Real-Time Systems, 27, 123–167, 2004

Resource Reservation in Dynamic Real-Time Systems

LUCA ABENI lucabe72@email.it
Broadsat S.r.l., Pisa, Italy

GIORGIO BUTTAZZO giorgio@sssup.it
University of Pavia, Pavia, Italy

Abstract. This paper focuses on the problem of providing efficient run-time support to multimedia applications in a real-time system, where different types of tasks (characterized by different criticality) can coexist. Whereas critical real-time tasks (hard tasks) are guaranteed based on worst-case execution times and minimum interarrival times, multimedia tasks are served based on mean parameters. A novel bandwidth reservation mechanism (the constant bandwidth server) allows real-time tasks to execute in a dynamic environment under a temporal protection mechanism, so that each task will never exceed a predefined bandwidth, independently of its actual requests. The paper also discusses how the proposed server can be used for handling aperiodic tasks efficiently and how a statistical analysis can be applied to perform a probabilistic guarantee of soft tasks. The performance of the proposed method is compared with that of similar service mechanisms (dynamic real-time servers and proportional share schedulers) through extensive simulation experiments.

Keywords: real-time scheduling, multimedia, quality of service, resource reservation

1. Introduction

A real-time system is a computer system whose behavior depends not only on the functional correctness of its outputs, but also on the time on which results are produced. For some critical applications (e.g., flight control systems, defense systems, nuclear power plants), missing a single temporal constraint (deadline) might cause a catastrophic consequence. In this case, we say that tasks are characterized by hard timing constraints that must be met in all anticipated workload conditions. An operating system able to support this kind of task is called a hard real-time system.

In these systems, in order to provide an off-line guarantee on the feasibility of each critical task, schedulability analysis is performed based on very pessimistic assumptions. This means assuming that each task instance always issues requests at its maximum activation rate and always uses its worst-case computation time. Moreover, to minimize conflicts with the other tasks, resources are statically allocated to tasks based on maximum requirements.

Such a design philosophy makes the system robust in tolerating peak load situations, but causes a tremendous waste of resources, which are underutilized for most of the time. In other words, high predictability is achieved at the price of low efficiency, which increases the overall cost of the system.

Although such an approach can be justified for safety-critical systems, it would not be appropriate for non-critical real-time applications. Indeed, most real-time systems, such as multimedia systems, control system simulators, virtual reality games, and even some robotic systems, are not characterized by hard timing constraints. For such systems, missing one or more deadlines does not generate catastrophic consequences on the environment, but only causes a performance degradation. For these systems, using a hard real-time approach to guarantee a certain level of performance would mean wasting a lot of resources and paying a much higher cost.

In addition, for some application, a hard real-time approach is also difficult to adopt, since estimating worst-case parameters can be very difficult or even impossible. Continuous media (CM) applications are typical examples. CM activities—such as audio and video streams—need real-time support because of their sensitivity to delay and jitter. However, the use of a hard real-time system for handling CM applications would 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 higher 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 would waste system's resources, especially the CPU.

- Providing a precise estimation of WCETs is a very difficult task, even for those applications running on a specific hardware. This problem is much more critical for multimedia applications, which in general run on many 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 jobs that manage such data may not be deterministic, so determining a minimum interarrival time (MIT) for such jobs, may not be possible. As a consequence, no *a priori* guarantee can be performed.

- Advanced multimedia systems tend to be more dynamic than classical real-time systems, so most of 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, such as Unix or Windows, to manage CM applications.

To provide adequate support for such soft real-time applications, the challenge is to increase efficiency on resource usage (to keep the overall cost low), and to remove any assumption on the exact knowledge of execution times, still ensuring a certain level of guarantee on system's performance. In these systems, the concept of absolute guarantee (given by a yes-or-no feasibility test) is too restrictive and should be replaced with the notion of quality of service (QoS), which can be specified within a much larger gray-level scale. More specifically, the features that would be desirable for supporting CM applications include:

- increasing resource utilization by adopting less pessimistic assumptions on tasks' behavior;
- handling task overruns to achieve graceful degradation on system's performance during transient overload conditions;
- providing task isolation to avoid reciprocal interference during overruns (temporal protection), so that a task overrun will not delay the execution of the other tasks;
- allowing efficient aperiodic task management, to improve aperiodic responsiveness while preserving the desired QoS level of periodic tasks; and
- introducing the notion of probabilistic guarantee, to relax hard timing constraints in favor of efficiency.

This paper presents a new scheduling approach which allows us to address all the properties listed above. In particular, it integrates and consolidates some preliminary work described in Abeni and Buttazzo (1998, 1999b, c).

The rest of the paper is organized as follows. Section 1.1 describes some related work presented in the literature. Section 2 describes our general approach and states our terminology and assumptions. Section 3 presents the scheduling algorithm and its main properties. Section 4 addresses the problem of aperiodic task management. Section 5 illustrates how statistical analysis can be used to perform a probabilistic guarantee. Section 6 discusses how the proposed method can be used to handle proportional share tasks and provides a comparison with other resource reservation techniques. Finally, Section 7 presents our conclusions and future work.

1.1. Related Work

Recently, significant work has been devoted at increasing the efficiency of real-time systems through more flexible algorithms and scheduling paradigms.

In Goyal et al. (1996), the authors proposed a variant of the Solaris operating system, based on the start fair queuing (SFQ) scheduling algorithm. In Stoica et al. (1996), the FreeBSD scheduler has been modified to implement the earliest eligible virtual deadline first (EEVDF) scheduling algorithm. In Waldspurger and Weihl (1995) the Linux kernel has been modified to provide stride scheduling. The three scheduling policies cited above are essentially based on a common approach referred as proportional share resource allocation (PSA). The essence of PSA is to allocate a resource to a task for a uniform fraction of any interval of time: in this way, each task makes progress as if it executes alone on a slower processor. Although such a ''fairness'' in tasks management allows to handle QoS requirements, spreading process execution can be too restrictive for some applications (see Section 6).

A different approach to the problem of providing QoS control in an operating system is to adapt the real-time theory to a more flexible environment. Govindan and Anderson

(1991) describe a multimedia operating system based on an earliest deadline first (EDF) scheduler. However, the 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).

Jeffay et al. (1992) present 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 MIT. 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 introduces the rate-based execution (RBE) task model, which is independent from the MIT (Jeffay and Bennet, 1995; Jeffay and Goddard, 1999). 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.

Mercer et al. (1993) 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 overruns 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 for an unpredictable amount of time.

Kaneko et al. (1996) propose a scheme based on a periodic process (the multimedia server) dedicated to the service of all multimedia requests. This solution allows to nicely integrate multimedia tasks together with hard real-time tasks; however, since a single server is used to handle different streams, it is not easy to control the QoS of each task.

Deng and Liu (1997) 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 (Deng et al., 1997) for tasks that do not use non-preemptable sections or global resources, and a total bandwidth server (Spuri and Buttazzo, 1994; 1996) 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 Zhang (1995) the virtual clock algorithm has been proposed to serve network packets within a reserved bandwidth. Since the method is based on the knowledge of the packet size, it cannot be used to serve tasks whose execution times are not known in advance. Moreover, the virtual clock algorithm results to be equivalent to the total bandwidth server, which is compared with the approach proposed in this paper.

In many practical real-time applications, the requirements for tasks are less stringent than real-time theory usually permits. A more flexible approach could be to guarantee a completion probability, rather than a hard deadline, since a task is usually acceptable if the probability of meeting a deadline is greater than a given value. In Tia et al. (1995), a probabilistic analysis of the rate monotonic (RM) scheduling algorithm is performed, and some solutions to guarantee a completion probability are presented. In Atlas and Bestavros (l998), the RM algorithm is modified to perform on-line acceptance of single jobs for ensuring (off-line) a minimum probability that a job is accepted; these two works are both limited to a static priority (RM) assignment and to a semi-periodic (fixed interarrival times) task model.

In Kang et al. (1997) a time division multiplexing (TDM) model is used to share resources among tasks in a distributed system. In this paper and in two successive works (Kang et al., 1998a, b) a statistical approach is used at design phase to compute the probability distribution of tasks' finishing times and check whether a distributed system verifies some given constraints on latency and minimum output rate.

In Lehoczky (1996) a statistical analysis of the EDF scheduling algorithm is presented. This analysis is based on the real-time queuing theory, that adapts the classical queuing theory to handle queues containing clients characterized by a deadline. Although this method can be used to estimate the system performance, it can hardly be used to enforce the desired timing behavior on individual tasks.

2. Our Approach

In this section we present a new resource reservation method for executing soft real-time tasks. We first introduce the task models and then show how these tasks can receive adequate service.

2.1. Task Models

We consider a system consisting of three types of tasks: hard, soft, and non-real-time tasks. A real-time task τ_i is a stream of requests, or jobs, $J_{i,j}$, $(j = 1, \dots, n)$, each characterized by an arrival time $r_{i,j}$, an execution time $c_{i,j}$, a finishing time $f_{i,j}$, and a deadline $d_{i,j}$. Real-time tasks can be hard or soft: a task τ_i is said to be hard if all its instances have to complete within their deadline $(\forall j\, f_{i,j} \leq d_{i,j})$, otherwise a critical failure may occur in the system. A task is said to be soft if a deadline miss in one or more instances does not cause a critical failure, but only a QoS degradation.

A hard real-time task is typically characterized by two parameters (C_i, T_i), where $C_i = \max\{c_{i,j}\}$ is the WCET of each job and $T_i = \min\{r_{i,j+1} - r_{i,j}\}$ is the MIT between successive jobs $J_{i,j+1}$ and $J_{i,j}$. Notice that a periodic task is a special case of real-time task, where T_i is equal to the task activation period, so that $r_{i,j+1} = r_{i,j} + T_i$ for any job $J_{i,j}$. The system must provide an *a priori* guarantee that all jobs of a hard task must complete within 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 characterized by more relaxed constraints and by a weaker knowledge of the task parameters. For each soft job $J_{i,j}$, a soft deadline is set at time $d_{i,j} = r_{i,j} + T_i$. When execution times are not known, the sequence $c_{i,j}$ can be seen as a sequence of values distributed according to a probability distribution function (PDF) $U(c) = P\{c_{i,j} = c\}$, or as a stochastic process. Since $U(c)$ does not depend on j, the stochastic process is stationary and time-invariant, so it is ergodic. Hence, the execution times expectation $E[c] = \sum cU(c)$ is equal to the mean execution time, calculated as $\bar{c} = \lim_{k\to\infty} \Sigma_{j=1}^{k} c_{i,j}/k$. If the execution and interarrival times PDFs are known, a statistical guarantee can be performed as described in Section 5.

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, non real-time tasks do not have timing constraints and can be scheduled in background with respect to hard and soft tasks (e.g., in a round robin fashion).

2.2. Bandwidth Reservation

Treating soft tasks as hard tasks is not appropriate for the reasons mentioned in Section 1. Moreover, when handling hybrid task sets consisting of hard and soft multimedia tasks, an underestimation of the WCET (or an overestimation of the MIT) would jeopardize the guarantee done on hard tasks. This problem is due to the fact that hard real-time systems do not enforce any kind of temporal protection among tasks (since worst-case parameters are known, the *a priori* analysis makes temporal protection useless for hard tasks). In order to serve soft tasks in a real-time system, however, an explicit temporal protection mechanism must be used to protect hard tasks from overruns generated by soft tasks. Temporal protection can be achieved by using a proper resource reservation strategy.

According to the approach we propose in this paper, each soft task is assigned a fraction of the CPU (a bandwidth) and it is scheduled in such a way that it will never demand more than its reserved bandwidth, independently of its actual requests. This is achieved by assigning each task a suitable (dynamic) deadline, computed as a function of the reserved bandwidth and its actual requests. If a task requires to execute more than its expected computation time, its deadline is postponed so that its reserved bandwidth is not exceeded. As a consequence, overruns occurring on task τ_a will only delay task τ_a, but will not steal the bandwidth assigned to the other tasks, which are then isolated and protected from reciprocal interference. By isolating the effects of task overloads, hard tasks can be guaranteed using classical schedulability analysis (Liu and Layland, 1973).

In our methodology, the bandwidth reservation mechanism is implemented by handling each soft task by a dedicated real-time server, which keeps the task demand within the reserved bandwidth through a suitable deadline assignment algorithm.

The bandwidth reservation mechanism is schematically illustrated in Figure 1. Hard tasks are guaranteed *a priori* based on worst-case parameters and are directly inserted in the ready queue, managed by the earliest deadline first EDF scheduling algorithm. Each

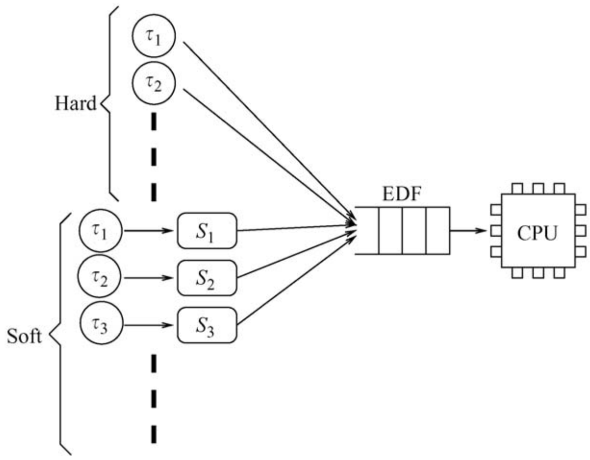

Figure 1. Implementing bandwidth reservation through dedicated servers.

soft task is handled by a dedicated server, which ensures that its served task will never contribute to the total processor load more than a given utilization U_{s_i}, specified by the user.

It is worth noting that, using this approach, a task is re-shaped before entering the EDF queue, so that it will not demand more than the reserved bandwidth. The server mechanism is responsible for such a re-shaping and it must provide the following important properties:

1. *Feasibility:* No deadline (hard or assigned by the scheduler) must be missed.

2. *Efficiency:* Tasks should be served in order to reduce the mean tardiness.

3. *Flexibility:* The server should handle tasks with variable or even unknown execution times and arrival patterns.

Any aperiodic server, like the Dynamic Polling Server (Spuri and Buttazzo, 1996), the Dynamic Deferrable Server (Ghazalie and Baker, 1995), the Dynamic Sporadic Server (DSS) (Ghazalie and Baker, 1995; Spuri and Buttazzo, 1996), the Total Bandwidth Server (TBS) (Spuri and Buttazzo, 1994, 1996), or the Constant Utilization Server (CUS) (Deng et al., 1997) can be used to serve soft tasks, but each of these servers fails in providing one of the previous properties. In particular, the TBS and the CUS require the WCET knowledge, hence they cannot be used in practical situations, whereas the other servers do not provide sufficient responsiveness in terms of mean tardiness. Hence, the problem is to develop a server mechanism that does not depend on any WCET or MIT knowledge, uses efficiently the CPU time (obtaining good performance), and provides temporal isolation handling periodic and aperiodic arrivals in a proper way.

In this paper we present the Constant Bandwidth Server (CBS), a novel server that achieves a performance similar to the one of the TBS, without requiring the WCET or

MIT knowledge in advance. Moreover, if the WCET and MIT are known, the server parameters can be set so that the served task behaves exactly as a hard task (hence, the CBS can properly serve also hard tasks).

3. The Constant Bandwidth Server

In this section, we present a novel service mechanism to properly implement a bandwidth reservation strategy. The service mechanisms that have inspired this work are the DSS and the TBS. 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 CUS, whose actual contributions are limited to 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.

The basic idea behind the CBS mechanism can be explained as follows: when a new job enters the system, it is assigned a suitable scheduling deadline (to keep its demand within the reserved bandwidth) and it is inserted in the EDF ready queue. If the job tries to execute more than expected, its deadline is postponed (i.e., its priority is decreased) to reduce the interference on the other tasks. Note that by postponing the deadline, the task remains eligible for execution. In this way, the CBS behaves as a work conserving algorithm, exploiting the available slack in an efficient (deadline-based) way, thus providing better responsiveness with respect to non-work conserving algorithms and to other reservation approaches that schedule the extra portions of jobs in background, like the one described by Mercer et al. (1993).

If a subset of tasks is handled by a single server, all the tasks in that subset will share the same bandwidth, so there is not isolation among them. Nevertheless, all the other tasks in the system are protected against overruns occurring in the subset.

In order not to miss any hard deadline, the deadline assignment rules adopted by the server must be carefully designed. The following section precisely defines the CBS algorithm, and formally proves its correctness for any (know or unknown) execution request and arrival pattern.

3.1. Definition of CBS

The CBS can be defined as follows:

- A CBS is characterized by a budget c_s and by an 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 at 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) 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 at 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 2 illustrates an example in which a hard periodic task, τ_1, with computation time $C_1 = 4$ and period $T_1 = 7$, is scheduled together with a soft task, τ_2, served by a CBS having a budget $Q_s = 3$ and a period $T_s = 8$. The first job of τ_2 ($J_{2,1}$), requiring four units of execution time, arrives at time $r_1 = 3$, when the server is idle. Being $c_s \geq (d_0 - r_1)U_s$,

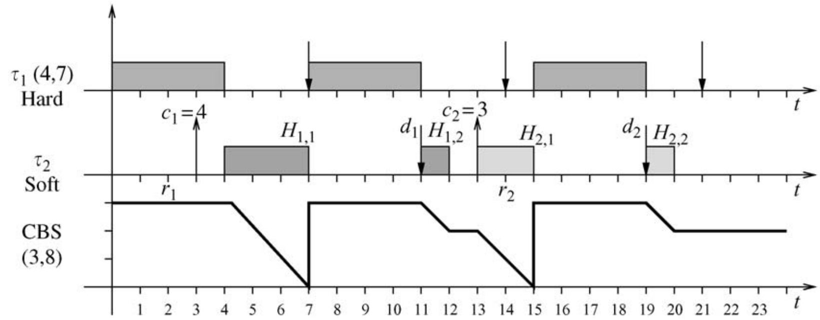

Figure 2. An example of CBS scheduling.

the job is assigned a deadline $d_1 = r_1 + T_s = 11$ and c_s is recharged at $Q_s = 3$. At time $t = 7$, the budget is exhausted, so a new deadline $d_2 = d_1 + T_s = 19$ is generated and c_s is replenished. Since the server deadline is postponed, τ_1 becomes the task with the earliest deadline and executes until completion. Then, τ_2 resumes and job $J_{2,1}$ (having deadline $d_2 = 19$) is finished at time $t = 12$, leaving a budget $c_s = 2$. The second job of task τ_2 arrives at time $r_2 = 13$ and requires three units of time. Since $c_s < (d_2 - r_2)U_s$, the last server deadline d_2 can be used to serve job $J_{2,2}$. At time $t = 15$, the server budget is exhausted, so a new server deadline $d_3 = d_2 + T_s = 27$ is generated and c_s is replenished at Q_s. For this reason, τ_1 becomes the highest priority task and executes until time $t = 19$, when job $J_{1,3}$ finishes and τ_2 can execute, finishing job $J_{2,2}$ at time $t = 20$ leaving a budget $c_s = 2$.

It is worth noting 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 the 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 theorems.

THEOREM 1 *The CPU utilization of a CBS S with parameters (Q_s, T_s) is $U_s = Q_s/T_s$, independently from the computation times and the arrival pattern of the served jobs.*

Proof: The proof is trivial for the case of periodic arrivals, but proving that no deadline is missed in the general case of arbitrary arrival patterns requires a more formal (and longer) verification. See Appendix for the complete proof. ∎

The following theorem provides a simple guarantee test for verifying the feasibility of a task set consisting of hard and soft tasks.

```
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 executes for a time unit
       c = c - 1;
When (c == 0)
       /*---------------Rule 3---------------*/
       k = k + 1;
       a_k = current_time();
       d_k = d_{k-1} + T_s;
       c = Q_s;
```

Figure 3. The CBS algorithm.

THEOREM 2 *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: *If.* Assume $U_p + U_s \leq 1$ and suppose there is an overflow at time t. The overflow is preceded by a period of continuous utilization of the processor. Furthermore, from a certain point t' on $(t' < t)$, only instances of tasks ready at t' or later and having deadlines less than or equal to t are run. Let C be the total execution time demanded by these instances. Since there is an overflow at time t, we must have

$$t - t' < C$$

Since C is due to periodic and aperiodic jobs, we can write:

$$C = \sum_{i=1}^{n} \left\lfloor \frac{t - t'}{T_i} \right\rfloor C_i + D_s(t', t)$$

where $D_s(t', t)$ is the computational demand due to the jobs served by the CBS in the interval $[t, t]$.

Applying Theorem 1 we have

$$\begin{aligned} C &\leq \sum_{i=1}^{n} \frac{t - t'}{T_i} C_i + (t - t')U_s \\ &\leq (t - t')(U_p + U_s) \end{aligned}$$

Thus, it follows that

$$U_p + U_s > 1$$

a contradiction.

Only If. If an aperiodic request enters the system periodically, with period T_s and execution time $C = Q_s$, a CBS behaves exactly as a periodic task with period T_s and execution time Q_s; in this case, the total utilization factor of the processor is $U_p + U_s$. Hence, if the whole task set is schedulable, from the EDF schedulability bound (Liu and Layland, 1973) we can conclude that $U_p + U_s \leq 1$. ■

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 soft tasks can be scheduled together with hard tasks without affecting the *a priori* guarantee, even in the case in which the execution times of the soft tasks are not known or the 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 algorithm 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 $r_{i,j+1} - r_{i,j} \geq T_i$ 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 is never postponed 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. (1993).

- Knowing the statistical distribution of the computation time of a task served by a CBS, it is possible to perform a QoS guarantee based on probabilistic deadlines (expressed in terms of probability for each served job to meet a deadline). Such a statistical analysis is presented in Section 5.

3.3. Implementation and Experimental Results

The proposed CBS algorithm has been implemented on the HARTIK kernel (Buttazzo, 1993; Lamastra et al., 1997), to support some sample multimedia applications. The algorithm can also be easily implemented in different real-time kernels where time accounting is provided. It is worth observing that a time accounting mechanism is provided in most of real-time OSs (e.g., RT-Linux, Linux/RK, Nemesis, Rialto, VxWorks, QNX/Neutrino), and in all POSIX compliant kernels. As an example, the CBS has been implemented also in Linux/RK (Rajkumar et al., 2000).

In order to show the effectiveness of the proposed approach on a concrete case study, we implemented a simple multimedia application consisting of a set of two MPEG players, which typically exhibit a highly variable load, since the frame decoding time depends on the frame type (traditionally denoted as I, P, or B). To create a more dynamic load, tasks are activated at different time instants. In particular, the first MPEG player, denoted as task τ_1, has a period $T_1 = 125$ ms (frame rate: 8 frames per second), whereas the second player, denoted as task τ_2, has a period $T_2 = 30$ ms (33 Fps).

To show the advantages presented by our approach, each player has been executed using EDF, with and without CBS. Figure 4 reports the number of decoded frames as a function of time, when the two periodic tasks are scheduled by EDF and τ_2 is activated at $t = 2000$ ms. When τ_1 is the only task in the system, it runs at the required frame rate (8 Fps), but when τ_2 is activated τ_1 slows down at 4.4 Fps, while τ_2 begins to execute at 17.96 Fps $\neq$ 33 Fps. Such a difference in the frame rate indicates the presence of an overload condition; indeed, some measurements on the execution times revealed that the τ_1's WCET is $C_1 = 49$ ms, and the τ_2's WCET is $C_2 = 53$ ms, thus $U = 49/125 + 53/30 = 2.158 > 1$.

When τ_2 terminates, τ_1 increases its frame rate to its maximum value (23.8 Fps, that corresponds to a period of about 42 ms, which results to be the mean execution time for τ_1). After this transient, τ_1 returns to execute at 8 Fps.

Figure 5 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

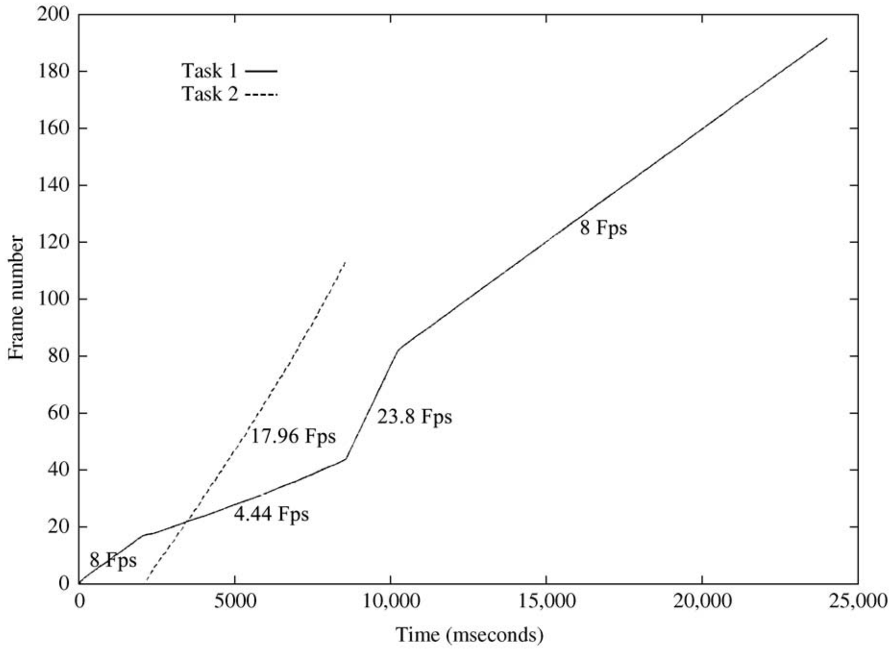

Figure 4. Two MPEG players scheduled by EDF.

$(Q_2, T_2) = (19, 30)$. 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 plots 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 its demand can be limited by assigning a maximum bandwidth to its dedicated server.

Notice that the proposed mechanism automatically arranges the task periods without using an *a priori* knowledge of tasks' execution times. The only information used by the CBS is the couple (Q_i, T_i) and the estimation of task's execution time given in the budget.

Figure 4 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 in order to recover from overruns.

As shown in Figure 6, a skip strategy eliminates accelerations in the movie, but introduces another problem, visible in an experiment illustrated in Figure 7, in which the same movie is decoded by two identical tasks, with $\overline{U_{\text{soft}}} \simeq 1$. Although the two tasks have

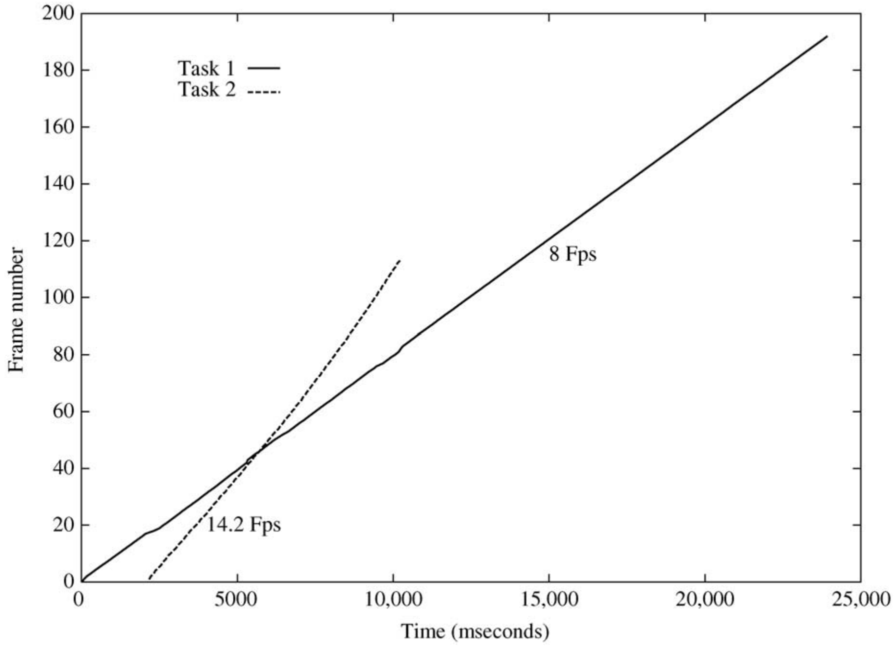

Figure 5. Two MPEG players scheduled by CBS.

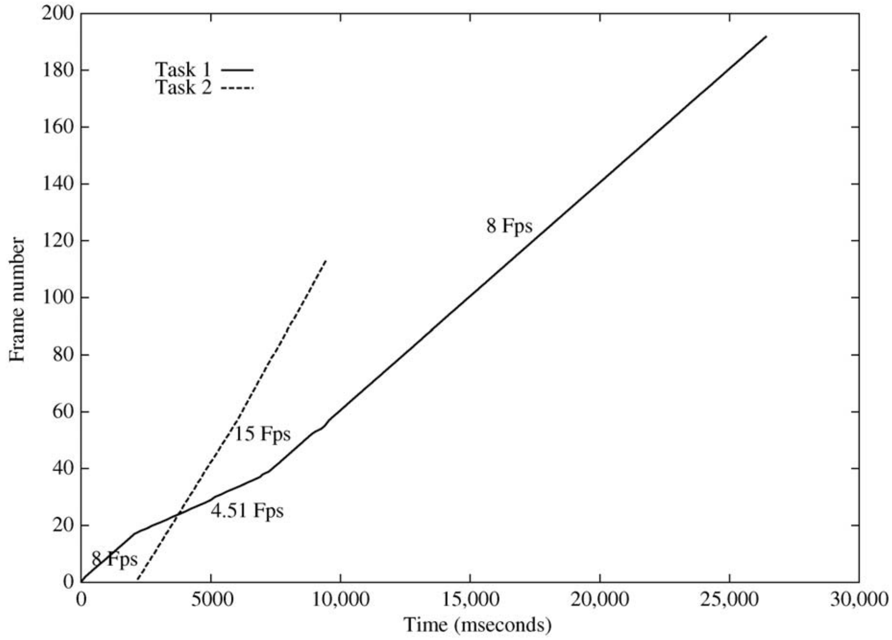

Figure 6. Two MPEG players scheduled by EDF with skip.

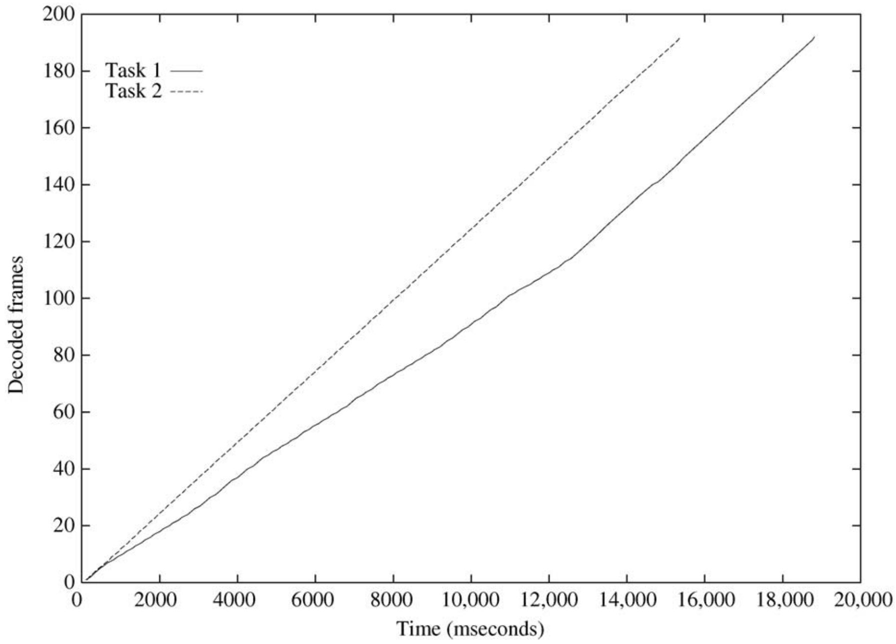

Figure 7. Two identical MPEG players scheduled by EDF with skip.

the same period, they proceed with different speeds. This is due to the fact that the system is overloaded. In fact, if

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

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

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

3.4. Notes on Resource Sharing

In a multiprogrammed system, tasks are rarely independent one from another, but must often cooperate in order to provide the expected service. In a shared memory programming paradigm, cooperation is achieved through shared resources, which must be used in mutual exclusion to preserve data consistency during concurrent accesses.

The use of classical semaphores for protecting critical sections of code, however, is prone to priority inversion, which causes a high priority task to be blocked by a low priority task for an unbounded amount of time, due to the interference of medium priority tasks. Different solutions have been proposed in the literature to bound the priority

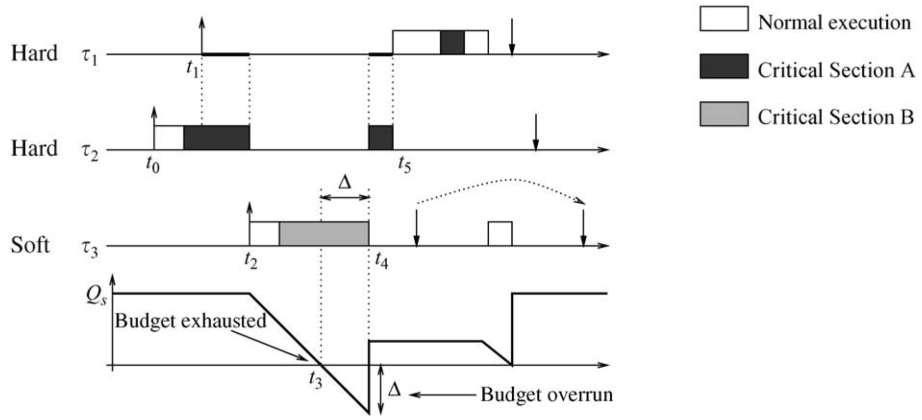

Figure 8. Example of budget overrun.

inversion phenomenon, both under fixed-priority (Sha et al., 1990) and dynamic-priority (Baker, 1991; Jeffay, 1992) assignments. In this paper, we analyze the problem by assuming that resources are accessed according to the Stack Resource Policy (SRP) (Baker, 1991).

When shared resources are accessed in mutual exclusion by soft tasks handled by a capacity-based server, as the CBS, an additional problem arises if the server exhausts its budget when a served task is inside a critical section. In this case, to prevent long blocking delays due to the budget replenishment rule, the soft task is allowed to continue executing with the same deadline (using extra budget) until it leaves the critical section. The extra budget used to complete the critical section must then be subtracted from the next replenishment to prevent server overruns from accumulating. This approach was first presented by Ghazalie and Baker (1995) for accounting for the blocking effects of critical sections in a number of dynamic aperiodic servers.

A typical example of budget overrun is depicted in Figure 8, where two hard tasks, τ_1 and τ_2, run together with a soft task, τ_3, handled by a CBS. At time t_1, according to the SRP, task τ_1 cannot preempt τ_2 inside the critical section because they share a common resource. As a consequence, τ_1 experiences a blocking time by τ_2 in $[t_1, t_2]$ and $[t_4, t_5]$. Notice that $[t_2, t_4]$ is not a blocking interval for τ_1, because τ_3 executes in that interval with a higher priority (i.e., with a shorter deadline). At time t_3, the CBS budget is exhausted while τ_3 is inside a critical section, so it is allowed to continue up to the end of the critical section using the same server deadline. For doing that, an amount of extra budget, Δ, must be used by the server to complete the operation. At time t_4, when τ_3 exits the critical section, the CBS recharges its budget at the value $Q_s - \Delta$ and postpones its deadline by T_s. Hence, τ_2 resumes execution and completes its critical section at time t_5.

Since the server execution is prolonged by an extra budget to allow the soft task to complete the critical section, the server utilization has to be computed to take such a budget overrun into account. The maximum interference created on task τ_1 by the budget overrun mechanism occurs when the soft task exhausts its budget immediately after entering its longest critical section. Thus, if ξ_s is the duration of the longest critical section of the tasks handled by the server (we assume $\xi_s < Q_s$), the bandwidth demanded by the server becomes $Q_s + \xi_s / T_s$.

3.5. Assigning Server Parameters

The CBS algorithm can be considered as a building block for properly implementing a bandwidth reservation strategy in hybrid real-time systems consisting of hard and soft tasks. When soft tasks are handled by a CBS, hard tasks can be guaranteed to meet their deadlines using the simple schedulability test expressed by Lemma 2, independently from soft tasks' requirements.

The performance experienced by soft tasks is influenced by the Q_s and T_s scheduling parameters. If some kind of QoS needs to be guaranteed to a soft task (e.g., in terms of tardiness, deadline miss ratio, or frame rate), the server parameters should be tuned according to the soft task requirements. This can be done either off-line (if some—even approximate—information is known on the task behavior) or on-line (if nothing is known about the task), using a feedback mechanism operating at a higher level with respect to the scheduling algorithm. Some possible criteria for assigning or tuning the server parameters are briefly discussed below.

If the WCET C_i and the MIT T_i are known for the served task, a hard guarantee can be performed on task τ_i by assigning $Q_s \geq C_i$ and $T_s = T_i$, as shown in Lemma 1.

If some (even approximate) knowledge about the execution and interarrival times is provided, in the form of probability distributions, it is still possible to perform a QoS guarantee based on probabilistic deadlines, as shown in Section 5.

If no *a priori* information is available on the execution and on the interarrival times of the served task, tuning the server parameters to guarantee some QoS becomes more difficult. However, the amount of reserved bandwidth can be dynamically adjusted using an adaptive strategy.

Such an approach has been proposed, for example, in Abeni and Buttazzo (1999a), where a QoS Manager, implemented as a user task, dynamically computes the (Q_s, T_s) pair based on the actual resource allocation error. Using a CBS, the resource allocation error can easily be evaluated by monitoring the difference between the current scheduling deadline assigned to the served job and its soft deadline. Such a measure is then used as a feedback signal for a controller (the QoS manager) which dynamically generates the new reserved CPU bandwidth as an output.

In conclusion, the CBS is flexible and powerful enough to support both fixed and dynamic, on-line and off-line parameter assignment strategies, thus it can be used in a wide range of different systems.

4. Aperiodic Task Handling

In this section we show how the CBS can be efficiently used as a service mechanism for improving responsiveness of soft aperiodic requests.

Aperiodic servers, such as the DSS (Spuri and Buttazzo, 1996), the TBS (Spuri and Buttazzo 1996) and the CUS (Deng et al., 1997) have been designed to handle all the aperiodic requests with a single server, in order to improve the average aperiodic response time without jeopardizing the periodic (hard) tasks schedulability.

Although the behavior of these servers seems similar to the one exhibited by the CBS, there are important differences to consider. For example, when all the aperiodic requests are handled together with a single server, it is impossible to control the QoS of individual soft tasks. In fact, an aperiodic task performing a large number of requests would severely affect the performance of the other aperiodic tasks (although it would not interfere with the periodic ones). On the contrary, by using a CBS for each soft task (either periodic or aperiodic), the individual performance of the served task can be controlled through the Q_s and T_s parameters. The CBS also provides temporal isolation (preventing the served task to demand more than the allocated bandwidth) and, if properly dimensioned, allows us to perform hard schedulability analysis (see Section 3.2), as well as statistical analysis (see Section 5), which cannot be performed using the traditional approach.

Moreover, the CBS achieves a good performance (comparable with the one provided by the TBS) in terms of tardiness. To show this result, we compared the CBS with other similar service mechanisms, namely the TBS and the DSS. The CUS is not shown in the graphs because its behavior is very similar to the one of 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 significant 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 WCETs, so it cannot be used for supporting CM applications. Moreover, since the CBS automatically reclaims any available idle time coming from early completions, an explicit reclaiming mechanism has also been added in the simulation of the TBS, as described in Spuri et al. (1995).

4.1. Possible Extensions

Since the CBS algorithm has been designed to reduce the tardiness, it behaves in a work conserving fashion. However, in some situations such a behavior can decrease the QoS of some multimedia applications, because it badly affects the jitter.

Consider, for example, a player task receiving an audio/video stream from the network: packets are supposed to arrive periodically, but due to some network congestion they can arrive in bursts. Hence, it may happen that different packets that were supposed to arrive at times t, $t + T$, and $t + 2T$ all arrive around time t'. Using the CBS to serve the task, if the CPU has enough idle time, the player can "speed up" around time $t' \geq t + 2T$, to slow down in the future (because serving the three activations postponed the deadline to a large value).

We believe that some different techniques can be used to address this problem, and that the ''right solution'' does not exist in general, but depends on the application semantic. For example, if fairness is important to the application, then the CBS behavior can be modified to make it non-work conserving. This can be achieved by suspending the server until the scheduling deadline when the budget is exhausted. Such a new semantic, however, would have a terrible impact on the tardiness, hence it is not always an appropriate solution. A better approach would be to give the application the possibility to switch between two CBS modes (work conserving and non-work conserving) at any instant, without compromising system schedulability.

A different solution to the previous problem can be to use application-level adaptation (as proposed in Abeni and Buttazzo, 2001), by skipping packets that ''arrive too late'', whenever possible. Of course, in this case the scheduler cannot help, because the overload management is completely application-dependent.

Finally, some reclaiming algorithm such as GRUB (Lipari and Baruah, 2000) or CASH (Caccamo et al., 2000) can be used to avoid an excessive postponement of the sever deadline in the presence of bursts.

4.2. Simulation Results

The performance of the CBS algorithm has been tested and compared against those of other similar algorithms using the real-time scheduling simulator presented in Casile et al. (1998).

All the simulations presented in this section have been conducted on a hybrid task set consisting of five periodic hard tasks with fixed parameters and five soft tasks with variable execution times and interarrival 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 interarrival times of the soft tasks are uniformly distributed in order to obtain a mean soft load $\overline{U_{\text{soft}}} = \Sigma_i \overline{c_{i,j}} / \overline{r_{i,j+1} - r_{i,j}}$ with $\overline{U_{\text{soft}}}$ going from 0 to $1 - U_{\text{hard}}$.

The metric used to measure the performance of the service algorithms is the mean tardiness experienced by soft tasks, where the tardiness of a task is defined by Equation (1). Such a metric was selected because in multimedia applications meeting all soft deadlines could be impossible or very inefficient; hence, the system should be designed to guarantee all the hard tasks and minimize the mean time that soft tasks execute after their deadlines. Standard deviations on the average values of the soft tardiness have been computed for all the experiments and they were never greater than 3%.

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 $U_{\text{hard}} = 0.5$. 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 above.

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

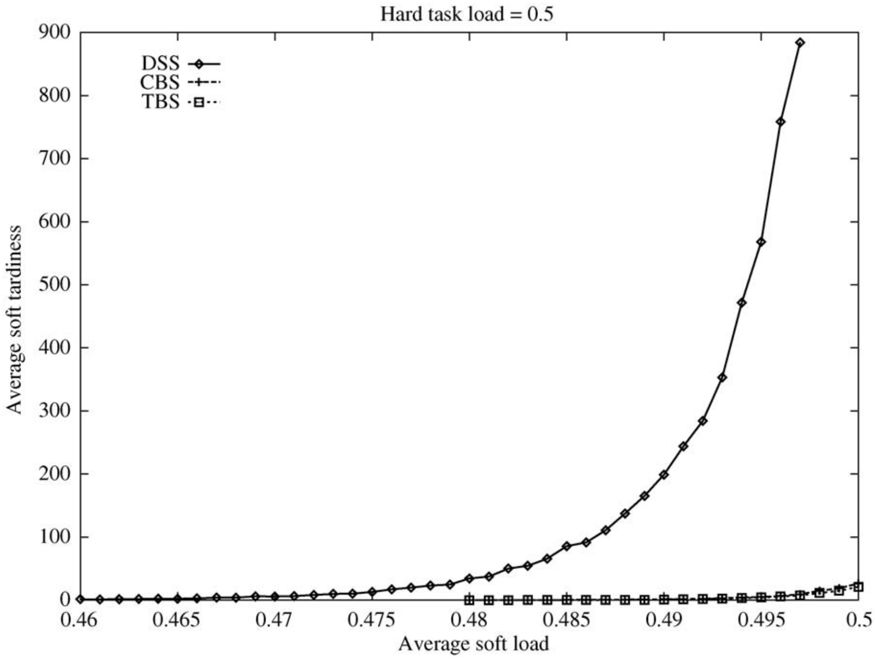

Figure 9. First experiment (TBS, CBS, and DSS).

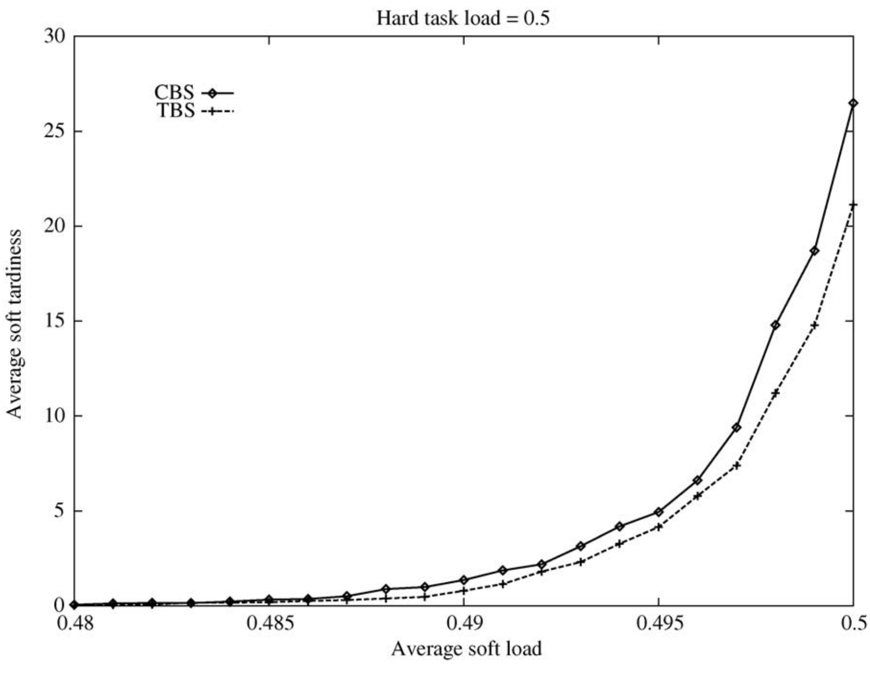

Figure 10. First experiment (TBS and CBS).

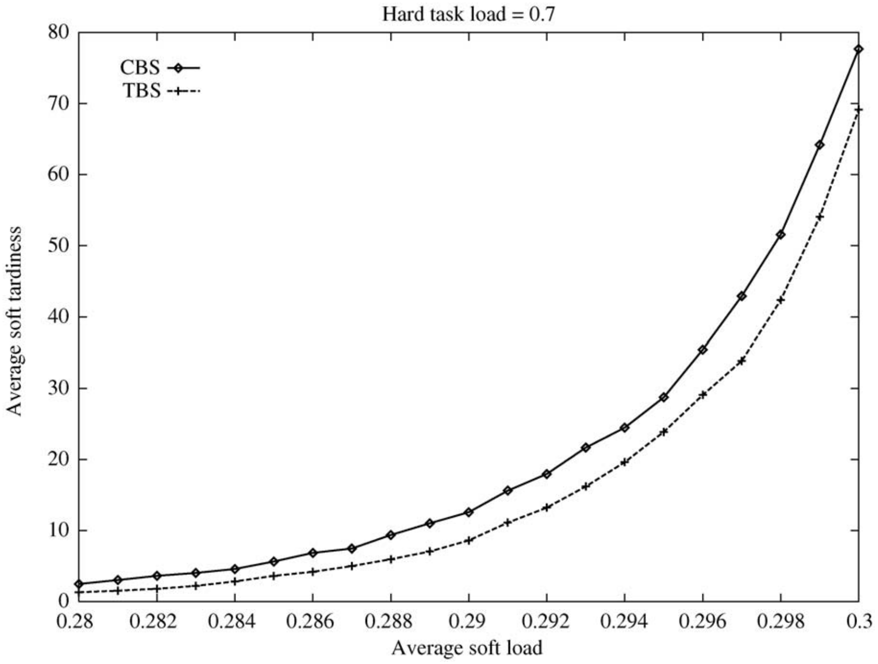

Figure 11. Second experiment.

that experienced using a TBS. However, the difference is so small that can be neglected for any practical purpose.

Figures 11 and 12 illustrate the results of similar experiments repeated with $U_{\text{hard}} = 0.7$ and $U_{\text{hard}} = 0.9$, respectively. As we can see, the major difference in the performance between CBS and TBS appears only for heavy hard loads. However, this situation is of little interest for most practical multimedia applications.

The advantage of the CBS over the TBS can be appreciated when $\text{WCET}_i \gg \overline{c_{i,j}}$. In this case, in fact, the TBS can cause an underutilization of the processor, due to its worst-case assumptions. This fact can be observed in Figure 13, which shows the results of a fourth experiment, in which $U_{\text{hard}} = 0.6$, $\overline{U_{\text{soft}}} = 0.4$, 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, the CBS performs better than the TBS when tasks' execution times have a high variance.

5. Statistical Analysis

As stated in Section 1, by relaxing the timing constraints of the application we can better utilize the processor and significantly increase the performance of the system. The new task model, however, requires an appropriate schedulability analysis aimed at providing an off-line guarantee on the QoS achieved by each task.

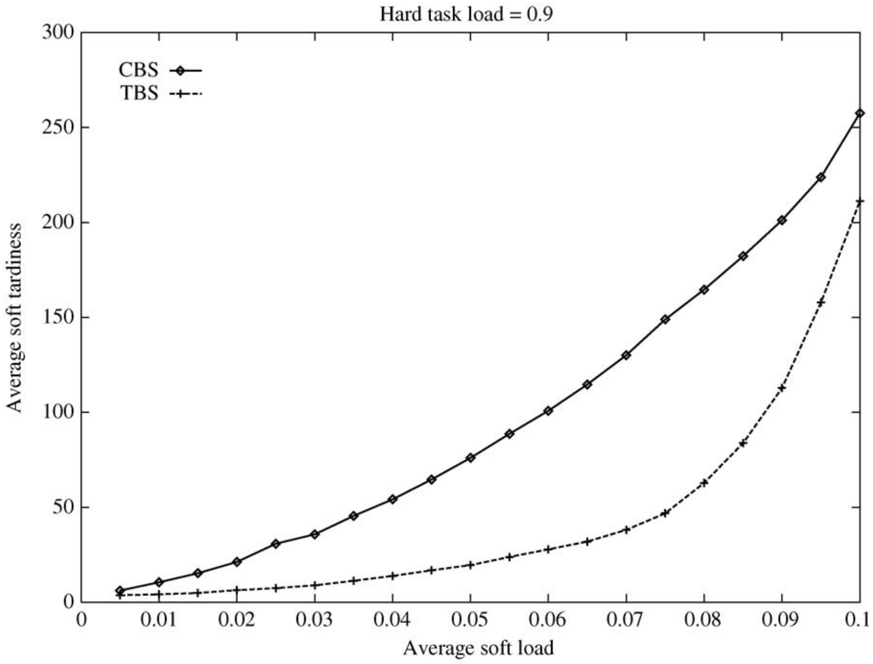

Figure 12. Third experiment.

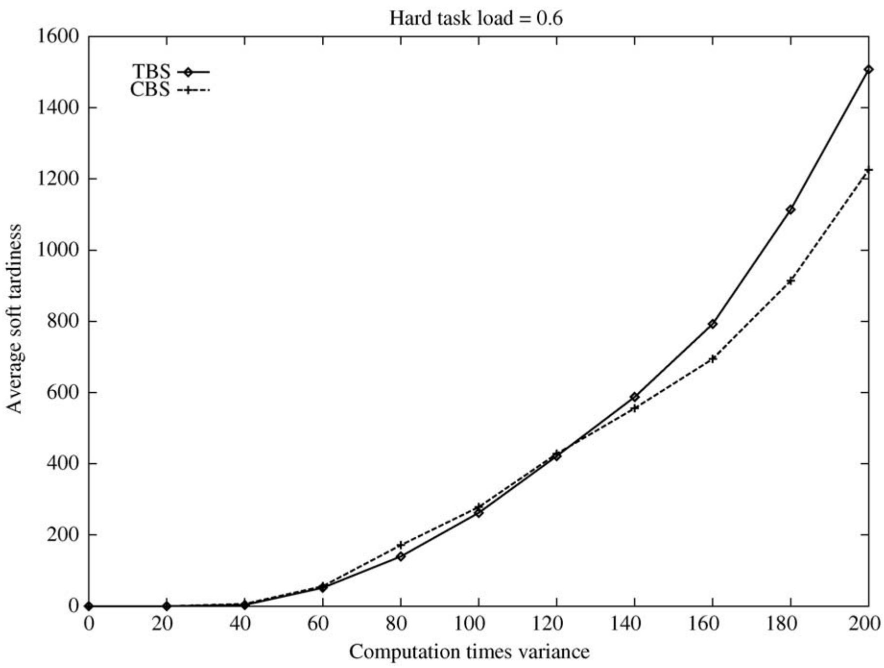

Figure 13. Fourth experiment.

We address this problem using the completion probability as a metric for expressing and guaranteeing a desired QoS: while the classical hard guarantee requires each job to finish within its deadline, the QoS guarantee ensures that a job will finish within its soft deadline with a given probability. To perform a QoS guarantee we need to describe the task's parameters in a probabilistic way and introduce the concept of probabilistic deadline.

According to this approach, each task is described by a pair of PDF $[U_i(c), V_i(t)]$, where $U_i(c)$ is the probability that job $J_{i,j}$ has execution time c, and $V(t)$ is the probability that jobs' interarrival time is t. Hence,

$$U_i(c) = P\{c_{i,j} = c\}$$
$$V_i(t) = P\{r_{i,j+1} - r_{i,j} = t\}$$

The QoS guarantee test is based on the notion of probabilistic deadline, δ. Our QoS guarantee does not require that the probabilistic deadline is always respected, but that it is respected with a given probability

$$X_i(\delta) = P\{f_{i,j} \leq r_{i,j} + \delta\}$$

where $f_{i,j}$ is the job's finishing time, and $X_i(\delta)$ is the probability that job $J_{i,j}$ finishes before a deadline $d_{i,j} = r_{i,j} + \delta$.

Definition 1 A task τ_i is guaranteed to respect a constraint (δ, p), if $X_i(\delta) \geq p$.

5.1. Schedulability Analysis

To perform a statistical guarantee on soft tasks served by a CBS the server is modeled 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 interarrival time; and (b) constant computation time and variable interarrival time.

5.1.1. Case a: c_i variable, T_i constant

This case treats the so called semi-periodic task model (Tia et al., 1995; Atlas and Bestavros, 1998), which can be used to model is typical in applications that manage CM. For example, a video stream requires frames to be played periodically, but the decoding/playing time of each frame is not constant.

Let $U(c)$ be a generic PDF for the task's computation times and let the interarrival times be deterministic, so that

$$V(t) = \begin{cases} 1, & \text{if} \quad t = T_i \\ 0, & \text{otherwise} \end{cases}$$

In this case, if we assign $T_s = T_i$, the system can be modeled as a queue in which each T_s units of time:

1. a request of c_j units arrives;

2. at most Q_s units can be served.

We can describe the system with a random process v_j defined as follows:

$$\begin{cases} v_0 = c_0 \\ v_j = \max\{0, v_{j-1} - Q_s\} + c_j \end{cases}$$

where v_j indicates the length of the queue (in time units) at time jT_s; that is, the units of times that are still to be served when job $J_{i,j}$ arrives. Since the CBS postpones the scheduling deadline by T_s each Q_s units of execution time, the last absolute deadline assigned by the server to $J_{i,j}$ will be

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

If the system is not overloaded $(U_p + U_s \leq 1)$, Theorem 1 ensures that each job will finish at or before its last assigned deadline, so the probability that a job finds a queue of length v_j at its arrival is a lower bound of the probability that the job finishes within a relative deadline

$$\delta = \left\lceil \frac{v_j}{Q_s} \right\rceil T_s$$

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

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

$$M = \begin{pmatrix} \overbrace{\begin{matrix} U(0) & U(0) & . & . & . & U(0) \\ U(1) & U(1) & . & . & . & U(1) \\ U(2) & U(2) & . & . & . & U(2) \\ U(3) & U(3) & & & & U(3) \\ . & . & & & & . \\ . & . & & & & . \\ . & . & & & & . \end{matrix}}^{Q_s+1} & \begin{matrix} 0 & 0 & . & . & . \\ U(0) & 0 & . & . & . \\ U(1) & U(0) & 0 & . & . \\ U(2) & U(1) & U(0) & 0 & . \\ . & . & . & . & . \\ . & . & . & & . \\ . & . & . & & \end{matrix} \end{pmatrix}$$

Figure 14. M for semi-periodic tasks.

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

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

Hence,

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

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

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

where Π is the vector of the state probabilities and M is described in Figure 14.

5.1.2. *Case b: c_i constant, T_i variable*

This case deals with a generalization of the sporadic task model that can be frequently encountered in CM applications. 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,

$$U(c) = \begin{cases} 1, & \text{if } c = C_i \\ 0, & \text{otherwise} \end{cases}$$

and $V(t)$ is a generic PDF.

It is worth noting that, if $Q_s = C_i$, the CBS behaves exactly like a TBS with a bandwidth $U_s = Q_s/T_s$. In fact, if $Q_s = C_i$, each job finishes exactly when the budget arrives to 0, and the server deadline is increased of T_s at the end of each job, which is assigned a deadline $d_{i,j} = \max\{r_{i,j}, d_{i,j-1}\} + T_s$, identical to that assigned by a TBS. It is also interesting to observe that, in this situation, the CBS is equivalent to a RBE model (Jeffay and Bennet, 1995; Jeffay and Goddard, 1999) with parameters $x = 1, y = T_i, D = T_i$.

In this situation, the CBS can be modeled by a $G/D/1$ queue: jobs arrive in the queue with randomly distributed interarrival times and the server processes a request each T_s time units. We can define a random process w_j, representing the time that job $J_{i,j}$ waits in the queue before being served, as

$$w_j = d_{i,j} - r_{i,j} - T_s$$

In this way, the distribution of the relative deadlines within which a job $J_{i,j}$ is served is given by

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

Note that, if the system is not overloaded $(U_p + U_s \leq 1)$, Theorem 1 guarantees that each job will finish at or before the deadline assigned by the server. Hence, $P\{d_{i,j} - r_{i,j} \leq \delta\}$ is a lower bound for $P\{f_{i,j} - r_{i,j} \leq \delta\}$. Since $d_{i,j} = \max\{r_{i,j}, d_{i,j-1}\} + T_s$, we have

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

where $a_{j+1} = r_{i,j+1} - r_{i,j}$.

Since a_j is a stochastic stationary and time-invariant process and w_j is 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 $V(h) = P\{a_j = h\}$, we have

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

In order to simplify the computation, we distinguish two cases: $k = 0$ and $k > 0$. For $k = 0$ we can write:

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

For $k > 0$ we have:

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

Matrix M describing the Markov chain is shown in Figure 15.

$$M = \begin{pmatrix}
\rho_0 & \rho_1 & \rho_2 & . & . \; . \; . \\
V(T_s-1) & V(T_s) & V(T_s+1) & . & . \; . \; . \\
V(T_s-2) & V(T_s-1) & V(T_s) & V(T_s+1) & . \; . \; . \\
V(T_s-3) & V(T_s-2) & V(T_s-1) & V(T_s) & . \; . \; . \\
. & . & . & . & \\
. & . & . & . & \\
. & . & . & . & \\
V(0) & V(1) & . & . & \\
0 & V(0) & . & . & \\
. & 0 & . & . & \\
. & & . & . & \\
. & & & . &
\end{pmatrix} \quad \text{with } \rho_i = \sum_{r=i+T_s}^{\infty} V(r)$$

Figure 15. M for generalized sporadic tasks.

5.1.3. Stability Considerations

For a generic queue, it is known that the queue is stable (i.e., the number of elements in the queue does not diverge to infinity) if

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

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

$$E[C] < Q_s \quad \text{in case (a)}$$
$$E[T] > T_s \quad \text{in case (b)}$$

where $E[C]$ is the execution time expectation and $E[T]$ is the interarrival time expectation. In general, the stability is guaranteed if

$$\frac{E[C]}{E[T]} < \frac{Q_s}{T_s}$$

If the condition above does not hold, the difference between the deadline $d_{i,j}$ assigned by the server to 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 the queue is stable, a stationary solution of the Markov chain describing the queue can be found; that is, there exists a solution Π such that $\Pi = \lim_{j \to \infty} \Pi^{(j)}$, so $\Pi = M$. This solution can be approximated by truncating the infinite dimension matrix M to an $N \times N$ matrix M' and solving the eigenvector problem $\Pi' = M'\Pi'$ with some numerical calculus technique.

The knowledge of the PDF of the relative deadlines before which a multimedia task is guaranteed to finish is useful for choosing the right server parameters (Q_s, T_s) for each soft task, in order to analyze and control the QoS of each task.

5.2. An Example

Consider a generalized sporadic task with fixed computation time (equal to 1) and interarrival times distributed as shown in Figure 16. The classical hard real-time analysis would guarantee this task with a worst-case utilization factor of 1/30.33. Using our approach, we can schedule the task with a CBS, with $Q_s = 1$ and $T_s < E[T] = 7.6$. As an example, consider $T_s = 6$, giving a utilization factor of $1/6 = 0.16$.

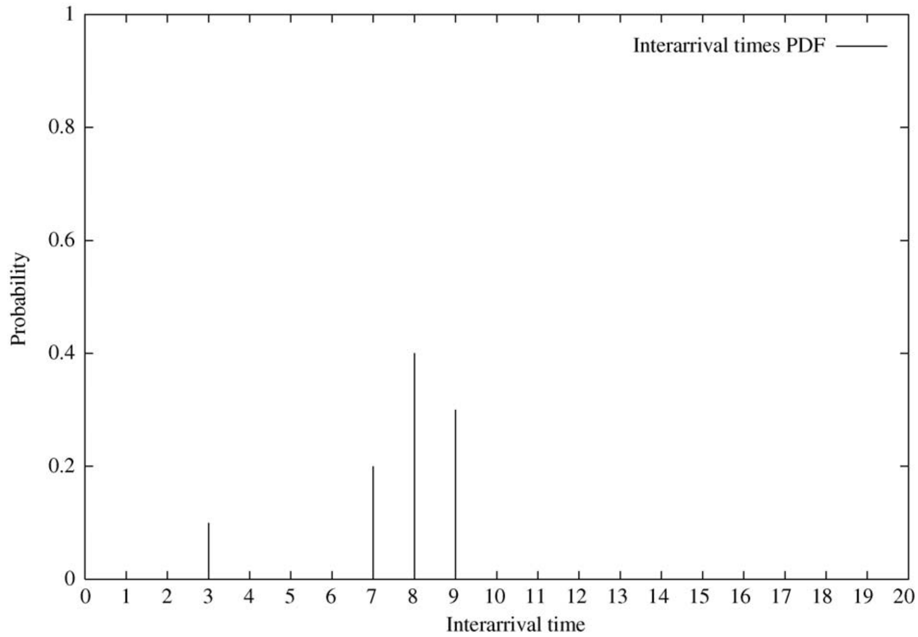

Figure 16. Interarrival time PDF $V(t)$.

The resulting M matrix is

$$M = \begin{pmatrix} 0.9 & 0.9 & 0.7 & 0.3 & 0 & 0 & . & . \\ 0 & 0 & 0.2 & 0.4 & 0.3 & 0 & 0 & . \\ 0 & 0 & 0 & 0.2 & 0.4 & 0.3 & 0 & . \\ 0.1 & 0 & 0 & 0 & 0.2 & 0.4 & 0.3 & . \\ 0 & 0.1 & 0 & 0 & 0 & 0.2 & 0.4 & . \\ . & 0 & 0.1 & 0 & 0 & 0 & 0.2 & . \\ . & & . & . & . & & . & . \\ . & & & . & . & . & & . \\ . & & & & . & . & . & . \end{pmatrix}$$

Figure 17 shows the solution vector $\Pi = M$, which gives the CDF $X(\delta)$ shown in Figure 18. In general, the CDF is a function of the bandwidth reserved to the task, increasing with the reserved bandwidth U_s. Notice that, if $C_i/T_i \leq U_s \leq 1$, the CFD becomes a step function:

$$X(\delta) = \begin{cases} 0 & \text{if } \delta < T_i \\ 1 & \text{otherwise} \end{cases}$$

$$\Pi = \begin{pmatrix} 0.815786 \\ 0.043039 \\ 0.023228 \\ 0.088615 \\ 0.009824 \\ 0.005252 \\ 0.009902 \\ 0.001696 \\ 0.000899 \\ 0.001138 \\ 0.000263 \\ 0.000138 \\ 0.000134 \\ 0.000038 \\ 0.000020 \\ 0.000016 \\ 0.000005 \\ 0.000003 \\ 0.000002 \\ 0.000001 \\ 0 \\ . \\ . \\ . \end{pmatrix}$$

Figure 17. PDF of the last assigned deadline.

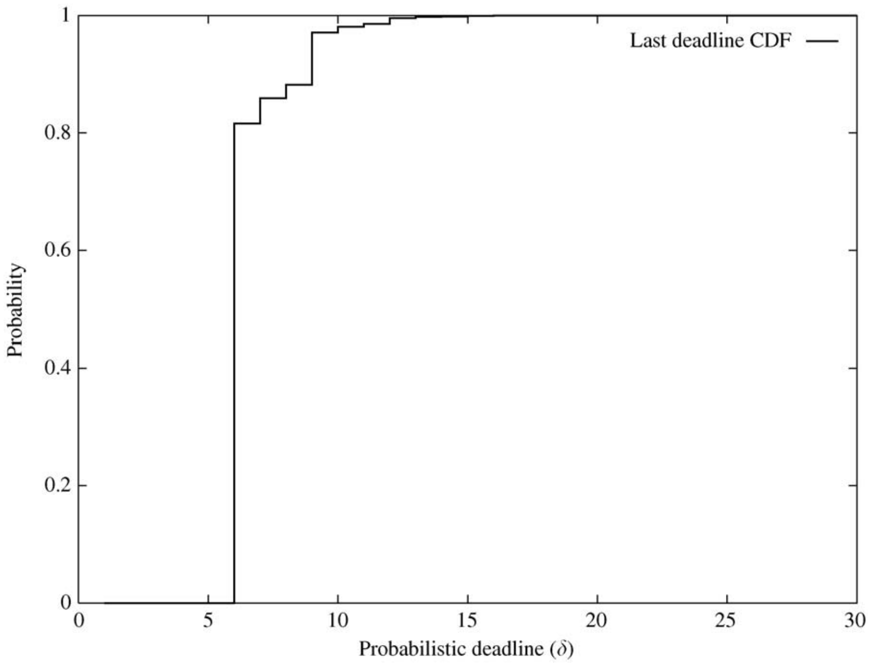

Figure 18. Probabilistic deadline CDF lower bound $X(\delta)$.

5.3. Experimental Validation

To validate the correctness of our statistical analysis we implemented the guarantee mechanism in a tool that calculates $X_i(\delta)$ for semi-periodic tasks and generalized sporadic tasks, given $U(c)$ or $V(t)$, respectively. We simulated the CBS scheduler and monitored the last server deadline assigned to a task and the task's finishing time. Then, we compared the simulation results with the theoretical ones.

In the first experiment, we considered a semi-periodic task with period $T_i = 1250$ and computation times uniformly distributed in $[100, 400]$, served by a CBS (Q_s, T_s) with $T_s = 1250$ and Q_s ranging from 260 to 400. Figure 19 shows the cumulative distribution function $X(\delta)$ computed by our tool (using the approximated Markov chain solution) and the one measured using the scheduling simulator, when $Q_s = 270$. The continuous line represents the finishing time CDF obtained by simulation. As the reader can see, theoretical and simulation results are consistent, and are a tight lower bound for the finishing time cumulative distribution function.

Table 1 shows the $X(\delta)$ values for three different Q_s values, computed using the proposed analysis and the simulator. It can be seen that the analysis results are close to the simulator's ones; it is worth noting that the analysis results are always less than or equal to the ones achieved by simulation (they are a lower bound of the exact cumulative distribution function). We also performed other similar experiments for semi-periodic and

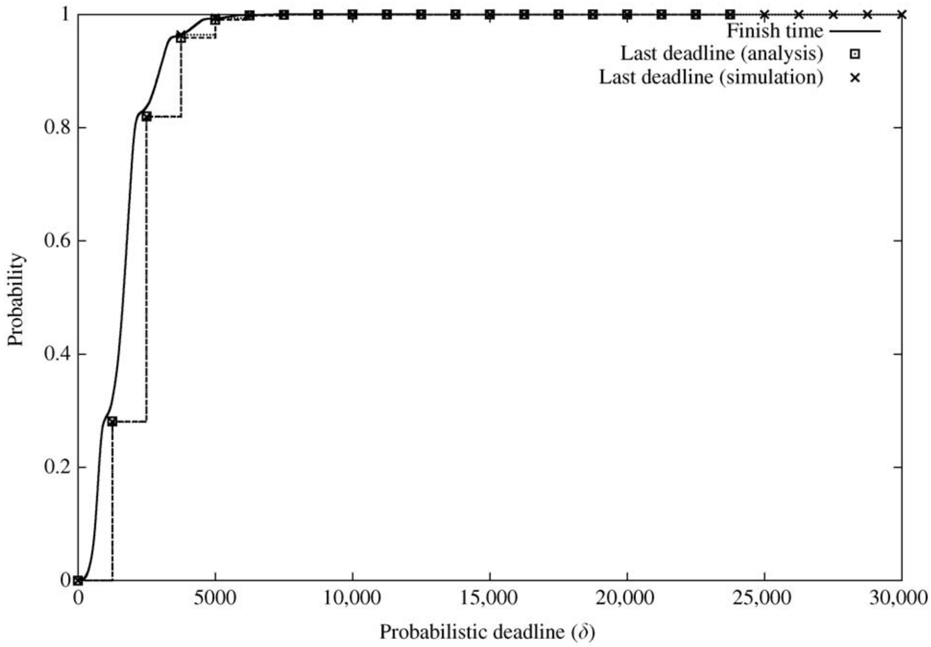

Figure 19. Simulation results versus analysis. The continuous curve represents the $X(\delta)$ CDF as a function of δ, whereas the step functions represents the lower bounds, computed by simulation and analysis.

Table 1. $X(\delta)$ lower bounds obtained by simulation and analysis for different δ and Q_s values.

	$Q_s = 280$		$Q_s = 320$		$Q_s = 400$	
δ	Simulation	Analysis	Simulation	Analysis	Simulation	Analysis
1250	0.388033	0.387972	0.677513	0.677459	1	1
2500	0.934535	0.934177	0.999879	0.999860	1	1
3750	0.994164	0.994103	1	1	1	1
5000	0.999548	0.999520	1	1	1	1
6250	0.999979	0.999979	1	1	1	1
7500	1	1	1	1	1	1
8750	1	1	1	1	1	1
10,000	1	1	1	1	1	1

sporadic tasks with different input PDFs, verifying the consistency between the stochastic analysis and the simulation.

6. Comparison with Proportional Share Scheduling

Proportional share (PS) scheduling is a scheduling methodology developed for supporting multimedia and soft real-time activities in a conventional operating system. In this section we compare our approach with such a methodology and illustrate how PS scheduling can be efficiently achieved using a CBS-based service.

6.1. The ideal model

The objective of the PS model is to emulate a fluid flow system on a discrete quantum-based allocation system. The ideal scheduling model is the generalized processor sharing (GPS) presented in Parekh and Gallager (1993). It can be thought as the limiting form of a weighted round robin policy: each task τ_i is assigned a weight w_i, which determines the minimum bandwidth sharing of the task. Given a task set $\Gamma = \{\tau_1, \ldots, \tau_n\}$, if $e_i(t_1, t_2)$ is the execution time of task τ_i in an interval $[t_1, t_2]$, the n tasks share the resource as follows:

$$\forall \tau_i \text{ active in } [t_1, t_2], \frac{e_i(t_1, t_2)}{e_j(t_1, t_2)} \geq \frac{w_i}{w_j}, \quad j = 1, 2, \ldots, n \tag{4}$$

Equation (4) expresses that in a GPS system each active task τ_i executes at least with a rate equal to $F_i = w_i / (\Sigma_{\tau_j \in \Gamma} w_j)$. The actual execution rate varies with time and is defined as the share of task τ_i: $f_i(t) = w_i / (\Sigma_{\tau_j \in \Gamma_a(t)} w_j)$ where $\Gamma_a(t)$ is the set of tasks which are active at time t.

As a consequence, the following properties hold for any time interval $[t_1, t_2]$:

$$e_i(t_1, t_2) \geq (t_2 - t_1)F_i \tag{5}$$

$$e_i(t_1, t_2) = \int_{t_1}^{t_2} f_i(t)dt \tag{6}$$

In a PS system, the resource is allocated in discrete time quanta of length Q. A task acquires the resource at the beginning of a time quantum and can release it either at the end of the quantum (in this case a new request is posted) or before the end of the quantum (in this case the process blocks and must be explicitly re-activated). This is done by dividing each task τ_i in requests q_i^k having maximum size Q.

Since the allocation is discrete in time, this approach generates an allocation error with respect to the ideal GPS model. In fact, in the ideal GPS system, in any interval $[t_1, t_2]$, a task τ_i executes for a time $\int_{t_1}^{t_2} f_i(t)dt$, whereas in a real system this is impossible because of the allocation error. The difference between the ideal and the real schedule is called the lag, defined as

$$\mathrm{lag}_i(t_1) = \int_{t_0}^{t_1} f_i(t)dt - e_i(t_0, t_1)$$

where t_0 is the activation time of task τ_i.

The goal of a PS algorithm is to reduce the allocation error experienced by tasks. To support some form of real-time execution it is important to guarantee that $\mathrm{lag}_i(t)$ is bounded. In fact, if an upper bound for $\mathrm{lag}_i(t)$ exists, the execution time accumulated by τ_i is

$$e_i(t_0, t) \geq \int_{t_0}^{t} f_i(t)dt - \max_t\{lag_i(t)\} \tag{7}$$

Hence, a task τ_i having execution time C_i is guaranteed to complete within a (relative) deadline D_i if

$$e_i(t_0, t_0 + D_i) \geq C_i$$

6.2. Emulating PS with CBS

Looking at the definitions given in Sections 3 and 6.1, it is easy to see that the bandwidth U_i demanded by a task τ_i is similar to the rate F_i. The only difference between these parameters is that, by definition, $\Sigma_\Gamma F_i = 1$, whereas $\Sigma_\Gamma U_i \leq 1$ (using the constant bandwidth abstraction, a fraction of the CPU bandwidth can be left available for other purposes).

As a consequence, the behavior of a PS scheduler on a set of n tasks can be emulated by serving each task through a CBS with bandwidth $U_i = F_i$ and period $T_i = Q/U_i$.

On the other hand, in Stoica et al. (1997), the authors show how the real-time tasks' share can be maintained constant at the arrival of a new task (Γ varies). This is done by re-arranging the tasks' weights. As shown in the cited paper, fixing the task share to a specified value allows to perform a real-time guarantee on the task.

As for the constant bandwidth allocation, there is an admission test $\Sigma_{\tau_i \in \Gamma} C_i/T_i \le 1$ (similar to $\Sigma_{\tau_i \in \Gamma} U_i \le 1$) and, if a new task is accepted, the weights are recomputed such that for all real-time tasks $F_i = f_i(t) = C_i/T_i$. This is not necessary under the CBS paradigm, where, if a new task is accepted, the other tasks' bandwidths do not have to be adjusted.

This difference is due to the fact that F_i is computed based on the executed time, whereas U_i is computed based on the demanded time in a given time interval (i.e., the sum of execution times of the jobs with deadlines within that interval).

Finally, using a CBS, the additional parameter T_s (period of the server) can be used to better model the tasks' temporal behavior of a real-time task.

6.3. PS versus CBS

There is a fundamental difference between the two theoretical models from which the PS and the CBS approaches derive: if $e_i(t_0, t_1)$ is the time executed by task τ_i in the time interval $[t_0, t_1]$, a PS scheduler tries to maintain the ratio $e_i(t_0, t_1)/(t_1 - t_0)$ constant over any time interval $[t_0, t_1]$, while a CBS scheduler maintains this ratio constant only between deadlines.

We believe that imposing a fair execution over any interval of time over-constraints the system. In fact, in most soft real-time applications it is important to schedule tasks before their deadlines, not to schedule them in a fair fashion. For example, in a video or audio player it is important to decode a frame before the start of the next frame, not to decode it fairly. Since no periods or deadlines are taken into account by a PS scheduler, but only the tasks' share, fairness is the only way to enforce QoS requirements. On the other hand, in the CBS approach, the knowledge of timing constraints can be used to generate a more efficient schedule (by preventing unnecessary context switches).

Another fundamental difference between the PS paradigm and the constant bandwidth paradigm is that the former allocates resources in time quanta, whereas the latter does not require so (it is fully preemptive). This makes PS easier to implement in a conventional operating system (such as Unix or Windows), whose scheduler is quantum-based, but introduces some limitations: for example, tasks can be activated only at quantum boundaries and tasks' periods must be multiple of Q. Moreover, the fairness property enforced by PS schedulers generates a larger number of preemptions, as shown in Figure 20, which can penalize the performance of the system (in a real system, context switches take some time!). On the other hand, if the value of Q is increased to reduce the preemption overhead, the allocation error becomes larger.

Another problem with the PS paradigm is that it does not model event-based systems intuitively. For example, a task activated by an external event (e.g., a task that manages data from the network, or that has to respond to an interrupt) can only be modeled as a periodic task, and it is not clear how to assign the weight or the share to this task. On the other hand, using a CBS, we can reserve a given bandwidth to an aperiodic event-driven task and use the server parameter T_s to model the expected interarrival time. Using a

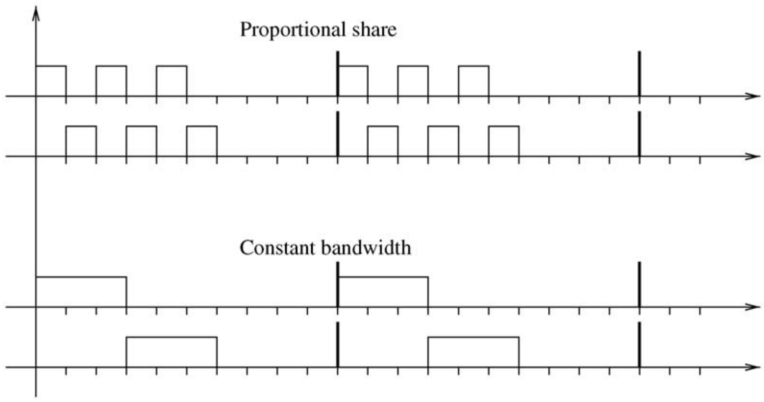

Figure 20. Examples of proportional share and constant bandwidth.

CBS, it is also simple to perform a deterministic or stochastic guarantee on the task's response time.

In summary, we can say that PS and CBS give different interfaces to similar (but not equal) allocation models: the CBS programming model is more suited for handling real-time constraints, so the CBS approach is to be preferred for integrating multimedia streams in a real-time system, whereas the PS is closer to the classical time-sharing approach, hence a PS scheduler can be more easily integrated in a conventional operating system.

As shown in Section 3.2, the CBS provides explicit support for hard real-time execution (reserving a bandwidth $U_i = (WCET_i/T_i)$ to each hard task and scheduling it directly with the EDF algorithm). A PS scheduler can also emulate it, but at the cost of the additional overhead of dynamically rearranging the tasks' weights.

On the other hand, PS schedulers are more flexible in partitioning the bandwidth among non-guaranteed tasks: this is useful to run non real-time applications (such as in a traditional workstation) together with multimedia ones. In this case, the notion of weight is more intuitive than the reserved bandwidth; the CBS can also emulate such a behavior, as shown above.

7. Conclusions

In this paper, we presented a novel service mechanism, the CBS, for integrating hard real-time and soft multimedia computing in a single system, under the EDF scheduling algorithm.

Temporal protection is one of the most important features offered by the proposed algorithm, which allows to execute a task so it will never exceed a predefined bandwidth, independently of its actual requests. We also discussed how the proposed server can be used for enhancing responsiveness of soft aperiodic tasks and how a statistical analysis can be applied to perform a probabilistic guarantee.

The server has been formally analyzed and compared with other known servers, obtaining interesting results in terms of some well defined QoS metrics, such as the experienced tardiness. The proposed model has also been implemented on the HARTIK kernel and on Linux/RK to support typical multimedia applications.

As a future work, we plan to extend the proposed model to designing flexible real-time systems that can be tuned on line according to the actual task requirements. For example, 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 by adjusting the QoS in overload conditions. The CBS mechanism can also be used to safely partition the CPU bandwidth among different applications that could coexist in the same system, as shown in Deng et al. (1997). A task can be used as a QoS manager to dynamically change the bandwidth reserved to each multimedia task.

Appendix

Proof of Theorem 1 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 2. To simplify the notation, we 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). The release time and the deadline of the kth chunk generated by the server will be denoted by a_k and d_k.

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

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

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:

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

We proceed by induction on $k_2 - k_1$, using the algorithmic definition of CBS shown in Figure 3.

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.

Case a: $d_k < a_k + T_s$.
If $d_k < a_k + T_s$, then d_k is generated by Rule 2, so $a_k + (c_s(f_{k-1})/Q_s)T_s < d_k$ 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

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

Case b: $d_k = a_k + T_s$.
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:

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

Inductive step

The inductive hypothesis

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

is used to prove that

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

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 $a_{k_2} + (c_s(f_{k_2-1})/Q_s)T_s \geq d_{k_2-1}$, hence $c(f_{k_2-1}) \geq (d_{k_2-1} - a_{k_2})Q_s/T_s$. 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

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

References

Abeni, L. 1997. Progetto e realizzazione di meccanismi di sistema per applicazioni real-time multimediali, (Aprile), Università degli studi di Pisa.

Abeni, L. 1998. Server mechanisms for multimedia applications. Scuola Superiore S. Anna, (RETIS TR98-01).

Abeni, L., and Buttazzo, G. 1998. Integrating multimedia applications in hard real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium* (December). Madrid, Spain.

Abeni, L., and Buttazzo, G. 1999. Adaptive bandwidth reservation for multimedia computing. In *Proceedings of the IEEE Real Time Computing Systems and Applications* (December). Hong Kong.

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

Abeni, L., and Buttazzo, G. 1999. QoS guarantee using probabilistic dealines. In *Proceedings of the IEEE Euromicro Conference on Real-Time Systems* (June). York, England.

Abeni, L., and Buttazzo, G. 2001. Hierarchical QoS management for time sensitive applications. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS 2001)* (May). Taipei, Taiwan.

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

Atlas, A. K., and Bestavros, A. 1998. Statistical rate monotonic scheduling. In *Proceedings of the IEEE Real-Time Systems Symposium* (December). Madrid, Spain.

Baker, T. P. 1990. A stack-based allocation policy for realtime processes. In *Proceedings of the 11th IEEE Real-Time Systems Symposium* (December). Lake Buena Vista, Florida, USA, pp. 191–200.

Baker, T. P. 1991. Stack-based scheduling of real-time processes. *Real-Time Systems* 3(1): 67–100.

Baruah, S. K., Cohen, N. K., Plaxton, C. G., and Varvel, D. A. 1996. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica* 6.

Baruah, S. K., Mok, A. K., and Rosier, L. E. 1993. Preemptively scheduling hard-real-time sporadic tasks on one processor, pp. 182–190.

Buttazzo, G. C. 1993. HARTIK: A real-time kernel for robotics applications. In *Proceedings of the 14th IEEE Real-Time Systems Symposium* (December). Raleigh-Durham, NC, USA, pp. 201–205.

Buttazzo, G. C., and Stankovic, J. 1993. RED: Robust earliest deadline scheduling. In *Third International Workshop on Responsive Computing Systems*. Austin, OPT.

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

Caccamo, M., Buttazzo, G., and Sha, L. 2000. Capacity sharing for overrun control. In *Proceedings of the IEEE Real-Time Systems Symposium* (December). Orlando, Florida.

Casile, A., Buttazzo, G., Lamastra, G., and Lipari, G. 1998. Simulation and tracing of hybrid task sets on distributed systems. In *Proceedings of the IEEE International Conference on Real-Time Computing Systems and Applications* (October). Hiroshima, Japan, pp. 249–256.

Chu, H.-H., and Nahrstedt, K. 1997. A soft real time scheduling server in UNIX operating system. In *European Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services* (September). Darmstadt, Germany.

Chu, H.-H., and Nahrstedt, K. 1998. CPU. Service classes for multimedia applications (August). Urbana Champaign, University of Illinois, (UIUCDCS-R-98-2068,UILU-ENG-98-1730).

Chu, H.-H., and Nahrstedt, K. 1999. CPU. Service classes for multimedia applications. In *Proceedings of the IEEE International Conference on Mutimedia Computing and Systems* (June). Florence, Italy.

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

Coulson, G., Blair, G. S., Robin, P., and others. 1993. Extending CHORUS micro-kernel to support continuous media applications. In *4th International Workshop on Network and Operating System Support for Digital Audio and Video*. Lancaster, UK.

Demers, A., Keshav, S., and Shenker, S. 1989. Analysis and simulation of a fair queueing alorithm. In *ACM SIGCOMM* (September), pp. 1–12.

Deng, Z., and Liu, J. W. S. 1997. Scheduling real-time applications in open envirovment. In *Proceedings of the 18th IEEE Real-Time Systems Symposium* (December). San Francisco, CA, USA, pp. 308–319.

Deng, Z., Liu, J. W. S., and Sun, J. 1997. A scheme for scheduling hard real-time applications in open system environment. In *Proceedings of the Ninth Euromicro Workshop on Real-Time Systems* (June). Toledo, Spain, pp. 191–199.

Diot, C., Huitema, C., and Turletti, T. Mutlimedia applications should be adaptive.

Ford, B., Back, G., Benson, G., Lepreau, J., Lin, A., and Shivers, O. 1997. The flux OSKit: A substrate for OS and language research. In *16 ACM Symposium on Operating Systems Principles* (October). Saint-Malo, France.

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

Gergeleit, M., and Streich, H. TaskPair-scheduling with optimistic case execution times - An example for an adaptive real-time system. Research Division of Responsive Systems National Center for Computer Science.

Ghazalie, T. M., and Baker, T. P. 1995. Aperiodic servers in a deadline scheduling environment. *Real-Time Systems* 9(1): 31–68.

Govindan, R., and Anderson, D. P. 1991. Scheduling and IPC mechanisms for continuous media. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles* (October). Asilomar, Pacific Grove, CA, pp. 68–80.

Goyal, P., Guo, X., and Vin, H. M. 1996. A hierarchical CPU scheduler for multimedia operating systems. In *Proceedings of the 2nd Symposium on Operating Systems Design and Implementation* (October). Seattle, WA, pp. 107–121.

Huang, T.-Y., and Liu, J. W. S. 1995. Predicting the worst-case execution time of concurren execution of instructions and cycle-stealing DMA I/O operations. In *ACM SIGPLAN Workshop on Language, Compilers and Tools for Real-Time Systems*. La Jolla, California.

Huang, T.-Y., Liu, J. W. S., and Chung, J.-Y. 1996. Allowing cycle-stealing direct memory access I/O concurrent with hard-real-time programs. In *Int. Conf. on Parallel and Distributed Systems*. Tokyo.

Hyden, E. A. 1994. Operating system support for quality of service (February). Computer Laboratory, University of Cambridge.

Jansen, P. G., and Wijgerink, E. 1996. Flexible scheduling by deadline inheritance in soft real-time kernels.

Jeffay, K. 1992. Scheduling sporadic tasks with shared resources in hard-real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium* (December). Phoenix, AZ.

Jeffay, K., and Bennet, D. 1995. A rate-based execution abstraction for multimedia computing. In *Proceedings of the Fifth International Workshop on Network and Operating System Support for Digital Audio and Video*. (April). Durham, NH.

Jeffay, K., and Goddard, S. M. 1999. A theory of rate-based execution. In *Proceedings of the IEEE Real-Time Systems Symposium* (December). Phoenix, AZ.

Jeffay, K., and Stone, D. L. 1993. Accounting for interrupt handling costs in dynamic priority task systems. In *IEEE Real Time System Symposium*, pp. 212–221.

Jeffay, K., Smith, F. D., Moorthy, A., and Anderson, J. H. 1998. Proportional share scheduling of operating system services for real-time applications. In *IEEE Real Time System Symposium* (December). Madrid, Spain.

Jeffay, K., Stanat, D. F., and Martel, C. U. 1991. On non-preemptive scheduling of periodic and sporadic tasks. *Real Time System Symposium* (December). San Antonio.

Jeffay, K., Stone, D. L., and Smith, F. D. 1992. Kernel support for live digital audio and video. *Computer Communications* 15(6): 388–395.

Jeffay, K., Stone, D. L., Talley, T., and Smith, F. D. 1993. Adaptive, best-effort delivery of digital audio and video across packet switched networks. In *Network and Operating System Support for Digital Audio and Video*.

Jones, M. B. 1993. Adaptive real-time resource management supporting composition of independently authored time-critical services.

Kaneko, H., Stankovic, J. A., Sen, S., and Ramamritham, K. 1996. Integrated scheduling of multimedia and hard real-time tasks. In *Proceedings of the IEEE Real-Time Systems Symposium* (December). Washington, DC, USA, pp. 206–215.

Kang, D.-I., Gerber, R., and Golubchik, L. 1998. Automated techniques for designing embedded signal processors on distributed platforms (October). University of Mariland, (UMIACS-TR-98-57).

Kang, D.-I., Gerber, R., and Sakena, M. 1997. Performance-based design of distributed real-time system. In *Proceedings of the 3rd IEEE Real-Time Technology and Applications Symposium (RTAS '97)* (June). Montreal, Canada, pp. 2–13.

Kang, D.-I., Gerber, R., and Sakena, M. 1998. Parametric design syntesis of distributed embedded systems (March). University of Mariland, (UMIACS-TR-98-18).

Kleinrock, L. 1975. *Queuing Systems*. Wiley-Interscience.

Koren, G., and Shasha, D. 1995. Skip-over: Algorithms and complexity for overloaded systems that allow skips. In *IEEE Real Time System Symposium*. Pisa.

Lamastra, G., Lipari, G., Buttazzo, G., Casile, A., and Conticelli, F. 1997. HARTIK 3.0: A portable system for developing real-time applications. In *Proceedings of the 4th IEEE International Conference on Real-Time Computing Systems and Applications (RTCSA '97)* (October). Taipei, Taiwan, pp. 43–50.

Lee, C., Rajkumar, R., Lehoczky, J., and Siewiorek, D. 1998. Pratical solutions for QoS-based resource allocation. In *Proceedings of the IEEE Real Time Systems Symposium* (December). Madrid, Spain.

Lehoczky, J. P. 1996. Real-time queueing theory. In *Proceedings of the 17th IEEE Real-Time Systems Symposium* (December). Washington, DC, USA, pp. 186–195.

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

Lipari, G. 1997. Resource constraints among periodic and aperiodic tasks (November). RETIS Lab—Scuola Superiore di Studi Universitari e Perfezionamento S. Anna, (9702).

Lipari, G., and Baruah, S. K. 2000. Greedy reclaimation of unused bandwidth in constant bandwidth servers. In *IEEE Proceedings of the 12th Euromicro Conference on Real-Time Systems* (June). Stokholm, Sweden.

Lipari, G., Buttazzo, G., and Abeni, L. 1998. A bandwidth reservation algorithm for multi-application systems. In *IEEE Real Time Computing Systems and Applications* (October). Hiroshima, Japan.

Liu, C. L., and Layland, J. W. 1973. Scheduling alghorithms for multiprogramming in a hard real-time environment. *Journal of the ACM* 20(1): 46–61.

Liu, J. W. S., Lin, K. J., and Natarjan, S. 1987. Scheduling real-time, periodic jobs using imprecise results. In *IEEE Real Time System Symposium* (December). San Jose, California.

Massalin, H. 1992. Synthesis: An efficient implementation of fundamental operating system services. Columbia University.

Mercer, C. W., Rajkumar, R., and Tokuda, H. 1993. Applying hard real-time technology to multimedia systems. In *Workshop on the Role of Real-Time in Multimedia/Interactive Computing System*.

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

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

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

Parekh, A. K., and Gallager, R. G. 1993. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking* 1(3): 344–357.

Parekh, A. K., and Gallager, R. G. 1994. A generalized processor sharing approach to flow control in intergrated services networks: the multiple node case. In *IEEE/ACM Transanctions on Networking*, pp. 137–150.

Rajkumar, R., Abeni, L., de Niz, D., Ghosh, S., Miyoshi, A., and Saewong, S. 2000. Recent developments with Linux/RK. In *Proceedings of the Second Real-Time Linux Workshop* (November). Orlando, Florida.

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

Rajkumar, R., Lee, C., Lehoczky, J., and Siewiorek, D. 1997. A resource allocation model for QoS management. In *Proceedings of the IEEE Real Time Systems Symposium* (December).

Schulzrinne, H., and Kurose, J. F. Distribution of the loss period for some queues in continuous and discrete time.

Sha, L., Rajkumar, R., and Lehoczky, J. P. 1990. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers* 39(9).

Sprunt, B., Sha, L., and Lehoczky, J. P. 1989. Aperiodic scheduling for hard real-time systems. *The Journal of Real-Time Systems*.

Spuri, M., and Buttazzo, G. 1996. Scheduling aperiodic tasks in dynamic priority systems. *Real-Time Systems* 10(2): 179–210.

Spuri, M., and Buttazzo, G. C. 1994. Efficient aperiodic service under the earliest deadline scheduling. In *Proceedings of the IEEE Real-Time Systems Symposium* (December).

Spuri, M., Buttazzo, G., and Sensini, F. 1995. Robust aperiodic scheduling under dynamic priority systems. In *Proceedings of the IEEE Real-Time Systems Symposium* (December).

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

Stoica, I., Abdel-Wahab, H., and Jeffay, K. 1997. On the duality between resource reservation and proportional share resource allocation . In *Proceedings of the SPIE Conference on Multimedia Computing and Networking* (February), 3020, San Jose, CA, pp. 207–214.

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

Streich, H. TaskPair-scheduling: An approach for dynamic real-time systems. Research Division of Responsive Systems National Center for Computer Science.

Talley, T., and Jeffay, K. 1996. A general framework for continuous media transmission control. In *21st IEEE Conference on Local Computer Network* (October). Minneapolis, OPT.

Tia, T.-S., Deng, Z., Shankar, M., Storch, M., Sun, J., Wu, L.-C., and Liu, J. W. S. 1995. Probabilistic performance guarantee for real-time tasks with varying computation times. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium* (January). Chicago, Illinois, pp. 164–173.

Tokuda, H., Nakajima, T., and Rao, P. 1990. Real-time mach: Toward a predictable real-time system. In *USENIX Mach Workshop* (October), pp. 73–82.

VESA BIOS Extensions—Core Functions—Version 2.0. VESA.

Waldspurger, C. A., and Weihl, W. E. 1994. Lottery scheduling: Flexible proportional-share resource management. In *First Symposium on Operating System Design and Implementation* (November). pp. 1–12.

Waldspurger, C. A., and Weihl, W. E. 1995. Stride scheduling: Deterministic proportional-share resource mangement (June). Massachusetts Institute of Technology, (MIT/LCS/TM-528).

Wang, Y. C., and Lin, K. J. 1998. Enanching the real-time capability of the Linux kernel. *Real Time Computing Systems and Applications* (October). Hiroshima, Japan.

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

Zhang, H. 1995. Service disciplines for guaranteed performance service in packet-switching networks. *Proceedings of the IEEE* 83(10): 1374–1396.

Luca Abeni graduated in Computer Engineering at the University of Pisa in 1998, and received a Ph.D. in Computer Engineering at the Scuola Superiore Sant'Anna of Pisa (Italy) in 2002. During 2000 he has been a visiting student at the Carnegie Mellon University, working with Professor Ragunathan Rajkumar on resource reservation algorithms for real-time kernels. During 2002 he has been a visiting student to the Oregon Graduate Institute (Portland), working on evaluating and enhancing the real-time performance of the Linux kernel. He currently works in Broadsat S.r.l., developing multimedia streaming solutions.

Giorgio Buttazzo is an Associate Professor of Computer Engineering at the University of Pavia, Italy. He graduated in Electronic Engineering at the University of Pisa in 1985, received a Master in Computer Science at the University of Pennsylvania in 1987, and a Ph.D. in Computer Engineering at the Scuola Superiore S. Anna of Pisa in 1991. During 1987, he worked on active perception and real-time control at the G.R.A.S.P. Laboratory of the University of Pennsylvania, in Philadelphia. From 1991 to 1998, he held a position of Assistant Professor at the Scuola Superiore S. Anna of Pisa, doing research on robot control systems and real-time scheduling. His main research interests include real-time operating systems, dynamic scheduling algorithms, quality of service control, multimedia systems, advanced robotics applications, and neural networks. He is a member of the IEEE and the IEEE Computer Society.