Adaptive Rate Control through Elastic Scheduling

Giorgio Buttazzo
Dept. of Computer Science
University of Pavia
buttazzo@unipv.it

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

Abstract

In real-time computing systems, timing constraints imposed on application tasks are typically guaranteed off line using schedulability tests based on fixed parameters and worst-case execution times. However, a precise estimation of tasks' computation times is very hard to achieve, due to the non deterministic behavior of several low-level processor mechanisms, such as caching, prefetching, and DMA data transfer.

The disadvantage of relying the guarantee test on a priori estimates is that an underestimation of computation times may jeopardize the correct behavior of the system, whereas an overestimation will certainly waste system resources and causes a performance degradation.

In this paper, we propose a new methodology for automatically adapting the rates of a periodic task set without forcing the programmer to provide a priori estimates of tasks' computation times. Actual executions are monitored by a runtime mechanism and used as feedback signals for predicting the actual load and achieving rate adaptation. Load balancing is performed using an elastic task model, according to which tasks utilizations are treated as springs with given elastic coefficients.

1 Introduction

Most of real-time applications require the execution of periodic activities, whose rate can usually be defined within a certain range. The higher the frequency, the better the performance. Depending on the application domain, some task rates are rigidly imposed by the environment (e.g., a PAL frame grabber produces a new half frame every 20 milliseconds), whereas other activities can be more flexible, producing significant results when their rates are within a certain range. For example, in multimedia systems, activities such a voice sampling, image acquisition, sound generation, data compression, and video playing, are performed periodically, but their execution rates are not so rigid. Missing a deadline while displaying an MPEG video may decrease the quality of service (QoS), but does not cause critical system faults. Depending on the requested QoS, tasks may increase or decrease their execution rate to accommodate the requirements of other concurrent activities.

Even in some control application, there are situations in which periodic tasks could be executed at different rates in different operating conditions. For example, in a flight control system, the sampling rate of the altimeters is a function of the current altitude of the aircraft: the lower the altitude, the higher the sampling frequency. A similar need arises in robotic applications in which robots have to work in unknown environments where trajectories are planned based on the current sensory information. If a robot is equipped with proximity sensors, in order to maintain a desired performance, the acquisition rate of the sensors must increase whenever the robot is approaching an obstacle.

In other situations, the possibility of varying tasks' rates increases the flexibility of the system in handling overload conditions, providing a more general admission control mechanism. For example, whenever a new task cannot be guaranteed by the system to meet its timing constraints, instead of rejecting the task, the system can try to reduce the utilizations of the other tasks (by increasing their periods in a controlled fashion) to decrease the total load and accommodate the new request. Unfortunately, there is no uniform approach for dealing with these situations.

For example, Kuo and Mok [7] propose a load scaling technique to gracefully degrade the workload of a system by adjusting the periods of processes. In this work, tasks are assumed to be equally important and the objective is to minimize the number of fundamental frequencies to improve schedulability under static priority assignments. In [14], Nakajima and Tezuka show how a real-time system can be used to support an adaptive application: whenever a deadline miss is detected, the period of the failed task is increased. In [15], Seto et al. describe a method for computing tasks' periods to minimize a performance index defined over the task set. This approach is effective at a design stage to optimize the performance of a discrete control system, but it cannot be used for on-line load adjustment. In [9], Lee, Rajkumar and Mercer propose a number of policies to dynamically adjust the tasks' rates in overload con-

ditions. In [1], Abdelzaher, Atkins, and Shin present a model for QoS negotiation to meet both predictability and graceful degradation requirements in cases of overload. In this model, the QoS is specified as a set of negotiation options, in terms of rewards and rejection penalties. In [12, 13], Nakajima shows how a multimedia activity can adapt its requirements during transient overloads by scaling down its rate or its computational demand. However, it is not clear how the the QoS can be increased when the system is underloaded. Recently, in [2, 3], a feedback control mechanism has been considered for observing and adjusting the system workload to reduce the number of deadline misses when tasks' execution times are not precisely known. However, this approach does not permit to control each task's utilization individually.

In [4], Beccari et al. propose several policies for handling overload through period adjustment. The authors, however, do not address the problem of increasing the task rates when the processor is not fully utilized. Moreover, all the proposed techniques are based on the knowledge of worst-case computation times.

Although these approaches may lead to interesting results in specific applications, we believe that a more general framework can be used to avoid a proliferation of policies and treat different applications in a uniform fashion. Moreover, most of the approaches proposed in the literature are based on the knowledge of task computation times. However, a precise estimation of tasks' computation times is very hard to achieve, due to the non deterministic behavior of several low-level processor mechanisms, such as caching, prefetching, and DMA data transfer.

The disadvantage of relying the guarantee test on a priori estimates is that an underestimation of computation times may jeopardize the correct behavior of the system, whereas an overestimation will certainly waste system resources and causes a performance degradation.

In this paper we present a novel approach in which task periods can be dynamically adjusted based on the current load. Load is estimated not by monitoring the number of deadline miss, but monitoring the actual computation time of each job. When the estimated load is found to be greater than a certain predefined threshold (it can be 1 under EDF), the elastic theory [6] is used to enlarge the task periods to find a feasible configuration. We will use the elastic approach to automatically find a feasible period configuration based on the actual tasks' demand, relieving the programmer of estimating the tasks' computation times.

The rest of the paper is organized as follows. Section 2 presents the task model and the basic assumptions. Section 3 briefly recalls the elastic approach. Section 4 illustrates the adaptive method for setting the task periods. Section 5 illustrates some experimental results achieved on the HARTIK kernel. Finally, Section 6 contains our conclusions and future work.

2 Task model

In our framework, each task is considered as flexible as a spring with a given rigidity coefficient and length constraints. In particular, the utilization of a task is treated as an elastic parameter, whose value can be modified by changing the period within a specified range.

Each task is characterized by four parameters: a computation time C_i, a minimum period T_{i_0} (considered as a nominal period), a maximum period $T_{i_{max}}$, and an elastic coefficient $E_i \geq 0$, which specifies the flexibility of the task to vary its utilization for adapting the system to a new feasible rate configuration. The greater E_i, the more elastic the task. Thus, an elastic task is denoted as:

$$\tau_i(C_i, T_{i_0}, T_{i_{max}}, E_i).$$

In the following, T_i will denote the actual period of task τ_i, which is constrained to be in the range $[T_{i_0}, T_{i_{max}}]$. Any task can vary its period according to its needs within the specified range. Any variation, however, is subject to an *elastic* guarantee and is accepted only if there exists a feasible schedule in which all the other periods are within their range. If $\sum \frac{C_i}{T_{i_0}} \leq 1$, all tasks can be created at the minimum period T_{i_0}, otherwise the elastic algorithm is used to adapt the tasks' periods to T_i so that $\sum \frac{C_i}{T_i} = U_d \leq 1$.

With respect to the classical elastic approach described in [6], in this paper we assume that tasks' computation times are not known a priori, but are estimated at runtime by a monitoring mechanism built in the kernel.

Since the estimated values of the execution times can change during program execution, the rate adaptation algorithm is periodically executed. The estimated execution times can be considered as a feedback to adapt the system load. In overload conditions, they are used to keep the number of missed deadlines as low as possible, whereas in underload conditions ($\sum \frac{C_i}{T_i} << 1$) the efficiency of the system is increased to utilize the processor up to a desired value U_d.

3 Task compression algorithm

For the sake of completeness, in this section we will briefly recall the compression algorithm presented in [6]. In the next section we will use this algorithm to

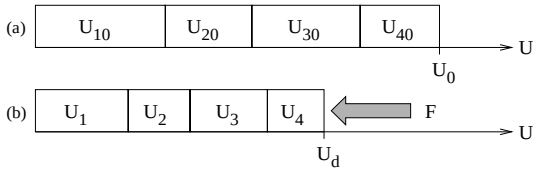

Figure 1: Processor utilizations of four periodic tasks before compression (a) and after compression (b).

automatically find a feasible period configuration based on the actual tasks' demand, relieving the programmer of estimating the tasks' computation times.

Using the same notation introduced by Liu and Layland [10], let U_{lub}^A be the *least upper bound* of the total utilization factor for a given scheduling algorithm A (we recall that for n tasks $U_{lub}^{RM} = n(2^{1/n} - 1)$ and $U_{lub}^{EDF} = 1$). Moreover, we define $U_0 = \sum_{i=1}^{n} C_i/T_{i_0}$ and $U_{min} = \sum_{i=1}^{n} C_i/T_{i_{max}}$. Hence, a task set can be schedulable by A at the nominal rates if $U_0 \leq U_{lub}^A$. Under EDF, such a schedulability condition becomes necessary and sufficient.

Under the elastic model, given a scheduling algorithm A and a set of n tasks with $U_0 > U_{lub}^A$, the objective of the guarantee is to compress tasks' utilization factors in order to achieve a desired utilization $U_d \leq U_{lub}^A$ such that all the periods are within their ranges. In [6] it has been shown that, if $U_{min} \leq U_d \leq U_{lub}^A$, a solution can quickly be found by compressing tasks' utilizations according to their elastic coefficients, as illustrated in Figure 1.

The algorithm[1] for compressing a set Γ of n elastic tasks up to a desired utilization U_d is shown in Figure 2.

All tasks' utilizations that have been compressed to cope with an overload situation can return at their nominal values when the overload is over. Notice that, to avoid missing deadlines, if a task wants to decrease its period, the period change must occur at its next release time. Thus, when the system receives a request of period variation, it calculates the new periods according to the elastic model: if the new configuration is found to be feasible, then it increases the periods of the decompressed tasks immediately, but decreases the periods of the compressed tasks only at their next release time.

4 Online Adaptation Algorithm

The elastic approach provides a powerful and flexible methodology for adapting the periodic tasks' rates to different workload conditions. The effectiveness of the

[1] The actual implementation of the algorithm contains more checks on tasks' variables, which are not shown here to simplify its description.

```
Algorithm Task_compress(Γ, U_d) {

  U_0 = Σ_{i=1}^{n} C_i/T_{i_0};
  U_min = Σ_{i=1}^{n} C_i/T_{i_max};
  if (U_d < U_min) return INFEASIBLE;

  do {

      U_f = E_v = 0;
      for (each τ_i) {
          if ((E_i == 0) or (T_i == T_{i_max}))
              U_f = U_f + U_i;
          else E_v = E_v + E_i;
      }

      ok = 1;
      for (each τ_i ∈ Γ_v) {
          if ((E_i > 0) and (T_i < T_{i_max})) {
              U_i = U_{i_0} − (U_0 − U_d + U_f)E_i/E_v;
              T_i = C_i/U_i;
              if (T_i > T_{i_max}) {
                  T_i = T_{i_max};
                  ok = 0;
              }
          }
      }

  } while (ok == 0);
  return FEASIBLE;
}
```

Figure 2: Algorithm for compressing a set of elastic tasks.

approach, however, strongly relies on the knowledge of the worst-case execution times (WCETs). If WCETs are not precisely estimated, the compression algorithm will lead to wrong period assignment. In particular, if WCETs are underestimated the compressed tasks may start missing deadlines, whereas if WCETs are overestimated, the algorithm will cause a waste of resources, as well as a performance degradation.

This problem has been addressed in [11, 13] by using a CPU reservation technique to enforce the maximum execution time per period, to each task. With this technique, however, the amount of time reserved to each task in each period must still be defined based on some off-line estimation. If the reserved budget is too small, the task will experience large overruns which cause the algorithm to increase its period too much. On the other hand, if the estimation is too big, the periods are not optimized, the reserved budget is never used completely, and the system is underutilized.

The solution proposed in this paper uses on-line esti-

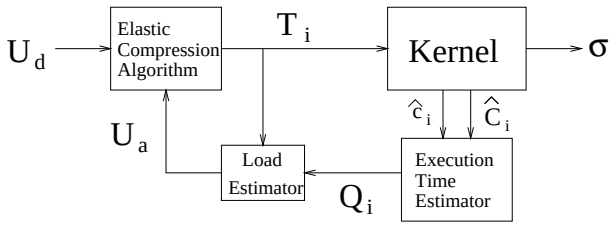

Figure 3: The execution times estimation architecture.

mates of tasks' execution times as feedback for achieving load adaptation. Such estimates are derived by a runtime monitoring mechanism embedded in the kernel. When a task starts its execution, it is created at its minimum rate, and, at the end of each period, a runtime monitoring mechanism updates the mean execution time $\hat{c}_i$ and the maximum execution time $\hat{C}_i$. Figure 3 shows the architecture used to perform the rate adaptation. The two values $\hat{C}_i$ and $\hat{c}_i$ derived by the monitoring kernel mechanism are used to compute an execution time estimate Q_i, used by the load estimator to compute the actual load $U_a = \sum \frac{Q_i}{T_i}$. Such a value is then used by the elastic algorithm (periodically invoked with a period P) to adapt tasks' rates. Thus the objective of the global control loop is to maintain the estimated actual load U_a close to a desired value U_d.

In this way, a new task can initially cause a lot of deadline misses, but when the execution times estimates begin to become accurate, the adaptation architecture enlarges the new task's period decreasing the total CPU utilization and reduces the number of missed deadlines. Since the WCET estimation is not necessarily exact, there can still be some sporadic deadline misses, but their number is reduced by the rate adaptation mechanism.

If the $\hat{C}_i$ estimate is used to compute tasks' utilizations to apply the elastic algorithm, tasks are assigned larger periods and the number of deadline misses quickly reduces to zero. However, this solution can cause a waste of resources, since tasks seldom experience their worst case simultaneously.

To increase efficiency, a more optimistic estimation can be used in order to exploit system resources: the resulting approach is a trade off between "rigid" reservation systems (in which each task is assigned a fixed amount of resources and cannot demand more, also if the other tasks are requiring less than the reserved amount) and completely unprotected system, such as bare EDF or RM. In this sense this approach is similar to the "Bandwidth Sharing Server" (BSS) [16], where tasks belonging to the same application can share the same resources. In the BSS case, however, the fraction of bandwidth that can be exchanged among tasks belonging to a particular application cannot be controlled, whereas in the elastic model it depends on the elastic coefficients.

In order to avoid that the number of deadline misses per time unit increases indefinitely to infinite, the execution time estimate used to perform the elastic compression must be greater than the task's mean execution time, so a value between $\hat{c}_i$ and $\hat{C}_i$ is considered acceptable. In our model, the elastic compression algorithm is invoked using a value

$$Q_i = \hat{c}_i + k(\hat{C}_i - \hat{c}_i)$$

where $k \in [0, 1]$ is the *guarantee factor*. Then, the utilization factor $\hat{U}_i$ is computed as

$$\hat{U}_i = \frac{Q_i}{T_i}$$

and the actual load U_a is estimated as

$$U_a = \sum \hat{U}_i.$$

It is worth noting that, if $k = 1$, the elastic algorithm results to be based on WCET estimations, so only few deadlines are missed during the transient overload caused by a task creation. A smaller value of k allows to increase the actual system utilization at the cost of an increasing number of possible deadline misses (we recall that a deadline is missed when many tasks require a long execution at the same time). A value of $k = 0$ allows maximum efficiency but is the limit under which the system overload becomes permanent.

5 Experimental results

To test the effectiveness of the adaptation algorithm, the elastic task model has been implemented on top of the HARTIK kernel [5, 8], as a middleware layer. In particular, the elastic guarantee mechanism has been implemented as a high priority task, the Elastic Manager (EM), activated by the other tasks when they are created or when they want to change their period. Whenever activated, the EM calculates the new periods and changes them atomically. Whenever a task asks to decrease its period, the EM will change it, if possible, at its next release time.

In a first experiment, we tested the behavior of the elastic compression mechanism using the task set shown in Table 1. Each task executes for a fixed amount of time, reported in the C_i column. The first three tasks start executing at time $t = 0$ at their nominal period, while the fourth task starts at time $t_1 = 10sec$.

When τ_4 is started, the task set is not schedulable with the current periods, thus the EM tries to accommodate the request of τ_4 by increasing the periods of the other

Task	C_i	T_{i_0}	$T_{i_{max}}$	E_i
τ_1	30	30	500	1
τ_2	60	30	500	1
τ_3	90	30	500	1
τ_4	24	30	500	1

Table 1: Task set parameters used for the first experiment. Periods and computation times are expressed in milliseconds.

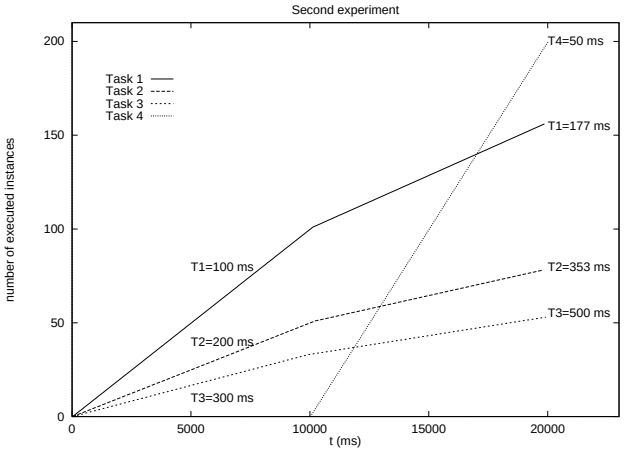

Figure 4: Dynamic task activation.

tasks according to the elastic model. The actual execution rates of the tasks are shown in Figure 4. Notice that, although the first three tasks have the same elastic coefficients, their periods are changed by a different amount, because tasks have different utilization factors.

In the following experiments we tested the on-line estimation mechanism. In order to do this, four tasks with variable execution times are started at time $t = 0$, activating the estimation and adaptation mechanisms. Tasks' periods and elastic factors are the same as in the previous experiment, shown in Table 1. During this experiment, we monitored the tasks' execution times and measured how the estimated values and the number of deadline misses change as the guarantee factor k varies from 0 to 1.

Figure 5 shows the estimated execution time Q_1 (from the jobs of task τ_1) for different values of the guarantee factor k. We can see that the value of the guarantee factor influences the stability, the conservativeness and the precision of the estimation. In fact, small values of k cause the estimate to converge around a stationary value after a long transient. For $k = 0$, the estimate converges in more than $60ms$, whereas for $k = 1$ it converges almost immediately. On the other hand, a high guarantee factor causes the system to be underutilized.

The influence of the guarantee factor on the number of missed deadlines is illustrated in Figure 6, which shows

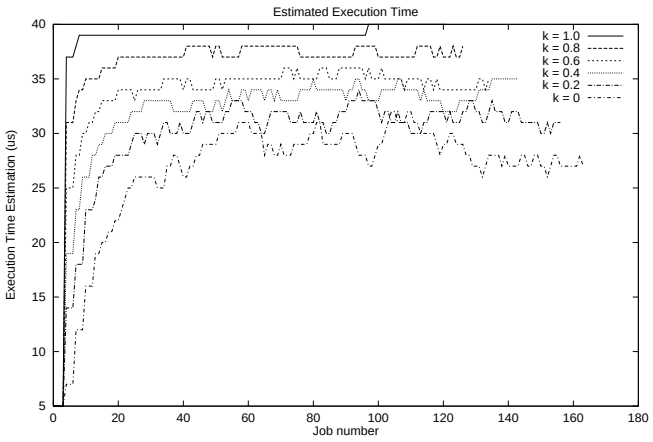

Figure 5: Estimated execution time of task τ_1 as a function of the guarantee factor k.

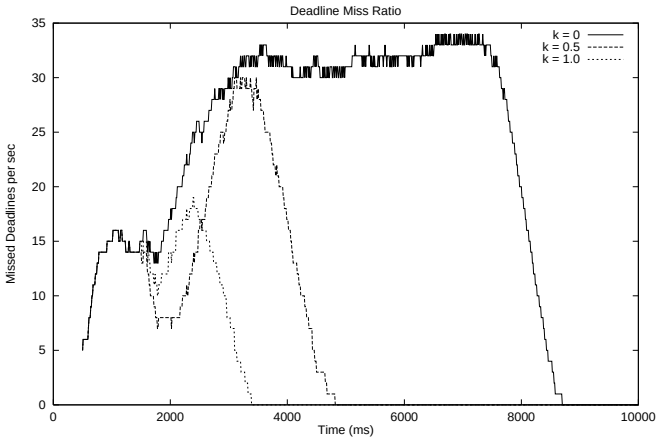

Figure 6: Missed deadlines per second as a function of the guarantee factor.

the evolution of the number of missed deadlines as a function of time for different values of k. From the figure, we can see that the number of missed deadlines per second always converges to 0, after an initial transient (even in the worst case, where $k = 0$, the system overload is not permanent). However, the duration of that transient depends on k: for small values of the guarantee factor, the overload is recovered after a long time, while for larger values the transient is shorter.

In order to obtain a fast adaptation of the task parameters without incurring in a system underutilization, a trade-off between high values and low values for the guarantee factor is required. Looking at the figures, it can be noted that in this case a value $k = 0.5$ can be a reasonable trade-off.

6 Conclusions

In this paper we presented a method for automatically adapting the computational demand of a periodic task set to the actual processor capacity. Tasks are treated as elastic springs whose utilizations can be adjusted (through proper period variations) to create a desired workload. Task execution times are estimated by an on-line monitoring mechanism embedded in the kernel, so avoiding an explicit measurement and specification of worst-case execution times.

Using this approach, periodic tasks enter the system executing at their maximum period, and increase their execution rate in order to control the system workload to a desired value U_d.

The proposed model can also be used to handle overload situations in a more flexible way. In fact, whenever a new task cannot be guaranteed by the system, instead of rejecting the task, the system can try to reduce the utilizations of the other tasks (by increasing their periods in a controlled fashion) to decrease the total load and accommodate the new request. As soon as a transient overload condition is over (because a task terminates or voluntarily increases its period) all the compressed tasks may expand up to their original utilization, eventually recovering their nominal periods.

The major advantage of the proposed method is that the policy for selecting a solution is implicitly encoded in the elastic coefficients provided by the user. Each task is varied based on its current elastic status and a feasible configuration is found, if there exists one.

The presented approach has been implemented on the HARTIK kernel [5, 8], where some experiments have been performed to show how the on line estimation can reduce the number of missed deadlines and allow to achieve a better system utilization.

References

[1] T. F. Abdelzaher, E. M. Atkins, and K. G. Shin, "QoS Negotiation in Real-Time Systems and Its Applications to Automated Flight Control," *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, Montreal, Canada, June 1997.

[2] J. A. Stankovic, C. Lu, and S. H. Son, "The Case for Feedback Control in Real-Time Scheduling", *IEEE Proceedings of the Euromicro Conference on Real-Time Systems*, York, England, June 1998.

[3] C. Lu, J. A. Stankovic, G. Tao, and S.H. Son, "Design and Evaluation of a Feedback Control EDF Scheduling Algorithm", *Proceedings of the 20th IEEE Real-Time Systems Symposium*, Phoenix, Arizona, December 1999.

[4] G. Beccari, S. Caselli, M. Reggiani, F. Zanichelli, "Rate Modulation of Soft Real-Time Tasks in Autonomous Robot Control Systems," *IEEE Proceedings of the 11th Euromicro Conference on Real-Time Systems*, York, June 9-11, 1999.

[5] G. Buttazzo, "HARTIK: A Real-Time Kernel for Robotics Applications", *Proceedings of the 14th IEEE Real-Time Systems Symposium*, Raleigh-Durham, pp. 201–205, December 1993.

[6] G. Buttazzo, G. Lipari, and L. Abeni, "Elastic Task Model for Adaptive Rate Control", *Proceedings of the IEEE Real-Time Systems Symposium*, December 1998.

[7] T.-W. Kuo and A. K, Mok, "Load Adjustment in Adaptive Real-Time Systems," *Proceedings of the 12th IEEE Real-Time Systems Symposium*, December 1991.

[8] G. Lamastra, G. Lipari, G. Buttazzo, A. Casile, and F. Conticelli, "HARTIK 3.0: A Portable System for Developing Real-Time Applications," *Proceedings of the IEEE Real-Time Computing Systems and Applications*, Taipei, Taiwan, October 1997.

[9] C. Lee, R. Rajkumar, and C. Mercer, "Experiences with Processor Reservation and Dynamic QOS in Real-Time Mach," *Proceedings of Multimedia Japan 96*, April 1996.

[10] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard real-Time Environment," *Journal of the ACM* 20(1), 1973, pp. 40–61.

[11] H. Fujita, T. Nakajima, and H. Tezuka, "A Processor Reservation System supporting Dynamic QOS control", *2nd International Workshop on Real-Time Computing Systems and Applications*, October 1995.

[12] T. Nakajima, "Dynamic QOS Control and Resource Reservation," *IEICE, RTP'98*, 1998.

[13] T. Nakajima, "Resource Reservation for Adaptive QOS Mapping in Real-Time Mach," *Sixth International Workshop on Parallel and Distributed Real-Time Systems*, April 1998.

[14] T. Nakajima and H. Tezuka, "A Continuous Media Application supporting Dynamic QOS Control on Real-Time Mach," *Proceedings of the ACM Multimedia '94*, 1994.

[15] D. Seto, J.P. Lehoczky, L. Sha, and K.G. Shin, "On Task Schedulability in Real-Time Control Systems," *Proceedings of the IEEE Real-Time Systems Symposium*, December 1997.

[16] G. Lipari, G. Buttazzo, and L. Abeni, "A Bandwidth Reservation Algorithm for Multi-Application Systems", *Proceedings of IEEE Real Time Computing Systems and Applications*, Hiroshima, Japan, October 1998.