Server Mechanisms for Multimedia Applications

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

Abstract

This paper focuses on the problem of providing efficient run-time support to multimedia applications in a real-time system, where three types of tasks can coexist simultaneously: multimedia, soft real-time, and hard real-time tasks. Hard tasks are guaranteed based on worst case execution times and minimum inter-arrival 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

Recent evolution of technology makes personal computers powerful enough to handle multimedia streams, so there is a great interest in providing support to continuous media applications (CM applications from now on). Nevertheless, allowing CM activities to coexist with other applications with different timing requirements (such as hard, soft real-time, and non real-time tasks) is still an open issue.

In fact, from one hand, CM activities 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 inter-arrival time of the tasks that process such data may not be deterministic, so it may be impossible to determine a minimum inter-arrival 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 deviced 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 [17, 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 will not consider this kind of solutions.

Anderson et al. in [5] describe a multimedia operating system based on an EDF scheduler, however the quality of service (QoS) can only be guaranteed based on the WCET of each task and based on a model of the external events that can activate a task (they use a Linear Bounded Arrival Process).

In [8], Jeffay presents a hard real-time system based on EDF scheduling to be used as a test-bet for video conference applications; the system can guarantee each task at its creation time based on its WCET and its minimum inter-arrival time. While a bound for the WCET can be found, the inter-arrival 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 inter-arrival 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 solves the problem 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 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 [2], 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 [3] for tasks that do not use nonpreemptable sections or global resources, and a Total Bandwidth Server [14, 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

[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

2.1 basic definitions

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 inter-arrival 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 inter-arrival 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. To predict the QoS that can be achieved by the system, a statistical guarantee could be requested; for example, by providing the probability for a job of finishing before its deadline. In this work, our goal is to schedule soft tasks to minimize the mean tardiness, 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 $J_{i,j}$.

A non real-time task is a task without timing constraints. Nothing is assumed to be known about non real-time tasks. Our goal is to schedule them as soon as possible, preserving the schedulability of the hard tasks and the QoS statistically guaranteed for soft tasks.

Finally, a periodic task is a task (hard or soft) in which the inter-arrival 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.

Let $D_i(t_1, t_2)$ the computation time required by task τ_i in the time intervals $[t_1, t_2]$:

$$D_i(t_1, t_2) = \sum_{j: r_{i,j} \geq t_1 \wedge d_{i,j} \leq t_2} c_{i,j}$$

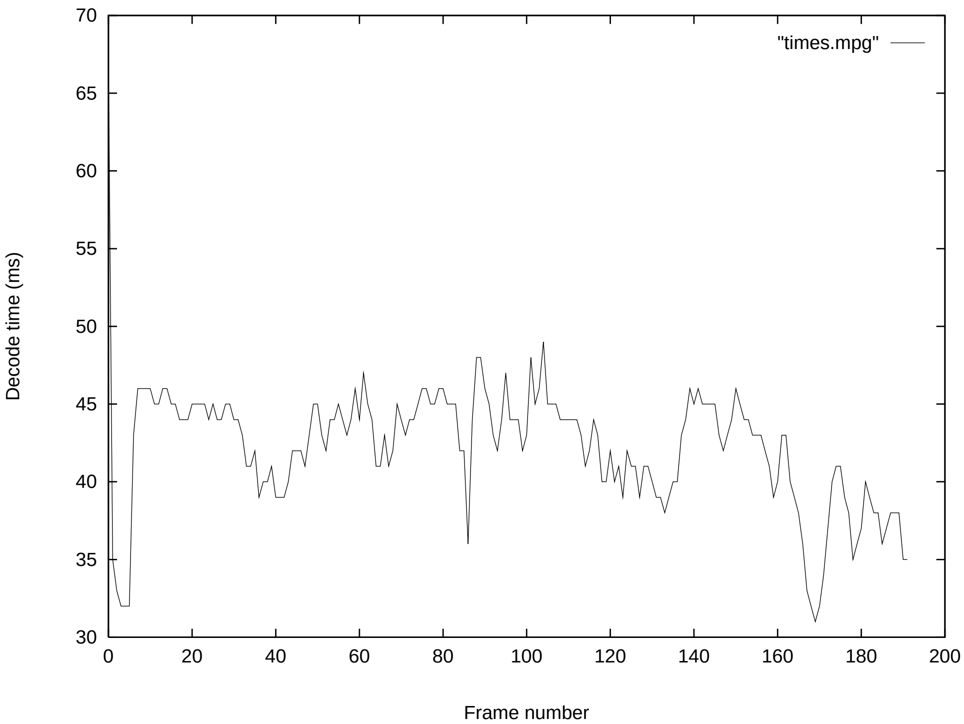

Figure 1: MPEG decoding times.

we say that task τ_i requires a bandwidth B_i if

$$B_i = \max_{t_1,t_2}\{\frac{D_i(t_1,t_2)}{t_2 - t_1}\}.$$

Since from [11] is known that the bandwidth B_i required by an hard real-time task τ_i is $\frac{C_i}{T_i}$ and that a task set Σ is schedulable if $\sum_i B_i \leq U_{lub}$ (with $U_{lub} = 1$ using EDF and $U_{lub} = 0.69$ using RM), it is easy to perform an a priori guarantee on hard tasks: if Σ is composed only of hard tasks, the guarantee test is

$$\sum_i \frac{C_i}{T_i} < U_{lub}$$

.

2.2 Soft tasks

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. As an example, Figure 1 shows the decoding times of various frames in a typical MPEG video stream.

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. For using a bandwidth reservation strategy, it is needed a mechanism to schedule a soft real-time task τ_i in order to ensure that its required bandwidth B_i is no more than the reserved bandwidth.

In this way, if a task needs more than its reserved bandwidth, it may slow down, but it will not jeopardize the schedulability of the other tasks: this is the isolation property, that ensure that a task overload will remain isolate to the task. By reserving a bandwidth B_i to each task and isolating the effects of task overloads, an a priori guarantee can be performed using the classical schedulability analysis

$$\sum_i B_i \leq U_{lub}.$$

2.3 Server mechanisms

Our scheduling scheme is based on EDF algorithm, because it permits a better CPU utilization (for EDF $U_{lub} = 1$, for RM $U_{lub} = 0.69$), so we propose to schedule hard tasks jobs $J_{i,j}$ by their absolute deadlines $d_{i,j} = r_{i,j} + T_i$, and to schedule each other job $J_{k,h}$ by an absolute deadline $d_{k,h}$ assigned to it in order to require a band B_k. For doing this, we need a mechanism, called server, to assign an absolute deadline to a job: a server receives computation time requests (jobs) in input and serves them assigning a dynamic absolute deadline to each of them.

The arriving jobs are enqueued in a queue of pending requests according to a given (arbitrary) non-preemptive discipline (e.g., FIFO) and the first job of the queue is served assigning a deadline to it and eventually putting it in the scheduler ready queue (since we use an EDF scheduler, this queue is ordered by absolute deadlines). If the first job of a server's pending request queue is in the ready queue, the server is said to be eligible, else it is not eligible; the server is said to be active when the first job of its pending request queue is executing.

A server S_s can serve a job $J_{i,j}$ dividing it in smaller blocks, each of them will be assigned a fixed absolute deadline, named chunks: $J_{i,j}$ is served dividing it in m chunks $H_{i,j,1}, H_{i,j,2}, ..., H_{i,j,m}$, each of them is characterized by a release time $r_{i,j,k}$ a deadline $d_{i,j,k}$, a finish time $f_{i,j,k}$ and a computation time $c_{i,j,k}$. In general, $r_{i,j,1} \geq r_{i,j}$ and $r_{i,j,k+1} \geq f_{i,j,k}$, where the major sign holds if the server is not eligible between $r_{i,j}$ and $r_{i,j,1}$ (or between $f_{i,j,k}$ and $r_{i,j,k+1}$).

The execution time required by a server S_s can be defined similarly to the execution time required by a task τ_i:

$$D_s(t_1, t_2) = \sum_{r_{i,j,k} \geq t_1 \wedge d_{i,j,k} \leq t_2} c_{i,j,k}$$

the bandwidth required by a server is defined exactly as that required by a task.

In order to be usable for realizing the isolation property, a server must require a limited bandwidth B_i: the server can limit its required band in two way:

- becoming not eligible every time that it can require too much execution time
- increasing the deadline of the served jobs in order to not require too much execution time

The first solution is used by some budget-based servers (like the Dynamic Sporadic Server): to each server is assigned a budget that decreases when the server is active; when the budget becomes 0, the server becomes not eligible and it will return eligible at a replenishing time calculated to limit the band to B_s.

The second solution is used by a class of servers (like the Total Bandwidth Server) that remains ever eligible while their pending request queue is not empty: we call them non-idle servers. Also if a budget is not used in the definition, a non idle server can be thought as characterized by a budget that is immediately replenished when it arrives to 0.

Scheduling a task τ_i by a dedicated limited band server S_s that requires a bandwidth B_s, we can ensure that the bandwidth required by τ_i is B_s, so a bandwidth reservation strategy can be used.

3 Multimedia Servers

To integrate the three classes of tasks in the same system, hard tasks are scheduled by the EDF algorithm based on their absolute deadlines, each soft task is handled by a dedicated server, whereas non real-time tasks can be handled by a single server: any conventional dynamic priority server like the Dynamic Sporadic Server or the Total Bandwidth Server or other can be used for this purpose.

3.1 Existent servers

The service mechanisms that have inspired this work are the Dynamic Sporadic Server (DSS) [14, 4] and the Total Bandwidth Server (TBS) [14, 15].

The DSS is a dynamic version of the Sporadic Server, originally proposed by Sprunt, Sha and Lehozky [13] for fixed priority systems: whereas the Sporadic Server has a fixed priority chosen according to the RM algorithm, the DSS has a dynamic priority assigned through a suitable deadline. The deadline assignment and the budget replenishment are defined by the following rules:

- When the server is created, its budget c_s is initialized to its maximum value.
- The next replenishment time RT and the current server deadline d_s are set as soon as $c_s > 0$ and there is an aperiodic request pending. If t_a is such time, then $RT = d_s = t_A + T_s$.
- The replenishing amount RA to be done at time RT is computed when the last aperiodic request is completed or c_s has been exhausted. If t_I is such a time, then the value of RA is set equal to the budget consumed within the interval $[t_a, t_I]$

The TBS is a simple and efficient aperiodic service mechanism based on the idea of assigning each aperiodic request a deadline such that the overall processor utilization due to the aperiodic load never exceeds the maximum value U_s. In particular, when the k^{th} aperiodic request arrives at time $t = r_k$, it receives a deadline

$$d_k = \max\{r_k, d_{k-1}\} + \frac{C_k}{U_s}$$

where C_k is the execution time of the request and U_s is the server utilization factor (that is, its bandwidth). By definition, $d_0 = 0$. An explicit budget is not used in the original definition

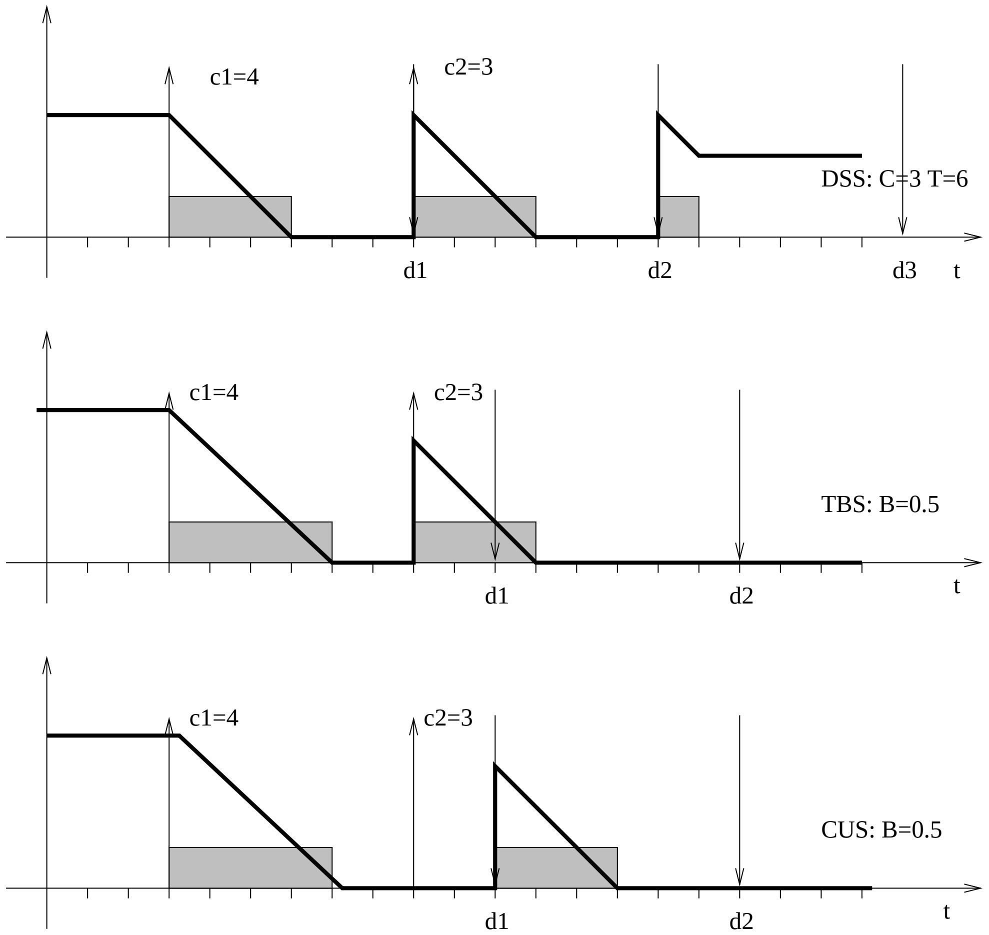

Figure 2: Comparison of three different dynamic servers: DSS, TBS, and CUS.

of the TBS, but the server can be thought as characterized by a budget that is replenished at C_k at time r_k.

Liu and Deng in [3] proposed a server mechanism very similar to TBS, the Constant Utilization Server (CUS), which differs only for the replenishing time, which occurs at time $\max\{r_k, d_{k-1}\}$ rather than at r_k as for the TBS.

While the isolation property can be realized independently by the kind of server used (it is only important that the server require a limited band),the QoS achieved by a soft task depends on the type of server adopted to serve it. If a single task is overloaded (it needs more than the reserved band), it is possible that some of its jobs finish after their soft deadlines: we say that this task is late. If there is some idle time (a fraction of the CPU bandwidth is free) a late task would have to use it to exit the late state. If the idle time is due to early completion of some previous jobs of the late task, also a server that not have the non-idle property can reclaim this time, but if the idle time is due to a total utilization factor less then 1 or to early completions of other tasks jobs, only a non-idle server can reclaim this time.

This difference between a non-idle server and a server that doesn't have this property is visible in figure 2. In this figure we can see the same soft task served by three different servers: the TBS, the DSS and the Constant Utilization Server (CUS), proposed by Dend and Liu in [3]. The TBS, that is a non-idle server, can use additional background time to reduce the response times, while the others two servers (the DSS or the CUS), that haven't the non-idle property, cannot execute for a time greater than $(t_2 - t_1)U_s$ in each interval $[t_1, t_2]$ even though the system is underloaded.

From a performance point of view, the TBS would be the right choice to serve multimedia tasks, so let's consider a multimedia task served by a dedicated TBS: each job $J_{i,j}$ is assigned a deadline

$$d_{i,j} = \max\{r_{i,j}, d_{i,j-1}\} + \frac{C_i}{U_s}$$

that can also be written as

$$d_{i,j} = \begin{cases} r_{i,j} + T_s & \text{if } j = 1 \\ \max\{r_{i,j}, d_{i,j-1}\} + T_s & \text{otherwise} \end{cases}$$

with $T_s = \frac{C_i}{U_s}$.

In [7], Jeffay proposed a Rate Based Execution (RBE) model, in which each task τ_i is characterized by three parameters (X_i, Y_i, D_i) and the jobs deadline are calculated according to the following rule:

$$d_{i,j} = \begin{cases} r_{i,j} + D_i & \text{if } 1 \leq j \leq X_i \\ \max\{r_{i,j}, d_{i,j-1}\} + T_s & \text{otherwise} \end{cases}$$

It is easy to see how the deadlines generated serving a soft task with a dedicated TBS are exactly the same deadlines assigned to the jobs by an $RBE(1, T_s, T_s)$ model.

3.2 The Constant Bandwidth Server

It is easy to understand how to give the maximum possible QoS to a multimedia task, it must be served by a non-idle server, like the TBS. Unfortunately, the TBS presents a problem: we said that behavior of the server must guarantee 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 in our hypothesis (no knowledge about the soft tasks WCETs and minimum inter-arrival times) this property, also belonging to a Dynamic Sporadic Server (DSS) [14, 4], is not valid for a Total Bandwidth Server (TBS) [14], nor for a Constant Utilization Server (CUS) [3], whose actual contributions are limited by U_s under the assumption that all the served jobs do not execute more than the declared WCET. Also the RBE model, that we said to assign the same deadlines as TBS, needs the knowledge of the tasks WCETs to guarantee each job to finish before its deadline.

We need a non-idle server (whose performance is comparable with the one achievable by a TBS) that doesn't need any information about WCETs and minimum inter-arrival times (like the DSS): this last property is obtainable only using a budget-based mechanism similar to which used by DSS. To provide these two properties we introduce the Constant Bandwidth Server (CBS), 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 $d_{s,k}$.

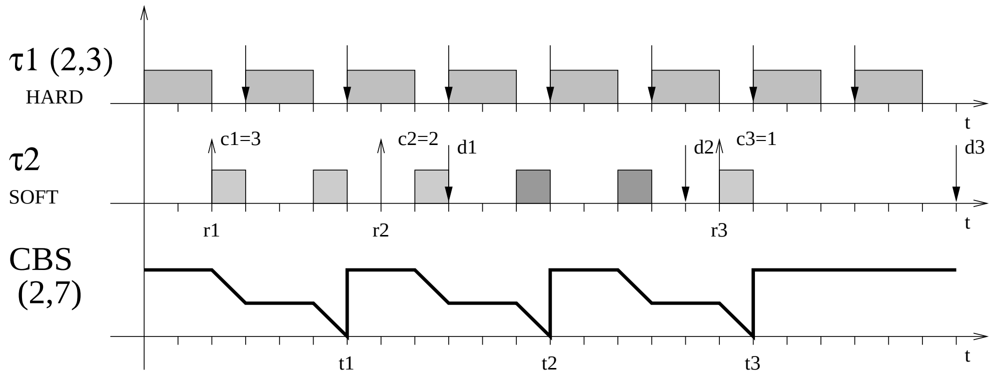

Figure 3: Simple example of CBS scheduling.

- 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 3 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 deadline assigned to the job is $d_{s,1} = r_1 + T_s = 9$ 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 $t_2 = 16$ 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 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.

In Figure 4, a hard periodic task, τ_1, is scheduled together with a soft task, τ_2, having fixed inter-arrival time ($T_2 = 7$) and variable computation time, with a mean value equal to $C_2 = 2$. This situation is typical in applications that manage continuous media: for example,

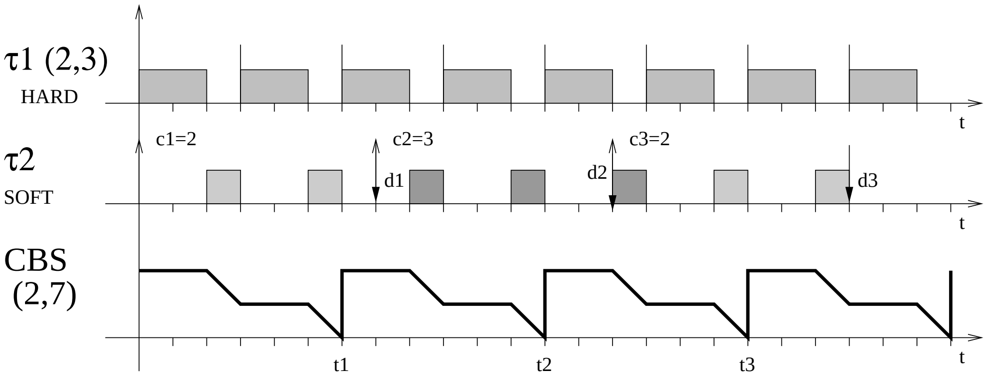

Figure 4: Example of CBS serving a task with variable execution time and constant inter-arrival time.

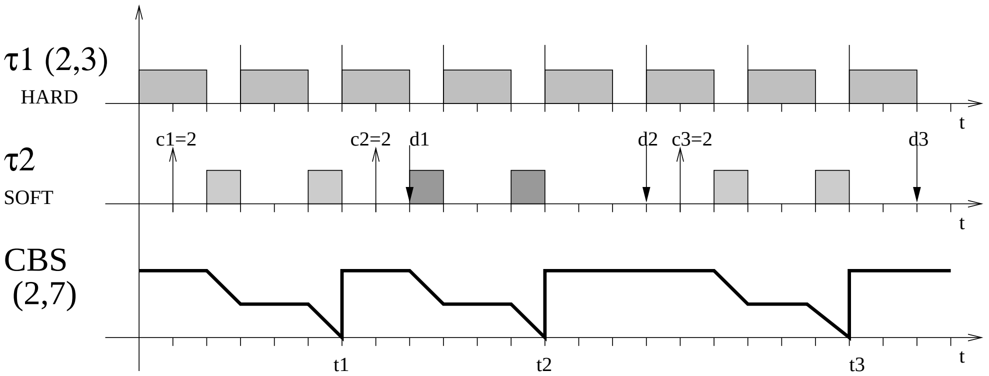

Figure 5: Example of CBS serving a task with constant execution time and variable inter-arrival time.

a video stream requires to be played periodically, but the decoding/playing time of each frame is not constant. In this example, to optimize the processor utilization, τ_2 is served by a CBS with a maximum budget equal to the mean computation time of the task ($Q_s = 2$) and a period equal to the task period ($T_s = 7$).

As we can see from Figure 4, the second job of task τ_2 is first assigned a deadline $d_{s,2} = r_2 + T_s$. At time t_2, however, since c_s is exhausted and the job is not finished, the job is scheduled with a new deadline $d_{s,3} = d_{s,2} + T_s$. As a result of a longer execution, only the soft task is delayed, while the hard task meets all its deadlines. Moreover, notice that the exceeding portion of the late job is not executed in background, but is scheduled with a suitable dynamic priority.

In other situations, frequently encountered in CM applications, tasks have fixed computation times but variable inter-arrival times. For example, this is the case of a task activated by external events, such a driver process activated by interrupts coming from a communication network. In this case, the CBS behaves exactly like a TBS with a bandwidth $U_s = Q_s/T_s$. In fact, if $C_i = Q_s$ each job finishes exactly when the budget arrives to 0, so the server deadline is increased of T_s. It is also interesting to observe that, in this situation, the CBS is also equivalent to a Rate-Based Execution (RBE) model [7] with parameters $x = 1, y = T_i, D = T_i$. An example of such a scenario is depicted in Figure 5.

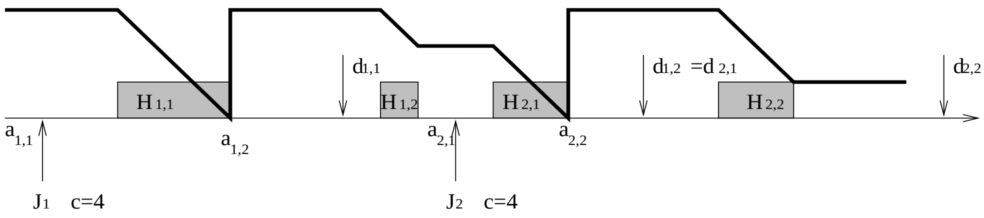

Figure 6: Serving some jobs divided in chunks.

3.3 CBS properties

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

Theorem 1 *A CBS with parameters* (Q_s, T_s) *requires a band* $U_s = \frac{Q_s}{T_s}$

Proof.
To prove the theorem, we show that a CBS with parameters (Q_s, T_s) cannot occupy a bandwidth greater than $U_s = Q_s/T_s$. That is, if $D_s(t_1, t_2)$ is the processor demand of the CBS in the interval $[t_1, t_2]$, we show that

$$\forall t_1, t_2 \in N : t_2 > t_1, \quad D_s(t_1, t_2) \leq \frac{Q_s}{T_s}(t_2 - t_1).$$

We recall 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 6. To simplify the notation, we indicate all the chunks generated by a server with an increasing index k (in the example of Figure 6, $H_{1,1} = H_1$, $H_{1,2} = H_2$, $H_{2,1} = H_3$, and so on).

The release time and the deadline of the k^{th} chunk generated by the server will be denoted by a_k and d_k, c will indicate the actual budget and n the number of requests in server queue. These variables are initialized in the following manner:

$$\begin{aligned} d_0 &= 0 \\ c &= 0 \\ n &= 0 \\ k &= 0 \end{aligned}$$

Using these notations, the server behavior can be expressed as in figure 7.

Indicating with e_k the server time demanded in the interval $[a_k, d_k]$ (that is, the execution time of chunk H_k), we can say that

$$\forall t_1, t_2, \quad \exists k_1, k_2 : \quad D_s(t_1, t_2) \; = \sum_{k: a_k \geq t_1 \wedge d_k \leq t_2} e_k \; = \sum_{k=k_1}^{k2} e_k.$$

```
When job J_j arrives at time r_j
       enqueue the request in the server pending request 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 queues;
       n = n - 1;
       if (n != 0) begin to serve the next job in 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;
       c = Q_s;
```

Figure 7: The CB algorithm.

If $c_s(t)$ is the server budget at time t and f_k is the time at which chunk H_k ends to execute, we can see that $c_s(f_k) = c_s(a_k) - e_k$, while $c_s(a_{k+1})$ is calculated from $c_s(f_k)$ in the following manner:

$$c_s(a_{k+1}) = \begin{cases} c_s(f_k) & \text{if } d_{k+1} \text{ was generated by Rule 2} \\ Q_s & \text{if } d_{k+1} \text{ was generated by Rule 1 or 3.} \end{cases}$$

Using these observations, the theorem can be proved by showing that:

$$D_s(a_{k_1}, d_{k_2}) + c_s(f_{k_2}) \leq (d_{k_2} - a_{k_1})\frac{Q_s}{T_s}.$$

We proceed by induction on $k_2 - k_1$, using the algorithmic definition of CBS shown in Figure 7.

Inductive base. If in $[t_1, t_2]$ there is only one active chunk ($k_1 = k_2 = k$), two cases have to be considered.

<u>*Case a*</u>: $d_k < a_k + T_s$.

If $d_k < a_k + T_s$, then d_k is generated by Rule 2, so $a_k + \frac{c_s(f_{k-1})}{Q_s} T_s < d_k$ and $a_k = f_{k-1}$, that is

$$a_k + \frac{c_s(a_k)}{Q_s} T_s < d_k.$$

Being $c_s(f_k) = c_s(a_k) - e_k = c_s(a_k) - D_s(a_k, d_k)$, we have

$$a_k + \frac{D_s(a_k, d_k) + c_s(f_k)}{Q_s} T_s < d_k$$

hence

$$D_s(a_k, d_k) + c_s(f_k) < (d_k - a_k)\frac{Q_s}{T_s}.$$

<u>*Case b*</u>: $d_k = a_k + T_s$.

If $d_k = a_k + T_s$, then $D_s(a_k, d_k) + c_s(f_k) = e_k + c_s(f_k) = Q_s$. Hence, in both cases, we have:

$$D_s(a_{k_1}, d_{k_2}) + c_s(f_{k_2}) = D_s(a_k, d_k) + c_s(f_k) \leq (d_k - a_k)\frac{Q_s}{T_s} = (d_{k_2} - a_{k_1})\frac{Q_s}{T_s}.$$

Inductive step. The inductive hypothesis

$$D_s(a_{k_1}, d_{k_2 - 1}) + c_s(f_{k_2 - 1}) \leq (d_{k_2 - 1} - a_{k_1})\frac{Q_s}{T_s}$$

is used to prove that

$$D_s(a_{k_1}, d_{k_2}) + c_s(f_{k_2}) \leq (d_{k_2} - a_{k_1})\frac{Q_s}{T_s}.$$

Given the possible relations between d_k and d_{k-1}, three cases have to be considered:

- $d_k \geq d_{k-1} + T_s$. That is, d_k is generated by Rule 3 or Rule 1 when $r_j \geq d_{j-1}$.

- $d_k = d_{k-1}$. That is, d_k is generated by Rule 2.
- $d_{k-1} < d_k < d_{k-1} + T_s$. That is, d_k is generated by Rule 1 when $r_j < d_{j-1}$.

Case a: $d_{k_2} = d_{k_2-1} + T_s$.

In this case d_{k_2} can be generated only by Rule 1 or 3. Adding e_{k_2} to both sides of the inductive hypothesis, we obtain:

$$\sum_{k=k_1}^{k_2-1} e_k + e_{k_2} \leq (d_{k_2-1} - a_{k_1})\frac{Q_s}{T_s} - c_s(f_{k_2-1}) + e_{k_2}$$

and, since $c_s(f_k) = c_s(a_k) - e_k$, we have

$$\sum_{k=k_1}^{k_2} e_k \leq (d_{k_2-1} - a_{k_1})\frac{Q_s}{T_s} - c_s(f_{k_2-1}) + c_s(a_{k_2}) - c_s(f_{k_2}).$$

Since d_{k_2} is generated by Rule 1 or 3, it must be $c_s(a_{k_2}) = Q_s$, therefore:

$$\sum_{k=k_1}^{k_2} e_k \leq (d_{k2-1} - a_{k_1})\frac{Q_s}{T_s} - c_s(f_{k_2-1}) + Q_s - c_s(f_{k_2})$$

$$\sum_{k=k_1}^{k_2} e_k + c_s(f_{k_2}) \leq (d_{k2-1} - a_{k_1})\frac{Q_s}{T_s} - c_s(f_{k_2-1}) + Q_s \leq (d_{k_2-1} - a_{k_1})\frac{Q_s}{T_s} + Q_s$$

and finally

$$D_s(a_{k_1}, d_{k_2}) + c_s(f_{k_2}) \leq (d_{k2-1} - a_{k_1})\frac{Q_s}{T_s} + Q_s = (d_{k2-1} + T_s - a_{k_1})\frac{Q_s}{T_s}$$

$$D_s(a_{k_1}, d_{k_2}) + c_s(f_{k_2}) \leq (d_{k_2} - a_{k_1})\frac{Q_s}{T_s}.$$

Case b: $d_{k_2} = d_{k_2-1}$.

If $d_{k_2} = d_{k_2-1}$, then d_{k_2} is generated by Rule 2. In this case,

$$\sum_{k=k_1}^{k_2-1} e_k + e_{k_2} \leq (d_{k_2-1} - a_{k_1})\frac{Q_s}{T_s} - c_s(f_{k_2-1}) + e_{k_2}$$

but, being $d_{k_2} = d_{k_2-1}$, $c_s(f_{k_2}) + e_k = c_s(a_{k_2})$ and $c_s(a_{k_2}) = c_s(f_{k_2-1})$ (by Rule 2), we have:

$$\sum_{k=k_1}^{k_2} e_k \leq (d_{k_2} - a_{k_1})\frac{Q_s}{T_s} - c_s(a_{k_2}) + e_{k_2} = (d_{k_2} - a_{k_1})\frac{Q_s}{T_s} - c_s(f_{k_2})$$

hence

$$D_s(k_1, k_2) + c_s(f_{k_2}) = \sum_{k=k_1}^{k_2} e_k \leq (d_{k_2} - a_{k_1})\frac{Q_s}{T_s}.$$

Case c: $d_{k_2-1} < d_{k_2} < d_{k_2-1} + T_s$.

If $d_{k_2} < d_{k_2-1} + T_s$, d_{k_2} is generated by Rule 1, so $a_{k_2} + \frac{c_s(f_{k_2-1})}{Q_s} T_s \geq d_{k_2-1}$, hence $c(f_{k_2-1}) \geq (d_{k_2-1} - a_{k_2}) \frac{Q_s}{T_s}$. Applying the inductive hypothesis, we obtain

$$\sum_{k=k_1}^{k_2-1} e_k + e_{k_2} \leq (d_{k_2-1} - a_{k_1}) \frac{Q_s}{T_s} - c_s(f_{k_2-1}) + e_{k_2}$$

from which we have

$$\sum_{k=k_1}^{k_2} e_k \leq (d_{k_2-1} - a_{k_1}) \frac{Q_s}{T_s} - (d_{k_2-1} - a_{k_2}) \frac{Q_s}{T_s} + e_{k_2}$$

$$\sum_{k=k_1}^{k_2} e_k \leq (d_{k_2-1} - d_{k_2-1} - a_{k_1} + a_{k_2}) \frac{Q_s}{T_s} + e_{k_2}.$$

Now, being $e_{k_2} = Q_s - c_s(f_{k_2})$, we have:

$$\sum_{k=k_1}^{k_2} e_k \leq (-a_{k_1} + a_{k_2}) \frac{Q_s}{T_s} + Q_s - c_s(f_{k_2}) = (a_{k_2} + T - a_{k_1}) \frac{Q_s}{T_s} - c_s(f_{k_2})$$

but, from Rule 1 and 3, we have $d_k = a_k + T$, so we can write

$$\sum_{k=k_1}^{k_2} e_k \leq (d_{k_2} - a_{k_1}) \frac{Q_s}{T_s} - c_s(f_{k_2})$$

hence

$$D_s(k_1, k_2) + c_s(f_{k_2}) = \sum_{k=k_1}^{k_2} e_k \leq (d_{k_2} - a_{k_1}) \frac{Q_s}{T_s}.$$

□

The isolation property allows us to use a bandwidth reservation strategy to allocate a fraction of the CPU time to each task that cannot be guaranteed a priori. The most important consequence of this result is soft task can be schedules 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.

- No hypothesis are required on the WCET and the minimum inter-arrival time of the served tasks: this allows the same program to be used on different systems without recalculating the computation times.

- **Lemma 1** *A hard task τ_i with parameters (C_i, T_i) is schedulable by a CBS with parameters $Q_s = C_i$ and $T_s = T_i$ if and only if τ_i is schedulable without the CBS.*

  Proof.
  For any job of a hard task we have that $r_{i,j+1} - r_{i,j} = T_i$ and $c_{i,j} \leq Q_i$. 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 $Q_i = C_i$. Moreover, since $c_{i,j} \leq Q_i$, 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.

- 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.

3.4 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 (see also Figure 4); and b) constant computation time and variable inter-arrival time (see also Figure 5).

Case a.

If the jobs inter-arrival times are constant and equal to T_s and the jobs execution times are randomly distributed with a given probability distribution function, the CBS can be modeled with a $D^G/D/1$ queue: each 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:

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

where v_j indicates the length of the queue (in time units) at time $(j-1)T_s$, that is the units of times that are still to be server when the job $J_{i,j}$ arrives. It can easily be shown that the absolute deadline before which $J_{i,j}$ will finish is

$$\delta_j = r_{i,j} + \left\lceil \frac{v_j}{Q_s} \right\rceil T_s.$$

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 execution time units (since c_j is time invariant, C_h doesn't depend on j), it is quite easy to calculate the value of $\pi_k^{(j)}$:

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

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

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

$$\begin{aligned}
\pi_k^{(j)} &= \sum_{h=0}^{\infty} P\{max\{v_{j-1} - Q_s, 0\} + c_j = k | v_{j-1} = h\} P\{v_{j-1} = h\} = \\
&= \sum_{h=0}^{\infty} P\{max\{h - Q_s, 0\} + c_j = k\} P\{v_{j-1} = h\} = \\
&= \sum_{h=0}^{Q_s} P\{c_j = k\} P\{v_{j-1} = h\} + \sum_{h=Q_s+1}^{\infty} P\{h - Q_s + c_j = k\} P\{v_{j-1} = h\} = \\
&= \sum_{h=0}^{Q_s} C_k \pi_h^{(j-1)} + \sum_{h=Q_s+1}^{\infty} P\{c_j = k - h + Q_s\} \pi_h^{(j-1)} = \\
&= \sum_{h=0}^{Q_s} C_k \pi_h^{(j-1)} + \sum_{h=Q_s+1}^{\infty} C_{k-h+Q_s} \pi_h^{(j-1)}
\end{aligned}$$

Equation

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

can be writen in matrix form

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

defining

$$M = \left(\begin{array}{cccccccccc}
\multicolumn{6}{c}{\overbrace{\hphantom{C_0 \quad C_0 \quad . \quad . \quad . \quad C_0}}^{Q_s+1}} & & & & \\
C_0 & C_0 & . & . & . & C_0 & 0 & 0 & . & . & . \\
C_1 & C_1 & . & . & . & C_1 & C_0 & 0 & . & . & . \\
C_2 & C_2 & . & . & . & C_2 & C_1 & C_0 & 0 & . & . \\
C_3 & C_3 & & & & C_3 & C_2 & C_1 & C_0 & 0 & . \\
. & . & & & & . & . & . & . & . & . \\
. & . & & & & . & . & . & . & & . \\
. & . & & & & . & . & . & . & &
\end{array} \right) \text{ and } \Pi^{(j)} = \left(\begin{array}{c}
\pi_0^{(j)} \\ \pi_1^{(j)} \\ \pi_2^{(j)} \\ \pi_3^{(j)} \\ . \\ . \\ .
\end{array} \right)$$

Case b.

In the case in which job execution times are constant and equal to Q_s ($\forall j, c_{i,j} = Q_s$) and jobs inter-arrival 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. We can define a random process w_j as $w_j = d_{i,j} - r_{i,j} - T_s$, so we have $d_{i,j+1} = r_{i,j+1} + T_s + w_{i,j+1}$. In this way, it is easy to find the distribution of the relative deadlines $d_{i,j} - r_{i,j}$ within which a job $J_{i,j}$ is served. In fact,

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

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

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

having defined $a_{j+1} = r_{i,j+1} - r_{i,j}$. Being a_j is 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 defining $\pi_k^{(j)} = P\{w_j = k\}$ and $A_h = P\{a_j = h\}$.

$$\begin{aligned} \pi_k^{(j)} &= P\{w_j = k\} = P\{\max\{0, w_{j-1} - a_j + T_s\} = k\} = \\ &= \sum_{h=-\infty}^{\infty} P\{\max\{0, w_{j-1} - a_j + T_s\} = k \wedge w_{j-1} = h\} = \\ &= \sum_{h=-\infty}^{\infty} P\{\max\{0, w_{j-1} - a_j + T_s\} = k | w_{j-1} = h\} P\{w_{j-1} = h\} = \\ &= \sum_{h=-\infty}^{\infty} P\{\max\{0, h - a_j + T_s\} = k\} P\{w_{j-1} = h\} = \end{aligned}$$

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

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

$$\begin{aligned} \forall k > 0, \pi_k^{(j)} &= \sum_{h=-\infty}^{\infty} P\{h - a_j + T_s = k\} P\{w_{j-1} = h\} = \\ &= \sum_{h=-\infty}^{\infty} P\{a_j = h - k + T_s\} \pi_h^{(j-1)} = \\ &= \sum_{h=0}^{\infty} A_{h-k+T_s} \pi_h^{(j-1)} \end{aligned}$$

Matrix M describing the Markov chain is so

$$M = \begin{pmatrix} \rho_0 & \rho_1 & \rho_2 & . & . & 0 & 0 & 0 & . & . & . \\ A_{T_s+1} & A_{T_s} & A_{T_s-1} & . & . & A_0 & 0 & 0 & . & . & . \\ A_{T_s+2} & A_{T_s+1} & A_{T_s} & A_{T_s-1} & . & A_1 & A_0 & 0 & 0 & . & . \\ A_{T_s+3} & A_{T_s+2} & A_{T_s+1} & A_{T_s} & . & A_2 & A_1 & A_0 & 0 & 0 & . \\ . & . & . & . & & . & . & . & . & . & . \\ . & . & . & . & & . & . & . & & . & . \\ . & . & . & . & & . & . & . & & & . \end{pmatrix}$$

with

$$\rho_i = \sum_{r=i+T_s}^{\infty} A_r.$$

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

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

$$\begin{array}{lcll} \overline{c_{i,j}} & \leq & Q_s & \text{in case a)} \\ \overline{r_{i,j+1} - r_{i,j}} & \geq & T_s & \text{in case b)} \end{array}$$

In general,

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

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

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

3.5 Conclusions on CBS

We introduced the CB server to serve soft tasks, but the CBS behavior is more general and this kind of server can be used to serve each kind of task. To see this, we distinguish four kind of tasks basing on the execution/inter-arrival times variance.

The first kind of tasks is characterized by known WCET and minimum inter-arrival time: we have previously referred this tasks as hard real-time tasks, saying that they can be scheduled using EDF performing an a priori guarantee. A task τ_i of this kind, with WCET C_i and minimum inter-arrival time T_i, can also be scheduled by a dedicated CBS with parameters $(Q_s = C_i, T_s = T - i)$ guaranteeing that each job $J_{i,j}$ will finish before the relative deadline T_i (see lemma 1).

The second kind of tasks is characterized by known WCET but unknown minimum inter-arrival time: we have previously introduced these tasks saying that they are frequently used to handle external events coming from an unpredictable environment. A task τ_i of this kind, with WCET C_i can be scheduled by a dedicated TBS with bandwidth U_s or with the RBE model; it can also be scheduled by a dedicated CBS with parameters $(Q_s = C_i, T_s = \frac{C_i}{U_s})$ guaranteeing that each job $J_{i,j}$ will finish before the same deadline assigned by the TBS or by the RBE. The QoS experimented by these tasks can be statistically guaranteed as showed above.

The third kind of tasks is characterized by unknown WCET and known minimum inter-arrival time: we have previously introduced these tasks saying that they are frequently used to manage continuous media. A task τ_i of this kind, can be scheduled by a dedicated CBS

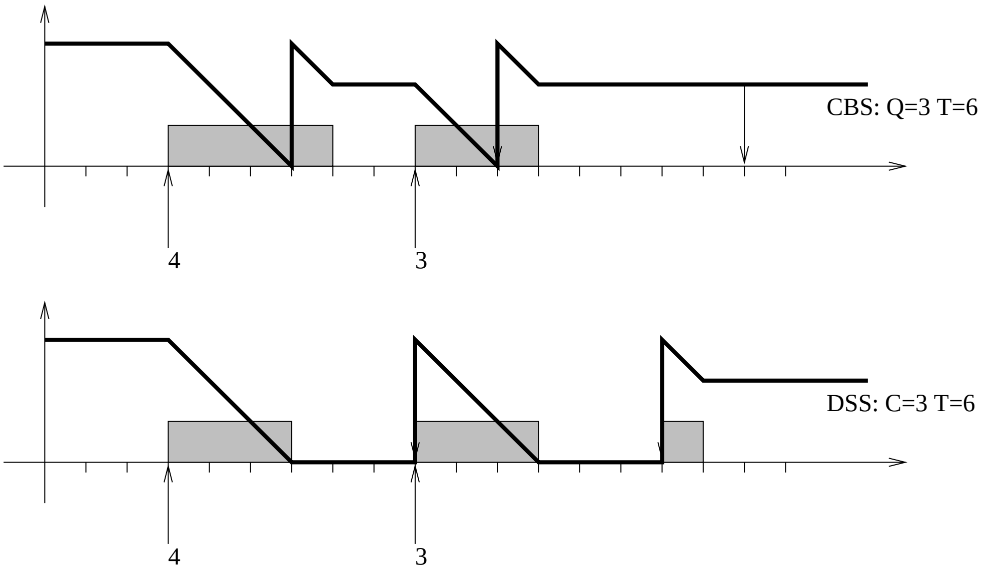

Figure 8: Behavior of a CBS and a DSS in the absence of other real-time tasks.

performing a statistical guarantee (EDF or TBS or RBE cannot be used on these tasks without jeopardizing the hard tasks schedulability).

The fourth kind of tasks is characterized by unknown WCET and minimum inter-arrival time: they can be scheduled by a dedicated CBS, but performing a statistical guarantee is too complex.

WCET	Min. interarr. time	task type	CBS behavior	guarantee
known	known	hard	EDF	deterministic
known	unknown	extern event handler	TBS, RBE	probabilistic
unknown	unknown	CM task		probabilistic
unknown	unknown			

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 for the reasons described in Section 3.1).

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 increasing the server's deadline and replenishing the budget immediately. This difference in the replenishing time, whose effects are illustrated in Figure 8, 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 been added to the plain TBS, as described in [16].

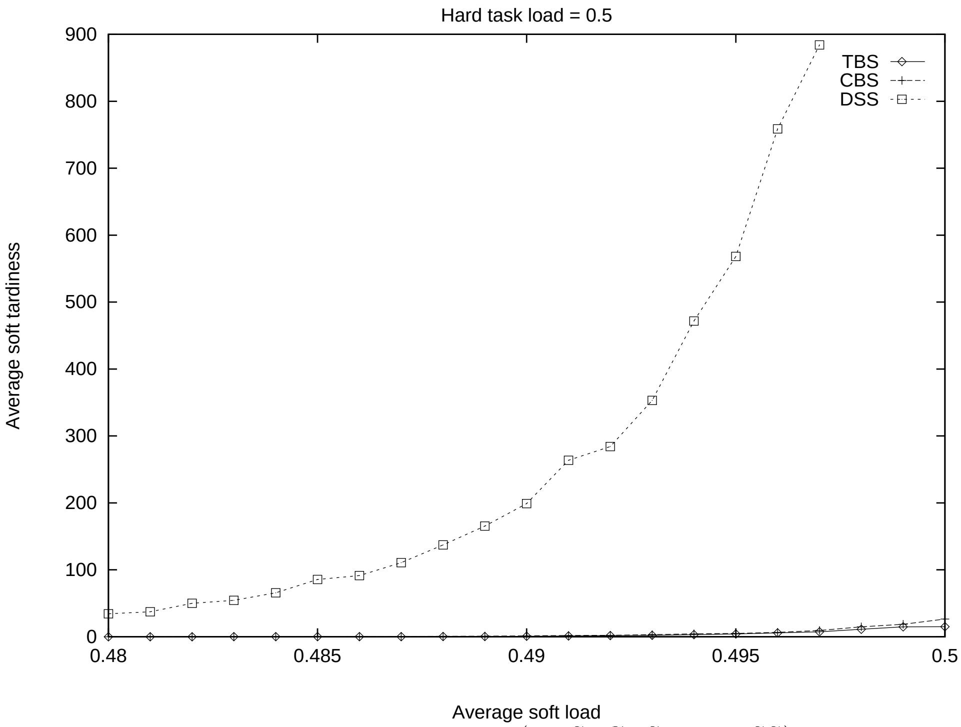

Figure 9: First experiment (TBS, CBS and DSS).

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 inter-arrival times. The execution times of the periodic hard tasks are randomly generated in order to achieve a desired processor utilization factor U_{hard}. The execution and inter-arrival times of the soft tasks are uniformly distributed in order to obtain a mean soft load $\overline{U_{soft}} = \sum_i \frac{\overline{c_{i,j}}}{\overline{r_{i,j+1} - r_{i,j}}}$ with $\overline{U_{soft}}$ going from 0 to $1 - U_{hard}$.

The metric used to measure the performance of the service algorithms is the mean tardiness experienced by soft tasks. In fact, as already mentioned above, since in multimedia applications meeting all soft deadlines could be impossible or very inefficient, the goal of the system should be to guarantee all the hard tasks and minimize the mean time that soft tasks execute after their deadlines.

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 9, 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 in Figure 8.

Figure 10 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.

Figure 11 and Figure 12 illustrate the results of a similar experiment 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 high periodic loads. Fortunately, this situation is of little interest for most practical multimedia applications.

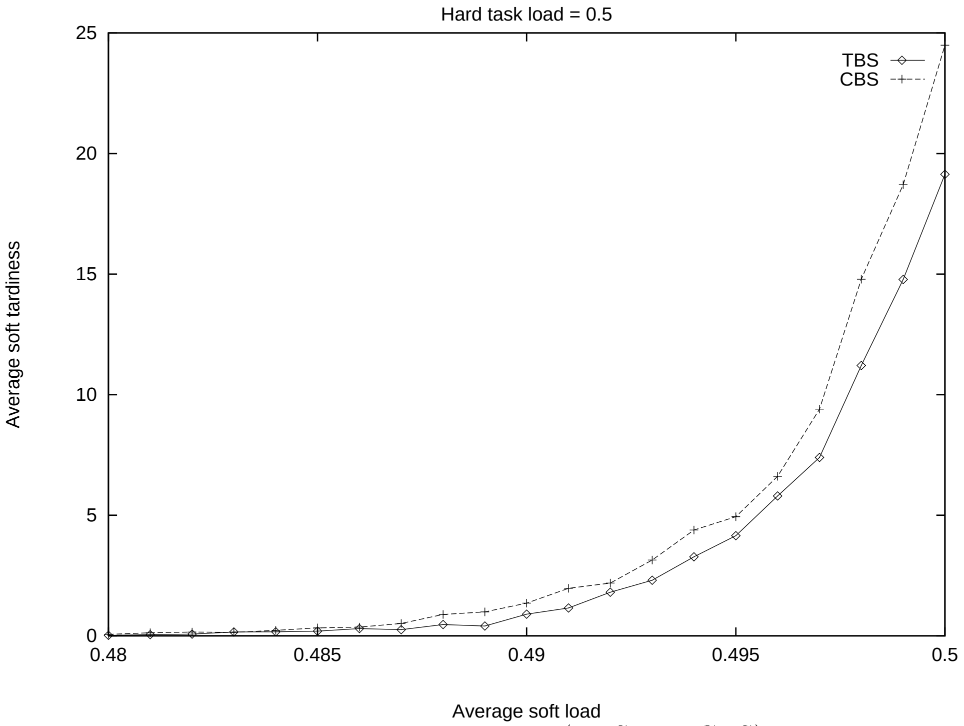

Figure 10: First experiment (TBS and CBS).

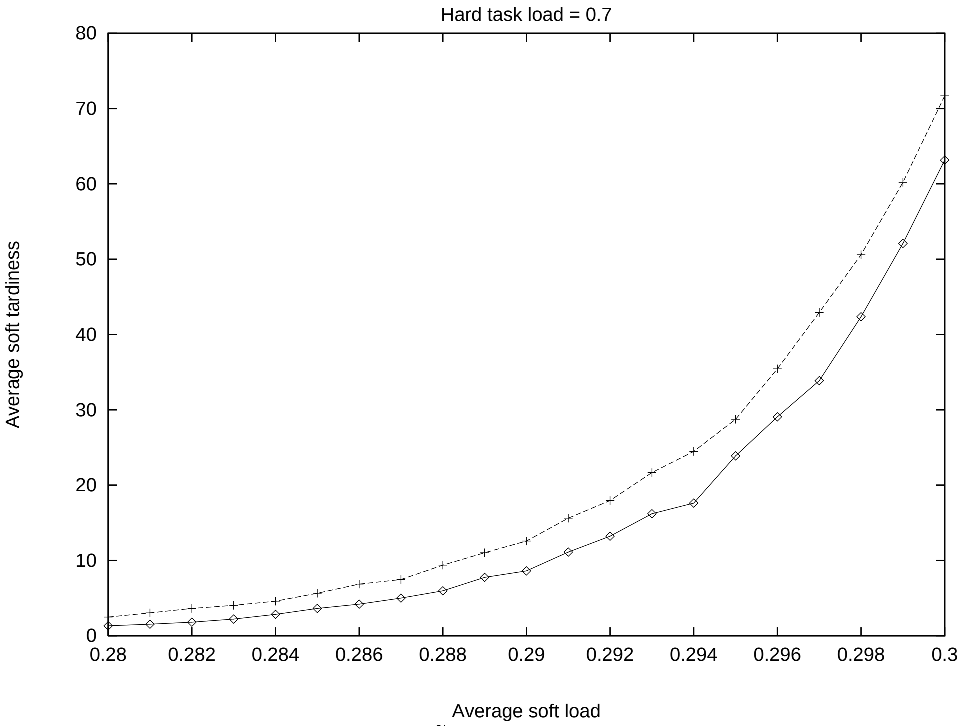

Figure 11: Second experiment.

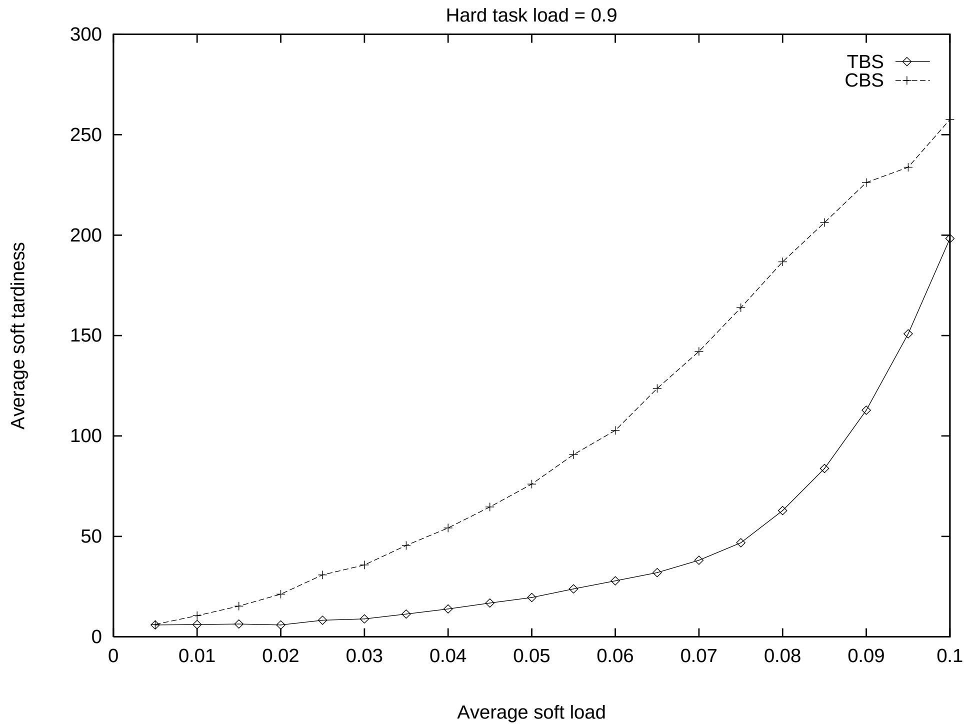

Figure 12: Third experiment.

When $WCET_i >> \overline{c_{i,j}}$ the TBS can cause an underutilization of the processor. This fact can be observed in Figure 13, which shows the results of a simulation similar to the previous one, in which $U_{hard} = 0.6$, $\overline{U_{soft}} = 0.4$, the inter-arrival 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.

5 Implementation and experimental results

The proposed CBS mechanisms has been implemented on the HARTIK kernel [1, 10], to support some sample multimedia applications.

The implementation of the CBS is relatively simple. In order to handle the budget exhausted event, the budget of the running task must be decreased by the system while the task executes. The simplest solution is to divide the time in ticks and assign each tick to a CBS, wich budget is decreased by 1. This solution is used by the most part of the existing operating systems (because it is very similar to the time quantum allocation in time sharig), but has a problem: all the times in the system (and in particular the arrival time $r_{i,j}$, execution time $c_{i,j}$ and finish time $f_{i,j}$ of each job) must be multiple of a system tick. Since the system tick cannot be too small (a tick less than 333 microseconds generates too overhead), the imposition made on $r_{i,j}$, $c_{i,j}$, and $f_{i,j}$ is too restrictive.

If we don't want these limitations on execution and inter-arrival time, we must manage the budget in an approximated way: each tick is arbitary considered as assigned to a CBS (that can be the CBS executing at the begin or at the end of the tick. This solution is simple, efficent (it doesn't introduce overhead) and has good performances, but doesn't ensure that the band required by a CBS is limited, so we decide to don't use it.

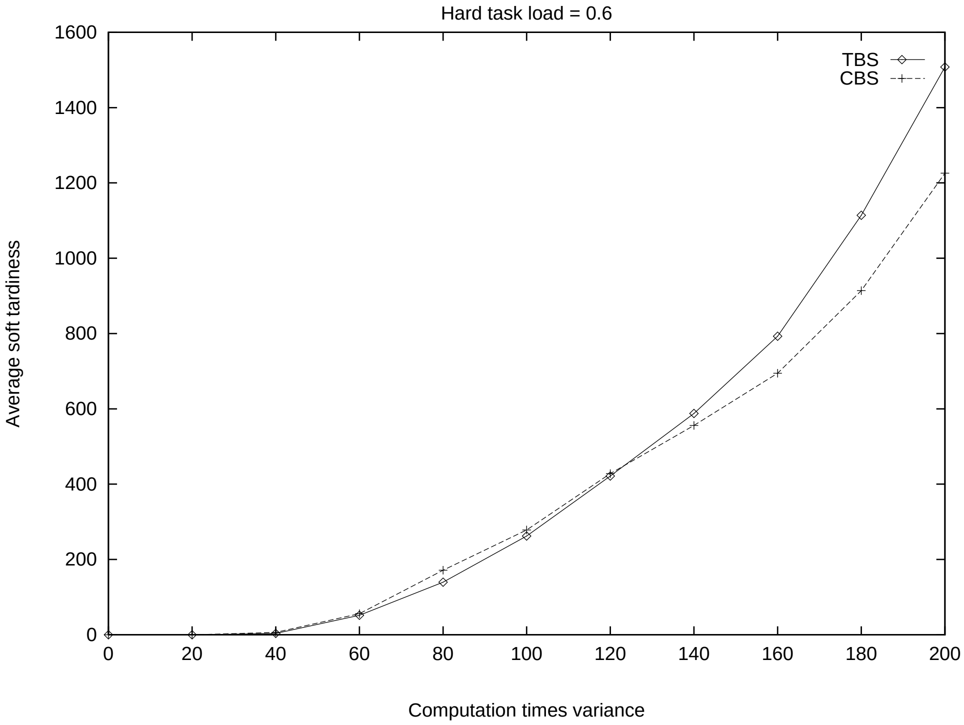

Figure 13: Fourth experiment.

Another solution is to decrease the budget every time that a context switch occurs: if a task executes from time t_1 to time t_2, its budget is decreased at time t_2 of a quantity equal to $t_2 - t_1$. The problem is that, in this way, it is difficult to catch the exhausted budget event; in fact, if a task executes from time t_1 to time t_2 and its budget at time t_1 is less than $t_2 - t_1$, the task uses more than its reserved bandwidth! To avoid this problem, the budget must be also updated at each tick; when at time t a system primitive performs a context switch, the time t_T of the next tick boundary is known, so the budget can be considered exhausted if it is less than $t_T - t$.

This technique also introduce an approximation in the budget management, but doesn't require any imposition on $r_{i,j}$, $c_{i,j}$, and $f_{i,j}$ and still ensure that a CBS doesn't require more than its reserved bandwidth.

The introduced approximation invalidates Lemma 1, so it seems to be impossible to schedule an hard task by a CBS. Moreover, it is possible to account this approximation in a formula similar to Lemma 1, but to do this we need some additional definitions.

Definition 1 *A TIM is the smallest time unit that can be measured (on a PC, $TIM = 1\mu sec$).*

Definition 2 *A Tick is a time unit composed of T^{tk} TIM: the timer generates an interrupt each Tick.*

Definition 3 *A Tick Boundary is a time $T^{tk}, 2T^{tk}, 3T^{tk}, ...$ in wich a timer interrupt arrives. Let's indicate the i^{th} Tick Boundary as $b_i = iT^{tk}$.*

Definition 4 *A R-CBS (Real CBS) is a CBS implementation such that the budget is managed in the following way:*

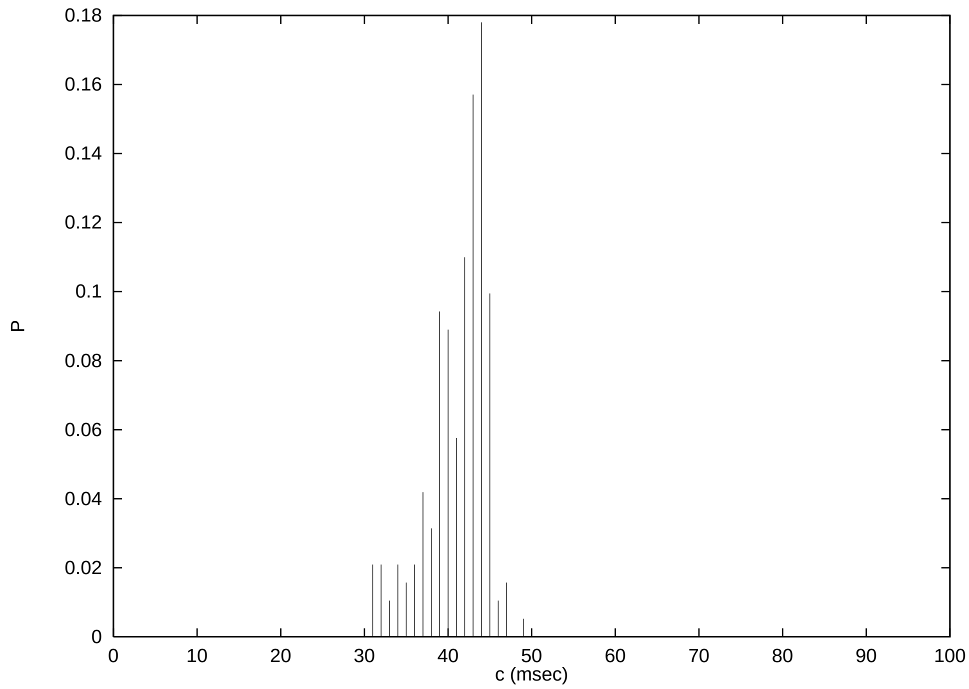

Figure 14: Task 1 execution times PDF.

- *the budged is updated ad the context switch and at the Tick Boundaries;*
- *let t_2 be a budget updating time, if the CPU was assigned to the R-CBS S_s in all the time intervall $[t_1, t_2]$, its budget is updated decreasing it by $t_2 - t_1$;*
- *when at time t_1 a job served by R-CBS S_s is scheduled, the budget c_s of the server is considered exhsusted if $c_s < t_2 - t_1$, being t_2 the next Tick Boundary ($t_2 = \min_i\{b_i : b_i > t_1\}$).*

Lemma 2 *A hard task τ_i having WCET C_i and minimum inter-arrival time T_i is schedulable by a R-CBS with parameters $(Q_s = C_i + T^{tk}, T_s = T_i)$ if and only if is schedulable by EDF.*

Proof.
Being

$$\forall j, r_{i,j+1} - r_{i,j} \geq T_i = T_s$$

each job is scheduled with an assigned absolute deadline $r_{i,j} + T_i$ and maximum budget $Q_s = C_i + T^{tk}$. But

$$\forall j, c_{i,j} \leq C_i = Q_s - T^{tk}$$

so each job finishes before that its budget becomes less than $1Tick$, hence also in the worst case (the job begins to execute an ϵ after a context switch, with ϵ small as you want) the budget will never be considered as exhausted. The absolute deadline assigned to the job doesn't change and remain equal to the deadline used by EDF scheduling alghotithm. Hence, the scheduled is the same obtained assigning to τ_i a relative deadline T_i and using EDF. $\square$

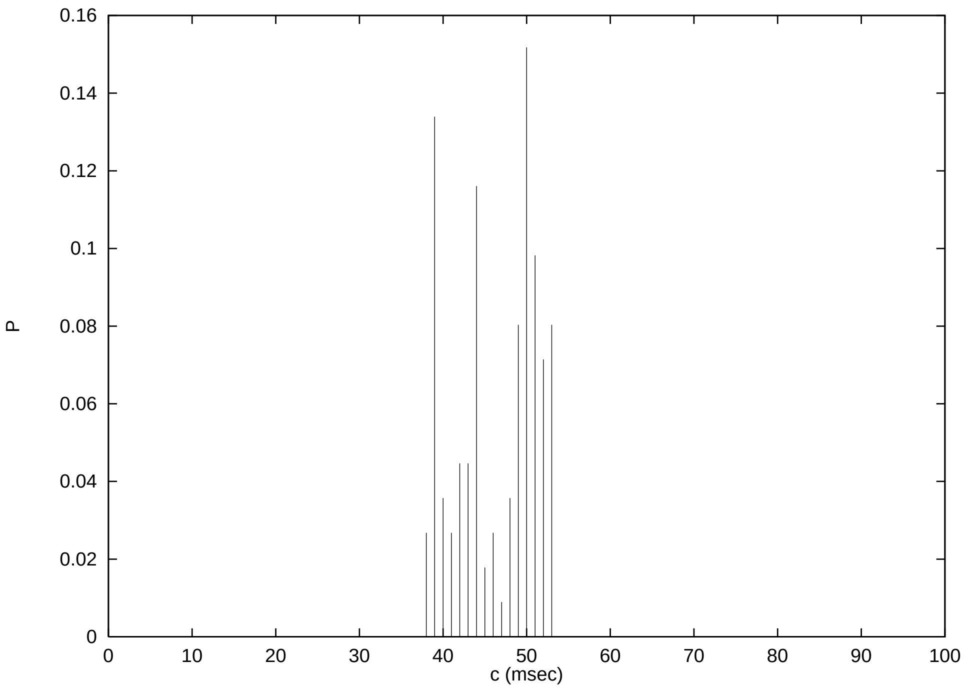

Figure 15: Task 2 execution times PDF.

Once the CBS mechanisms has been built into the HARTIK kernel, we have implemented an MPEG player and compared the performance with respect to a simple EDF scheduling scheme. In our experiment, two periodic tasks decode the MPEG videos which frame decoding time distributions are shown in Figures 14 and 15. Task τ_1 has required period $T_1 = 125ms$ (frame rate: 8 frames per second) and task τ_2 has a required period $T_2 = 30ms$ (33 Fps). Figure 16 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 rete (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 $17.96Fps$. When τ_2 terminates, τ_1 increase its frame rate to the maximum possible ($23, 8Fps$, that corresponds to a period of about $42ms$, which is the mean execution time for τ_1). After this transition time, τ_1 returns to execute at $8Fps$.

Figure 17 shows the number of decoded frames as a function of time, when the same periodic tasks are scheduled by a 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 $Q_1 = 42 \approx \overline{c_1}$, τ_1 will execute at a frame rate near to the required.

From the figure is clear how the frame rate of τ_1 is about constant and has only 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 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 right that it is penalized by CBS.

Notice that the proposed mechanism automatically arrange the task periods without using a-priori knowledge about the tasks' execution times (clearly, a similar result could be obtained changing T_1 and T_2 according with C_1 and C_2 in order to not overload the system). The only informations used by the CBS are the couple (Q_i, T_i) and the extimation of task execution

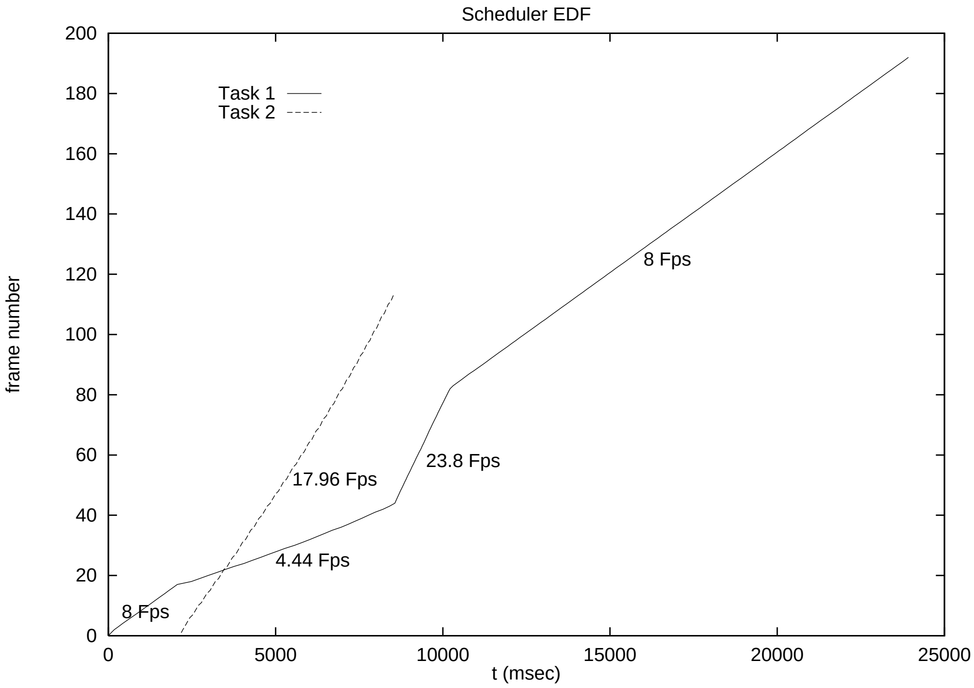

Figure 16: Two MPEG players scheduled by EDF.

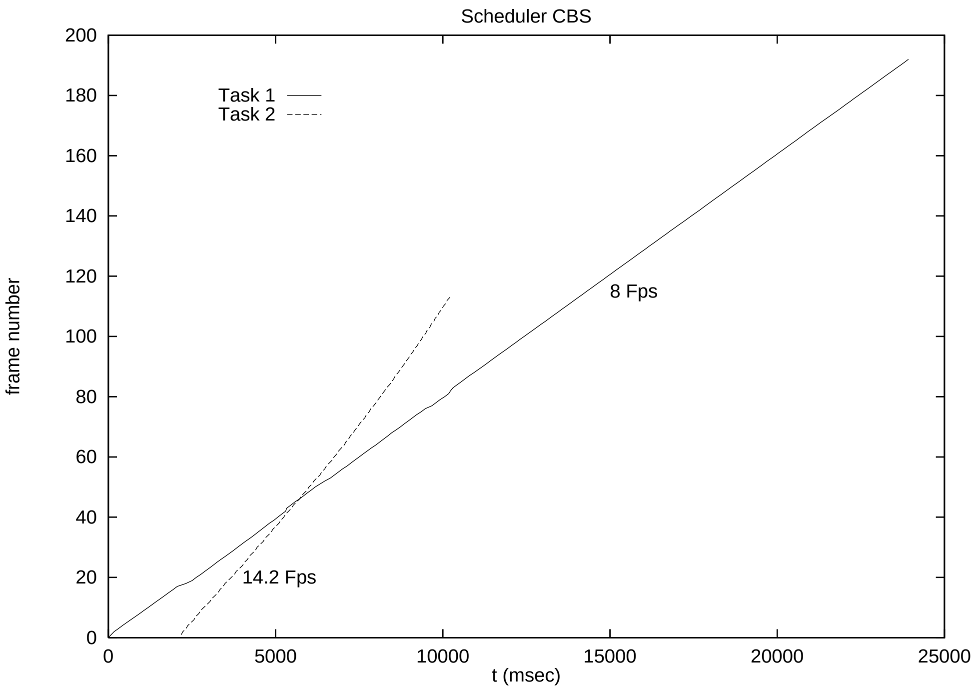

Figure 17: Two MPEG players scheduled by CBS.

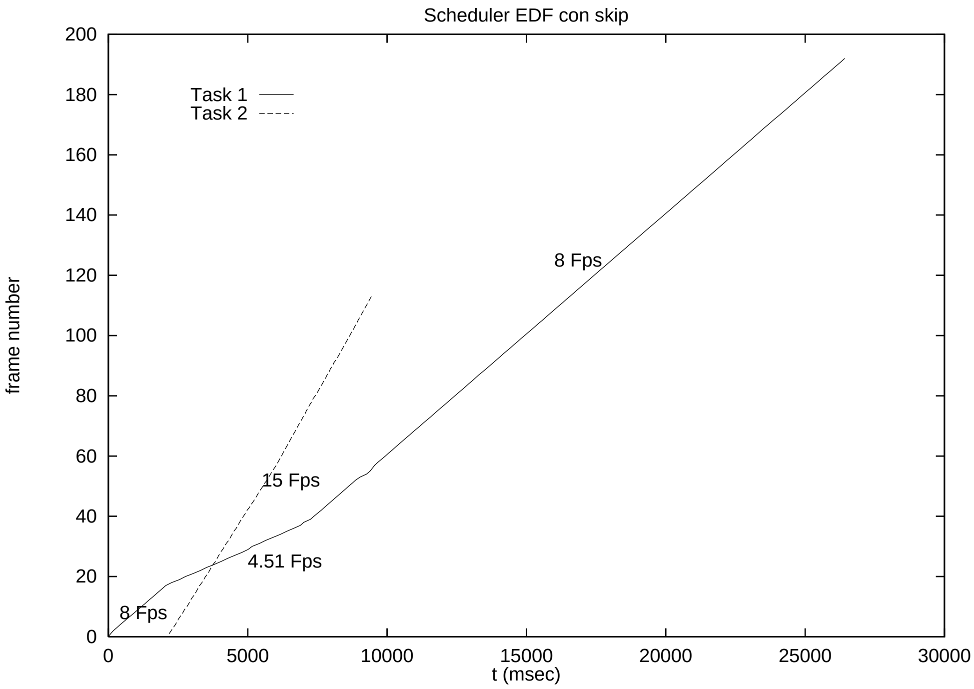

Figure 18: Two MPEG players scheduled by EDF with skip.

time given by the budget.

Figure 16 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 finish after its absolute deadline, the next jobs is skipped in order to finish correctly.

Implementing a skip strategy to serve soft periodic tasks, the indesiderable effects disappares (as shown in Figure 18), but there is another problem, visible in an exeperiment in wich the same movie is decoded by two identical tasks, with $\overline{U_{soft}} \approx 1$.

It is easy to see that, although the two tasks have the same period, they proceed with different speeds. This is due to the fact that the system is overloaded. In fact, if

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

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

Figure 19 shows the number of decoded frames as a function of time when the two tasks are served by an EDF algorith implementing skip: it is visible how the two movies aren't reproduced at the same rate.

In Figure 20 is showed how the CBS scheduling doesn't soffer this problem. This figure shows the results of an experiment in which the two tasks are served by two identical CBSs with parameters $Q_s = \overline{c_{1,j}} = \overline{c_{2,j}}$ and $T_s = 2Q_s$ (the parameters are equal because the two tasks play the same video). The result of this test clearly shows that the CBS introduces a form of fairness in the scheduling and allows the two tasks to proceed at the same rate.

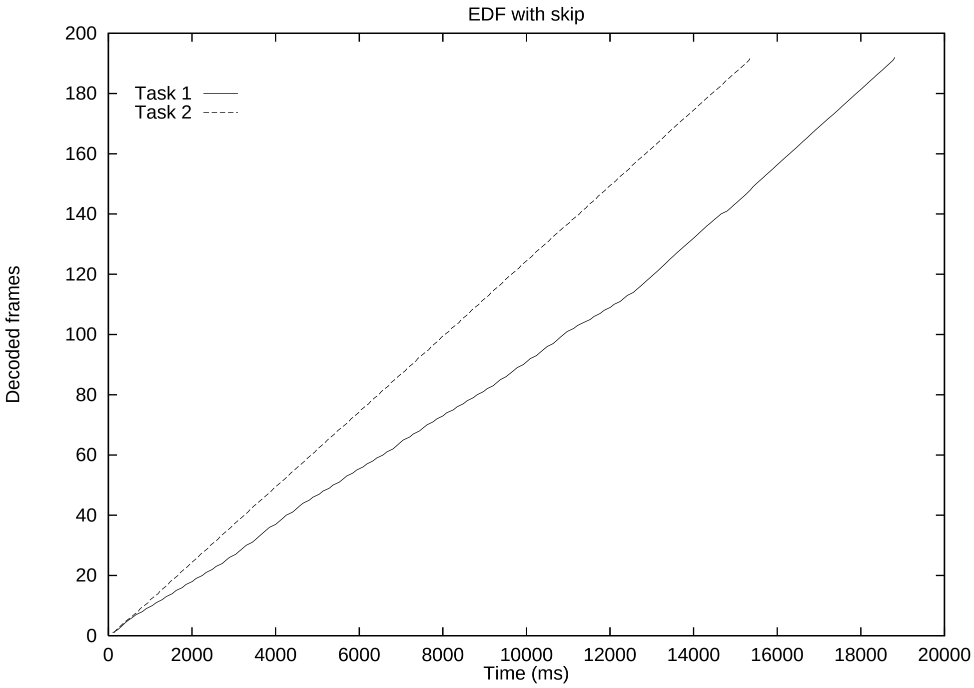

Figure 19: Two identical MPEG players scheduled by EDF with skip.

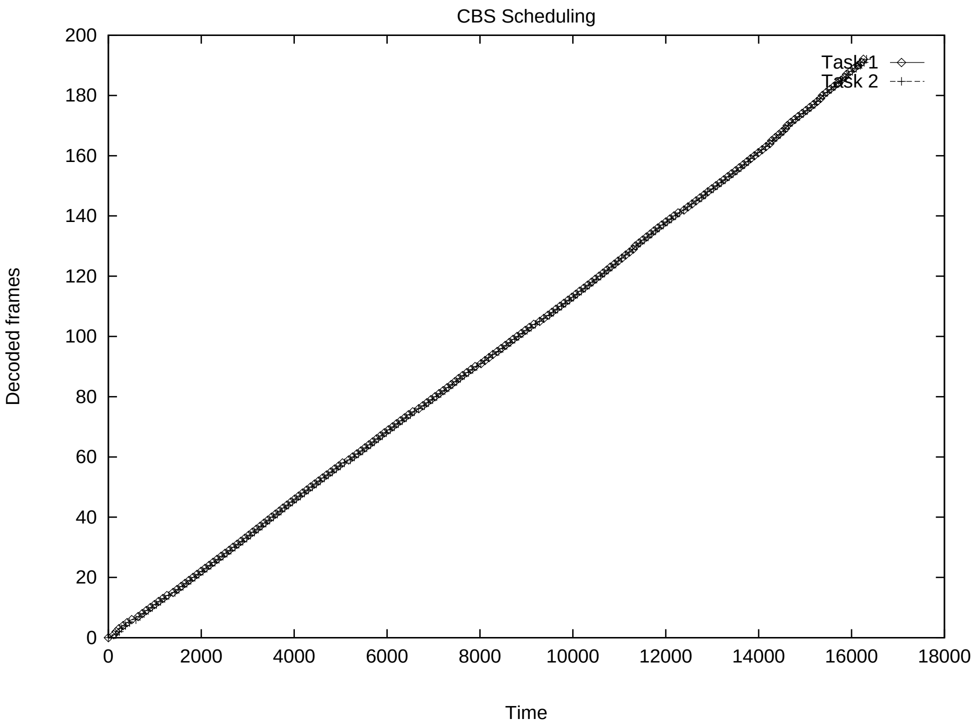

Figure 20: Two identical MPEG players scheduled by CBS.

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 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.

In order to use proposed model in more general situations, the following issues need to be investigated:

- handling resource constraints: a concurrency control protocol needs to be integrated with the method to avoid priority inversion when accessing shared resources;
- supporting adaptive applications: a served task could use the difference between the current CBS deadline and its deadline to evaluate the request in excess and react accordingly. Such a kind of feedback could be used for adjusting the QoS in overload conditions.
- supporting more applications in the same system: 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 [2].
- dynamic QoS management: a task can be used as a QoS manager to change dynamically the bandwidth reserved to each multimedia task. The strategies for changing the parameters of each CBS still have to be investigated.

References

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

[2] Z. Deng and J. W. S. Liu. "scheduling real-time applications in open envirovment". In *IEEE Real-Time Systems Symposium*, San Francisco, 1997.

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

[4] T. M. Ghazalie and T.P. Baker. "aperiodic servers in a deadline scheduling environment". *Real-Time Systems*, (9), 1995.

[5] Ramesh Govindan and D. P. Anderson. "scheduling and ipc mechanisms for continuous media". In *ACM Symposium on Operating Systems Principles*, Pacific Grove, 1991.

[6] Pawan Goyal, Xingang Guo, and Harrik M. Vin. "a hierical cpu scheduler for multimedia operating systems". In *2nd OSDI Symposium*, 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] Hiroyuki Kaneko, John A. Stankovic, Subhabrata Sen, and Krithi Ramamritham. "integrated scheduling of multimedia and hard real-time tasks". In *IEEE Real Time System Symposium*, 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*, 1973.

[12] Clifford W. Mercer, Stefan Savage, and Hideyuki Tokuda. "processor capacity reserves for multimedia operating systems". Technical report, Carnegie Mellon University, Pittsburg, May 1993.

[13] B. Sprunt, L. Sha, and J. P. Lehoczky. "aperiodic scheduling for hard real-time systems". *The Journal of Real-Time Systems*, (1), 1989.

[14] M. Spuri and G. Buttazzo. "scheduling aperiodic tasks in dynamic priority systems". *Real-Time Systems*, 10, 1996.

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

[16] Marco Spuri, Giorgio Buttazzo, and Fabrizio Sensini. "robust aperiodic scheduling under dynamic priority systems". In *IEEE Real-Time Systems Symposium*, December 1995.

[17] Ian Stoica, Hussein Abdel-Wahab, Kevin Jeffay, Sanjoy K. Baruah, Johannes E. Gehrke, and C. Greg Plaxton. "a proportional share resource allocation algorithm for real-time, time-shared systems". In *IEEE Real Time System Symposium*, 1996.