per set of streams, for three players, for the same range of AT. We notice again that our proposed mechanism is on average better than the native UNIX scheduler. However, if taking the absolute maxima we can notice that it is usually higher for RM, and we will discuss this aspect in the next section.

Figure 6 shows the miss percentages from the total number of deadlines to be met by the tasks within a set of players. Thus, for the first set of results represented we were calculating the

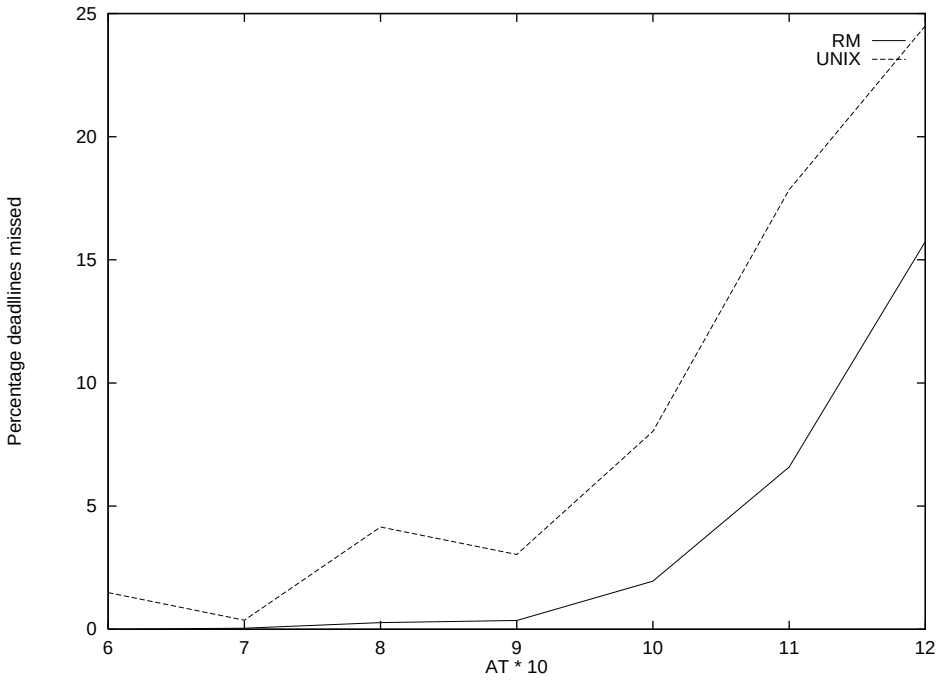

Figure 6: Miss percentages for sets of three players

percentage of deadlines missed per stream, and then we were averaging for all the streams in the set. Now, we are calculating the total number of deadlines that have to be met by the tasks in a task set, and then we compute the percentage of that number of deadlines that were missed.

This experiment will give information about the useful workload that "approximate" RM and the native scheduler are able to perform. A more useful measurement would have been to determine the amount of computation that has been wasted by the tasks that, being admitted, have occupied the CPU and then have missed their deadline. We plan to perform this measurement in the future. We notice that again the "approximate" RM scheduling mechanism is superior to the UNIX scheduler.

Additionally, in figures 3 and 4 we have plotted the average percentages of deadlines missed from the first experiment, for sets of one, two, and three players. The figures show that the miss rates are higher for a smaller number of streams. We attribute this to the fact that from our

random distribution of data, it is more likely that streams with high frames rates are accepted if they are part of smaller sets of tasks. Thus, tasks within these small sets have on average higher frame rates. At high frames rates, and thus short periods, effects like timer interrupt errors and nonpreemption at granularity smaller than the time-slice, are more important. Notice that we do not have values for sets of four players, since for our random distribution of stream characteristics, no set of four players has passed the admission test.

We have also measured jitter values and have observed the same trend: Average jitter values are higher for the native UNIX scheduler than for our scheduling mechanism. However jitter is characterized by high unpredictability. In general, jitter is less than twice the length of a time-slice, since timer interrupt errors and non-preemptive scheduling have effects only at this granularity. Additionally, RM does not provide good jitter guarantees.

7 Discussion

Besides the experiments described, we have observed that our scheduling mechanism and the native UNIX scheduler exhibit a different behavior related to the distribution of deadlines missed within a set of streams. Under the UNIX scheduler, tasks with higher ratios between computation times and periods tend to miss more deadlines, since the UNIX scheduling mechanism dynamically degrades the priority of tasks with high computational demands. If the periods are similar for two tasks of the set, the task with shorter period tends to miss more deadlines, because the UNIX features that impede on real-time performance are more important at smaller granularity.

For the "approximate" RM scheduling mechanism, the task with the longest period tends to miss more deadlines, since it has the lowest priority. Depending on user preference, this effect can be tolerable or not. With RM, our results indicate that a smaller number of deadlines will be missed than with the UNIX scheduler. However, since the frame rate for the stream that misses in the RM case is lower, missing a few number of frames can result in catastrophic quality degradation. Thus, it is essential to provide a QoS specification, and interpret the importance of different phenomena from this point of view.

For all the experiments, we have observed a high variance for the result data collected. We believe that this is due to the fact that the admission test itself does not capture information about the non-real-time UNIX effects. These effects are dependent on other task characteristics than those considered by the Lehoczky admission test. We intend to perform more experiments in order to increase the degree of confidence in our results. Given the large number of experiments required, we have usually tested only ten sets of streams for each AT and each number of players. This is insufficient, as the data exhibit high variance.

8 Related Work

Current research addresses diverse CPU management issues for continuous media applications. Ramakrishnan et al. [15] have addressed the problem of continuous media tasks coexisting with interactive and general-purpose tasks. Their scheduler and CPU admission tester offer performance guarantees for periodic tasks, while insuring satisfactory performance of interactive and general-purpose tasks. Mercer et al. [10] have described a mechanism of reserving CPU for periodic tasks on microkernel architectures and have provided facilities for monitoring and policing in this environment. Han and Shin [5] have addressed the particular case of scheduling MPEG streams on the server and have described a scheduler and an admission tester for such streams based on the EDF algorithm. A common feature of all these approaches is that they are based on the RM or EDF scheduling algorithms.

Despite the diversity of CPU management issues proposed, we notice that their applicability on most of the platforms is limited. Usually, the mechanisms are implementing on real-time operating systems [10], or the studies are purely theoretical [5]. The lack of operating system support for resource reservation has impeded on end-to-end resource reservation attempts [12]. We also notice that the resource interplay that occurs at end-system level and between end-systems and network has been insufficiently studied in the context of reservations. Even though QoS-driven resource management is often mentioned as desirable, practical solutions are rarely described. Thus, our work focuses on real applications and widespread systems, and studies the applicability of theoret-

ical studies on existing platforms.

9 Conclusions and Future Work

Our work investigates the real-time performance and guarantees that can be offered for video applications running on UNIX platforms. We have implemented the RM scheduling algorithm and admission tester on UNIX and have studied the statistical guarantees offered for tasks that simulate MPEG video streams. While our work is focusing only on CPU management and ignores other resources, our future work will address the resource interplay in multimedia systems, from an end-to-end perspective. We intend to consider different network environments (with guaranteed traffic rate or with burstiness). We will continue to perform QoS-driven resource management: we will start from application-level QoS parameters and will translate them into resource characteristics.

We are currently integrating the RM admission tester with our MPEG player and we will estimate the guarantees that can be offered for a real application. We will study the effects of the complexity of real applications, whose resource needs are hard to characterize. In addition, for real applications task communication, system calls and significant memory usage cannot be neglected.

While so far we have studied the performance of the RM scheduling algorithm and admission tester on UNIX for MPEG streams, we plan to generalize the results for any kind of tasks. We also plan to provide a formal characterization of the RM admission test for the UNIX system, where preemption is limited by time-slice length. Our solution will thus refer to a situation in between preemptive scheduling (characterized by Liu and Layland and Lehoczky et al.), and nonpreemptive scheduling (characterized by Nagarajan and Vogt).

We will also analyze the coexistence of continuous media tasks, interactive tasks and general-purpose tasks on UNIX. We intend to study reservation mechanisms for UNIX that allow guarantees for continuous media tasks, without compromising performance of other tasks. Finally, we intend to provide a UNIX extension with an in-kernel admission tester.

References

[1] Andrew Campbell, Geoff Coulson, Francisco Garcia, David Hutchison, and Helmut Leopold.

Integrated Quality of Service for Multimedia Communications. In *IEEE INFOCOM 93*, 1993.

[2] Shanwei Cen, Calton Pu, Richard Staehli, Crispin Cowan, and Jonathan Walpole. A Distributed Real-Time MPEG Video Audio Player. In *Proceedings of the 1995 International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'95)*, pages 151–162, New Hampshire, April 1995.

[3] Sally Floyd, Van Jacobson, Ching-Gung Liu, Steven McCanne, and Lixia Zhang. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. In *ACM SIGCOMM 95*, pages 342–356, 1995.

[4] Didier Le Gall. MPEG: A Video Compression Standard for Multimedia Applications. *CACM*, 34(4):46–58, April 1991.

[5] Ching-Chih Han and Kang G. Shin. Scheduling MPEG-Compressed Video Streams with Firm Deadline Constraints. In *Proceedings of the ACM Multimedia'95*, pages 411–422, San Francisco, CA, 1995.

[6] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *Proc. IEEE 10th Real-Time Systems Symp.*, pages 166–171, December 1989.

[7] Kwei-Jay Lin and Ching-Shan Peng. Scheduling Algorithms for Real-Time Agent Systems. In *Proceedings of the Sixth International Workshop on Research Issues in Data Engineering (RIDE'96)*, New Orleans, Louisiana, February 1996.

[8] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *J. ACM*, 20(1):46–61, January 1973.

[9] Steven McCanne and Martin Vetterli. Joint Source/Channel Coding for Multicast Packet Video. In *Proceedings of the IEEE International Conference on Image Processing*, Washington, DC, 1995.

[10] C. W. Mercer, S. Savage, and H. Tokuda. Processor Capacity Reserves: Operating System Support for Multimedia Applications. In *Proc. of the International Conference on Multimedia Computing and Systems*, pages 90–99, May 1994.

[11] R. Nagarajan and C. Vogt. Performance of Multimedia Traffic over the Token Ring. Tech. report, IBM-ENC, Heidelberg, 1992.

[12] K. Nahrstedt and J.M. Smith. Design, Implementation and Experiences of the OMEGA Architecture. Tech. report, University of Pennsylvania, May 1995.

[13] K. Nahrstedt and J.M. Smith. The QoS Broker. *IEEE Multimedia*, 2(1):53–67, Spring 1995.

[14] Jason Nieh, James G. Hanko, J. Duane Northcutt, and Gerard A. Wall. SVR4 UNIX Scheduler Unacceptable for Multimedia Applications. In *Proceedings of the 1993 International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'93)*, pages 35–47, Lancaster, U.K., November 1993.

[15] K.K. Ramakrishnan, Lev Vaitzblit, Cary Gray, Uresh Vahalia, Dennis Ting, Percy Tzelnic, Steve Glaser, and Wayne Duso. Operating Systems Support for a Video-On-Demand File Service. In *Proceedings of the 1993 International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'93)*, pages 225–236, Lancaster, U.K., November 1993.

[16] A. Silberschatz and P. Galvin. *Operating System Concepts*. Addison-Wesley, Reading, Massachusetts, 1994.

[17] Richard Staehli, Jonathan Walpole, and David Maier. Quality of Service Specifications for Multimedia Presentations. *Multimedia Systems*, 3(5/6):251–263, November 1995.

[18] John A. Stankovic. Misconceptions About Real-Time Computing. *IEEE Multimedia*, 1988.

[19] Ralf Steinmetz. Analyzing the Multimedia Operating System. *IEEE Multimedia*, 1995.

[20] Andreas Vogel, Brigitte Kerherve, Gregor von Bochmann, and Jan Gecsei. Distributed Multimedia and QoS: A Survey. *IEEE Multimedia*, May 1995.