Real-Time Systems, 23, 7–24, 2002
© 2002 Kluwer Academic Publishers. Manufactured in The Netherlands.

Adaptive Workload Management through Elastic Scheduling

GIORGIO BUTTAZZO buttazzo@unipv.it
Department of Computer Science, University of Pavia, Italy

LUCA ABENI luca@sssup.it
RETIS Lab, Scuola Superiore S. Anna, Pisa, Italy

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.

Keywords: feedback scheduling, adaptive real-time systems, overload management, elastic rate adaptation

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 ms), 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.

In the literature related to real-time systems there is no uniform approach for dealing with these situations. For example, Kuo and Mok (1991) 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. Nakajima and Tezuka (1994) 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. Seto et al. (1997) 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. Lee et al. (1996) propose a number of policies to dynamically adjust the tasks' rates in overload conditions. Abdelzaher et al. (1997) 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.

Nakajima (1998) 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 QoS can be increased when the system is underloaded. Recently, in Stankovic et al. (1998) and Lu et al. (1999), 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 Lu et al. (2000), the previous approach has been extended to enable system designers to specify a desired behavior in terms of a set of performance metrics and apply existing control methods to design an adaptive real-time system to achieve the specified requirements.

Beccari et al. (1999) 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. Unfortunately, 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 detecting the number of deadline miss, but by monitoring the actual computation time of each job, which is assumed to be unknown a priori. When the estimated load is found to be greater than a certain predefined threshold (it can be 1 under EDF), the elastic scheduling method proposed in Buttazzo et al. (1998) is used to enlarge the task periods to find a feasible configuration. We 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 paper integrates and consolidates a preliminary work described in Buttazzo and Abeni (2000).

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. To provide a minimum level of guarantee off line, we assume that for each task an upper bound of the worst-case execution time (WCET) is known in advance. Note, however, that overestimating this value does not degrade the actual system performance, since the upper bound is just used to guarantee the feasibility of the schedule when all the tasks run with their maximum period. At run time, an on-line estimate of the actual task computation time is used to perform period adaptation.

Each task is characterized by four parameters: an upper bound of the worst-case computation time C_i^{ub}, 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

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

From a design perspective, elastic coefficients can be set equal to values which are inversely proportional to tasks' importance. For example, in a real-time application, those tasks whose activation rate is imposed by peripheral devices (e.g., image acquisition from

a frame grabber) should have $E_i = 0$. Other periodic computations whose rate is not rigidly determined by the hardware could have some elasticity. For instance, in a robot control application, trajectory planning would have a higher elasticity than obstacle avoidance, which in turn would be more elastic than force control. We recall that the elastic coefficient only determines how the task period is varied within its range.

Also notice that the elastic task model automatically solves an optimal task frequency assignment for tasks whose control performance can be described by a quadratic cost function with respect to the sampling frequency (a spring system naturally minimizes the elastic potential energy, which is proportional to the square of the displacements). According to this approach, the elastic coefficients can be used to express a proper cost function.

In the following, T_i will denote the actual period of task τ_i, which is constrained to be in the range $[T_{i_0}, T_{i_{\max}}]$, whereas C_i will denote its actual exectution time (considered to be unknown *a priori*). In the case of tasks with variable computation times, C_i will denote the actual worst-case execution time. Any period variation, is always 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. In our framework, tasks are scheduled by the Earliest Deadline First algorithm [13]. Hence, if $\sum(C_i^{ub}/T_{i_0}) \leq 1$, 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 such that $\sum(C_i/T_i) = U_d \leq 1$, where C_i is the actual on-line execution estimate and U_d is some desired utilization factor. It can easily been shown (see Section 3.2) that a solution can always be found if $\sum(C_i^{ub}/T_{i_{\max}}) \leq 1$.

With respect to the classical elastic approach described in Buttazzo et al. (1998), in this paper we assume that tasks' actual computation times are not known a priori, but are estimated at run time by a monitoring mechanism built in the kernel, able to record the actual execution time of each task instance. Such a monitoring mechanism is available on the HARTIK kernel, where the adaptive elastic approach has been implemented and tested Abeni and Buttazzo (2000). As demonstrated in Section 5, the on-line estimation mechanism can effectively be applied to tasks with fixed computation times which are not known a priori, as well as to tasks whose computation times vary significantly from instance to instance.

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 $(\sum(C_i/T_i) \ll 1)$ 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 Buttazzo et al. (1998). For a more general description of the elastic approach see Buttazzo et al. (2002), where the algorithm has also been extended to be used in the presence of resource constraints. In the next section we will use the

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

To understand how an elastic guarantee is performed in this model, it is convenient to compare an elastic task τ_i with a linear spring S_i characterized by an elastic coefficient E_i, a nominal length x_{i_0}, and a minimum length $x_{i_{\min}}$. In the following, x_i will denote the actual length of spring S_i, which is constrained to be in the range $[x_{i_{\min}}, x_{i_0}]$. In this comparison, the length x_i of the spring is equivalent to the task's utilization factor $U_i = C_i/T_i$, hence, a set of n tasks with total utilization factor $U_p = \sum_{i=1}^{n} U_i$ can be viewed as a sequence of n springs with total length $L = \sum_{i=1}^{n} x_i$.

Using the same notation introduced by Liu and Layland (1973), 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 periodic 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 a linear spring system of total length $L_0 = \sum_{i=1}^{n} x_{i_0}$, this is equivalent to compressing the springs by a force F, so that the new total length becomes $L_d < L_0$. This concept is illustrated in Figure 1.

In the absence of length constraints (i.e., if $x_{\min} = 0$) the length x_i of each compressed spring can be computed as follows:

$$\forall i \quad x_i = x_{i_0} - (L_0 - L_d)\frac{K_p}{k_i} \tag{1}$$

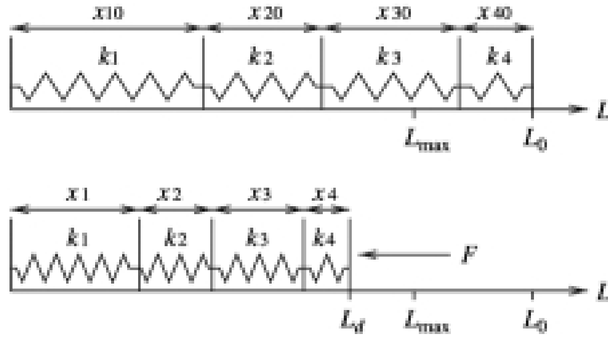

Figure 1. A linear spring system: the total length is L_0 when springs are uncompressed (a); and $L_d < L_0$ when springs are compressed (b).

where

$$K_p = \frac{1}{\sum_{i=1}^{n} \frac{1}{k_i}} \tag{2}$$

If each spring has a length constraint, in the sense that its length cannot be less than a minimum value $x_{i_{\min}}$, then the problem of finding the values x_i requires an iterative solution. In fact, if during compression one or more springs reach their minimum length, the additional compression force will only deform the remaining springs. Thus, at each instant, the set Γ of springs can be divided into two subsets: a set Γ_f of fixed springs having minimum length, and a set Γ_v of variable springs that can still be compressed. Applying Equation (1) to the set Γ_v of variable springs, we have

$$\forall S_i \in \Gamma_v \quad x_i = x_{i_0} - (L_{v_0} - L_d + L_f)\frac{K_v}{k_i} \tag{3}$$

where

$$L_{v_0} = \sum_{S_i \in \Gamma_v} x_{i_0} \tag{4}$$

$$L_f = \sum_{S_i \in \Gamma_f} x_{i_{\min}} \tag{5}$$

$$K_v = \frac{1}{\sum_{S_i \in \Gamma_v} \frac{1}{k_i}} \tag{6}$$

Whenever there exists some spring for which Equation (3) gives $x_i < x_{i_{\min}}$, the length of that spring has to be fixed at its minimum value, sets Γ_f and Γ_v must be updated, and Equations (3)–(6) recomputed for the new set Γ_v. If there exists a feasible solution, that is, if the desired final length L_d is greater than or equal to the minimum possible length of the array $L_{\min} = \sum_{i=1}^{n} x_{i_{\min}}$, the iterative process ends when each value computed by Equation (3) is greater than or equal to its corresponding minimum $x_{i_{\min}}$.

When dealing with a set of elastic tasks, Equations (3), (5) and (6) can be rewritten by substituting all length parameters with the corresponding utilization factors, and the rigidity coefficients k_i and K_v with the corresponding elastic coefficients E_i and E_v. Similarly, at each instant, the set Γ of periodic tasks can be divided into two subsets: a set Γ_f of fixed tasks having minimum utilization, and a set Γ_v of variable tasks that can still be compressed. If $U_{i_0} = C_i / T_{i_0}$ is the nominal utilization of task τ_i, U_{v_0} is the sum of all the nominal utilizations in Γ_v, and U_f is the total utilization factor of tasks in Γ_f, then to achieve a desired utilization $U_d < U_0$ each task has to be compressed up to the following utilization:

$$\forall \tau_i \in \Gamma_v \quad U_i = U_{i_0} - (U_{v_0} - U_d + U_f)\frac{E_i}{E_v} \tag{7}$$

where

$$U_{v_0} = \sum_{\tau_i \in \Gamma_v} U_{i_0} \tag{8}$$

$$U_f = \sum_{\tau_i \in \Gamma_f} U_{i_{\min}} \tag{9}$$

$$E_v = \sum_{\tau_i \in \Gamma_v} E_i \tag{10}$$

If there exist tasks for which $U_i < U_{i_{\min}}$, then the period of those tasks has to be fixed at its maximum value $T_{i_{\max}}$ (so that $U_i = U_{i_{\min}}$), sets Γ_f and Γ_v must be updated (hence, U_f and E_v recomputed), and Equation (7) applied again to the tasks in Γ_v. If there exists a feasible solution, that is, if the desired utilization U_d is greater than or equal to the minimum possible utilization $U_{\min} = \sum_{i=1}^{n} (C_i/T_{i_{\max}})$, the iterative process ends when each value computed by Equation (7) is greater than or equal to its corresponding minimum $U_{i_{\min}}$. The algorithm[1] for compressing a set Γ of n elastic tasks up to a desired utilization U_d is shown in Figure 2.

3.1. Decompression

All tasks' utilizations that have been compressed to cope with an overload situation can return toward their nominal values when the overload is over. Let Γ_c be the subset of compressed tasks (that is, the set of tasks with $T_i > T_{i_0}$), let Γ_a be the set of remaining tasks in Γ (that is, the set of tasks with $T_i \leq T_{i_0}$), and let U_d be the current processor utilization of Γ. Whenever a task in Γ_a decreases its rate or returns to its nominal period, all tasks in Γ_c can expand their utilizations according to their elastic coefficients, so that the processor utilization is kept at the value of U_d.

Now, let U_c be the total utilization of Γ_c, let U_a be the total utilization of Γ_a, and let U_{c_0} be the total utilization of tasks in Γ_c at their nominal periods. It can easily be seen that if $U_{c_0} + U_a \leq U_{lub}$ all tasks in Γ_c can return to their nominal periods. On the other hand, if $U_{c_0} + U_a > U_{lub}$, then the release operation of the tasks in Γ_c can be viewed as a compression, where $\Gamma_f = \Gamma_a$ and $\Gamma_v = \Gamma_c$. Hence, it can still be performed by using Equations (7), (9) and (10) and the algorithm presented in Figure 2.

3.2. Theoretical Results

The following theorem states the convergence and the complexity of the elastic approach, and provides a condition under which a feasible solution is always found.

```
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;
      }

      U_{v_0} = U_0 − U_f;

      ok = 1;
      for (each τ_i ∈ Γ_v) {
          if ((E_i > 0) and (T_i < T_{i_max})) {
              U_i = U_{i_0} − (U_{v_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.

THEOREM 1 *Given a set* $\Gamma = \{\tau_i(C_i^{ub}, T_i, T_{i_{\max}}) | i = 1, \ldots, n\}$ *of elastic tasks, if*

$$\sum_{i=1}^{n} \frac{C_i^{ub}}{T_{i_{\max}}} \leq 1 \tag{11}$$

the compression algorithm converges to a feasible solution in $O(n^2)$ *time, in the worst case.*

Proof: If inequality (11) holds, the feasibility of the solution is guaranteed because, in the worst case, task periods can be extended up to their maximum value $T_i = T_{i_{\max}}$, and

since C_i^{ub} is an upper bound of the actual worst-case computation time $(C_i \leq C_i^{ub})$ we have

$$\sum_{i=1}^{n} \frac{C_i}{T_{i_{\max}}} \leq \sum_{i=1}^{n} \frac{C_i^{ub}}{T_{i_{\max}}} \leq 1$$

Moreover, if condition (11) is verified, the convergence of the iterative compression algorithm can be proved as follows. After each compression step, two situations may occur:

1. $T_i < T_{i_{\max}} \quad \forall \tau_i \in \Gamma_v$.

2. There exists some k, such that $\tau_k \in \Gamma_v$ and $T_k = T_{k_{\max}}$.

If condition 1 is true, a feasible solution has been found and the algorithm stops. If condition 2 is true, set Γ_v is updated by removing those tasks that reached their maximum period and another iteration is performed on the remaining tasks in Γ_v. Every time condition 2 holds, the set Γ_v is reduced at least by one task. Since Γ_v initially contains n tasks, it follows that the compression algorithm will iterate at most for n times. Since, each compression step has an $O(n)$ complexity, the complexity of the overall compression algorithm is $O(n^2)$ in the worst case. ■

In Buttazzo et al. (2000) it is shown that, in order to keep the task set schedulable during compression, periods must be changed at opportune instants. In particular, the following theorem states a property of compression: if at time t tasks increase their periods from T_i to T_i', then from a certain time t^* on, the total utilization factor is decreased from U_p to $U_p' = \sum_{i=1}^{n} (C_i/T_i')$.

THEOREM 2 *Let Γ be a task set with utilization U_p and let Γ_c be the subset of tasks that at time t increase their period, so that the total processor utilization U_p is compressed to $U_p' < U_p$. Let $t^* = \max_{\tau_i \in \Gamma_c} (d_i - c_i(t)/U_i)$, where $c_i(t)$ is the remaining computation time of task τ_i. Then, from time t^* on, the bandwidth used by the task set is not greater than U_p'.*

The reason why the saved bandwidth $U_p - U_p'$ is not available immediately at time t is that some of the compressed tasks already executed with the previous period, thus consuming the processor bandwidth originally allocated to them. As a consequence, interval $[t, t^*]$ is needed for estinguishing the transient.

It is worth noting that in case of decompression (that is, when utilizations are increased by period reduction), a task cannot reduce its period immediately, but has to wait until its next activation.

Consider the example shown in Figure 3, where two tasks, with computation times $C_1 = 3$ and $C_2 = 2$ and periods $T_1 = 10$ and $T_2 = 3$, start at time 0. The processor utilization is $U_p = (29/30)$, thus the task set is schedulable by EDF. Suppose that, at time $t = 14$, τ_1 wants to change its period from $T_1 = 10$ to $T_1' = 5$, so that the compression

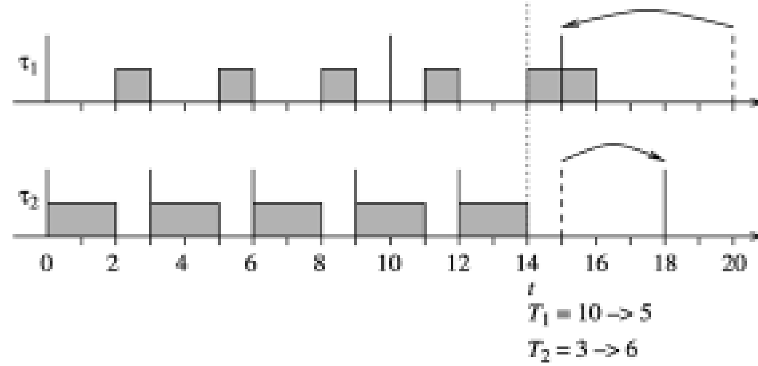

Figure 3. A task can miss its deadline if a period is decreased at arbitrary time.

algorithm increases the period of τ_2 from $T_2 = 3$ to $T_2' = 6$ to keep the system schedulable. The new processor utilization is $U_p' = (28/30)$, so the task set is still schedulable; however, if periods are changed immediately (i.e., at time $t = 14$), task τ_1 misses its deadline at time $t = 15$.

In general, although the periods of the tasks that decrease their rate can be changed immediately, the periods of the tasks that increase their rate can be changed only at their 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 (i.e., $U_p' < 1$), then it increases the periods of the decompressed tasks immediately, but decreases the periods of the compressed tasks only at their next release time. Theorem 2 ensures that the total processor demand in any interval $[t, t + L]$ will never exceed LU_p and no deadline will be missed.

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 approach, however, strongly relies on the knowledge of the 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 Fujita et al. (1995) and Nakajima (1998) 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

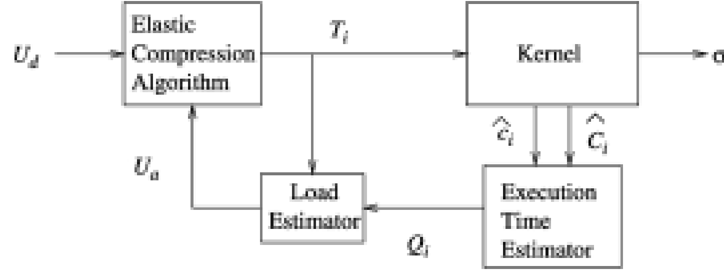

Figure 4. Feedback-based architecture for elastic rate adaptation.

optimized, the reserved budget is never used completely, and the system is underutilized.

The solution proposed in this paper uses on-line estimates 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 4 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 $U_a = \sum(Q_i/T_i)$. 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.

The advantage of using the elastic compression algorithm is that rate variations can be controlled individually for each task by means of elastic coefficients, whose values can be set to be inversely proportional to tasks' importance.

Using this approach, the application is adapted to the actual computational power of the hardware platform without any a priori knowledge on task computation times. The effectiveness of the adaptation depends on whether tasks' utilizations are computed based on worst-case or average-case estimates. If the $\hat{C}_i$ estimate is used to compute tasks' utilizations for 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) (Lipari et al., 1998), where tasks belonging to the same application can share the same resources. In the BSS case, however, the fraction of CPU 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

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

The guarantee factor k is used to balance predictability versus efficiency. If $k = 1$, the elastic algorithm results to be based on WCET estimations, so only a few deadlines can be missed when the estimated WCET $\hat{C}_i$ is smaller than the real one. In general, if no information about execution times is provided, the first $\hat{C}_i$ values will likely be underestimated and may cause some deadline miss during task's startup time.

Smaller values of k allow increasing the actual system utilization at the cost of a larger 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.

It is worth noting that the proposed approach can be used both when no *a priori* information about execution times is provided, and when an approximated estimation of the mean or maximum execution time is know. The method can also be successfully applied to task sets characterized by variable computation times, allowing task periods to vary according to execution times variations. In fact, at run time, the mean execution time estimation is computed iteratively (that is, $\hat{c}_i$ is periodically updated based on the last execution time experienced by the task) starting from an initial value c_i^0. If nothing is known about the task parameters, an arbitrary value is assumed for c_i^0. On the other hand, if an approximate estimation of the execution time is known in advance, it can be used as c_i^0, so reducing the initial transient during which $\hat{c}_i$ converges to a reasonable estimation and increasing the speed at which periods converge towards a ''stable value''.

When the proposed mechanism is applied to tasks characterized by unknown but fixed execution times, if a feasible solution exists, the stable period configuration can be reached without any deadline miss. In this case, since the mean execution time is equal to the WCET, the guarantee factor must be set to $k = 1$.

5. Experimental Results

To test the effectiveness of the adaptation algorithm, the elastic task model has been implemented on top of the HARTIK kernel (Buttazzo, 1993; Lamastra et al., 1997), 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. According to the result of Theorem 2, periods are changed at the next release time of the task whose period is decreased. If more tasks ask to decrease their period, the EM will change them, if possible, at their next release time.

In a first experiment, we tested the behavior of the elastic compression mechanism (without the on-line estimation mechanism) using the task set shown in Table 1. In this case, tasks' parameters are assumed to be known a priori, and each task is characterized by a constant execution time, reported in the C_i column. At time $t = 0$, the first three tasks start executing at their nominal period, whereas the fourth task starts at time $t_1 = 10$ seconds, so creating a dynamic workload variation.

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 tasks according to the elastic model. The actual execution rates of the tasks are shown in Figure 5. Notice that, although the first three tasks have the same elastic coefficients and the same initial utilization, their periods are changed by a different amount, because τ_3 reaches its maximum period.

In a second experiment, we tested the on-line estimation mechanism on a set of tasks characterized by fixed execution times, where each task instance was implemented as a simple dummy loop. As previously discussed, by setting $k = 1$, we verified that the estimation mechanism was able to reach a stable period configuration in one step, without causing any deadline miss.

In the next experiment, we tested the adaptive algorithm in the case of variable execution times, using four tasks started at time $t = 0$. Tasks' periods and elastic factors are the same as the ones used in the first experiment, shown in Table 1, whereas execution time varies randomly with uniform distribution between 5 and 55 ms. During this experiment, we monitored tasks' execution times and measured how the estimated values and the number of deadline misses changed as the guarantee factor k varied from 0 to 1. We note that tasks' execution times are assumed to be unknown and a value $c_i^0 = 5$ ms has been used as an initial estimate for $\hat{c}_i$.

Figure 6 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

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

Task	C_i	T_{i_0}	$T_{i_{max}}$	E_i
τ_1	30	100	500	1
τ_2	60	200	500	1
τ_3	90	300	500	1
τ_4	24	50	500	0

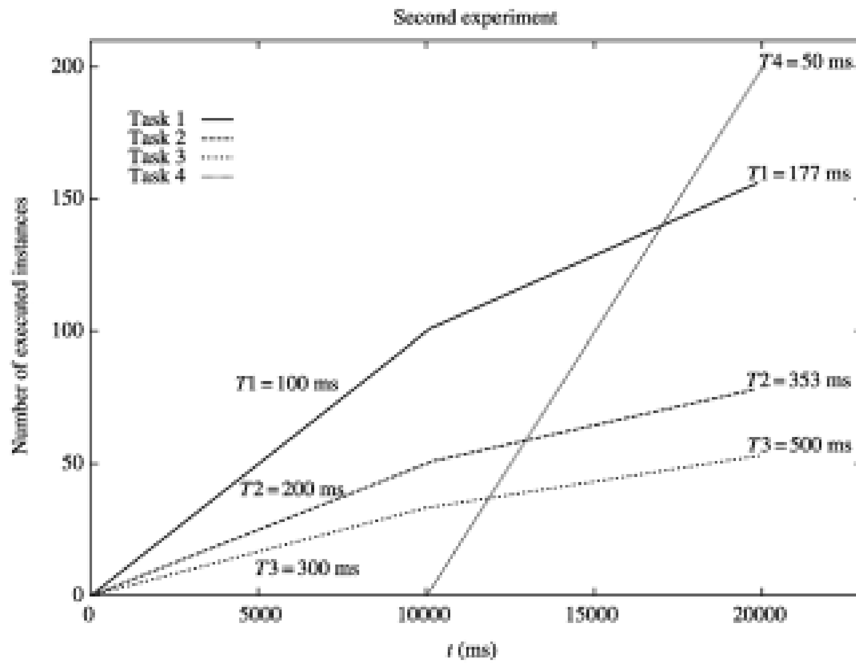

Figure 5. Dynamic task activation.

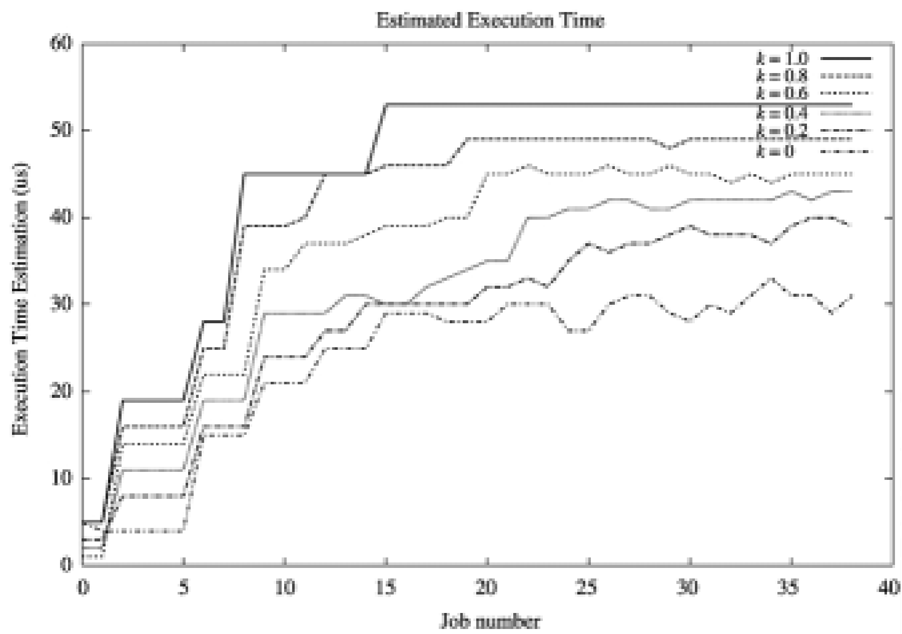

Figure 6. Estimated execution time of task τ_1 as a function of the guarantee factor k.

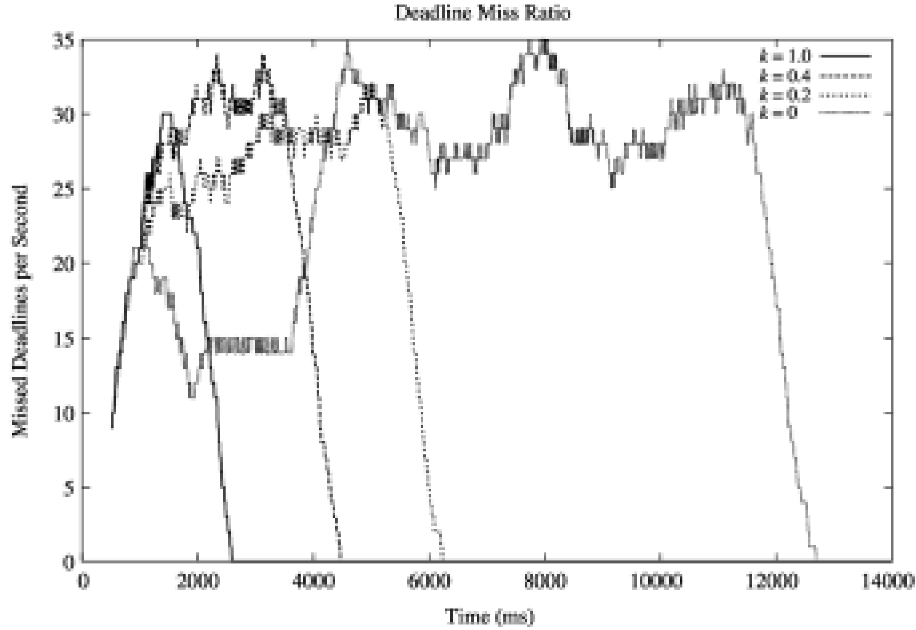

Figure 7. Missed deadlines per second as a function of the guarantee factor.

transient. 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 7, which shows 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 the value of k: for small values of the guarantee factor, the overload is recovered after a long time, while for larger values the transient is shorter. It is worth noting that, since the execution time is not constant, during the first task instances the WCET can be underestimated; for this reason, during the initial transient the deadline miss ratio is greater than 0 even for $k = 1$.

In order to speed up the transient adaptation phase without incurring in a system under utilization, a trade off on the values of the guarantee factor is required. Experimental results show that a value $k = 0.4$ gives a reasonable trade off.

Another way to reduce the number of deadline misses in the transient adaptation phase is to provide a more accurate c_i^0 value. To show how the performance of the algorithm is affected by the initial c_i^0 estimate, we repeated the previous experiment with different c_i^0 values. For example, we observed that with $c_i^0 = 25\,\text{ms}$ (which is still not very accurate) no task missed its deadline. The estimated execution times obtained in this experiment are reported in Figure 8.

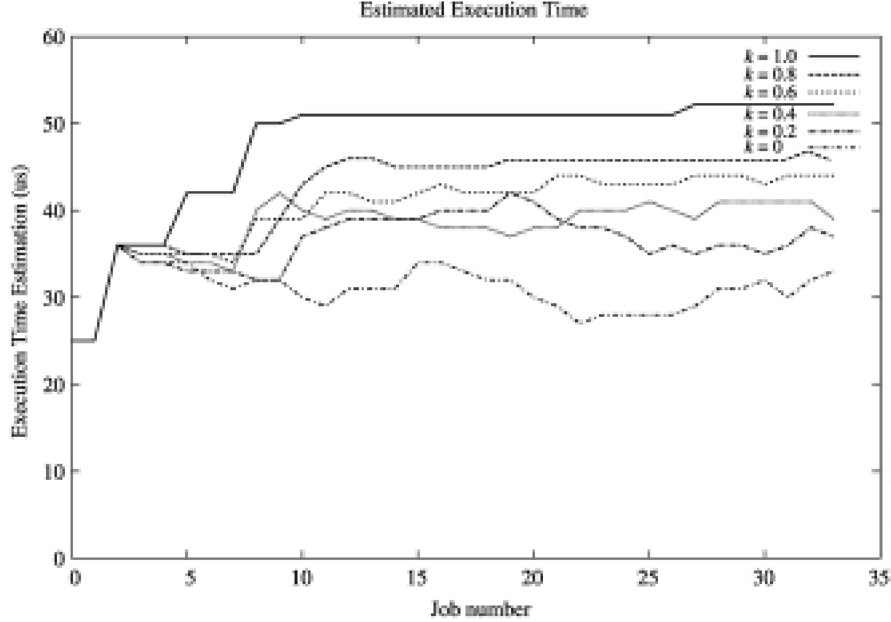

Figure 8. Estimated execution time of task τ_1 obtained using an initial estimate $c_1^0 = 25\,\text{ms}$.

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. A simple way to set

the elastic coefficients is to make them inversely proportional to task importance. Then, 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 (Buttazzo, 1993; Lamastra et al., 1997), 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.

Notes

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

References

Abdelzaher, T. F., Atkins, E. M., and Shin, K. G. 1997. QoS negotiation in real-time systems and its applications to automated flight control. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*. Montreal, Canada.

Abeni, L., and Buttazzo, G. 2000. Support for dynamic QoS in the HARTIK kernel. In *Proceedings of the 7th IEEE Real-Time Computing Systems and Applications*. Cheju Island, South Korea.

Beccari, G., Caselli, S., Reggiani, M., and Zanichelli, F. 1999. Rate modulation of soft real-time tasks in autonomous robot control systems. In *IEEE Proceedings of the 11th Euromicro Conference on Real-Time Systems*. York.

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

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

Buttazzo, G., and Abeni, L. 2000. Adaptive rate control through elastic scheduling. In *Proceedings of the 39th IEEE Conference on Decision and Control*. Sydney, Australia.

Buttazzo, G., Lipari, G., Caccamo, M., and Abeni, L. 2002. Elastic scheduling for flexible workload management. *IEEE Transactions on Computers* 51(3): 289–302.

Fujita, H., Nakajima, T., and Tezuka, H. 1995. A processor reservation system supporting dynamic QOS control. In *Second International Workshop on Real-Time Computing Systems and Applications*.

Kuo T.-W., and Mok, A. K. 1991. Load adjustment in adaptive real-time systems. In *Proceedings of the 12th IEEE Real-Time Systems Symposium*.

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 IEEE Real-Time Computing Systems and Applications*. Taipei, Taiwan.

Lee, C., Rajkumar, R., and Mercer, C. 1996. Experiences with processor reservation and dynamic QOS in real-time mach. In *Proceedings of Multimedia Japan 96*.

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

Liu, C. L., and Layland, J. W. 1973. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM* 20(1): 40–61.

Lu, C., Stankovic, J., Abdelzaher, T., Tao, G., Son, S., and Marley, M. 2000. Performance specifications and metrics for adaptive real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*. Orlando, Florida.

Nakajima, T., and Tezuka, H. 1994. A continuous media application supporting dynamic QOS control on real-time mach. In *Proceedings of the ACM Multimedia '94*.

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

Seto, D., Lehoczky, J. P., Sha, L., and Shin, K. G. 1997. On task schedulability in real-time control systems. In *Proceedings of the IEEE Real-Time Systems Symposium*.

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

Lu, C., Stankovic, J. A., Tao, G., and Son, S. H. 1999. Design and evaluation of a feedback control EDF scheduling algorithm. In *Proceedings of the 20th IEEE Real-Time Systems Symposium*. Phoenix, Arizona.