Adaptive Bandwidth Reservation for Multimedia Computing

Luca Abeni
RETIS Lab
Scuola Superiore S. Anna, Pisa
luca@hartik.sssup.it

Giorgio Buttazzo
University of Pavia (Italy)
INFM - Pavia research unit
giorgio@sssup.it

Abstract

In this paper we present a framework for dynamically allocating the CPU resource to tasks whose execution times are not known a priori. Tasks are partitioned in three classes: the ones that require a uniform execution but do not impose any temporal constraint, periodic tasks that operate on Continuous Media, and event driven tasks that respond to external interrupts. For the last two classes, we show how to adjust the fraction of the CPU bandwidth assigned to each task using a feedback mechanism.

1. Introduction

Due to recent applications of real-time technology to multimedia computing, there is an increasing demand for real-time operating systems (OSs) capable of supporting traditional time-sharing applications together with multimedia ones.

One of the key features that the OS should have to handle multimedia applications is the ability to allocate system resources to tasks according to some Quality of Service (QoS) requirement. For example, considering the CPU allocation problem, an adequate amount of processing time has to be pre-allocated to tasks, in order to ensure that they will meet their timing constraints. Unfortunately, such a pre-allocation requires an a priori knowledge of the tasks' execution times.

In classical hard real-time systems the worst-case execution times are used to perform a guarantee test for admission control [12], whereas using a reservation strategy [1, 15] this knowledge is used to compute the fraction of the resource assigned to the task. In the first case, a wrong estimation of the execution time can jeopardize the entire system schedulability, while in the second case only the tasks whose execution times are wrongly estimated will be affected (temporal protection).

Reservations can be used to precisely allocate resources to tasks, and are widely used in static allocation strategies. For example, in [16], the authors propose a model that can be used to allocate resources to tasks in order to maximize some utility function. This method requires the a priori knowledge of tasks' execution times and of the utility that a task can achieve receiving a given amount of a resource. In [2], a method for allocating CPU time to tasks based on the probability to meet a deadline is presented. This method does not require the exact knowledge of tasks' execution times and can also be used if an approximate execution time distribution function is known. However, due to the lack of feedback, reservation is not adaptive and the method can only be used off-line.

Proportional Share (PS) scheduling [19, 8] is often proposed as an alternative to real-time scheduling algorithms, but it also requires the knowledge of execution times. Using a PS scheduler it is possible to ensure that a task will execute at a constant rate, but if the task execution time is unknown, it is impossible to impose a desired rate. In [18], the authors show how to dynamically adjust the weights based on tasks' execution times. In [3], the PS resource allocation strategy is compared with the resource reservation approach, and the two methods are shown to be similar.

Although an a priori knowledge of execution times allows to perform admission control, such an a priori knowledge is somewhat unrealistic in a multimedia environment: the same applications can run on hundreds of different systems and architectures, making impossible to estimate the execution times a priori. For this reason, we propose a dynamic adaptation of the reserved resources, based on some form of feedback mechanism.

A dynamic QoS adaptation is tested in [14, 7], while feedback control schemes are widely used in multimedia applications to scale resource utilization (for example, see [11]). In [14], the task periods are changed in order to prevent system overload, and the deadline miss event is used as feedback. In [5], tasks' utilizations are tuned through period changes in order to keep the processor load close to one. In [7], or more recently in [13], a processor capacity reserve mechanism is used to detect task overruns and construct a feedback mechanism, but the controlled variable is still the task period. In [4], the authors describe a QoS manager capable of adjusting task periods according to some user definable strategy. It uses an open loop to control the QoS, assuming that each task CPU requirement is known. All these works present some form of dynamic QoS adaptation, but the utilization is scaled only by changing the tasks' rates/periods.

In [6], the authors claim that tasks' QoS can be achieved also without OS support (i.e., without using real-time or PS resource allocation), if the applications shed their resource requirements based on system load. The proposed method uses the global resource usage as a feedback, but it is not clear how multiple applications can simultaneously adjust their load without obtain instability. We believe that an application can use some form of feedback to scale down its requirements, but the load adjustment must be a system responsibility.

In [11], the authors present a general framework to control the application requests for system resources using the amount of allocated resources as a feedback. In particular, they show that a Proportional Integrative Derivative (PID) controller can be used to bound tasks' resource usage in a stable and fair way.

A similar method is presented in [9], where the authors describe a scheduler providing resource reservations. The system can adjust the reserved time based on the execution times of the last task instances, but it is not clear how the system behaves in overload situations (when all the reservations require to be increased). This problem is common to other approaches concerning rate adaptation, such as [20], that presents a scheduling environment similar to the one proposed in this paper. This work, however, does not consider the case in which there is no free CPU bandwidth for task rates adaptation.

Recently, in [17], a feedback control mechanism has been considered for observing and adjusting the system workload to reduce the number of deadline misses when tasks' execution times are not precisely known.

In this paper, we propose a method for controlling the fraction of CPU bandwidth allocated to tasks whose execution times are not known a priori. The approach is similar to the one proposed in [11]. However, while in [11] the feedback is used to control the tasks' requests, in our approach it is used for achieving adaptive scheduling. A measure of the achieved QoS is used as a feedback to control the assigned resources (through a bandwidth reservation strategy). The task's QoS can be tuned by changing either the periods or the reserved CPU time.

The rest of the paper is organized as follows: Section 2 presents some task models suitable for multimedia applications; Section 3 describes a method for using the proposed models in a real-time system that provides a bandwidth reservation mechanism; Section 4 shows some experimental results; and Section 5 states our conclusions and future work.

2. Task Models

Real-time tasks are traditionally scheduled using fixed priorities (Rate or Deadline Monotonic), dynamic priorities (Earliest Deadline First), or using some form of Proportional Share, giving the programmer a direct visibility of low-level scheduling parameters such as priorities, Worst Case Execution Times (WCETs), deadlines, or weights.

In many cases, this is very different from what a user really needs, thus the programmer is forced to assign low-level parameters according to complex mapping functions. Moreover, this approach has the following disadvantages:

- the system schedulability strongly relies on the exact knowledge of WCETs, which cannot always be easily estimated;
- in some cases, tasks' parameters (e.g., the PS weights) do not have an easy interpretation, so the user can only assign them using heuristic rules;
- tasks' parameters are too low-level to support complex features, such as bandwidth adaptation.

The problems mentioned above can be addressed by introducing high-level task models providing an interface closer to the real needs. For example, in a multimedia environment, the following features can be identified for the application tasks:

- Each task is characterized by an importance value with respect to all the other tasks: when the system resources are not enough to fulfill each task request, the resources will be shared according to tasks' importance.
- Some tasks need to execute with a constant rate, without respecting any explicit time constraint.
- Some tasks need to execute periodically, respecting some time constraints (for example, each periodic instance should be served before the next activation).
- Some tasks need to respond to internal or external events, serving a minimum number of events per time unit.

Each task τ_i in the system is characterized by a weight w_i, representing the task's importance with respect to the others. Moreover, some tasks can be characterized by some *temporal constraints* (such as a period T_i). Based on these characteristics, we identify three classes of tasks (see Figure 1):

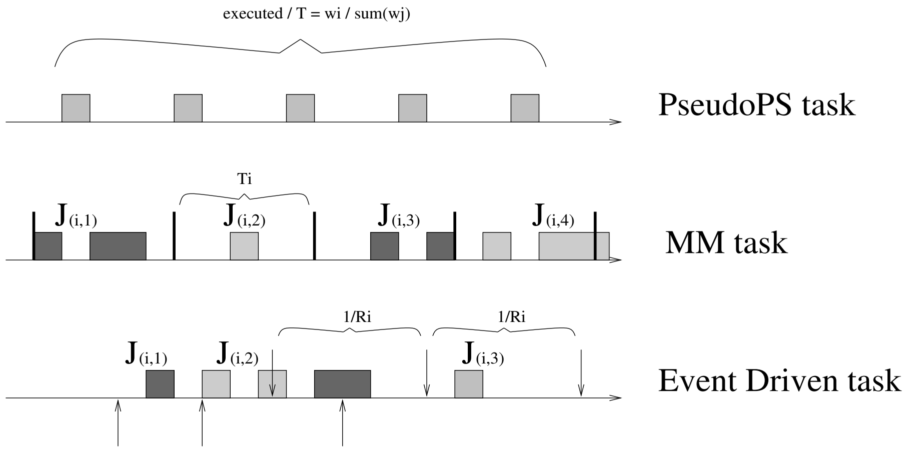

Figure 1. Example of the three classes of tasks.

- **PseudoPS (PPS) Tasks** $\tau_i = (w_i)$: they are flows of instructions that execute uniformly, receiving a processor share proportional to the task weight w_i.
- **MultiMedia (MM) Tasks** $\tau_i = (w_i, T_i)$: they are streams of jobs $J_{i,j}$ periodically activated with a period T_i, so that job $J_{i,j}$ arrives in the system at time $r_{i,j} = r_{i,j-1} + T_i$, and should finish before the next job starts. Using the real-time terminology, we say that $J_{i,j}$ has a soft deadline $d_{i,j} = r_{i,j+1} = r_{i,j} + T_i$. The objective of the system is to assign each task a fraction of the processor bandwidth sufficient to meet this requirement.
- **Event Driven (ED) Tasks** $\tau_i = (w_i, R_i)$: they are streams of aperiodic jobs $J_{i,j}$ activated by external or internal events. The user specifies the average number R_i of jobs that should be executed per time unit, and the objective of the system is to automatically adjust the fraction of processor bandwidth assigned to each task in order to meet this requirement.

PseudoPS Tasks are equivalent to tasks scheduled by a PS system: they will execute at a uniform rate, but, since their execution time is not known, no temporal constraint can be guaranteed, although a suitable (system dependent) tuning of the tasks' weights may allow to serve conventional applications in a timely fashion, without modifying them.

MM Tasks are designed to manage Continuous Media (CM) streams. Since they are composed of distinct jobs, the system can monitor each job's execution time in order to arrange the CPU bandwidth reserved to the task. Using this task model, the programmer has to specify the task period, but the task execution time does not need to be estimated.

Event Driven Tasks are similar to MM Tasks, in the sense that the programmer is not required to specify the task execution time: the only mandatory task parameter is the number R_i of jobs that must execute in a time unit. The difference with MM Tasks is that Event Driven Tasks are not periodic; the system can monitor the time difference between two consecutive activations, and use it to tune some scheduling parameters.

If the system is overloaded, and the CPU bandwidth is not sufficient to fulfill each task's requirement, an explicit bandwidth compression algorithm corrects the fraction of CPU bandwidth assigned to each task using the task weight w_i (the tasks with the higher weights will receive a bandwidth nearest to the requested one). This model permits to distinguish the task temporal constraint (the period T_i or the rate $R_i = 1/T_i$) from the task importance, expressed by the weight w_i. In facts, one of the biggest problems of classical real-time scheduling algorithms (such as Rate Monotonic or Earliest Deadline First) was that the task importance resulted to be proportional to the inverse of the task period.

3. Adaptive Bandwidth Reservation

In the previous section, we defined three task models, characterizing each task τ_i with a weight w_i, and characterizing time-sensitive tasks (MM and Event Driven Tasks) with a temporal parameter. In this section we describe how these parameters can be mapped to an adequate low-level scheduling algorithm.

In order to handle the proposed task models, we have to:

1. choose a suitable low-level **scheduling algorithm**;
2. **map** the task model's parameters to the scheduling algorithm parameters;
3. design a **feedback mechanism** to adjust the tasks' reserved bandwidth on line.

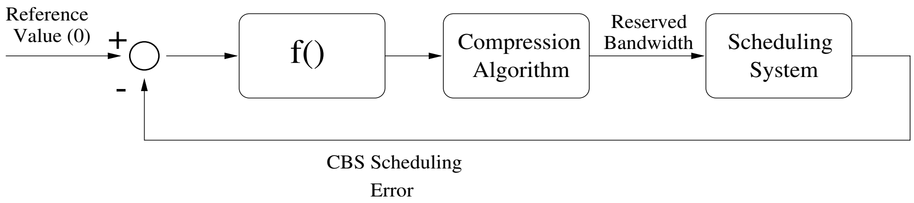

Figure 2. The control scheme.

We use the Earliest Deadline First (EDF) [12] algorithm as a low-level scheduler, since it achieves full CPU utilization, whereas the bandwidth reservation mechanism is realized by assigning each task a dedicated Constant Bandwidth Server (CBS), a service mechanism described in [1]. Since a CBS is characterized by a *maximum budget* Q_s and a *server period* T_s (being $B_s = \frac{Q_s}{T_s}$ the fraction of CPU bandwidth used by the server), the proposed task models must be mapped to such server parameters.

For PseudoPS tasks, Q_s can be considered as the *scheduling quantum*: it is assigned an initial default value of $10msec$, which can be changed using a specific system call. Each PPS task τ_i is assigned a bandwidth

$$B_i = \frac{w_i}{\sum_{j:\tau_j \in PPS} w_j} B_{PPS} \qquad (1)$$

where B_{PPS} is the total bandwidth assigned to PPS tasks. Thus, T_s is computed as $T_s = \frac{Q_s}{B_i}$. Equation 1 shows how the total PPS bandwidth is shared among the PPS tasks according to the tasks weights (it results to be identical to the share f_i in a PS system). In this way the fraction of CPU bandwidth B_{PPS} assigned to PPS tasks will be shared among them proportionally to the weights, like in a PS system (each PPS task will receive an amount Q_s of CPU time every T_s, so Q_s can be considered as the scheduling quantum in a conventional time sharing system, as said above).

For MM and Event Driven tasks, the T_s parameter is fixed and equal to $T_i = 1/R_i$, where R_i is the *task rate* specified by the user. The Q_s parameter is adjusted by the system in order to meet the tasks' temporal requirements. This can be done using a feedback: as **observed value** we choose the *CBS scheduling error*

$$\epsilon_s = d_s - (r_{i,j} + T_i)$$

defined as the difference between the CBS scheduling deadline d_s and the task's soft deadline.

Since the underlying scheduling algorithm is EDF, if the server is schedulable each instance is guaranteed to finish within the assigned server deadline d_s. Then, the CBS scheduling error ϵ_s represents the difference between the deadline d_s that the job *is guaranteed* to respect and the deadline that it *should* respect. When this value is 0, the task is guaranteed to respect its soft deadline, while when the scheduling error is different from 0, some task instance *can* terminate after its (soft) deadline because the reserved bandwidth $\frac{Q_s}{T_s}$ is not enough to serve the task within its soft deadline. Hence, the objective of the system is to control ϵ_e to 0: if this value increases, Q_s has to be increased accordingly, otherwise it can be left unchanged[1].

Based on this idea, a control mechanism $Q_s = f(\epsilon_s)$ can be built to adjust the server maximum budget Q_s. The choice of the control function $f()$ can be crucial to rapidly adjust the reserved bandwidth: in this work we used a simple feedback $\Delta Q_s = \frac{\epsilon_s}{T_i} * Q$ where Q is a system parameter (the default scheduling quantum), but more complex control strategies (such as a PID controller) are under investigation.

When the feedback mechanism increases the budgets "too much" and $\sum_{j:\tau_j \in MM \cup ED} B_j > B_{max}$, the tasks' reserved bandwidth have to be reduced to maintain the system schedulable. This is done using the tasks' weights.

A simple solution is to scale the tasks' utilizations in a proportional way:

$$B'_i = B_i s_i$$

being s_i the scaling factor. Since the compression must be done according to the tasks' weights, s_i must be proportional to w_i: $s_i = w_i M$. Imposing $\sum_{j:\tau_j \in MM \cup ED} B'_j = B_{max}$, we obtain:

$$\begin{aligned}
\sum_{j:\tau_j \in MM \cup ED} B'_j &= B_{max} \Rightarrow \\
\sum_{j:\tau_j \in MM \cup ED} B_j s_j &= B_{max} \Rightarrow \\
\sum_{j:\tau_j \in MM \cup ED} B_j w_j M &= B_{max} \Rightarrow \\
M \sum_{j:\tau_j \in MM \cup ED} B_j w_j &= B_{max} \Rightarrow \\
M &= \frac{B_{max}}{\sum_{j:\tau_j \in MM \cup ED} B_j w_j}
\end{aligned}$$

Hence,

$$B'_i = B_i w_i \frac{B_{max}}{\sum_{j:\tau_j \in MM \cup ED} B_j w_j} \qquad (2)$$

[1] Note that the budget Q_s is decreased only in overload conditions according to the specified task weights. This solution avoids unnecessary bandwidth adaptations.

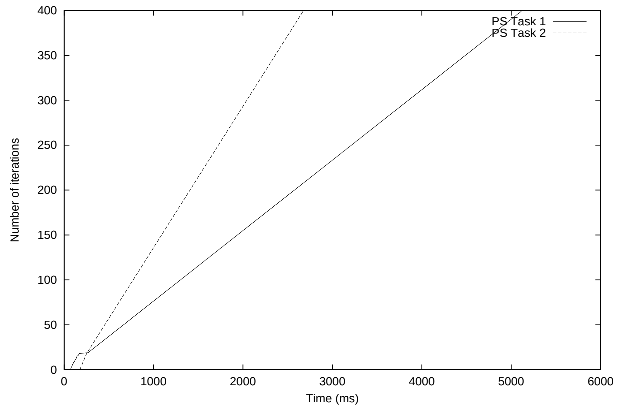

Figure 3. PSTasks throughputs.

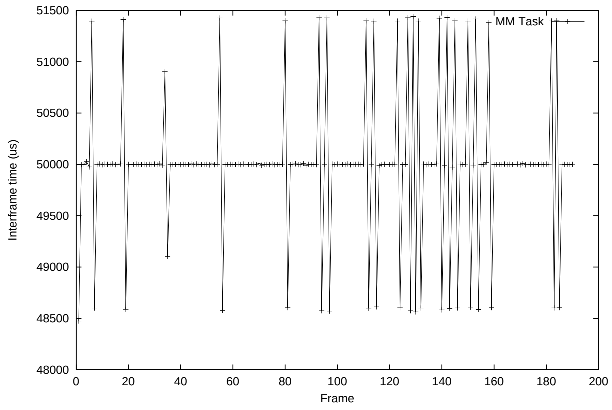

Figure 4. MM task inter start times.

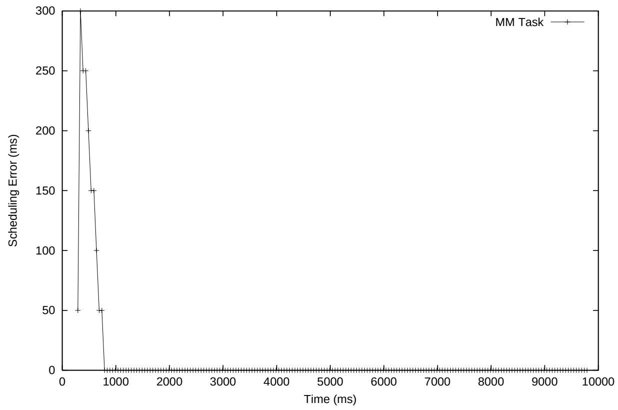

Figure 5. CBS scheduling error.

This simple rule can be slightly modified to guarantee a minimum bandwidth B_{min} to each task.

Tasks' utilizations can also be computed using the *Elastic Model* presented in [5]; in this case the task weight is considered as an elastic coefficient.

The described closed loop control used to adjust the reserved bandwidth is shown in Figure 2.

In order to experiment the effectiveness of the proposed approach, the mechanism described above has been implemented in the HARTIK real-time kernel [10]. A *QoS Manager* task is used to manage all the other task's parameters. When the QoS Manager is activated, it asks the system for all the free CPU bandwidth; then any task creation or termination is handled by the manager. In this way it can share the available CPU bandwidth among the various tasks as described above. When an MM or ED task completes one of its jobs, it signals this event to the QoS Manager, that evaluates the CBS scheduling error and adjusts the tasks' bandwidths.

4. Experimental Results

Several experiments have been conducted to test the performance of our adaptive bandwidth reservation algorithm. In the first experiment, we monitored the execution of PPS tasks to verify that they proceed according to the assigned weights. As an example, Figure 3 reports the execution of two PPS tasks, τ_1 and τ_2, with $w_2 = 2w_1$. It can be seen that τ_2 executes two times faster than τ_1.

Once verified that PPS tasks behave as expected, we tested the behavior of the QoS manager in handling MM tasks, by executing a JPEG player and verifying that it plays its video stream at a desired frame rate. In order to do this, we implemented an MM task that plays sequences of JPEG images with requested period $T = 50msec$ $(20Fps)$.

First, we tested the ability of the QoS Manager to handle a single MM task, executing the JPEG player together with two time consuming PPS activities to stress the temporal protection mechanism. The result of this experiment is reported in Figure 4, which shows the differences between the start times of two successive jobs. Since the task is periodic, such a difference should be constant and equal to the task period $T = 50msec$: as we can see, the plotted values result to be stable around the requested value T. Notice that the deviations with respect to the requested value are less than 3% of the period and are due to the scheduling effect.

Figure 5 shows the values of the controlled CBS scheduling error ϵ, during the same experiment (as said in Section 3 this value should go to 0). Since the task is assigned a very low initial bandwidth ($B = 0.02$) the scheduling error at the second job is quite high (about 6 times the period), but it reduces to 0 after 8 jobs.

Once we tested the correctness of the bandwidth adaptation with a single MM task, we performed a new experiment playing two video streams with two MM tasks, τ_1 and τ_2, with $T_1 = T_2 = 50msec$, $w_1 = 5$ and $w_2 = 10$. Since τ_2

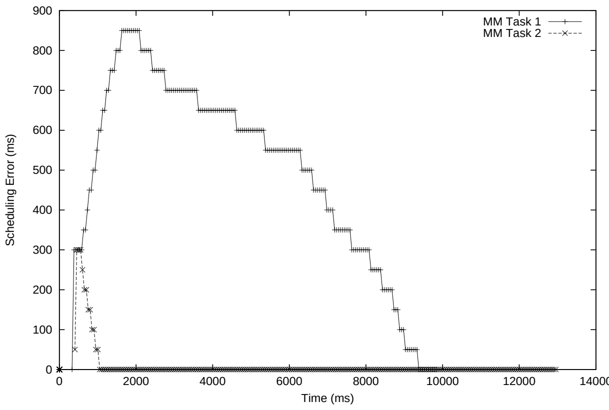

Figure 6. CBS scheduling errors.

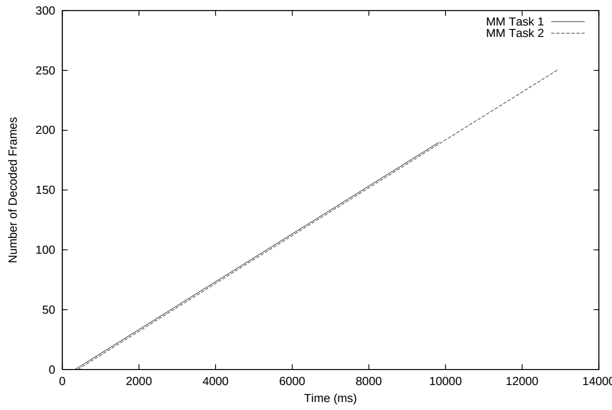

Figure 8. MM Tasks throughput.

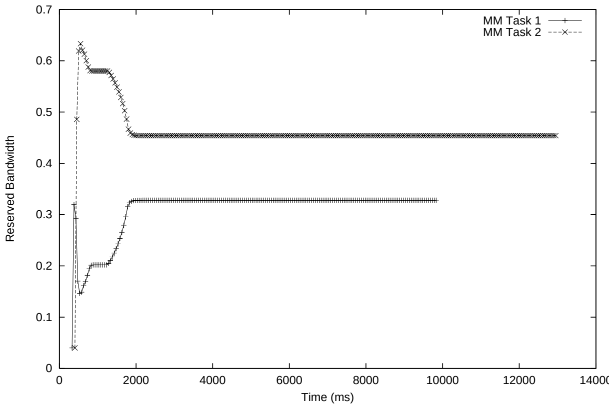

Figure 7. Reserved bandwidths.

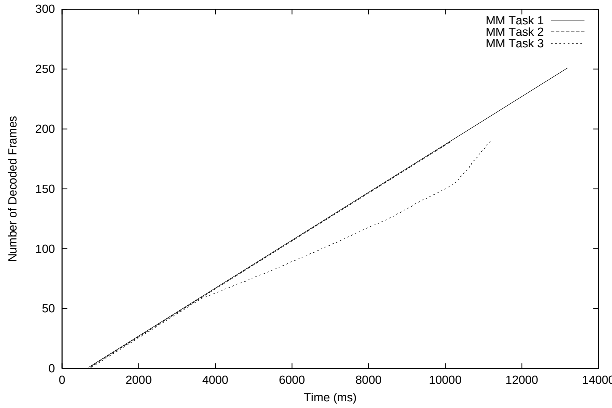

Figure 9. MM Tasks throughput.

is more important than τ_1 ($w_2 > w_1$), the CBS scheduling error of τ_2 goes to 0 more quickly than the scheduling error of τ_1 (see Figure 6). This is due to the fact that when the feedback mechanism tries to assign two "big" bandwidth values B_1 and B_2 ($B_1 + B_2 > B_{max}$), the compression algorithm described by equation 2 assigns τ_2 a bandwidth B_2 greater than B_1 (see Figure 7). It is worth noting that, since the system is not globally overloaded, the two streams are played at their requested rates and the bandwidth adaptation produces no negative effects in terms of number of decoded frames per second. This is illustrated in Figure 8, which represents the decoded frames as a function of time.

In the next experiment we tested the system behavior in overload conditions, by activating three MM tasks (JPEG decoders) with $T_1 = T_2 = T_3 = 50msec$, $w_1 = 5$, $w_2 = 10$ and $w_3 = 1$. As shown in Figure 9 (that represents the number of decoded frames as a function of time), the two most important tasks (τ_1 and τ_2) play the stream at the requested rate of $20Fps$, while τ_3 runs slower. The graphs representing τ_1 and τ_2 are identical to the ideal model (a line with slope of 20 frames per second), while the graph representing τ_3 deviates from it. This is due to the fact that Equation 2 assigns bandwidths proportional to weights, and w_3 is very small with respect to w_1 and w_2 (a ratio of $1/5$ with respect to w_2).

The inter-start times are reported in Figure 10, where we can see again that the only task experiencing a significant start-time jitter is τ_3, which has the lowest weight.

In Section 2, we stated that the MM Task model permits to separate the task rate from the task importance. We proved this fact with another experiment, in which two video streams are played by two MM tasks τ_1 and τ_2 with $T_1 = 50$, $T_2 = 33$, $w_1 = 5$, $w_2 = 10$. In this case, the worst-case load generated by the two tasks is greater than one, whereas the average load is very close to one: for this reason the system will be able to play the tasks with an average rate equal to the requested one, but they will experience a significant inter start-time jitter. The resulting inter-start times are shown in Figure 11. One more time, the graphs

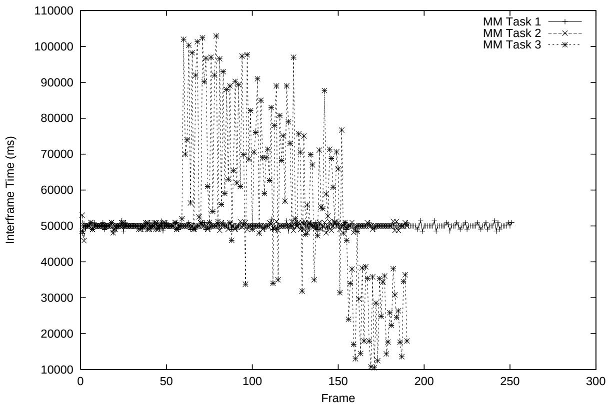

Figure 10. MM tasks inter start times.

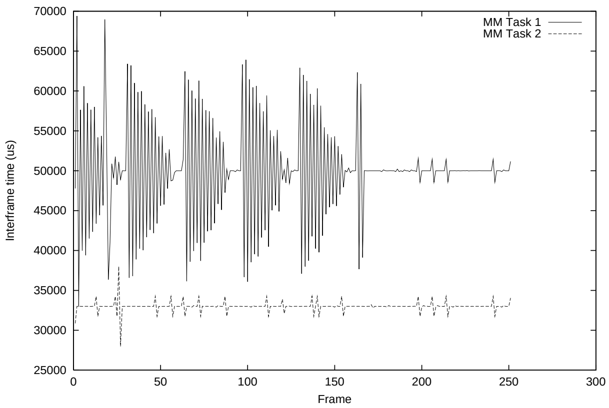

Figure 11. inter start times with $w_1 = 5, w_2 = 10$.

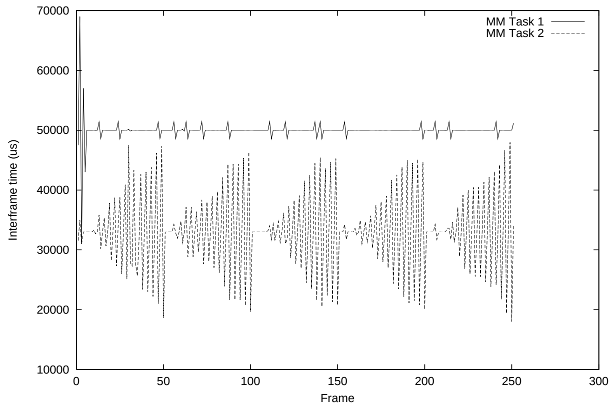

Figure 12. inter start times with $w_1 = 10, w_2 = 5$.

show how the task with the lowest weight (τ_1) is the one with the highest jitter.

In order to prove how the task importance can be controlled, we exchanged the tasks' weights, by assigning $w_1 = 10$ and $w_2 = 5$. The inter-start times obtained with the new weights are shown in Figure 12: in this case, the less important task is τ_2, and it experiences a jitter bigger than τ_1's one. By comparing Figures 11 and 12 we can see how the tasks' weights allow to control the QoS experienced by each single task independently by its period. Notice that this behavior represents a novel feature of our approach that cannot usually be achieved using classical real-time scheduling algorithms (such as Rate Monotonic or Earliest Deadline First), or PS scheduling schemes, where weights are not independent from task periods and execution times.

Similar experiments have been performed to test the behavior of ED tasks, but they are not reported due to space limitations.

5. Conclusions

In this paper we proposed three task models to support MultiMedia applications in a real-time environment, and we introduced an adaptive resource reservation algorithm for supporting the proposed framework.

From experimental validation, it results that the system fulfills its objective by:

- not requiring any a priori knowledge about tasks' execution times;
- being machine independent;
- separating time constraints from importance of a task.

As a future work, we plan to investigate more sophisticated feedback control schemes and bandwidth tuning algorithms. In particular, we are interested in experimenting a feedback control function that could be used to dynamically adjust the reserved bandwidth in order to allow a probabilistic deadline guarantee. Others experiments are planned to test how the Event Driven Task model can be used to manage hardware interrupts (such as the interrupts raised by a network card).

Moreover, since our approach supports a task model (i.e., PPS) which is similar to that considered in Proportional Share schemes, we intend to compare such task model with the adaptive task models through more specific experiments.

References

[1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the IEEE*

Real Time Systems Symposium, Madrid, Spain, December 1998.

[2] L. Abeni and G. Buttazzo. Qos guarantee using probabilistic dealines. In *Proceedings of the IEEE Euromicro Conference on Real-Time*, York, England, June 1998.

[3] L. Abeni and G. Buttazzo. Constant bandwidth vs proportional share resource allocation. In *Proceedings of the IEEE International Conference on Mutimedia Computing and Systems*, Florence, Italy, June 1999.

[4] D. Aparah. Adaptive resource management in a multimedia operating system. In *Proceedings of the 8th International Workshop on Network and Operating System Support for Digital Audio and Video*, Cambridge, UK, July 1998.

[5] G. Buttazzo, L. Abeni, and G. Lipari. Elastic task model for adaptive rate control. In *Proceedings of the IEEE Real Time Systems Symposium*, Madrid, Spain, December 1998.

[6] C. L. Compton and D. L. Tennenhouse. Collaborative load shedding for media-based applications. In *Proceedings of the International Conference on Multimedia Computing and Systems*, 1994.

[7] H. Fujita, T. Nakajima, and H. Tezuka. A processor reservation system supporting dynamic qos control. In *2nd International Workshop on Real-Time Computing Systems and Applications*, October 1995.

[8] P. Goyal, X. Guo, and H. M. Vin. A hierarchical cpu scheduler for multimedia operating systems. In *2nd OSDI Symposium*, October 1996.

[9] H. hua Chu and K. Nahrstedt. CPU service classes for multimedia applications. In *Proceedings of the IEEE International Conference on Mutimedia Computing and Systems*, Florence, Italy, June 1999.

[10] G. Lamastra, G. Lipari, G. Buttazzo, A. Casile, and F. Conticelli. Hartik 3.0: A portable system for developing realtime applications. In *Proceedings of the IEEE Conference on Real-Time Computing Systems and Applications*, October 1997.

[11] B. Li and K. Nahrstedt. A control theoretical model for quality of service adaptations. In *Proceedings of Sixth International Workshop on Quality of Service*, 1998.

[12] C. L. Liu and J. Layland. Scheduling alghorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1), 1973.

[13] T. Nakajima. Resource reservation for adaptive qos mapping in real-time mach. In *Sixth International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, April 1998.

[14] T. Nakajima and H. Tezuka. A continuous media application supporting dynamic qos control on real-time mach. In *ACM Multimedia*, 1994.

[15] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa. Resource kernels: A resource-centric approach to real-time and multimedia systems. In *Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking*, January 1998.

[16] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. A resource allocation model for qos management. In *Proceedings of the IEEE Real Time Systems Symposium*, 1997.

[17] J. A. Stankovic, C. Lu, and S. H. Son. The case for feedback control in real-time scheduling. In *Proceedings of the IEEE Euromicro Conference on Real-Time*, York, England, June 1998.

[18] I. Stoica, H. Abdel-Wahab, and K. Jeffay. On the duality between resource reservation and proportional share resource allocation. In *SPIE Multimedia Computing and Networking*, volume 3020, pages 207–214, San Jose, CA, February 1997.

[19] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. K. Baruah, J. E. Gehrke, and C. G. Plaxton. A proportional share resource allocation algorithm for real-time, time-shared systems. In *Proceedings of the IEEE Real Time Systems Symposium*, December 1996.

[20] D. K. Y. Yau and S. S. Lam. Adaptive rate controlled scheduling for multimedia applications. *IEEE/ACM Transactions on Networking*, August 1997.