Proc. of the 15th Int. Conference on Real-Time and Network Systems (RTNS 2007), Nancy, France, March 29-30, 2007.

Reducing Delay and Jitter in Software Control Systems

Hoai Hoang
Centre for Research on Embedded Systems
Halmstad University
Halmstad, Sweden
Hoai.Hoang@ide.hh.se

Giorgio Buttazzo
Real-Time Systems Laboratory
Scuola Superiore Sant'Anna
Pisa, Italy
giorgio@sssup.it

Abstract

Software control systems may be subject to high interference caused by concurrency and resource sharing. Reducing delay and jitter in such systems is crucial for guaranteeing high performance and predictability. In this paper, we present a general approach for reducing delay and jitter by acting on task relative deadlines. The method allows the user to specify a deadline reduction factor for each task to better exploit the available slack according to specific jitter sensitivity. Experimental results confirm the effectiveness and the generality of the proposed approach with respect to other methods available in the literature.

1 Introduction

Complex software systems are often implemented as a number of concurrent tasks that interact with a given set of resources (processor, memories, peripherals, etc.). Tasks related to control activities are typically periodic, and are activated with a specific rate derived by the system's designer. Other tasks related to specific input/output devices (e.g., serial lines, data buses, networks) may be aperiodic and can be activated by interrupts or by the occurrence of particular events.

Although the activation rates of periodic tasks can be precisely enforced by the operating system through proper kernel mechanisms, the execution pattern of each task depends on several factors, including the scheduling algorithm running in the kernel, the overall system workload, the task set parameters, the interaction with the shared resources, and the interference introduced by interrupts. As a consequence, control tasks may experience variable delays and jitter that can degrade the system performance, if not properly handled.

The effects of delays and jitter on real-time control applications have been extensively studied in the literature [10, 19] and several methods have been proposed to cope with them. Marti el al. [18] presented a control technique to compensate the effect of jitter with proper control actions computed based on the temporal distance between successive samples. Cervin et al. [11] presented a method for finding an upper bound of the input-output jitter of each task by estimating the worst-case and the best-case response time under EDF scheduling, but no method is provided to reduce the jitter. Rather, the concept of *jitter margin* is introduced to simplify the analysis of control systems and guarantee their stability when certain conditions on jitter are satisfied.

Other authors proposed suitable scheduling methods for reducing the delay and jitter caused by complex intertask interference. For example, Di Natale and Stankovic [12] proposed the use of simulated annealing to find the optimal configuration of task offsets that minimizes jitter, according to some user defined cost function. Baruah et al. [4] followed a different approach to reduce both delay and jitter by reducing the relative deadline of a task, so limiting the execution interval of each job. Two methods have been illustrated for assigning shorter relative deadlines to tasks while guaranteeing the schedulability of the task set: the first method is based on task utilizations and runs in polynomial time, whereas the second one (more effective) has a pseudo-polynomial complexity since it is based on the processor demand criterion [2].

Brandt et al. [6] also addressed the problem of reducing the deadline of a periodic task, but their approach is based on the processor utilization, hence it cannot be used to find the shortest possible deadline.

Zheng et al. [20] presented a method for computing the minimum deadline of a newly arrived task, assuming the existing task set is feasibly schedulable by EDF; however, their approach is tailored for distributed applications and requires some off-line computation. When the utilization of all the tasks in the task set is high, the number of off-line computations are very large, that make this method become not efficiently, high computational complexity.

Buttazzo and Sensini [7] also presented an on-line algorithm to compute the minimum deadline to be assigned to

a new incoming task in order to guarantee feasibility under EDF. However, their approach only applies to aperiodic requests that have to be executed in a periodic environment.

Hoang et al. [16] and Balbastre et al. [5] independently proposed a method for minimizing the relative deadline of a single periodic task, while keeping the task set schedulable by EDF. Although the approach can be applied sequentially to other tasks in a given order, the deadline reduction achievable on the first task is much higher than that achievable on the other tasks in the sequence. To avoid this problem, in the same paper, Balbastre et al. also proposed a method to perform a uniform scaling of all relative deadlines. The problem with a uniform reduction, however, is that jitter and delay may not necessarily reduce as expected (and for some task they could even increase).

To allow more flexibility in controlling the delay and jitter in software controlled systems, in this paper we present a general approach for reducing task deadlines according to individual task requirements. The method allows the user to specify a deadline reduction factor for each task, to better exploit the available slack according to tasks actual requirements. The deadline reduction factor can be specified as a real number in [0,1], with the meaning that a value equal to one allows the relative deadline to be reduced up to the minimum possible value (corresponding to the task computation time), whereas a value equal to zero means no reduction.

As special cases, the method can minimize the deadline of a single task (by setting its reduction factor to 1 and the others to zero), or perform a uniform deadline rescaling in the task set (by setting all reduction factors to 1). If two tasks have the same delay/jitter requirements and need to reduce their relative deadlines as much as possible, this can simply be achieved by setting both reduction factors to 1 and all the others to zero. Note that this could not be achieved by applying a deadline minimization algorithm [16, 5] to the tasks in a given order, because the first task would steal all the available slack for itself, leaving small space for the second.

The rest of the paper is organized as follows. Section 2 presents the system model and the terminology adopted throughout the paper. Section 3 illustrates the addressed problem with some concrete examples. Section 4 describes the deadline reduction algorithm. Section 5 presents some experimental results and compares the proposed method with other deadline reduction approaches. Finally, Section 6 states our conclusions and future work.

2 Terminology and assumptions

We consider a set $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ of n periodic tasks that have to be executed on a uniprocessor system under the Earliest Deadline First (EDF) algorithm [17]. Each task τ_i consists of an infinite sequence of jobs, or task instances, having the same worst-case execution time and the same relative deadline. All tasks are fully preemptive. The following notation is used throughout the paper:

$\tau_{i,j}$ denotes the j-th job of task τ_i, (where $j = 1, 2, \ldots$), that is the j-th instance of the task execution.

$r_{i,k}$ denotes the release time of job $\tau_{i,k}$, that is the time at which the job is activated and becomes ready to execute.

$s_{i,k}$ denotes the start time of job $\tau_{i,k}$, that is the time at which the first instruction of $\tau_{i,k}$ is executed.

$f_{i,k}$ denotes the finishing time of job $\tau_{i,k}$, that is the time at which the job completes its execution.

C_i denotes the worst-case execution time of task τ_i.

T_i denotes the period of task τ_i, or the minimum inter-arrival time between successive jobs.

D_i denotes the relative deadline of task τ_i, that is, the maximum finishing time (relative to its release time) allowed for any job.

$d_{i,j}$ denotes the absolute deadline of job $\tau_{i,j}$, that is the maximum absolute time before which job $\tau_{i,j}$ must complete ($d_{i,j} = r_{i,j} + D_i$).

U_i denotes the utilization of task τ_i, that is the fraction of cpu time used by τ_i ($U_i = C_i/T_i$).

U denotes the total utilization of the task set, that is, the sum of all tasks utilizations ($U = \sum_{i=1}^{n} U_i$).

$R_{i,j}$ denotes the response time of job $\tau_{i,j}$, that is the interval between its release time and its finishing time:

$$R_{i,j} = f_{i,j} - r_{i,j}. \tag{1}$$

$IOD_{i,j}$ denotes the input-output delay of job $\tau_{i,j}$, that is the interval between its start time and its finishing time:

$$IOD_{i,j} = f_{i,k} - s_{i,k}. \tag{2}$$

RTJ_i denotes the response time jitter of a task, that is the maximum variation in the response time of its jobs:

$$RTJ_i = \max_k\{R_{i,k}\} - \min_k\{R_{i,k}\} \tag{3}$$

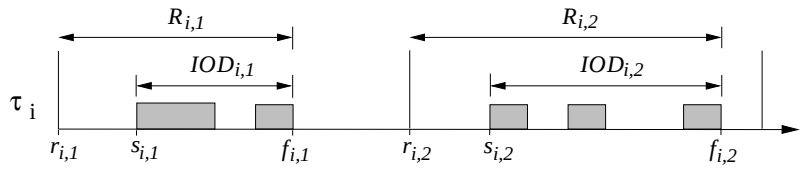

Figure 1. Example of a real-time task.

Figure 1 illustrates some of the parameters defined above.

Moreover, each task τ_i is characterized by a maximum relative deadline D_i^{max} (considered to be the nominal one) and a minimum relative deadline D_i^{min}, specified by the application designer. When not explicitly assigned, we assume $D_i^{max} = T_i$ and $D_i^{min} = C_i$.

3 Problem statement

The method proposed in this work to reduce delay and jitter in periodic tasks requires the application designer to specify an additional task parameter, called the *deadline reduction factor*, δ_i, which is a real number in [0,1]. A value $\delta_i = 1$ indicates that task τ_i is very sensitive to delay and jitter, hence its relative deadline is allowed to be reduced up to its minimum possible value (D_i^{min}). A value $\delta_i = 0$ indicates that task τ_i is not sensitive to delay and jitter, so its relative deadline does not need to be modified. In general, we assume that the sensitivity of τ_i to delay and jitter is proportional to δ_i.

Once all deadline reduction factors have been specified according to delay and jitter sensitivity, the problem we want to solve is to shorten all deadlines as much as possible to respect the proportions dictated by the reduction factors, while keeping the task set feasible.

Note that task specific jitter coefficients have also been defined by Baruah et al. [4] (they were denoted by ϕ_i and called jitter tolerance factors). In that work, however, the objective was to minimize the weighted jitter of the task set, defined as

$$\text{WtdJitter}(\mathcal{T}) = \max_i \left\{ \frac{RTJ_i}{\phi_i} \right\},$$

rather than reducing deadlines proportionally to sensitivity, as done in this paper.

To better motivate the proposed approach, we now illustrate an example that shows the advantage of specifying individual reduction factors.

3.1 A motivating example

Consider a set of three periodic tasks with periods $T_1 = 6$, $T_2 = 9$, $T_3 = 12$, and computation times $C_1 = 1$, $C_2 = 2$, $C_3 = 5$. Suppose that τ_1 and τ_2 are control tasks sensitive to delay and jitter, whereas τ_3 is not and can be executed anywhere within its period. Assuming $D_i = D_i^{max} = T_i$ for each task, the schedule produced by EDF is shown in Figure 2. The response time jitters of the tasks are $RTJ_1 = 2$, $RTJ_1 = 3$, and $RTJ_1 = 2$.

Now observe that, for this task set, the uniform scaling algorithm proposed by Balbastre et al. [5] does not produce any change in the schedule, so it cannot reduce any jitter. For this particular case, in fact, the maximum common reduction factor that guarantees a feasible schedule is 1/3, meaning that for each task we can set $D_i = (2/3)T_i$. As shown in Figure 3, however, the schedule produced by EDF with such deadlines is exactly the same as that shown in Figure 2.

Also notice that minimizing the deadline of a single task (using the algorithm proposed by Hoang et al. [16] or by Balbastre et al. [5]) may not necessarily have the desired effect on the other jitter sensitive tasks. For example, as depicted in Figure 4, minimizing D_2 the jitter of τ_2 becomes zero, but the jitter of τ_1 cannot be reduced below 2 (even if D_1 is minimized after D_2).

In this case, a better solution to reduce the delay and jitter of τ_1 and τ_2 is to reduce the deadlines of both tasks simultaneously, leaving D_3 unchanged. As an example, assuming $\delta_1 = \delta_2 = 1$, the maximum common reduction factor that can be applied to both tasks to keep the task set feasible is 2/3, meaning that we can set $D_1 = T_1/3$, $D_2 = T_2/3$, and $D_3 = T_3$. Figure 5 shows the schedule produced by EDF with such deadlines.

The next section describes the algorithm that computes the new feasible deadlines according to the specified reduction factors δ_i.

4 The algorithm

Before describing the algorithm, it is worth observing that, for the feasibility constraint, the actual deadline reduction will be less than or equal to the one specified by the reduction factor, that is

$$\frac{D_i^{max} - D_i}{D_i^{max} - D_i^{min}} \leq \delta_i.$$

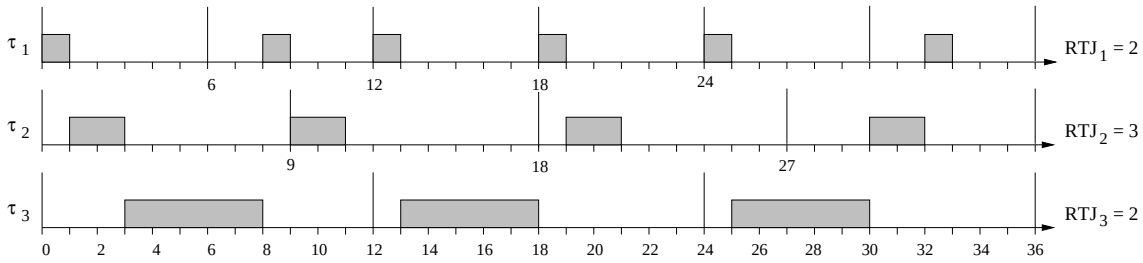

Figure 2. EDF schedule with relative deadlines equal to periods.

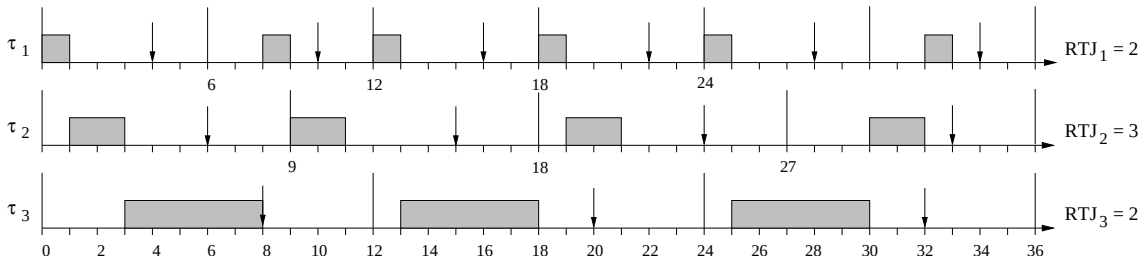

Figure 3. EDF schedule with uniformly scaled deadlines.

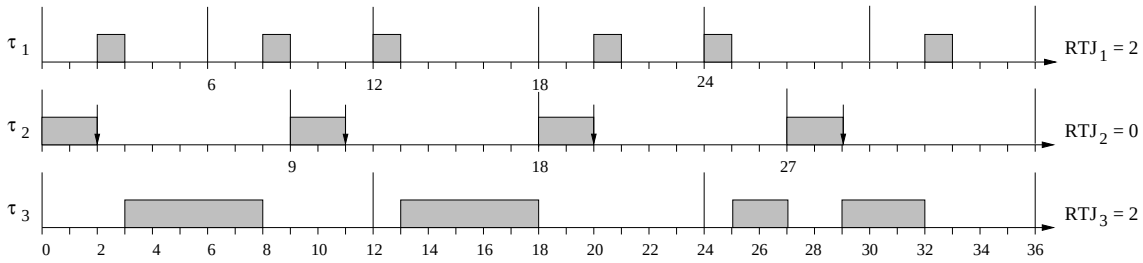

Figure 4. EDF schedule when only τ_2 deadline is minimized.

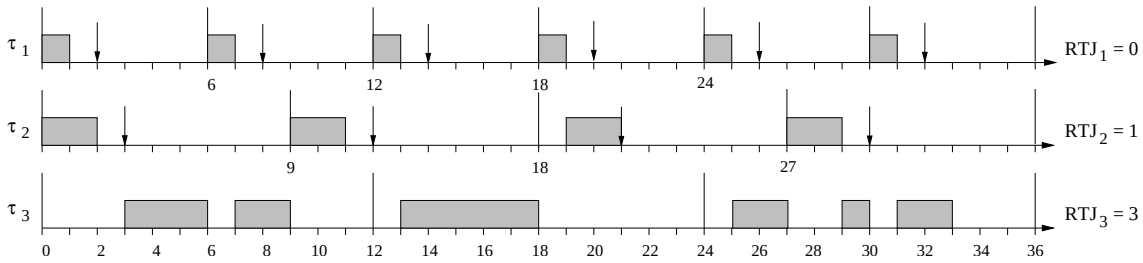

Figure 5. EDF schedule when deadlines of both τ_1 and τ_2 are reduced.

However, to respect the proportions specified by the reduction factors, we must compute the new deadlines in such a way that $\forall i, j = 1, \ldots, n$ (and δ_i, $\delta_j \neq 0$) we have

$$\frac{D_i^{max} - D_i}{D_i^{max} - D_i^{min}}/\delta_i \;=\; \frac{D_j^{max} - D_j}{D_j^{max} - D_j^{min}}/\delta_j.$$

This means that

$$\forall i = 1, \ldots, n \quad \frac{D_i^{max} - D_i}{D_i^{max} - D_i^{min}}/\delta_i = \alpha$$

where α is a constant less than or equal to one. Hence, the problem consists in finding the greatest value of α that keeps the task set feasible, where deadlines are computed as

$$D_i = D_i^{max} - \alpha\delta_i(D_i^{max} - D_i^{min}) \qquad (4)$$

The highest value of α that guarantees feasibility can be found by binary search.

The search algorithm assumes that the task set $\mathcal{T}$ is feasible for $\alpha = 0$ (that is, when all tasks are scheduled with the maximum deadlines), and starts by trying feasibility with $\alpha = 1$ (that is, with all tasks having their minimum deadlines). If $\mathcal{T}$ is found feasible with $\alpha = 1$, then the algorithm exits with the best solution, otherwise the binary search is started.

The feasibility test as a function of α can be performed using the function reported in Figure 6, where all relative deadlines are first computed according to Equation (4), and then the test is performed using the Processor Demand Criterion [2, 3]. In particular, the $Processor\_demand\_test(\mathcal{T})$ function returns 1 if the task set $\mathcal{T}$ is feasible, 0 otherwise.

```
Feasible(T, α)

    for i = 0 to n
        D_i = D_i^max − α δ_i (D_i^max − D_i^min);
    end for

    F = Processor_demand_test(T);

    return (F);
end
```

Figure 6. Feasibility test as a function of α.

The binary search algorithm to find the highest value of α is reported in Figure 7. Note that, besides the task set parameters, the algorithm requires a value ε, needed to bound the complexity of the search and stop the algorithm when the search interval ($\Delta = \alpha_{max} - \alpha_{min}$) becomes smaller than a given error.

```
Best_alpha(T, ε)
    α_max = 1;
    α_min = 0;
    Δ = α_max − α_min;

    if Feasible(T, α_max) then return(α_max);

    while (Δ > ε) do
        α = (α_max + α_min)/2;

        if Feasible(T, α) then α_min = α;
        else α_max = α;

        Δ = α_max − α_min;
    end while

    return(α_min);

end
```

Figure 7. Binary search algorithm for finding the highest α that guarantees feasibility.

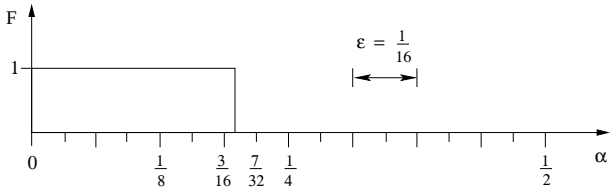

Figure 8. Search values for $\varepsilon = 1/16$.

For example, for the case illustrated in Figure 8, if $\varepsilon = 1/16$, the algorithm will try six values (1, 1/2, 1/4, 1/8, 3/16, and 7/32) and stops by returning the last feasible value, that is $\alpha = 3/16$. In general, the complexity of the algorithm is logarithmic with respect to ε, and the number of steps to find the best α is given by

$$N = 2 - \log_2 \varepsilon.$$

For the given example, $\log_2(1/16) = -4$, so we have $N = 6$. Once the highest feasible α is found, the task deadlines are reduced according to Equation (4), which takes into account the individual reduction factors.

5 Experimental results

This section describes a set of simulation experiments that have been conducted to evaluate the effectiveness of the proposed algorithm to reduce delay and jitter of specific tasks according to given reduction factors. For special cases, the method is also compared with the algorithm that uniformly scales all deadlines [5] and with the algorithm that minimizes the relative deadline of a single task [16, 5].

We have investigated different application scenarios, generated through synthetic task sets with random parameters within given ranges and distributions. To generate a feasible task set of n periodic tasks with given utilization $U_d < 1$, we first generated n random utilizations uniformly distributed in (0,1) and then normalized them to have

$$\sum_{i=1}^{n} U_i = U_d.$$

Then, we generated n computation times as random variables uniformly distributed in $[C_{min}, C_{max}]$ (with $C_{min} = 5$ and $C_{max} = 30$) and then calculated the period of each task as

$$T_i = \frac{C_i}{U_i}.$$

For each task τ_i, D_i^{max} has been set equal to T_i and D_i^{min} has been set equal to C_i.

For each generated task set $\mathcal{T}$, we measured the maximum response time of each task ($R_i = \max_k\{R_{i,k}\}$) and the maximum response time jitter ($RTJ_i = \max_k\{RTJ_{i,k}\}$) caused by EDF under three different deadline setting:

1. **Plain**: all tasks run with their maximum deadlines: $D_i = D_i^{max}$;
2. **Scaled**: all deadlines are uniformly scaled by the same factor according to the algorithm proposed by Balbastre et al. [5];
3. **New**: task deadlines are computed by the proposed algorithm according to given reduction factors.

In the first experiment, a simulation has been carried out with a set of 10 periodic tasks, having fixed utilization $U = 0.9$. The proposed algorithm has been applied to a group of four tasks with the same reduction factor ($\delta_i = 1$), while leaving the remaining tasks with their original deadlines ($\delta_i = 0$). In particular, the four tasks with the longest periods (from τ_7 to τ_{10}) have been selected for reduction. The worst-case response time and the response time jitter (RTJ) have been measured for each task and then averaged over 1000 simulation runs. A value $\varepsilon = 10^{-4}$ was used to find the best α.

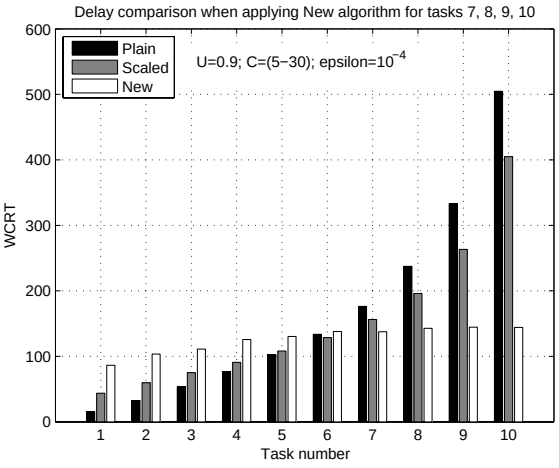

Figure 9. Worst-case response times when applying the proposed algorithm to task 7, 8, 9 and 10.

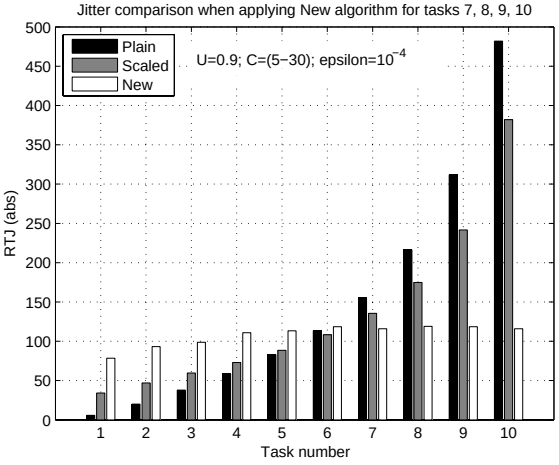

Figure 10. Response Time Jitter when applying the proposed algorithm to task 7, 8, 9 and 10.

Figures 9 and 10 respectively show the response time and jitter achieved for each individual task in this experiment. Note that, the X-axis shows the task identification number, where tasks are ordered by increasing period, so that task number 1 is the one with the shortest period. In particular, Figure 11 reports the jitter experienced by each task under the proposed algorithm, showing the 95% confidence interval around each average value.

As expected, the results show that restricting deadline reduction only to a subset of sensitive tasks allows better control of delay and jitter.

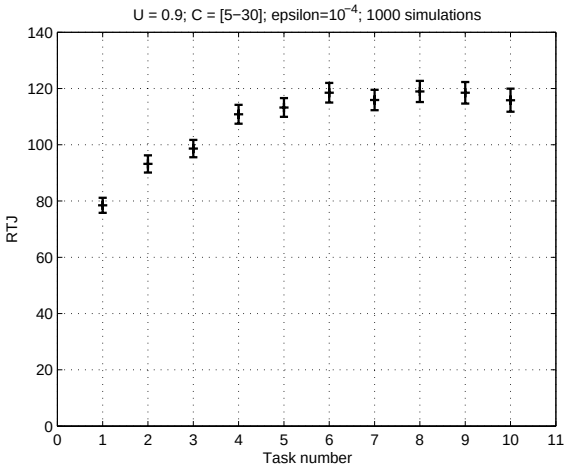

Figure 11. Confidence intervals (95%) of the jitter measures achieved in the first experiment under the proposed algorithm.

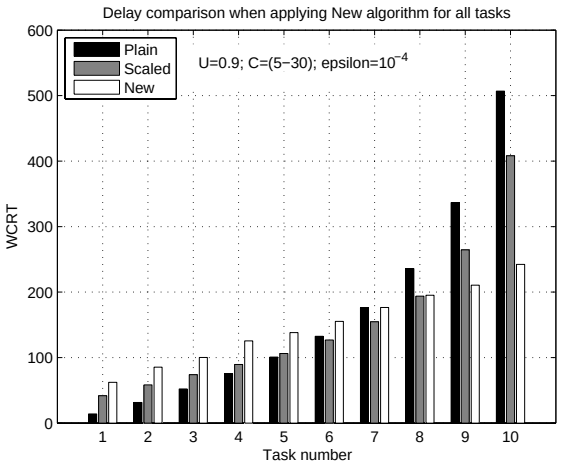

Figure 12. Worst-case response times when applying the proposed algorithm uniformly to all the tasks.

Also note that reducing all task deadlines by the same scaling factor (as done by the Scaled algorithm) has not a significant effect on jitter reduction with respect to the Plain scenario (where all deadlines are equal to the periods), thus justifying the need for adopting selective reduction factors.

A second experiment has been carried out to compare our algorithm against the uniform scaling algorithm [5] when all relative deadlines are uniformly scaled by the same reduction factor ($\delta_i = 1$ for $i = 1, \ldots, 10$). We have applied both methods on the same task set taken for the first experiment, using the same value of ε (10^{-4}).

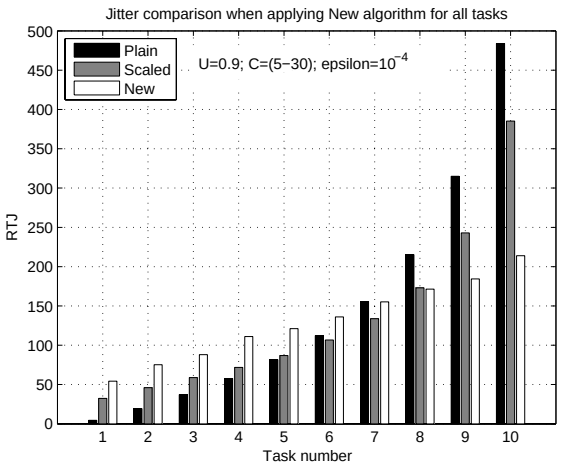

Figure 13. Response Time Jitter when applying the proposed algorithm uniformly to all the tasks.

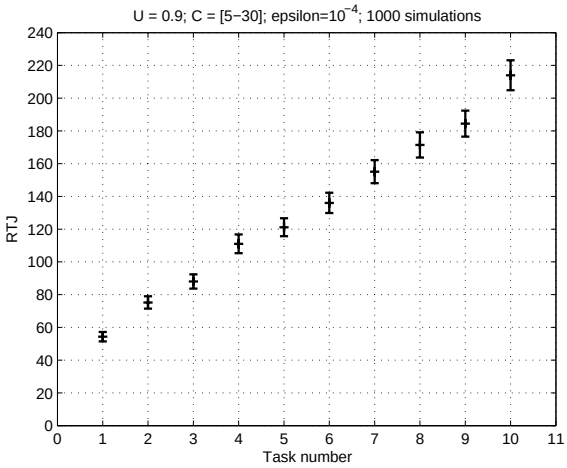

Figure 14. Confidence intervals (95%) of the jitter measures achieved in the second experiment under the proposed algorithm.

As shown in Figures 12 and 13, our algorithm performs almost the same as the uniform scaling algorithm for tasks with short periods, whereas it performs slightly better for tasks with longer periods. All values plotted in the graphs represent the average over 1000 simulations, and the 95% confidential intervals on the average jitter achieved under the proposed algorithm are shown in Figure 14.

Finally, the performance of the proposed method has also been compared against the plain EDF scheduler and the scaled method when the task set has a lower utilization equal to $U = 0.6$.

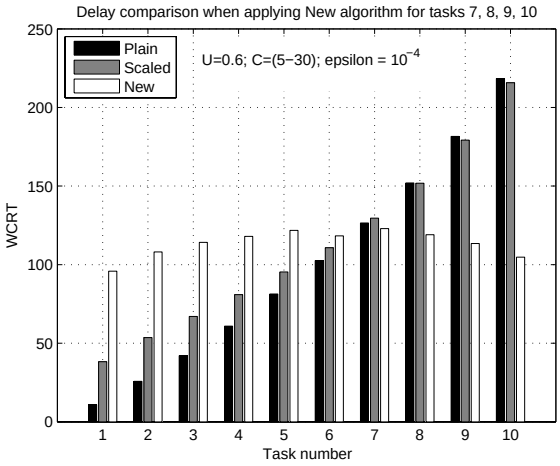

Figure 15. Worst-case Response Time when applying the proposed algorithm to task 7, 8, 9 and 10 ($U = 0.6$).

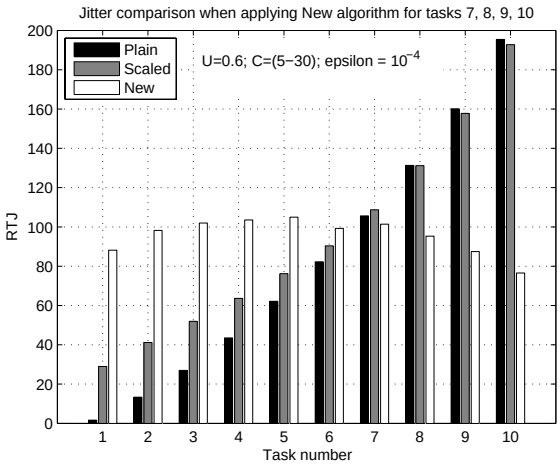

Figure 16. Response Time Jitter when applying the proposed algorithm to task 7, 8, 9 and 10 ($U = 0.6$).

Figures 15 and 16 show the response time and jitter of each tasks when applying the new algorithm to tasks 7, 8, 9, 10 only, whereas Figures 17 and 18 show the response time and jitter of the tasks when applying the new algorithm uniformly to all the tasks.

All the experiments confirm that the proposed approach is able to reduce both delay and control jitter of specific control tasks according to desired scaling factors and under different load conditions. With respect to the plain EDF and uniform scaling algorithm, the proposed algorithm is more effective when the task set has a high utilization.

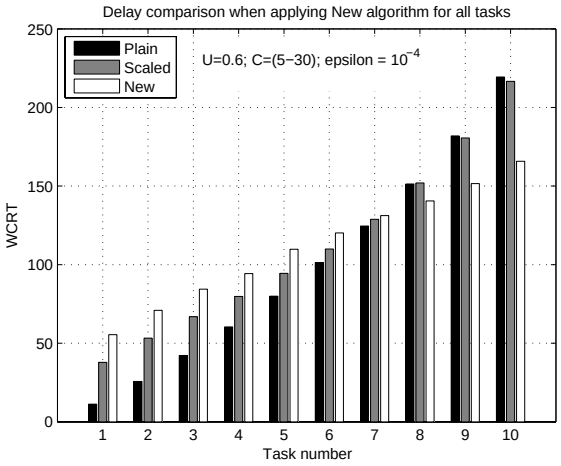

Figure 17. Worst-case Response Time when applying the proposed algorithm uniformly to all the tasks ($U = 0.6$).

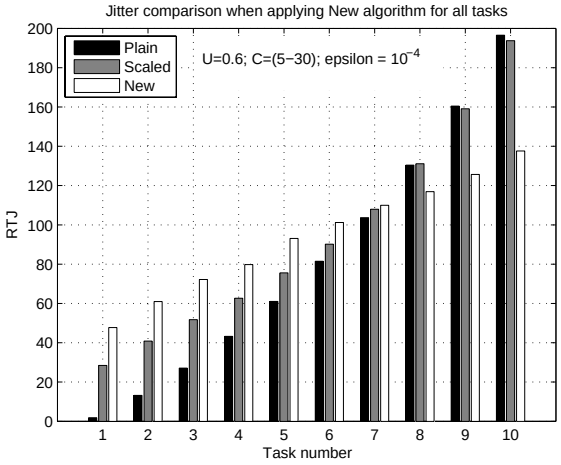

Figure 18. Response Time Jitter when applying the proposed algorithm uniformly to all the tasks ($U = 0.6$).

6 Conclusions

In this paper we presented a method for reducing the relative deadlines of a set of periodic tasks according to given reduction factors, $\delta_i \in [0, 1]$, denoting task sensitivity to jitter and delay. A value $\delta_i = 1$ denotes high sensitivity to delay and jitter, indicating that the task relative deadline can be reduced as much as possible, up to the minimum possible value which guarantees the task schedulability, whereas a value $\delta_i = 0$ denotes no sensitivity, indicating that the task relative deadline does not need to be modified.

Note that shortening the relative deadline decreases the admissible execution interval of a task, affecting both its response time and jitter.

Moreover, the proposed approach generalizes two other methods presented in the real-time literature for jitter reduction: the deadline minimization algorithm, independently developed by Hoang et al. [16] and by Balbastre et al. [5], and the uniform deadline scaling method, proposed by Balbastre et al. in the same paper. In fact, using the proposed approach, the relative deadline of a single periodic task τ_k can be minimized simply by setting $\delta_k = 1$ and all other reduction factors to zero. Similarly, a uniform reduction of all task deadlines can simply be achieved by setting all reduction factors to 1.

Experimental results confirm the effectiveness of the proposed approach, showing that deadline reductions are more significant when acting only on a subset of selected tasks.

As a future work, we plan to investigate the issue also under fixed priorities. Here, the deadline reduction algorithm cannot be trivially extended, because changing relative deadlines may also affect the priority order, and hence the feasibility test.

Acknowledgement

This work has been partly funded by the CERES research profile grant from The Knowledge Foundation.

References

[1] K. J. Astrom and B. Wittenmark, *Computer Controller Systems: Theory and Design*, Prentice-Hall, 1984.

[2] S. K. Baruah, R. R. Howell, and L. E. Rosier, "Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic Real-Time Tasks on One Processor," *Real-Time Systems*, 2, 1990.

[3] S. K. Baruah, A. K. Mok, and L. E. Rosier, "Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor," *Proc. of the 11th IEEE Real-Time Systems Symposium*, Orlando, FL, USA, Dec. 1990.

[4] S. Baruah, G. Buttazzo, S. Gorinsky, and G. Lipari, "Scheduling Periodic Task Systems to Minimize Output Jitter," *Proc. of the 6th IEEE International Conference on Real-Time Computing Systems and Applications*, Hong Kong, Dec. 1999.

[5] P. Balbastre, I. Ripoll, and A. Crespo, "Optimal Deadline Assignment for Periodic Real-Time Tasks in Dynamic Priority Systems," *Proc. of the 18th Euromicro Conference on Real-Time Systems (ECRTS 2004)*, Dresden, Germany, July 5-7, 2006.

[6] S. A. Brandt, S. A. Banachowski, C. Lin, and T. Bisson: "Dynamic Integrated Scheduling of Hard Real-Time, Soft Real-Time and Non-Real-Time Processes," *Proc. of the 24th IEEE Real-Time Systems Symposium*, Cancun, Mexico, USA, December 2003.

[7] G. Buttazzo and F. Sensini, "Optimal Deadline Assigment for Scheduling Soft Aperiodic Task in Hard Real-Time Environments," *IEEE Transactions on Computers*, Vol. 48, No. 10, Oct. 1999.

[8] G. Buttazzo, M. Velasco, P. Marti, and G. Fohler, "Managing Quality-of-Control Performance Under Overload Conditions," *Proc. of the 16th Euromicro Conference on Real-Time Systems (ECRTS 2004)* , Catania, Italy, July 2004.

[9] G. Buttazzo. Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications - Second Edition, Springer, 2005.

[10] A. Cervin, "Integrated Control and Real-Time Scheduling," Doctoral Dissertation, ISRN LUTFD2/TFRT-1065-SE, Department of Automatic Control, Lund, Sweden, April 2003.

[11] A. Cervin, B. Lincoln, J. Eker, K.-E. Arzn, and G. C. Buttazzo, "The Jitter Margin and Its Application in the Design of Real-Time Control Systems," *Proceedings of the 10th International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA 2004)*, Gothenburg, Sweden, August 25-27, 2004.

[12] M. Di Natale and J. Stankovic, "Scheduling Distributed Real-Time Tasks with Minimum Jitter," IEEE Transactions on Computers, Vol. 49, No. 4, pp. 303-316, 2000.

[13] J. A. Stankovic, M. Spuri, K. Ramamritham, G. C. Buttazzo, *Deadline Scheduling for Real-Time Systems - EDF and Related Algorithms*. Kluwer Academic Publishers, 1998.

[14] D. Ferrari and D. C. Verma, "A Scheme for Real-Time Channel Establishement in Wide-Area Networks," *IEEE Journal of Selected Areas in Communications*, Vol. 8, No. 3, pp. 368-379, Apr. 1990.

[15] P. Gai, L. Abeni, M. Giorgi, and G. Buttazzo, "A New Kernel Approach for Modular Real-Time systems Development," *Proc. of the 13th IEEE Euromicro Conference on Real-Time Systems*, Delft, Netherlands, June 2001.

[16] H. Hoang, G. Buttazzo, M. Jonsson, and S. Karlsson, "Computing the Minimum EDF Feasible Deadline in Periodic Systems," *Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, Sydney, Australia, August 2006.

[17] C. L. Liu and J. W. Layland, "Scheduling algorithms for Multipprogramming in Hard Real-Time traffic environment," *Journal of the Association for Computing Machinery*, Vol. 20, No. 1, Jan. 1973.

[18] P. Marti, G. Fohler, K. Ramamritham, and J.M. Fuertes, "Jitter Compensation for Real-time Control Systems," *Proc. of the 22rd IEEE Real-Time System Symposium*, London, UK, December 2001.

[19] P. Marti, "Analysis and Design of Real-Time Control Systems with Varying Control Timing Constraints," PhD Thesis, Department of Automatic Control, Technical University of Catalonia, Barcelona, Spain, July 2002.

[20] Q. Zheng and K. G. Shin, "On the Ability of Establishing Real-Time Chennals in Point-to-Point Packet-Switched Networks," *IEEE Transactions on Communications*, Vol. 42, No. 2/3/4, Feb./March/Apr. 1994.