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

Comparative Assessment and Evaluation of Jitter Control Methods

Giorgio Buttazzo
Scuola Superiore S. Anna
Pisa, Italy
giorgio@sssup.it

Anton Cervin
Lund University
Lund, Sweden
anton@control.lth.se

Abstract

Most control systems involve the execution of periodic activities, which are automatically activated by the operating system at the specified rates. When the application consists of many concurrent tasks, each control activity may experience delay and jitter, which depend on several factors, including the scheduling algorithm running in the kernel, the overall workload, the task parameters, and the task interactions. If not properly taken into account, delays and jitter may degrade the performance of the system and even jeopardize its stability.

In this paper, we evaluate three methodologies for reducing the jitter in control tasks: the first one consists of forcing the execution of inputs and outputs at the period boundaries, so trading jitter with delay; the second method reduces jitter and delay by assigning tasks shorter deadlines; whereas, the third method relies on non preemptive execution. We compare these techniques by illustrating examples, pointing out advantages and disadvantages, and evaluating their effects in control applications by simulation. It is found that the deadline advancement method gives the better control performance for most configurations.

1 Introduction

Real-time control applications typically involve the execution of periodic activities to perform data sampling, sensory processing, control, action planning, and actuation. Although not strictly necessary, periodic execution simplifies the design of control algorithms and allows using standard control theory to guarantee system stability and performance requirements. In a computer controlled system, periodic activities are enforced by the operating system, which automatically activates each control task at the specified rate.

Nevertheless, when the system involves the execution of many concurrent tasks, each activity may experience delay and jitter, which depend on several factors, including the scheduling algorithm running in the kernel, the overall workload, the task parameters, and the task interactions. If not properly taken into account, delay and jitter may degrade the performance of the system and even jeopardize its stability [18, 20, 12].

The problem of jitter in real-time control applications has received increased attention during the last decade and several techniques have been proposed to cope with it. Nilsson [24] analyzed the stability and performance of real-time control systems with random delays and derived an optimal, jitter-compensating controller. Martí *et al.* [23] proposed a compensation technique for controllers based on the pole placement design method. Di Natale and Stankovic [13] proposed the use of simulated annealing to find the optimal configuration of task offsets that minimizes jitter, according to some user defined cost function. Cervin *et al.* [10] 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 [22], but no method is provided to reduce the jitter by shortening task deadlines. 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.

Another way of reducing jitter and delay is to limit the execution interval of each task by setting a suitable relative deadline. Working on this line, Baruah at al. [5] proposed two methods for assigning shorter relative deadlines to tasks and guaranteeing the schedulability of the task set. Shin *et al.* [25] presented a method for computing the minimum deadline of a newly arrived task, assuming the existing task set is feasibly schedulable by EDF. Buttazzo and Sensini [8] 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.

Another common practice to reduce jitter in control applications is to separate each control task into three distinct subtasks performing data input, processing, and control output [11]. The input-output jitter is reduced by postponing the input-output subtasks to some later point in time, so trading jitter with delay. While it has been shown that task splitting in general may improve the schedulability of a task set [16], the method also introduces a number of problems that have not been deeply investigated in the real-

time literature. Finally, another possible technique to reduce scheduling-induced jitter is to execute the application in a non preemptive fashion.

In general, none of the techniques above can reduce the jitter all the way down to zero. There will always be some variability in the execution time of the (sub)tasks themselves, and there might be additional jitter caused by poor timer resolution, tick scheduling, non-preemptable kernel calls, etc. In this paper, however, we will focus on scheduling-induced jitter, and we will consider standard control algorithms that can be assumed to have near-constant computation times.

Although the techniques above have often been used in control applications, a comprehensive assessment and a comparative evaluation of their effect on control performance is still missing. In this paper, we provide a systematic description of these techniques, illustrating examples and pointing out possible problems introduced by each method. Then, we discuss a number of simulation experiments aimed at evaluating the effect of these approaches in control applications.

The rest of the paper is organized as follows. Section 2 presents the system model and the basic assumptions. Section 3 provides a definition of jitter and identifies the possible causes. Section 4 introduces the three approaches considered in this work for jitter reduction and discusses pros and cons of the methods. Section 5 describes some experimental results carried out to evaluate the three approaches. Section 6 states our conclusions and future work.

2 Terminology and Assumptions

We consider a set $\mathscr{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ of periodic tasks that have to be executed on a uniprocessor system. Each periodic task τ_i consists of an infinite sequence of task instances, or jobs, having the same worst-case execution time (WCET), the same relative deadline, and the same interarrival period. The following notation is used throughout the paper:

$\tau_{i,k}$ denotes the k-th job of task τ_i, with $k \in \mathscr{N}$.

C_i denotes the worst-case execution time (WCET) of task τ_i, that is, the WCET of each job of τ_i.

T_i denotes the period of task τ_i, or worst-case minimum inter-arrival time.

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

$r_{i,k}$ denotes the release time of job $\tau_{i,k}$. If the first job is released at time $r_{i,1} = \Phi_i$, also referred to as the task phase or the offset, the generic k-th job is released at time $r_{i,k} = \Phi_i + (k-1)T_i$.

$s_{i,k}$ denotes the start time of job $\tau_{i,k}$.

$f_{i,k}$ denotes the finishing time of job $\tau_{i,k}$.

$R_{i,k}$ denotes the response time of job $\tau_{i,k}$, that is, the difference of its finishing time and its release time ($R_{i,k} = f_{i,k} - r_{i,k}$).

$INL_{i,k}$ denotes the input latency of a control job $\tau_{i,k}$, that is, the interval between the release of the task and the reading of the input signal. If the input is performed at the beginning of the job, then $INL_{i,k} = s_{i,k} - r_{i,k}$.

$IOL_{i,k}$ denotes the input-output latency of a control job $\tau_{i,k}$, that is, the interval between the reading of the input and the writing of the output. If the input is performed at the beginning of the job and the output at the end, then $IOL_{i,k} = f_{i,k} - s_{i,k}$.

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

We assume all tasks are fully preemptive, although some of them can be executed in a non preemptive fashion. Moreover, we allow relative deadlines to be less than or equal to periods.

3 Jitter characterization

Due to the presence of other concurrent tasks that compete for the processor, a task may evolve in different ways from instance to instance; that is, the instructions that compose a job can be executed at different times, relative to the release time, within different jobs. The maximum time variation (relative to the release time) in the occurrence of a particular event in different instances of a task defines the jitter for that event. The jitter of an event of a task τ_i is said to be *relative* if the variation refers to two consecutive instances of τ_i, and *absolute* if it is computed as the maximum variation with respect to all the instances.

For example, the response time jitter (RTJ) of a task is the maximum time variation between the response times of the various jobs. If $R_{i,k}$ denotes the response time of the k^{th} job of task τ_i, then the relative response time jitter of task τ_i is defined as

$$\mathrm{RTJ}_i^{rel} = \max_k |R_{i,k+1} - R_{i,k}| \tag{1}$$

whereas the absolute response time jitter of task τ_i is defined as

$$\mathrm{RTJ}_i^{abs} = \max_k R_{i,k} - \min_k R_{i,k}. \tag{2}$$

Of particular interest for control applications are the time instants at which inputs are read and outputs are written. An overview of control task timing is given in Figure 1. The time interval between the release of the task and the reading of the input signal $y_i(t)$ is called the input latency and is denoted by $\mathrm{INL}_{i,k}$. The interval between the reading of the

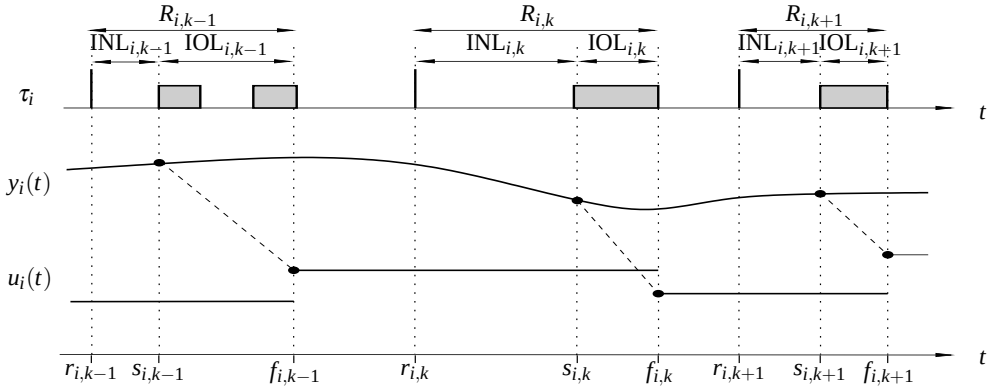

Figure 1. Control task timing.

input and the writing of the output is called the input-output latency and is denoted by $\mathrm{IOL}_{i,k}$.

In the figure, it is assumed that the input signal $y_i(t)$ is sampled at the start time $s_{i,k}$, and that the control signal $u_i(t)$ is updated at the finishing time $f_{i,k}$. Under this assumption, the response-time jitter is equivalent to the output jitter.

Similarly, the input jitter (INJ) of a task is the maximum time variation of the instants at which the input is performed in the various jobs. Thus, the relative input jitter of task τ_i can be defined as

$$\mathrm{INJ}_i^{rel} = \max_k |\mathrm{INL}_{i,k+1} - \mathrm{INL}_{i,k}| \tag{3}$$

whereas the absolute input jitter of task τ_i is defined as

$$\mathrm{INJ}_i^{abs} = \max_k \mathrm{INL}_{i,k} - \min_k \mathrm{INL}_{i,k}. \tag{4}$$

Another type of jitter of interest in control applications is the input-output jitter (IOJ), that is, the maximum time variation of the interval between the reading of the input and the writing of the output. The relative input-output jitter of task τ_i is defined as

$$\mathrm{IOJ}_i^{rel} = \max_k |\mathrm{IOL}_{i,k+1} - \mathrm{IOL}_{i,k}| \tag{5}$$

whereas the absolute input-output jitter of task τ_i is defined as

$$\mathrm{IOJ}_i^{abs} = \max_k \mathrm{IOL}_{i,k} - \min_k \mathrm{IOL}_{i,k}. \tag{6}$$

The jitter experienced by a task depends on several factors, including the scheduling algorithm running in the kernel, the overall workload, the task parameters, and the task interactions through shared resources.

The example shown in Figure 2 illustrates how the jitter is affected by the scheduling algorithm. The task set consists of three periodic tasks with computation times $C_1 = 2$, $C_2 = 3$, $C_3 = 2$, and periods $T_1 = 6$, $T_2 = 8$, $T_3 = 12$. Notice that, if the task set is scheduled by the Rate Monotonic (RM) algorithm (Figure 2a), the three tasks experience a response time jitter (both relative and absolute) equal to 0, 2, and 8, respectively. Under the Earliest Deadline First (EDF) algorithm (Figure 2b), the same tasks experience a response time jitter (both relative and absolute) equal to 1, 2, and 3, respectively. Also the input-output jitter changes with the scheduling algorithm; in fact, under RM, the three tasks have an input-output jitter (both relative and absolute) equal to 0, 2, 5, respectively, whereas under EDF the input-output jitter is zero for all the tasks. In general, EDF significantly reduces the jitter experienced by tasks with long period by slightly increasing the one of tasks with shorter period. A more detailed evaluation of these two scheduling algorithms for different scenarios can be found in [9].

Using the same example shown in Figure 2, it is easy to see that, if task τ_3 has a shorter computation time (e.g., $C_3 = 1$), the response time jitter experienced by τ_3 under RM decreases from 8 to 3, while the input-output jitter becomes 0. Hence, the jitter is heavily dependent on the workload, especially for fixed priority assignments. Also note the dependency on the task set parameters. In fact, by slightly increasing the period T_3, under EDF, the first job of τ_3 would be preempted by the second job of τ_1, so the jitter of τ_3 would increase significantly. It is also clear that the task offsets have an influence on the jitter. Finally, more complex interferences may occur in the presence of shared resources, which can introduce additional blocking times that may increase the jitter.

4 Jitter control methods

We now introduce three common techniques typically adopted to reduce jitter in real-time control systems. They are described in the following sections.

4.1 Reducing jitter by task splitting

The first approach exploits the fact that most control activities have a common structure, including an input phase, where sensory data acquisition is performed, a processing phase, where the control law is computed, and an output

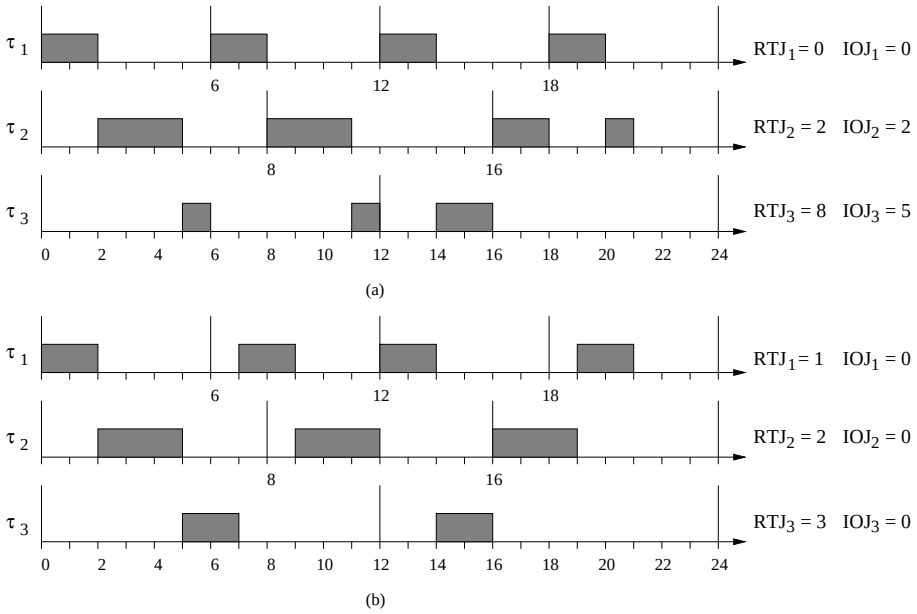

Figure 2. Jitter under RM (a) and EDF (b).

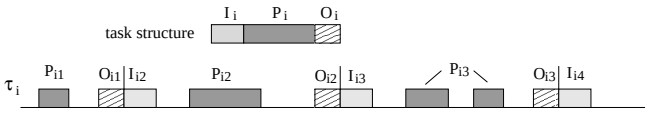

Figure 3. Task input and output can be forced to occur at period boundaries.

phase, where the proper control actions are sent to the controlled plant. Hence, each control task τ_i is divided into three subtasks: input, processing, and output. The computation times of these three subtasks will be denoted as I_i, P_i, and O_i, respectively, and the total computation time is given by $C_i = I_i + P_i + O_i$.

The key idea behind this method is to force the input subtask to be executed at the beginning of the period and the output subtask to be executed at the end, as illustrated in Figure 3.

The input and output subtasks can be forced to execute at desired time instants by treating them as interrupt routines activated by dedicated timers. The processing subtask, instead, is handled as a normal periodic task, subject to preemption according to the scheduling algorithm. This method will be referred to as *Reducing Jitter by Task Splitting* (RJTS). To avoid using two distinct timers, the output part of the k-th job can be executed at the beginning of the next period, that is just before the execution of the input part of the $(k+1)$-th job.

Figure 4 shows how the task set illustrated in Figure 2 would be handled by using the RJTS technique. Note that the output part of each job is always executed at the beginning of the next period, just before the input part of the next

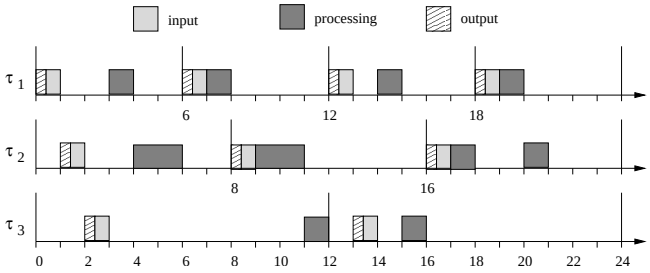

Figure 4. A task set executed according to the RJTS method.

job. As it can be seen from the example, treating the input-output parts as interrupt handling routines can significantly reduce the jitter for all tasks. The advantages of this method are the following:

- The fixed input-output delay greatly simplifies both the controller design and the assertion of system stability. The case of a one-sample delay is especially simple to handle in the control analysis [1]. In contrast, analyzing stability under jitter is much more difficult and requires either knowledge of the statistical distributions of the delays [24] or leads to potentially conservative results [19].

- In theory, the task splitting method can be applied to all the tasks and under any workload, whenever the task splitting overhead can be considered negligible.

However, there are other concerns that must be taken into account when applying this technique:

- The jitter reduction is obtained by inserting extra delay in the task execution. In fact, when applying this method, input and output parts are always separated by exactly a period, while normally the average delay could be smaller. The effect of having a higher delay in the control loop has to be carefully analyzed, since it could be more negative than the effect of jitter.

- Executing the input/output parts as high priority interrupt routines causes extra interference on the processing parts of the control tasks. Such an interference needs to be taken into account in the guarantee test. Feasibility analysis becomes more complex, but can still be performed using appropriate methods, like the one proposed by Jeffay and Stone [17], or by Facchinetti *et al.* [14].

- The extra interference caused by the input/output parts running as high priority interrupt routines decreases the schedulability of the system. As a consequence, tasks must run at lower rates with respect to the normal case.

- The input and output parts of different tasks may compete for the processor among themselves, hence they need to be scheduled with a policy that, in general, could be different than that used for the control tasks (e.g., it could be preemptive or non preemptive). As a consequence, a two-level scheduler is required in the real-time kernel to support such a mechanism. Figure 5 shows an example in which the input and output parts of different tasks overlap in time and require a scheduling policy. In the example, input and output parts always preempt processing parts, but among themselves they are scheduled with a priority equal to the task priority they belong. More specifically, if $P_1, \ldots, P_n$ are the priorities assigned to the n control tasks (where $P_1 \geq P_2 \geq \ldots \geq P_n$), each corresponding input and output part of task τ_i can be assigned a priority $P_i^* = P_0 + P_i$, where P_0 is a priority level higher than P_1.

- Finally, the implementation of the RJTS method requires an extra effort to the user, who has to program a timer for each task to trigger the input and output parts at the period boundaries.

4.2 Reducing jitter by advancing deadlines

Another common approach that can be applied to reduce jitter is to shorten the task relative deadlines. In fact, if a task τ_i is guaranteed to be executed with a relative deadline D_i, clearly its input-output jitter, as well as its response time jitter, cannot be greater than $D_i - C_i$. This method will

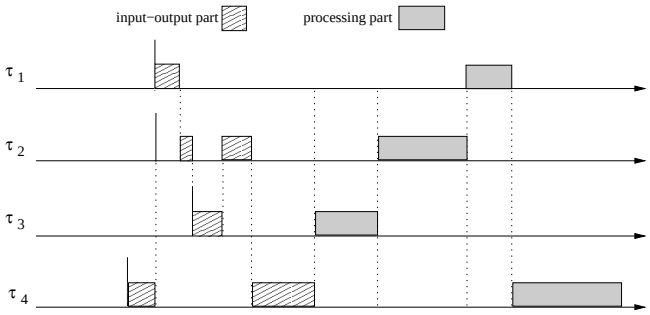

Figure 5. Input and output subtasks need to be scheduled when they overlap in time.

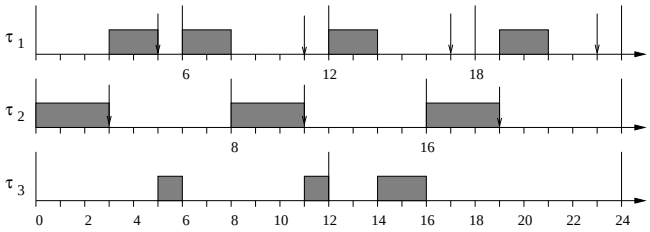

Figure 6. A task set executed according to the RJAD method.

be referred to as *Reducing Jitter by Advancing Deadlines* (RJAD). Following this approach, Baruah *et al.* [5] proposed two methods for assigning shorter relative deadlines to tasks and guaranteeing the schedulability of the task set.

Figure 6 illustrates an example in which the three tasks shown in Figure 2 are required to execute with a jitter no higher that 3, 0, and 10, so their deadlines are set to $D_i = C_i + Jitter$, that is to 5, 3, and 12, respectively.

The RJAD method has the following advantages with respect to the RJTS approach:

- The method does not require any special support from the operating system, since jitter constraints are simply mapped into deadline constraints.

- No extra effort is required to the user to program dedicated timers. Once jitter constraints are mapped into deadlines, the kernel scheduler can do the job.

- There are no interrupt routines creating extra interference in the schedule, so the guarantee test can be performed with the classical response time analysis [21, 2] or more efficient techniques [6], under fixed priorities, or with the processor demand criterion [4], under EDF.

- Advancing deadlines, jitter and delay are both reduced, implying better achievable control performance.

However, there are also the following disadvantages with respect to the RJTS approach:

- The major problem of this method is that it cannot reduce the jitter of all the tasks, but only a few tasks can be selected to run with zero (or very low) jitter. A better result could be achieved by exploiting task release offsets, but the analysis becomes of exponential complexity.

- Advancing task deadlines, the system schedulability could be reduced. As a consequence, as in the RJTS method, tasks could be required to run at lower rates with respect to the normal case.

4.3 Reducing jitter by non preemption

A third method for reducing the input-output jitter of a task is simply to execute it in a non preemptive fashion. This method will be referred to as *Reducing Jitter by Non Preemption* (RJNP). For example, if the task set illustrated in Figure 2 is executed using non preemptive rate-monotonic scheduling, the resulting schedule would be, for this particular case, the same as that one generated by EDF, depicted in Figure 2b.

The RJNP method has the following advantages:

- Using a non preemptive scheduling discipline, the input-output jitter becomes very small for all the tasks (assuming that the task execution times are constant), since the interval between the input and output parts is always equal to the task computation time. This makes it easy to compensate for the delay in the control design.

- Another advantage of this method is that the input-output delay is also reduced to the minimum, which is also equal to the task computation time. This probably gives the largest performance improvement, since control loops are typically more sensitive towards delay than jitter.

- Non preemptive execution allows using stack sharing techniques [3] to save memory space in small embedded systems with stringent memory constraints [15].

On the other hand, the RJNP approach introduces the following problems:

- A general disadvantage of the non preemptive discipline is that it reduces schedulability. In fact, a non preemptive section of code introduces an additional blocking factor in higher priority tasks that can be taken into account with the same guarantee methods used for resource sharing protocols.

- There is no least upper bound on the processor utilization below which the schedulability of any task set can be guaranteed. This can easily be shown by considering a set of two periodic tasks, τ_1 and τ_2, with priorities $P_1 > P_2$ and utilization $U_i = \varepsilon$, arbitrarily small. If $C_2 > T_1$, $C_1 = \varepsilon T_1$, and $T_2 = C_2/\varepsilon$, the task set is unschedulable, although having an arbitrarily small utilization.

- Achieving non preemptive execution for all user tasks is easy in standard operating systems (a single semaphore can be used), but making only one or a few tasks non preemptible requires a larger programming effort.

In order to evaluate the impact of the different approaches on control performance, the three methods have been compared by simulation under different scenarios. The results of the simulations are illustrated and discussed in the next section.

5 Experimental Results

5.1 Simulation Set-Up

We consider a system with $n = 7$ periodic tasks that are scheduled under EDF[1] [22]. The tasks are distinguished in two categories: a subset of control tasks, whose jitter and delay must be bounded, and a subset of hard real-time tasks, with no jitter and delay requirements, except for schedulability. The number of control tasks is changed in the experiments as a simulation parameter.

The performance of the various jitter reduction methods is evaluated by monitoring the execution of a control task, τ_1, with period $T_1 = 50$ ms and constant execution time $C_1 = 5$ ms. The input and output operations are assumed to take $I_1 = O_1 = 0.5$ ms, leaving $P_1 = 4$ ms for the processing part. Notice that τ_1 is not necessarily the task with the shortest deadline. In fact, the other six tasks, τ_2–τ_7, are generated with random attributes, with fixed execution times C_i uniformly distributed in [1, 10] ms, and utilizations U_i chosen according to a 6-dimensional uniform distribution to reach the desired total utilization (algorithm UUniFast in [7]). Task periods are then given by $T_i = C_i/U_i$. Relative deadlines are set equal to periods ($D_i = T_i$) for all hard real-time tasks, whereas they can be reduced by the RJAD method for the control tasks. The offset Φ_i of each task is uniformly distributed in [0, T_i].

The following parameters are varied in the simulation experiments:

- The total utilization U is varied between 0.2 to 0.9 in steps of 0.1. We also include the case $U = 0.99$.

[1] For lack of space we decided to perform the experiments only under EDF, which guarantees full processor utilization in the fully preemptive case, but similar simulations can be carried out under any fixed priority assignment.

- The number of control tasks is varied between 1 and 7. For each control task, $I_1 = O_1 = 0.5$ ms is assumed for the input and output phases.
- The implementation of the control tasks is varied depending on the specific technique used to reduce the jitter.

(We have assumed fixed execution times in an attempt to keep down the number of simulation parameters. In future work, we would like to study the effects of stochastic execution times as well.)

To better evaluate the enhancements achievable with the three methods discussed in Section 4, we also monitor the results when no special treatment is applied on control tasks. Thus, we consider the following four methods:

Standard Task Model (STM). Each task, including the control tasks, is assigned a relative deadline equal to the period.

Reducing Jitter by Task Splitting (RJTS) The output and input operations of the control tasks are implemented in non-preemptible timer interrupt routines. An extra overhead of 0.5 ms was assumed for this implementation, making the non-preemptible section 1.5 ms. This can potentially cause hard real-time tasks to miss their deadlines.

Reducing Jitter by Advancing Deadlines (RJAD) The deadlines of the control tasks are advanced according to Method 2 of Baruah *et al.* [5].

Reducing Jitter by Non Preemptive Execution (RJNP). Control tasks are implemented as non-preemptible tasks, whereas hard tasks can be normally preempted. This can potentially cause hard real-time tasks to miss their deadlines.

For each parameter configuration, $N = 500$ random task sets are generated, and the system is simulated for 50 s (corresponding to 1000 invocations of τ_1). The absolute input jitter (INJ), the absolute input-output jitter (IOJ), and the average input-output latency (IOL) for τ_1 are recorded for each experiment. To evaluate the effect of each method on task schedulability, we also recorded the percentage of unfeasible task sets, i.e., the number of task sets where one or more hard real-time tasks miss a deadline. When a deadline is missed, the simulation for the current task set is aborted and the performance is not counted.

Two sets of experiments were carried out: one to evaluate the effects of the methods on the timing behavior, and one to see their impact on the control performance. For the INJ, IOJ, and IOL results, standard deviations for the average values were computed and they were never greater than 0.35 ms for any configuration. For the control cost evaluation, the standard deviation was less than 0.2%, not counting cases where the control loop went unstable.

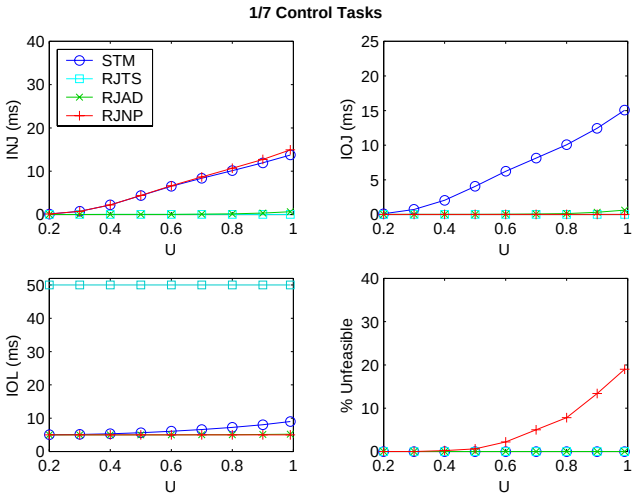

Figure 7. Jitter results with one control task and six hard real-time tasks.

5.2 Jitter Results

We first tested the jitter reduction methods with one control task and six hard real-time tasks. The results are reported in Figure 7. Note that RJAD is very successful in reducing both the input jitter (INJ) and the input-output jitter (IOJ) while minimizing the input-output latency (IOL) for all load cases. RJTS also performs very well in reducing both INJ and IOJ for any utilization, but gives a very long IOL, as expected. Finally, RJNP is able to reduce the IOJ and the IOL to a minimum, but actually increases the INJ slightly compared to the standard task model. RJNP is also the most invasive method in terms of schedulability (bottom-right graph), causing hard real-time tasks to miss deadlines for utilizations higher than 0.5.

Figure 8 illustrates the results achieved with four control tasks and three hard real-time tasks. In this case, RJAD does not work quite as good anymore, because four control tasks to advance their deadlines. RJNP keeps the IOJ and the IOL and a minimum, while the INJ increases. Moreover, it causes even more deadlines to be missed with respect to the previous case. Note that some deadlines are also missed with RJTS at high utilizations, due to the non-preemptible interrupt routines needed for executing the input-output parts at the end of the periods. For the same reason, some jitter is now also experienced by RJTS (upper graphs).

A final simulation experiment has been performed with seven control tasks and the results are shown in Figure 9. In this case, a very high sampling jitter occurs under RJNP (upper-left graph), whereas RJAD shows virtually no improvement at all with respect to the standard task model (STM, upper graphs). Also RJTS experiences more jitter than before, due to the higher number of interrupts. Notice that no deadlines are missed (bottom-right graph) since there are no hard tasks anymore.

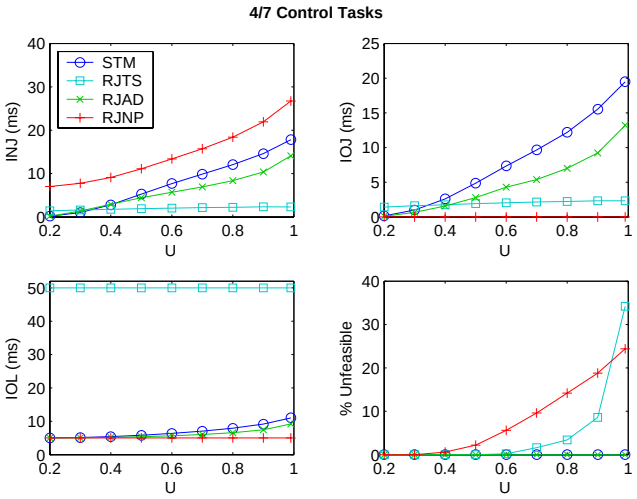

Figure 8. Jitter results with four control tasks and three hard real-time tasks.

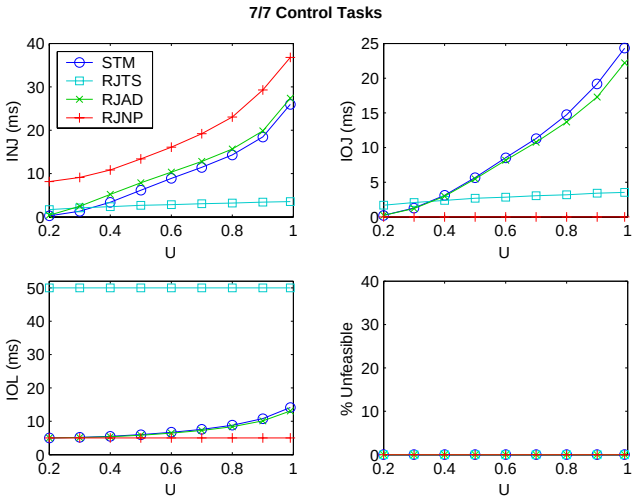

Figure 9. Jitter results with seven control tasks.

5.3 Control Performance Results

The actual control performance degradation resulting from scheduling-induced delay and jitter depends very much on the control application. In general, however, controllers with a high sampling rate (compared to, e.g., the cross-over frequency) are less sensitive to delay and jitter.

In this section, we study two benchmark control problems: one assuming fast sampling and the other assuming slow sampling. In each case, it is assumed that task τ_1 implements a Linear Quadratic Gaussian (LQG) controller [1] to regulate a given plant. The sampling period is hence $T_1 = 50$ ms in both cases. Associated with each plant is a quadratic cost function V that is used both for the LQG control design and for the performance evaluation of the control loop. The two plants and their cost functions are given below:

Plant 1:

$$\frac{dx}{dt} = \begin{bmatrix} 0 & 1 \\ 9 & 0 \end{bmatrix} x + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u + \begin{bmatrix} 1 \\ 0 \end{bmatrix} v$$
$$y = \begin{bmatrix} 0 & 1 \end{bmatrix} x + \sqrt{0.1} e$$

$$V = \mathrm{E} \lim_{T\to\infty} \frac{1}{T} \int_0^T \left(x^T \begin{bmatrix} 0 & 0 \\ 0 & 10 \end{bmatrix} x + u^2 \right) dt$$

Plant 2:

$$\frac{dx}{dt} = \begin{bmatrix} 0 & 1 \\ -3 & -4 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u + \begin{bmatrix} 35 \\ -61 \end{bmatrix} v$$
$$y = \begin{bmatrix} 2 & 1 \end{bmatrix} x + e$$

$$V = \mathrm{E} \lim_{T\to\infty} \frac{1}{T} \int_0^T \left(x^T \begin{bmatrix} 2800 & 80\sqrt{35} \\ 80\sqrt{35} & 80 \end{bmatrix} x + u^2 \right) dt$$

Here, x is the plant state vector, u is the control signal, y is the measurement signal, v is a continuous-time zero-mean white noise process with unit intensity, and e is a discrete-time zero-mean white noise process with unit variance. Further, v and e are assumed to be independent.

Plant 1 is a linear model of an inverted pendulum (an unstable plant) with a natural frequency of 3 rad/s. Assuming a constant input-output delay of 5 ms, the cost function produces an LQG controller that achieves a cross-over frequency of 5.2 rad/s and a phase margin of 31°. This can be seen as a typical, quite robust control design with a high enough sampling rate. The jitter margin [10] is computed to 82 ms, which is larger than the sampling period and indicates that the control loop cannot be destabilized by scheduling-induced jitter.

Plant 2 (a stable plant) and its associated cost function represent a pathological case where the LQG design method gives a controller that is very sensitive to delay and jitter [24]. Assuming a delay of 5 ms, the resulting cross-over frequency is 20.6 rad/s while the phase margin is only 17°. The jitter margin is found to be only 10 ms, indicating that latency and jitter from the scheduling might destabilize the control loop.

Assuming the same simulation set-up as in the jitter evaluation, for each parameter configuration, $N = 500$ random task sets were generated and the control performance of task τ_1 was recorded for 50 s. For STM, RJAD, and RJNP, the controller was designed assuming a constant delay of 5 ms, while for RJTS, the delay was assumed to be 50 ms. The whole procedure was repeated for each of the two plants, the results being reported in Figure 10. In the figure, the costs have been normalized such that the performance under minimum delay and jitter is 1.

For Plant 1, it can be noted that RJTS performs uniformly worse than the other implementations, including

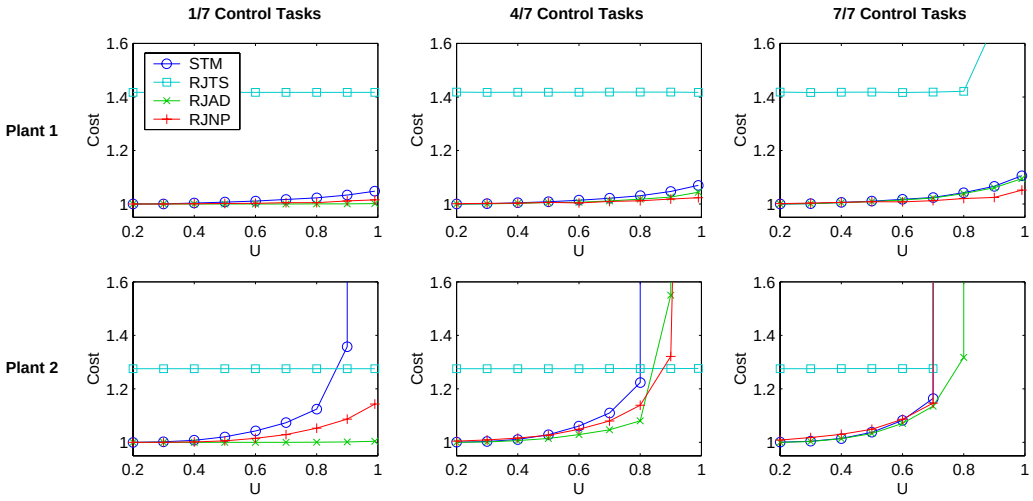

Figure 10. Control performance evaluation for two different plants.

STM. This is due to the long input-output latency, which, despite the exact delay compensation, destroys the performance. As the number of control tasks and the load increase, the performance of STM, RJAD, and RJNP degrades only slightly, RJNP performing the best in the case of multiple controllers, while RJAD works the best for a single controller. Also notice the performance break-down of RJTS for 7 control tasks and high loads, where the extra scheduling overhead causes task τ_1 to miss its outputs.

For Plant 2, the issue of jitter is more critical. While RJTS gives a constant performance degradation (except for the case of seven control tasks and high load) the other implementations exhibit degradations that seem to relate to the total amount of jitter (INJ + IOJ) reported in Figures 7–9. As a consequence, RJAD has an edge over RJNP for almost all system configurations. Note that, for sufficiently high utilizations, some implementations actually cause the control loop to go unstable (the cost approaches infinity) for high loads.

In summary, RJAD gives the better control performance for most system configurations. At the same time, it does not cause any deadlines to be missed. RJTS is the "safest" implementation, in that it gives a more or less constant performance degradation, even for an unrobust control design, many control tasks, and a high CPU load.

6 Conclusions

In this paper we studied three scheduling techniques for reducing delay and jitter in real-time control systems consisting of a set of concurrent periodic tasks. The first method (RJTS) reduces jitter by splitting a task in three subtasks (input, processing and output) and forces the execution of input and output parts at the period boundaries. The second method (RJAD) reduces jitter and delay by assigning the tasks a shorter deadline. The third method (RJNP) simply relies on non preemptive execution to eliminate input-output jitter and delay.

Advantages and disadvantages of the three approaches have been discussed and a number of simulation experiments have been carried out to compare these techniques and illustrate their effect on control system performance. In the simulation results, it was seen that RJTS and RJNP can compromise the schedulability of the system if this issue is not taken into account at design time. RJAD performed very well for a single control task and reasonably well for multiple control tasks. RJTS was able to reduce both the input jitter and the input-output jitter to a minimum but produced a long input-output latency.

In conclusion, for a robust control design (with sufficient phase and delay margins), the performance degradation due to jitter is very small, even for the standard task model. For a single control task, the RJAD method can reduce this degradation all the way down to zero in most cases. On the other hand, a constant one-sample delay gives a very large penalty in comparison. Hence, it is clear that the RJTS method should be avoided for robust control systems.

For unrobust control systems with very small phase and delay margins, RJTS could be considered to be a "safe" choice of implementation. For many system configurations, however, even STM performs better than RJTS. One has to go to quite extreme situations to find examples where RJTS actually gives better control performance than STM. In these situations, the computing capacity is probably severely under-dimensioned, and it is questionable whether *any* implementation can actually meet the control performance requirements.

Finally, in terms of control performance, the RJNP method sometimes performs better and sometimes worse than RJAD in the various configurations. However, one

should keep in mind, that RJNP may cause hard tasks to miss their deadlines and requires a more difficult off-line analysis.

As a future work, we plan to provide support for these techniques in the Shark operating system, in order to evaluate the effectiveness of these approaches on real control applications. Also, we would like to explore the trade-off between jitter and delay for a wider class of plants and controllers.

References

[1] K. J. Åström and B. Wittenmark, *Computer-Controlled Systems: Theory and Design*. Third edition. Prentice-Hall, 1997.

[2] N. C. Audsley, A. Burns, M. Richardson, K. W. Tindell, and A. J. Wellings, "Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling," *Software Engineering Journal*, Vol. 8, No. 5, pp. 284–292, September 1993.

[3] T. P. Baker, "Stack-Based Scheduling of Real-Time Processes," *Real-Time Systems* Vol. 3, No. 1, pp. 67–100, 1991.

[4] 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.

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

[6] E. Bini and G. Buttazzo, "Schedulability Analysis of Periodic Fixed Priority Systems," *IEEE Transactions on Computers*, Vol. 53, No. 11, pp. 1462–1473, November 2004.

[7] E. Bini and G. Buttazzo, "Biasing Effects in Schedulability Measures," in *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, Catania, Italy, July 2004.

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

[9] G. Buttazzo, "Rate Monotonic vs. EDF: Judgment Day", *Real-Time Systems*, Vol. 28, pp. 1-22, 2005.

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

[11] A. Crespo, I. Ripoll and P. Albertos, "Reducing delays in RT control: the control action interval," in *Proceedings of the 14th IFAC World Congress*, pp. 257–262, 1999.

[12] C. Davidson, "Random sampling and random delays in optimal control," PhD Dissertation 21429, Department of Optimization and Systems Theory, Royal Institute of Technology, Sweden, 1973.

[13] 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.

[14] T. Facchinetti, G. Buttazzo, M. Marinoni, G. Guidi, "Non-Preemptive Interrupt Scheduling for Safe Reuse of Legacy Drivers in Real-Time Systems," in *Proceedings of the 17th Euromicro Conference on Real-Time Systems*, Palma de Mallorca, Spain, July 2005.

[15] P. Gai, G. Lipari, and M. di Natale, "Minimizing Memory Utilization of Real-Time Task Sets in Single and Multi-Processor Systems-on-a-chip," in *Proceedings of the 22th IEEE Real-Time Systems Symposium*, London, UK, December 2001.

[16] R. Gerber and S. Hong, "Semantics-Based Compiler Transformations for Enhanced Schedulability," in *Proceedings of the 14th IEEE Real-Time Systems Symposium*, December 1993.

[17] K. Jeffay and D. L. Stone, "Accounting for Interrupt Handling Costs in Dynamic Priority Task Systems," in *Proceedings of the IEEE Real-Time Systems Symposium*, Raleigh-Durham, NC, USA, pp. 212–221, December 1993.

[18] R. E. Kalman and J. E. Bertram, "A unified approach to the theory of sampling systems," *J. Franklin Inst.*, Vol. 267, pp. 405–436, 1959.

[19] C.-Y. Kao and B. Lincoln, "Simple Stability Criteria for Systems with Time-Varying Delays," *Automatica*, 40:8, pp. 1429–1434, August 2004.

[20] H. J. Kushner and L. Tobias, "On the stability of randomly sampled systems," *IEEE Transactions on Automatic Control*, Vol 14, No 4, pp. 319–324, 1969.

[21] J. P. Lehoczky, L. Sha, and Y. Ding, "The Rate-Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," in *Proceedings of the 10th IEEE Real-Time Systems Symposium*, Santa Monica, CA, USA, pp. 166–171, December 1989.

[22] 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, January 1973.

[23] P. Martí, G. Fohler, K. Ramamritham, and J.M. Fuertes, "Jitter Compensation for Real-time Control Systems," in *Proceedings of the 22nd IEEE Real-Time System Symposium*, London, UK, December 2001.

[24] J. Nilsson, B. Bernhardsson and B. Wittenmark, "Stochastic Analysis and Control of Real-Time Systems with Random Time Delays," *Automatica*, 34:1, pp. 57–64, 1998.

[25] 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.