Integrating Multimedia Applications in Hard Real-Time Systems

Luca Abeni and Giorgio Buttazzo
Scuola Superiore S. Anna, Pisa
luca@hartik.sssup.it, giorgio@sssup.it

Abstract

This paper focuses on the problem of providing efficient run-time support to multimedia applications in a real-time system, where two types of tasks can coexist simultaneously: multimedia soft real-time tasks and hard real-time tasks. Hard tasks are guaranteed based on worst case execution times and minimum interarrival times, whereas multimedia and soft tasks are served based on mean parameters. The paper describes a server-based mechanism for scheduling soft and multimedia tasks without jeopardizing the a priori guarantee of hard real-time activities. The performance of the proposed method is compared with that of similar service mechanisms through extensive simulation experiments and several multimedia applications have been implemented on the HARTIK kernel.

1. Introduction

Continuous Media (CM) activities, such as audio and video streams, need real-time support because of their sensitivity to delay and jitter. On the other hand, however, the use of a hard real-time system for handling CM applications can be inappropriate for the following reasons:

- If a multimedia task manages compressed frames, the time for coding/decoding each frame can vary significantly, hence the worst case execution time (WCET) of the task can be much bigger than its mean execution time. Since hard real-time tasks are guaranteed based on their WCET (and not based on mean execution times), CM applications can cause a waste of the CPU resource.
- Providing a precise estimation of WCETs is very difficult even for those applications always running on the same hardware. This problem is even more critical for multimedia applications, which in general can run on a large number of different machines (think of a video conferencing system running on several different PC workstations).
- When data are received from an external device (for instance, a communication network) the interarrival time of the tasks that process such data may not be deterministic, so it may be impossible to determine a minimum interarrival time for such tasks. As a consequence, no a priori guarantee can be performed.
- Advanced multimedia systems tend to be more dynamic than classical real-time systems, so all the scheduling methodologies devised for static real-time systems are not suited for CM applications.

For the reasons mentioned above, a large part of the multimedia community continues to use classical operating systems, as Unix or Windows, to manage CM. Recently, some scheduling algorithms have been proposed [16, 6] to mix some form of real-time support with a notion of fairness, but they do not make use of conventional real-time theory. Since we are interested in systems based on a conventional RT scheduler (such as EDF or RM), we do not consider this kind of solutions.

In [8], Jeffay presents a hard real-time system based on EDF scheduling to be used as a test bed for video conference applications; the system can guarantee each task at its creation time based on its WCET and its minimum interarrival time. While a bound for the WCET can be found, the interarrival time may not have a lower bound, because of the unpredictability of the network (which may even reverse the order of messages at the reception site). For this reason, Jeffay in [7] introduces the Rate-Based Execution (RBE) task model, which is independent from the minimum interarrival time. Although this kind of task cannot be guaranteed to complete within a given deadline, it is possible to guarantee that it will not jeopardize the schedulability of other hard real-time tasks present in the system.

In [12], Mercer, Savage, and Tokuda propose a scheme based on CPU capacity reserves, where a fraction of the CPU bandwidth is reserved to each task. A reserve is a couple (C_i, T_i) indicating that a task τ_i can execute for at most C_i units of time in each period T_i. This approach removes the need of knowing the WCET of each task, because it fixes the maximum time that each task can execute in its period. Since the periodic scheduler is based on the Rate Monotonic algorithm, the classical schedulability analysis can be applied to guarantee hard tasks, if they are present. The only problem with this method is that overload situations on multimedia tasks are not handled efficiently. In fact, if a task instance executes for more than C_i units of time, the remaining portion of the instance is scheduled in background, prolonging its completion of an unpredictable time.

In [9], Kaneko et al. propose a scheme based on a periodic process (the multimedia server) dedicated to the service of all multimedia requests. This allows to nicely integrate multimedia tasks together with hard real-time tasks; however, being the server only one, it is not easy to control the QoS of each task.

In [3], Liu and Deng describe a scheduling hierarchy which allows hard real-time, soft real-time, and non real-time applications to coexist in the same system, and to be created dynamically. According to this approach, which uses the EDF scheduling algorithm as a low-level scheduler, each application is handled by a dedicated server, which can be a Constant Utilization Server [4] for tasks that do not use nonpreemptable sections or global resources, and a Total Bandwidth Server [13, 15] for the other tasks. This solution can be used to isolate the effects of overloads at the application level, rather than at the task level. Moreover, the method requires the knowledge of the WCET even for soft and non real-time tasks.

In this paper, we propose a scheduling methodology based on reserving a fraction of the processor bandwidth to each task (in a way similar to processor capacity reserves of Mercer et al.[12]). However, to efficiently handle the problem of task overloads, each task is scheduled by a dedicated server, which does not require the knowledge of the WCET and assigns a suitable deadline to the served task whenever the reserved time is consumed.

The rest of the paper is organized as follows: Section 2 specifies our notation, definitions and basic assumptions; Section 3 describes our scheduling scheme in detail and its formal properties; Section 4 compares the proposed algorithm with other server mechanisms, and presents some simulation results; Section 5 describes an implementation of the proposed algorithm on the HARTIK kernel and shows some experimental results; and, finally, Section 6 presents our conclusions and future work.

2. Terminology and assumptions

We consider a system consisting of three types of tasks: hard, soft, and non real-time tasks. Any task τ_i consists of a sequence of jobs $J_{i,j}$, where $r_{i,j}$ denotes the arrival time (or request time) of the j^{th} job of task τ_i.

A hard real-time task is characterized by two additional parameters, (C_i, T_i), where C_i is the WCET of each job and T_i is the minimum interarrival time between successive jobs, so that $r_{i,j+1} \geq r_{i,j} + T_i$. The system must provide an a priori guarantee that all jobs of a hard task must complete before a given deadline $d_{i,j}$. In our model, the absolute deadline of each hard job $J_{i,j}$ is implicitly set at the value $d_{i,j} = r_{i,j} + T_i$.

A soft real-time task is also characterized by the parameters (C_i, T_i), however the timing constraints are more relaxed. In particular, for a soft task, C_i represents the *mean* execution time of each job, whereas T_i represents the *desired* activation period between successive jobs. For each soft job $J_{i,j}$, a soft deadline is set at time $d_{i,j} = r_{i,j} + T_i$. Since mean values are used for the computation time and minimum interarrival times are not known, soft tasks cannot be guaranteed a priori. In multimedia applications, soft deadline misses may decrease the QoS, but do not cause critical system faults.

The objective of the system is to minimize the mean tardiness of soft tasks, without jeopardizing the schedulability of the hard tasks. The tardiness $E_{i,j}$ of a job $J_{i,j}$ is defined as

$$E_{i,j} \ = \ max\{0, \ f_{i,j} - d_{i,j}\} \tag{1}$$

where $f_{i,j}$ is the finishing time of job $J_{i,j}$.

Finally, a periodic task is a task (hard or soft) in which the interarrival time between successive jobs is exactly equal to T_i for all jobs ($r_{i,j+1} = r_{i,j} + T_i$). Periodic tasks do not have special treatment in this model.

Tasks that manage CM can be modeled as soft real-time tasks, because missing deadlines may decrease the QoS without causing catastrophic consequences. Moreover, CM activities are typically characterized by highly variable execution times, causing the WCET to be much greater than the mean execution time.

For the reasons mentioned above, treating CM tasks as hard real-time tasks is not appropriate, firstly because an underestimation of the WCET would compromise the guarantee done on the other tasks, and secondly because it would be very inefficient, since trying to guarantee a task with a WCET much greater than its mean execution time would cause a waste of the CPU resource.

This problem can be solved by a bandwidth reservation strategy, which assigns each soft task a maximum bandwidth, calculated using the mean execution time and the desired activation period, in order to increase CPU utilization. If a task needs more than its reserved bandwidth, it may slow down, but it will not jeopardize the schedulability of the hard real-time tasks. By isolating the effects of task overloads, hard tasks can be guaranteed using classical schedulability analysis [11].

To integrate hard and soft tasks in the same system, hard tasks are scheduled by the EDF algorithm based on their absolute deadlines, whereas each soft task is handled by a dedicated server, the *Constant Bandwidth Server* (CBS), whose behavior and properties are described in the next section.

3. The Constant Bandwidth Server

The service mechanisms that have inspired this work are the Dynamic Sporadic Server (DSS) [13, 5] and the Total Bandwidth Server (TBS) [13, 15]. As the DSS, the CBS guarantees that, if U_s is the fraction of processor time assigned to a server (i.e., its bandwidth), its contribution to the total utilization factor is no greater than U_s, even in the presence of overloads. Notice that this property is not valid for a TBS, nor for a Constant Utilization Server (CUS) [4], whose actual contributions are limited by U_s only under the assumption that all the served jobs execute no more than the declared WCET. With respect to the DSS, however, the CBS shows a much better performance, comparable with the one achievable by a TBS.

3.1. Definition of CBS

The CBS can be defined as follows:

- A CBS is characterized by a budget c_s and by a ordered pair (Q_s, T_s), where Q_s is the maximum budget and T_s is the period of the server. The ratio $U_s = Q_s/T_s$ is denoted as the server bandwidth. At each instant, a fixed deadline $d_{s,k}$ is associated with the server. At the beginning $d_{s,0} = 0$.
- Each served job $J_{i,j}$ is assigned a dynamic deadline $d_{i,j}$ equal to the current server deadline $d_{s,k}$.
- Whenever a served job executes, the budget c_s is decreased by the same amount.
- When $c_s = 0$, the server budget is recharged to the maximum value Q_s and a new server deadline is generated as $d_{s,k+1} = d_{s,k} + T_s$. Notice that there are no finite intervals of time in which the budget is equal to zero.
- A CBS is said to be active at time t if there are pending jobs (remember the budget c_s is always greater than 0); that is, if there exists a served job $J_{i,j}$ such that $r_{i,j} \leq t < f_{i,j}$. A CBS is said to be idle at time t if it is not active.
- When a job $J_{i,j}$ arrives and the server is active the request is enqueued in a queue of pending jobs according to a given (arbitrary) non-preemptive discipline (e.g., FIFO).
- When a job $J_{i,j}$ arrives and the server is idle, if $c_s \geq (d_{s,k} - r_{i,j})U_s$ the server generates a new deadline $d_{s,k+1} = r_{i,j} + T_s$ and c_s is recharged to the maximum value Q_s, otherwise the job is served with the last server deadline $d_{s,k}$ using the current budget.
- When a job finishes, the next pending job, if any, is served using the current budget and deadline. If there are no pending jobs, the server becomes idle.
- At any instant, a job is assigned the last deadline generated by the server.

Figure 1 illustrates an example in which a hard periodic task, τ_1, is scheduled together with a soft task, τ_2, served by a CBS having a budget $Q_s = 2$ and a period $T_s = 7$. The first job of τ_2 arrives at time $r_1 = 2$, when the server is idle. Being $c_s \geq (d_{s,0} - r_1)U_s$, the job is assigned the deadline $d_{s,1} = r_1 + T_s = 9$ and c_s is recharged at $Q_s = 2$. At time $t_1 = 6$, the budget is exhausted, so a new deadline $d_{s,2} = d_{s,1} + T_s = 16$ is generated and c_s is replenished. At time r_2, the second job arrives when the server is active, so the request is enqueued. When the first job finishes, the second job is served with the actual server deadline ($d_{s,2} = 16$). At time $t_2 = 12$, the server budget is exhausted so a new server deadline $d_{s,3} = d_{s,2} + t_s = 23$ is generated and c_s is replenished to Q_s. The third job arrives at time $r_3 = 17$, when the server is idle and $c_s = 1 < (d_{s,3} - r_3)U_s = (23 - 17)\frac{2}{7} = 1.71$, so it is scheduled with the actual server deadline $d_{s,3}$ without changing the budget.

It is worth to notice that under a CBS a job J_j is assigned an absolute time-varying deadline d_j which can be postponed if the task requires more than the reserved bandwidth. Thus, each job J_j can be thought as consisting of a number of chunks $H_{j,k}$, each characterized by a release time $a_{j,k}$ and a fixed deadline $d_{j,k}$. An example of chunks produced by a CBS is shown in Figure 2. To simplify the notation, we will indicate all the chunks generated by a server with an increasing index k (in the example of Figure 2, $H_{1,1} = H_1$; $H_{1,2} = H_2$, $H_{2,1} = H_3$ and so on).

In order to provide a formal definition of the CBS, let a_k and d_k be the release time and the deadline of the k^{th} chunk generated by the server, and let c and n be the actual server budget and the number of pending requests in the server queue (including the request currently being served). These variables are initialized as follows:

$$d_0 = 0 \quad c = 0 \quad n = 0 \quad k = 0$$

Using this notation, the server behavior can be described by the algorithm shown in Figure 3.

3.2. CBS properties

The proposed CBS service mechanism presents some interesting properties that make it suitable for supporting CM applications. The most important one, the *isolation property*, is formally expressed by the following theorem.

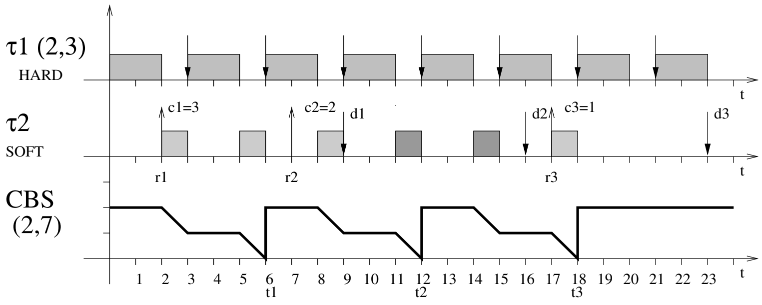

Figure 1. An example of CBS scheduling.

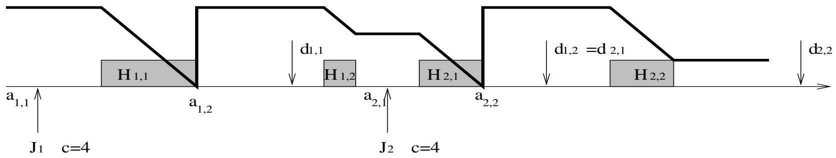

Figure 2. Example of jobs divided to chunks.

Theorem 1 *Given a set of n periodic hard tasks with processor utilization U_p and a CBS with processor utilization U_s, the whole set is schedulable by EDF* if *and* only if

$$U_p + U_s \leq 1.$$

Proof.
See [1]. □

The isolation property allows us to use a bandwidth reservation strategy to allocate a fraction of the CPU time to soft tasks whose computation time cannot be easily bounded. The most important consequence of this result is that such tasks can be scheduled together with hard tasks without affecting the a priori guarantee, even in the case in which soft requests exceed the expected load.

In addition to the isolation property, the CBS has the following characteristics.

- The CBS behaves as a plain EDF if the served task τ_i has parameters (C_i, T_i) such that $C_i \leq Q_s$ and $T_i = T_s$. This is formally stated by the following lemma.

  Lemma 1 *A hard task τ_i with parameters (C_i, T_i) is schedulable by a CBS with parameters $Q_s \geq C_i$ and $T_s = T_i$ if and only if τ_i is schedulable with EDF.*

  Proof.
  For any job of a hard task we have that $r_{i,j+1} - r_{i,j} = T_i$ and $c_{i,j} \leq Q_s$. Hence, by definition of the CBS, each hard job is assigned a deadline $d_{i,j} = r_{i,j} + T_i$ and it is scheduled with a budget $Q_s \geq C_i$. Moreover, since $c_{i,j} \leq Q_s$, each job finishes no later than the budget is exhausted, hence the deadline assigned to a job does not change and is exactly the same as the one used by EDF. □

- The CBS automatically reclaims any spare time caused by early completions. This is due to the fact that whenever the budget is exhausted, it is always immediately replenished at its full value and the server deadline is postponed. In this way, the server remains eligible and the budget can be exploited by the pending requests with the current deadline. *This is the main difference with respect to the processor capacity reserves* proposed by Mercer et al. [12].

- Knowing the statistical distribution of the computation time of a task served by a CBS, it is possible to perform a statistical guarantee, expressed in terms of probability for each served job to meet its deadline.

3.3. Statistical guarantee

To perform a statistical guarantee on soft tasks served by CBS, we can model a CBS as a queue, where each arriving job $J_{i,j}$ can be viewed as a request of $c_{i,j}$ time units. At any time, the request at the head of the queue is served using the current server deadline, so that it is guaranteed that Q_s units of time can be consumed within this deadline.

We analyze the following cases: a) variable computation time and constant inter-arrival time; and b) constant computation time and variable inter-arrival time.

```
When job J_j arrives at time r_j
     enqueue the request in the server queue;
     n = n + 1;
     if (n == 1) /* (the server is idle) */
       if (r_j + (c / Q_s) * T_s >= d_k)
         /*---------------Rule 1---------------*/
         k = k + 1;
         a_k = r_j;
         d_k = a_k + T_s;
         c = Q_s;
       else
         /*---------------Rule 2---------------*/
         k = k + 1;
         a_k = r_j;
         d_k = d_{k-1};
         /* c remains unchanged */
When job J_j terminates
     dequeue J_j from the server queue;
     n = n - 1;
     if (n != 0) serve the next job in the queue with deadline d_k;
When job J_j served by S_s executes for a time unit
     c = c - 1;
When (c == 0)
     /*---------------Rule 3---------------*/
     k = k + 1;
     a_k = actual_time();
     d_k = d_{k-1} + T_s;
     c = Q_s;
```

Figure 3. The CBS algorithm.

<u>*Case a.*</u>

If job interarrival times are constant and equal to T_s, and job execution times are randomly distributed with a given probability distribution function, the CBS can be modeled with a $D^G/D/1$ queue: every T_s units of time, a request of c_j units arrives and at most Q_s units can be served. We can define a random process v_j as follows:

$$\begin{cases} v_1 & = & c_1 \\ v_j & = & max\{0, v_{j-1} - Q_s\} + c_{i,j} \end{cases}$$

where v_j indicates the length of the queue (in time units) at time $(j-1)T_s$, that is the unit of times that are still to be served when job $J_{i,j}$ arrives. Hence, since Q_s units of time are served every period T_s, the job will finish no later than

$$d_{j_{max}} = r_{i,j} + \left\lceil \frac{v_j}{Q_s} \right\rceil T_s$$

which is also the latest deadline assigned by the server to job $J_{i,j}$.

If $\pi_k^{(j)} = P\{v_j = k\}$ is the state probability of process v_j and $C_h = P\{c_j = h\}$ is the probability that an arriving job requires h units of time (since c_j is time invariant, C_h does not depend on j), the value of $\pi_k^{(j)}$ can be calculated as follows:

$$\pi_k^{(j)} = P\{v_j = k\} = P\{max\{v_{j-1} - Q, 0\} + c_j = k\}$$

$$\pi_k^{(j)} = \sum_{h=-\infty}^{\infty} P\{max\{v_{j-1} - Q, 0\} + c_j = k \wedge v_{j-1} = h\}.$$

Being v_j greater than 0 by definition, the sum can be calculated for h going from 0 to infinity:

$$\pi_k^{(j)} = \sum_{h=0}^{\infty} P\{max\{h - Q, 0\} + c_j = k\} P\{v_{j-1} = h\}$$

$$= \sum_{h=0}^{Q} C_k \pi_h^{(j-1)} + \sum_{h=Q+1}^{\infty} P\{c_j = k - h + Q\} \pi_h^{(j-1)}$$

$$= \sum_{h=0}^{Q} C_k \pi_h^{(j-1)} + \sum_{h=Q+1}^{\infty} C_{k-h+Q} \pi_h^{(j-1)}.$$

Hence

$$\pi_k^{(j)} = \sum_{h=0}^{Q} C_k \pi_h^{(j-1)} + \sum_{h=Q+1}^{\infty} C_{k-h+Q} \pi_h^{(j-1)}. \quad (2)$$

Using a matrix notation, equation (2) can be written as

$$\Pi^{(j)} = M \Pi^{(j-1)} \quad (3)$$

where M and Π are described in Figure 4

$$M = \begin{pmatrix} \overbrace{C_0 \quad C_0 \quad . \quad . \quad . \quad C_0}^{Q+1} & 0 & 0 & . & . & . \\ C_1 \quad C_1 \quad . \quad . \quad . \quad C_1 & C_0 & 0 & . & . & . \\ C_2 \quad C_2 \quad . \quad . \quad . \quad C_2 & C_1 & C_0 & 0 & . & . \\ . \quad\quad . \quad\quad\quad\quad\quad . & . & . & . & & \end{pmatrix} \text{ and } \Pi^{(j)} = \begin{pmatrix} \pi_0^{(j)} \\ \pi_1^{(j)} \\ \pi_2^{(j)} \\ . \end{pmatrix}$$

Figure 4. Matrix describing the Markov chain for case a)

<u>*Case b.*</u>

In the case in which jobs' execution times are constant and equal to Q_s ($\forall j, c_{i,j} = Q_s$) and jobs' interarrival times are distributed according to a given distribution function, each job is assigned a deadline $d_{i,j} = \max\{r_{i,j}, d_{i,j-1}\} + T_s$, identical to that assigned by a TBS. In this situation, the CBS can be modeled by a $G/D/1$ queue: jobs arrive in the queue with a randomly distributed arrival time and the server can process a request each T_s time units. If we define a random process w_j as $w_j = d_{i,j} - r_{i,j} - T_s$, the distribution of the relative deadlines $d_{i,j} - r_{i,j}$ of job $J_{i,j}$ can be computed from the distribution of w_j, because

$$d_{i,j} - r_{i,j} = w_{i,j} + T_s.$$

Since $d_{i,j} = max\{r_{i,j}, d_{i,j-1}\} + T_s$, we have

$$\begin{aligned} w_{j+1} &= d_{i,j+1} - T_s - r_{i,j+1} = \\ &= max\{r_{i,j+1}, d_{i,j}\} + T_s - T_s - r_{i,j+1} = \\ &= max\{0, d_{i,j} - r_{i,j+1}\} = \\ &= max\{0, r_{i,j} + w_j + T_s - r_{i,j+1}\} = \\ &= max\{0, w_j - a_{j+1} + T\} \end{aligned}$$

having defined $a_{j+1} = r_{i,j+1} - r_{i,j}$. Being a_j a stochastic stationary and time invariant process and w_j a Markov process, the matrix M describing the w_j Markov chain can be found. By defining $\pi_k^{(j)} = P\{w_j = k\}$ and $A_h = P\{a_j = h\}$, we have

$$\pi_k^{(j)} = P\{w_j = k\} =$$

$$= P\{\max\{0, w_{j-1} - a_j + T_s\} = k\}$$

$$= \sum_{h=-\infty}^{\infty} P\{\max\{0, w_{j-1} - a_j + T\} = k \wedge w_{j-1} = h\}$$

$$= \sum_{h=-\infty}^{\infty} P\{\max\{0, h - a_j + T\} = k\} P\{w_{j-1} = h\}$$

In order to simplify the calculus, we distinguish two cases: $k = 0$ and $k > 0$:

$$\begin{aligned} \pi_0^{(j)} &= \sum_{h=-\infty}^{\infty} P\{h - a_j + T \le 0\} P\{w_{j-1} = h\} = \\ &= \sum_{h=-\infty}^{\infty} P\{a_j \ge h + T\} P\{w_{j-1} = h\} = \\ &= \sum_{h=0}^{\infty} \sum_{r=h+T}^{\infty} P\{a_j = r\} \pi_h^{(j-1)} = \\ &= \sum_{h=0}^{\infty} \sum_{r=h+T}^{\infty} A_r \pi_h^{(j-1)} \end{aligned}$$

$\forall k > 0,$

$$\pi_k^{(j)} = \sum_{h=-\infty}^{\infty} P\{h - a_j + T = k\} P\{w_{j-1} = h\} =$$

$$= \sum_{h=-\infty}^{\infty} P\{a_j = h - k + T\} \pi_h^{(j-1)} =$$

$$= \sum_{h=0}^{\infty} A_{h-k+T} \pi_h^{(j-1)}$$

Thus, matrix M describing the Markov chain is shown in Figure 5. For a generic queue, it is known that the queue is stable (i.e., the number of elements in the queue do not diverge to infinity) if

$$\rho = \frac{mean\ interarrival\ rate}{mean\ service\ rate} < 1.$$

Hence, the stability can be achieved under the following conditions:

$$\begin{cases} \overline{c_{i,j}} < Q_s & \text{in case a)} \\ \overline{r_{i,j+1} - r_{i,j}} > T_s & \text{in case b)} \end{cases}$$

In general,

$$\frac{\overline{c_{i,j}}}{\overline{r_{i,j+1} - r_{i,j}}} < \frac{Q_s}{T_s}.$$

If this condition is not satisfied the difference between the deadline $d_{i,j}$ assigned by the server to a job $J_{i,j}$ and the job release time $r_{i,j}$ will increase indefinitely. This means that, for preserving the schedulability of the other tasks, τ_i will slow down in an unpredictable manner.

If a queue is stable, a stationary solution of the Markov chain describing the queue can be found; that is, there exists

$$M = \begin{pmatrix} \rho_0 & \rho_1 & \rho_2 & . & . & . & . & . & . & . \\ A_{T+1} & A_T & A_{T-1} & . & . & A_0 & 0 & 0 & . & . & . \\ A_{T+2} & A_{T+1} & A_T & A_{T-1} & . & A_1 & A_0 & 0 & 0 & . & . \\ A_{T+3} & A_{T+2} & A_{T+1} & A_T & . & A_2 & C_1 & A_0 & 0 & 0 & . \\ . & . & . & . & & . & . & . & . & . & . \end{pmatrix} \text{ with } \rho_i = \sum_{r=i+T}^{\infty} A_r.$$

Figure 5. Matrix describing the Markov chain for case b)

a solution Π such that $\Pi = \lim_{j\to\infty} \Pi^{(j)}$, and $\Pi = M\Pi$. This solution can be approximated by truncating matrix M (having infinite dimension) to an $N \times N$ matrix M' and solving the eigenvector problem $\Pi' = M'\Pi'$ with some numerical calculus technique.

The knowledge of the probability distribution function of the relative deadlines before which a multimedia task job is guaranteed to finish is useful for guaranteeing a QoS to each task and for choosing the right server parameters (Q_s, T_s) for each soft task.

4. Simulation results

In this section we compare the CBS with other similar service mechanisms, namely the Total Bandwidth Server (TBS) and the Dynamic Sporadic Server (DSS). The Constant Utilization Server (CUS) is not considered in the graphs because it is very similar to the TBS (indeed, slightly worse in performance).

The main difference between DSS and CBS is visible when the budget is exhausted. In fact, while the DSS becomes idle until the next replenishing time (that occurs at the server's deadline), the CBS remains eligible by increasing its deadline and replenishing the budget immediately. This difference in the replenishing time, causes a big difference in the performance offered by the two servers to soft real-time tasks. The TBS does not suffer from this problem, however *its correct behavior relies on the exact knowledge of job's WCETs, so it cannot be used for supporting CM applications*. Moreover, since the CBS automatically reclaims any available idle time coming from early completions, a reclaiming mechanism has also been added in the simulation of the TBS, as described in [14].

All the simulations presented in this section have been conducted on a hybrid task set consisting of 5 periodic hard tasks with fixed parameters and 5 soft tasks with variable execution times and interarrival times. The periods and the execution times of the periodic hard tasks are randomly generated in order to achieve a desired processor utilization factor U_{hard}, while their relative deadlines are equal to the periods. The execution and interarrival times of the soft tasks are uniformly distributed in order to obtain a mean soft load $\overline{U_{soft}} = \sum_i \frac{\overline{c_{i,j}}}{r_{i,j+1} - r_{i,j}}$ with $\overline{U_{soft}}$ going from 0 to $1 - U_{hard}$. All the soft tasks have the same relative deadline.

The metric used to measure the performance of the service algorithms is the mean tardiness $\overline{E_i}$ computed over all instances of each soft task. The reason for choosing such a metric is motivated by the fact that, as already mentioned above, in multimedia applications meeting all soft deadlines could be impossible or very inefficient. Thus, a more realistic objective is to guarantee all the hard tasks and minimize the mean time that soft tasks execute after their deadlines. Notice that, since all the soft tasks have the same relative deadline, the tardiness is not dependent on task's deadline.

In the first experiment, we compare the mean tardiness experienced by soft tasks when they are served by a CBS, a TBS and a DSS. In this test, the utilization factor of periodic hard tasks is $U_{hard} = 0.5$. The simulation results are illustrated in Figure 6, which shows that the performance of the DSS is dramatically worse than the one achieved by the CBS and TBS. This result was expected for the reasons explained above.

Figure 7 shows the same results, but without the DSS: the only difference is in the scale of the y-axis. In this figure, the TBS and CBS curves can be better distinguished, so we can see that the tardiness experienced by soft tasks under a CBS is slightly higher than that experienced using a TBS. However, the difference is so small that can be neglected for any practical purposes.

Figures 8 and 9 illustrate the results of similar experiments repeated with $U_{hard} = 0.7$ and $U_{hard} = 0.9$ respectively. As we can see, the major difference in the performance between CBS and TBS appears only for heavy hard loads. Fortunately, this situation is of little interest for most practical multimedia applications.

When $WCET_i >> \overline{c_{i,j}}$ the TBS can cause an underutilization of the processor. This fact can be observed in Figure 10, which shows the results of a fourth experiment, in which $U_{hard} = 0.6$, $\overline{U_{soft}} = 0.4$, the interarrival times are fixed, and the execution times of the soft tasks are uniformly distributed with an increasing variance.

As can be seen from the graph, CBS performs better than TBS when c_i varies a lot among the jobs.

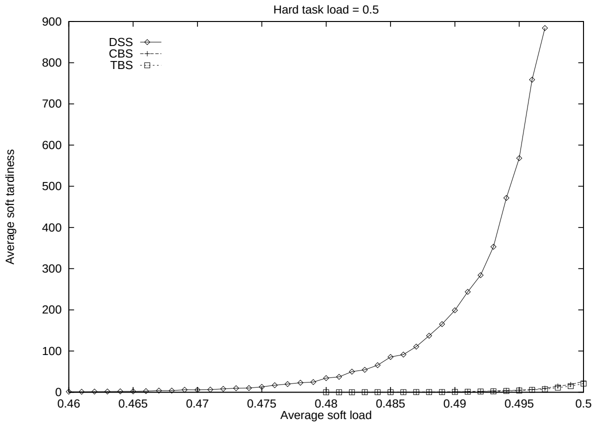

Figure 6. First experiment (TBS, CBS and DSS).

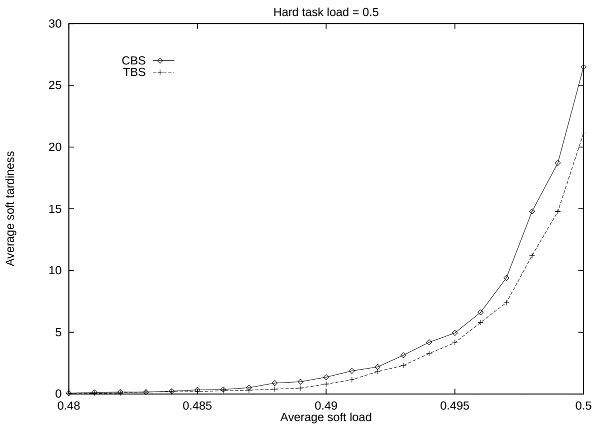

Figure 7. First experiment (TBS and CBS).

5. Implementation and experimental results

The proposed CBS mechanism has been implemented on the HARTIK kernel [2, 10], to support some sample multimedia applications (see [1] for implementation details).

For example, an MPEG player has been executed using EDF, with and without CBS. The application consists of two periodic tasks: task τ_1 with a period $T_1 = 125ms$, corresponding to 8 frames per second (Fps), and task τ_2 with a period $T_2 = 30ms$ (33 Fps). Figure 11 reports the number of decoded frames as a function of time, when the two periodic tasks are scheduled by EDF, activating τ_2 at $t = 2000$. Since $C_1 = 49ms$, $C_2 = 53ms$ and $49/125 + 53/30 = 2.158 > 1$, when τ_2 is activated the system becomes overloaded. In fact, when τ_1 is the only task in the system, it runs at the required frame rate (8 Fps), but when at time $t = 2000$ τ_2 is activated, τ_1 slows down to $4.4Fps$, while τ_2 begins to execute at $17.96Fps$.

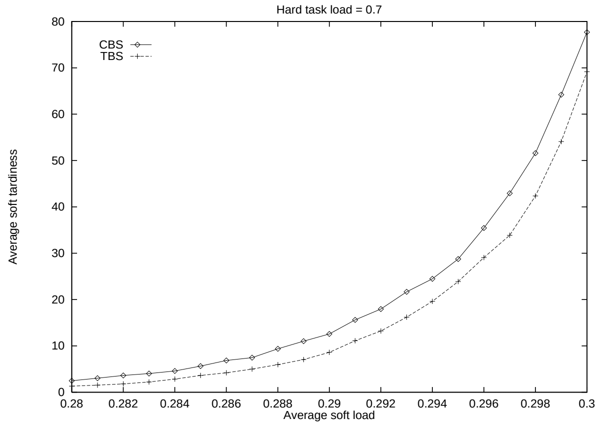

Figure 8. Second experiment.

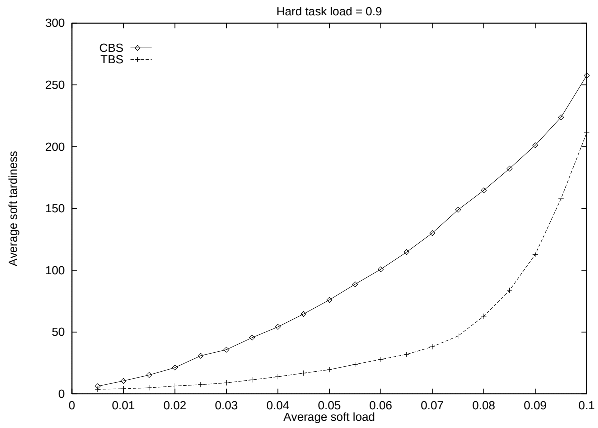

Figure 9. Third experiment.

When τ_2 terminates, τ_1 increases its frame rate to its maximum value ($23, 8Fps$, that corresponds to a period of about $42ms$, which is the mean execution time for τ_1). After this transient interval, τ_1 returns to execute at $8Fps$.

Figure 12 shows the number of decoded frames as a function of time, when the same periodic tasks are scheduled by two CBSs with parameters $(Q_1, T_1) = (42, 125)$ and $(Q_2, T_2) = (19, 30)$. Being $42/125 + 19/30 = 0.969 < 1$, the two servers are schedulable, and being $Q_1 = 42 \simeq \overline{c_1}$, τ_1 will execute at a frame rate near to the required one.

From the figure we can see that the frame rate of τ_1 is about constant except for two little variations corresponding to the activation and the termination of τ_2 (remember that $Q = \overline{c}$ is a limit condition). This is obtained by slowing down the frame rate of τ_2 to 14.2 Fps: this task is clearly overloaded ($T_2 < \overline{c_2}$), so it is penalized by the CBS.

Notice that the proposed mechanism automatically arrange the task periods without using a-priori knowledge about the tasks' execution times. The only information used

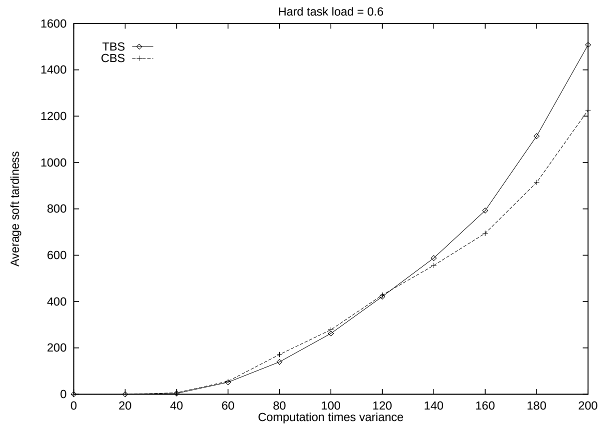

Figure 10. Fourth experiment.

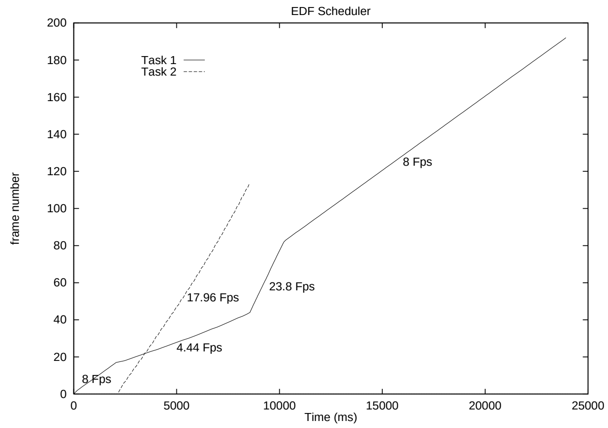

Figure 11. Two MPEG players scheduled by EDF.

by the CBS is the couple (Q_i, T_i) and the estimation of task execution time given by the budget.

Figure 11 shows another undesirable effect: when τ_2 terminates, the frame rate of τ_1 increases to its maximum value (more than the required rate), in order to terminate in the same time instant in which it would terminate if τ_2 was not activated. This phenomenon causes an acceleration of the movie that appears unnatural and unpleasant. This problem can be solved using a skip strategy to serve soft tasks: when a job finishes after its absolute deadline, the next job is skipped.

As shown in Figure 13, a skip strategy eliminates accelerations in the movie, but it introduces another problem, which is presented in the next experiment, where the same movie is decoded by two identical tasks, with $\overline{U_{soft}} = 1$.

From Figure 14 it is easy to see that, although the two tasks have the same period, they proceed with different speed. This is due to the fact that the system is overloaded.

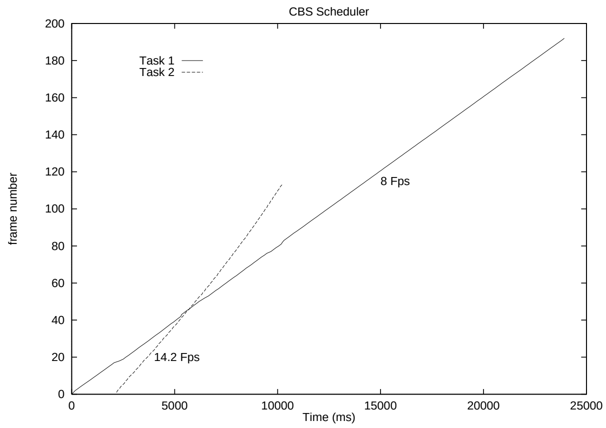

Figure 12. Two MPEG players scheduled by CBS.

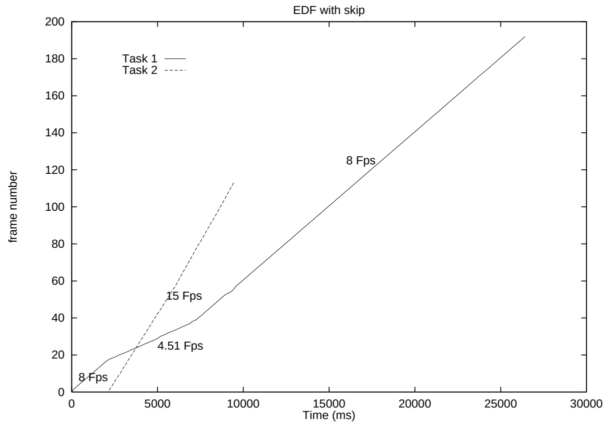

Figure 13. Two MPEG players scheduled by EDF with skip.

In fact, if

$$\overline{U_{soft}} = \frac{\overline{c_{1,j}}}{\overline{r_{1,j+1} - r_{1,j}}} + \frac{\overline{c_{2,j}}}{\overline{r_{2,j+1} - r_{2,j}}} = 1$$

then $U_{soft} = \frac{C_1}{T_1} + \frac{C_2}{T_2} > 1$.

Serving the two tasks by two identical CBSs with parameters $Q_s = \overline{c_{1,j}} = \overline{c_{2,j}}$ and $T_s = 2Q_s$, they proceed at the same rate (tasks' parameters are equal because the two tasks play the same video).

6. Conclusions

In this paper, we presented a novel service mechanism, the Constant Bandwidth Server, for integrating hard real-time and soft multimedia computing in a single system, under the EDF scheduling algorithm. The server has been

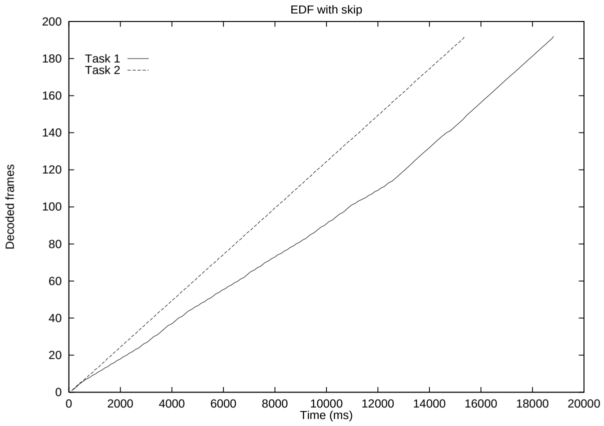

Figure 14. Two identical MPEG players scheduled by EDF with skip.

formally analyzed and compared with other known servers, obtaining very interesting results. The proposed model has also been implemented on the HARTIK kernel and used to support typical multimedia applications.

As a future work, in order to extend the proposed model to more general situations, the following issues need to be investigated. A concurrency control protocol needs to be integrated with the method to avoid priority inversion when accessing shared resources. The difference between the first and the current CBS deadline can be used as a kind of feedback for evaluating the request in excess and react accordingly adjusting the QoS in overload conditions. The CBS mechanism can be used to safely partition the CPU bandwidth among different applications that could coexist in the same system, as shown in [4]. A task can be used as a QoS manager to dynamically change the bandwidth reserved to each multimedia task. The strategies for changing the parameters of each CBS still have to be investigated.

References

[1] L. Abeni. Server mechanisms for multimedia applications. Technical Report RETIS TR98-01, Scuola Superiore S. Anna, 1998.

[2] G. C. Buttazzo. Hartik: A real-time kernel for robotics applications. In *IEEE Real-Time Systems Symposium*, December 1993.

[3] Z. Deng and J. W. S. Liu. Scheduling real-time applications in open envirovment. In *IEEE Real-Time Systems Symposium*, December 1997.

[4] Z. Deng, J. W. S. Liu, and J. Sun. A scheme for scheduling hard real-time applications in open system environment. In *Ninth Euromicro Workshop on Real-Time Systems*, 1997.

[5] T. M. Ghazalie and T. Baker. Aperiodic servers in a deadline scheduling environment. *Real-Time Systems*, 9, 1995.

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

[7] K. Jeffay and D. Bennet. A rate-based execution abstraction for multimedia computing. In *Network and Operating System Support for Digital Audio and Video*, 1995.

[8] K. Jeffay, D. L. Stone, and F. D. Smith. Kernel support for live digital audio and video. *Computer Communications*, 15(6), 1992.

[9] H. Kaneko, J. A. Stankovic, S. Sen, and K. Ramamritham. Integrated scheduling of multimedia and hard real-time tasks. In *IEEE Real Time System Symposium*, December 1996.

[10] G. Lamastra, G. Lipari, G. Buttazzo, A. Casile, and F. Conticelli. Hartik 3.0: A portable system for developing real-time applications. In *Real-Time Computing Systems and Applications*, October 1997.

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

[12] C. W. Mercer, S. Savage, and H. Tokuda. Processor capacity reserves for multimedia operating systems. Technical Report CMU-CS-93-157, Carnegie Mellon University, Pittsburg, May 1993.

[13] M. Spuri and G. Buttazzo. Scheduling aperiodic tasks in dynamic priority systems. *Real-Time Systems*, 10(2), 1996.

[14] M. Spuri, G. Buttazzo, and F. Sensini. Robust aperiodic scheduling under dynamic priority systems. In *IEEE Real-Time Systems Symposium*, December 1995.

[15] M. Spuri and G. C. Buttazzo. Efficient aperiodic service under the earliest deadline scheduling. In *IEEE Real-Time Systems Symposium*, December 1994.

[16] 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 *IEEE Real Time System Symposium*, December 1996.