Jitter Control in On-line Scheduling of Dependent Real-Time Tasks

L. David[(1)], F. Cottet[(1)] and N. Nissanke[(2)]

[(1)] LISI–ENSMA*
1, Av. Clément Ader,
Téléport 2
86961 Futuroscope - France
{david,cottet}@ensma.fr

[(2)] SCISM†
South Bank University
103, Borough Road
London SE1 0AA - U.K.
nissanke@sbu.ac.uk

Abstract

A typical characteristic of real-time systems is concurrent processing of tasks under strict timing requirements. These timing requirements may impose not only direct constraints, such as deadlines, but also indirect timing constraints in terms of inter–task dependencies. However, scheduling policies which can deal with some of these constraints effectively are limited, especially for the on-line context, and are not widely known within the real-time community. In this context, this paper presents a technique to control execution irregularities, namely, jitter. The technique is based on the modi£cation of task temporal parameters in the paradigm of the well known Deadline Monotonic (DM) and Earliest Deadline First (EDF) algorithms. The technique presents a way either to remove jitter completely (in the case of regularity constrained tasks) or to bound jitter releases. Moreover, the approach takes into account temporal dependencies of tasks on one another.

Keywords: Real-time, on-line scheduling, jitter, dependent periodic tasks.

1. Introduction

A real-time system, controlling a physical device or a physical process, invariably performs two basic functions: sampling sensor readings and responding to different situations by sending control signals to actuators. With respect to process requirements, these tasks (sampling data systems, control systems, etc.) must often be periodic. However, this is rarely the case in practice since periodicity cannot be achieved in an absolute sense. Indeed there are often timing requirements on the design in the form of permissible jitter [2] or timing tolerances. Basically, jitter arises due to the way algorithms, such as Rate Monotonic (RM), Deadline Monotonic (DM) and Earliest Deadline First (EDF) [19] algorithms, schedule multiple tasks for concurrent execution. These algorithms do not consider jitter requirements. Indeed, in the periodic task model of Liu and Layland [19], each instance of a task (an execution of a task within a given period) is considered completely independent from other instances. However, most real-time applications (e.g. multimedia data transmission [11], fault-tolerant systems, precision robotics [18], etc.) have regularity constraints on various tasks. Besides, jitter can arise due to different reasons at the implementation level, an example being the unpredictability in the actual timing of periodic events as generated by the RTOS [20]. However, this paper addresses only the jitter induced by scheduling at the application level.

Scheduling algorithms for controlling jitter are limited. An off-line scheduling technique based on simulated annealing methods due to Di Natale *et al.* [10] deals with end-to-end jitters in a distributed context. In this context, Coutinho *et al.* [8] propose a jitter minimization technique based on a genetic algorithm. Another off-line solution due to [6, 7] consists of an exhaustive search of all valid sequences. Han and Lin [14] also suggest a distance constraint task system model (DCTS) in which execution regularity constraints are taken into account in some speci£c scheduling algorithms. In £xed priority scheduled systems, Bate *et al.* [2] describe a task attribute handling mechanism which can deal with timing requirements, including jitter. Baruah *et al.* [1] propose an interesting technique for minimizing maximum jitter of a relatively small number of task in the EDF context, but without taking advantage of the pos-

*LISI – Laboratory of Applied Computer Science; ENSMA – National Engineering School in Mechanics and Aerotechnics.

†SCISM–SBU – School of Computing, Information Systems and Mathematics

L. David, F. Cottet and N. Nissanke. Jitter Control in On-Line Scheduling of Dependent Real-Time Tasks.
Proc. of the 22^{nd} Real-Time Systems Symposium, pages 49-58, RTSS'01, December 2001, London, UK.

sibility to set offsets in order to reduce jitter and to handle systems of interdependent tasks. Based on a deadline assignment technique, Kim *et al.* [15] present a technique to reduce jitter in a set of independent tasks. In our work, we study in greater depth a) the issue of jitter control in the general on-line scheduling context, and b) con£gurations of interdependent tasks in the deadline based scheduling context, both in a uni–processor context.

Our discussion on jitter control proceeds as follows. Section 2 introduces different types of jitter. Section 3 introduces a set of task parameter handling tools to be used in Sections 4 and 5 in describing a complete jitter cancellation technique and a jitter bounding technique.

2. De£nitions of Jitter

Our work uses the periodic task model [19]. Each ith task is characterized by four parameters: $r_{i,1}$ (the £rst request time or offset), C_i (maximum execution time), D_i (relative deadline) and T_i (task period), each being, as appropriate, a non–negative or a positive integer; see Figure 1.

We consider two types of jitter: regularity jitter (execution irregularity within one task) and end-to-end jitter (response time irregularity, either within a set of two or more tasks forming an activity or within the same task in a repetitive sense). Both the above types of jitter can be de£ned over a scheduling period, or a major cycle (H) – a reference pattern used for scheduling simulation. Authors of [13] generalize the results of [17] such that in every case H is bounded by $max_i(r_{i,1}) + 2LCM_i(T_i)$. A common approach to jitter de£nition is to use the maximum duration between successive activation times and completion times [1, 5]. It is not unusual, however, to come across sequences of task instances where the intervals between successive completion times are more or less identical, resulting in a less accurate estimate of jitter. An advantage of using H is that in de£ning jitter it allows the use of statistical concepts, such as the mean, covering a range of completion times spread over H.

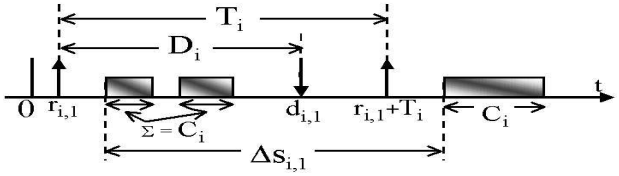

Figure 1. Task model.

2.1. Regularity Jitter

Let $s_{i,k}$ and $e_{i,k}$ be, respectively, the k^{th} execution start date and the k^{th} execution end date of task τ_i; see Figures 1 and 2. The length of time between two successive start dates of task τ_i within the k^{th} scheduling period is denoted by $\Delta s_{i,k}$, i.e. $\Delta s_{i,k} = s_{i,k+1} - s_{i,k}$. Analogously, $\Delta e_{i,k} = e_{i,k+1} - e_{i,k}$ denotes the length of time between two successive end dates. Both $\Delta s_{i,k}$ and $\Delta e_{i,k}$ vary from 0 to $2T_i$, the two extremities corresponding to negligibly small execution times. Let us also introduce

$$j_{s_{i,k}} = \frac{|\Delta s_{i,k} - T_i|}{T_i} 100\,\% \qquad (1)$$

as a percentage relative measure of the execution start date irregularity between the k^{th} and the $(k+1)^{th}$ instances. Note that $j_{s_{i,k}} = 0\,\%$ when the difference $s_{i,k}$ equals T_i (i.e. the case of regular tasks) and $j_{s_{i,k}} = 100\,\%$ when it equals 0 or $2T_i$ (i.e. the case of maximum irregularity in execution). Analogous expressions apply for $e_{i,k}$. Let us now introduce the de£nition:

DEFINITION 1 : *Given a task τ_i with characteristics $(r_{i,1}, C_i, D_i, T_i)$, the mean regularity jitter of τ_i is de£ned by: $J_{Mean}(\tau_i) = \frac{1}{N_i}\sum_{k=1}^{N_i} j_{s_{i,k}}$, N_i being the number of instances of task τ_i during H, and $s_{i,k}$ and $j_{s_{i,k}}$ being as de£ned above.*

See Figure 2 for an illustration of the above for $N_i = 4$.

Similarly, other statistics can be de£ned on jitter based on $j_{s_{i,k}}$, for example, the maximum jitter, the minimum jitter, etc. Note that if preemption is not permitted then regularity jitter based on $s_{i,k}$ or $e_{i,k}$ will be identical. Otherwise, they are linked through the cohesion jitter – a topic to be introduced in the next section.

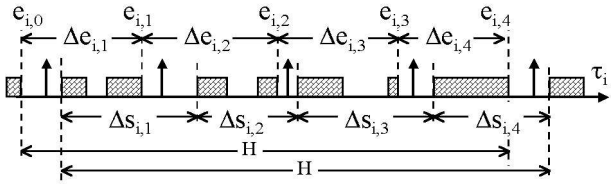

Figure 2. Regularity jitter.

2.2. End-to-End Jitter

This subsection de£nes end-to-end jitter for an activity such as a control activity. An activity may consist of just one periodic task or, alternatively, of several periodic tasks. In the former case, the end-to-end jitter measures the cohesion jitter and, in the latter, the response time irregularities. Thus, the end-to-end jitter is the irregularity in the elapsed time between the execution start date of one instance of a task and the execution end date of another instance of a task, with the two task instances not necessarily belonging to the same task. The corresponding response times are denoted by $RT_k(a)$, a being a given activity and k being an index

L. David, F. Cottet and N. Nissanke. Jitter Control in On-Line Scheduling of Dependent Real-Time Tasks.
Proc. of the 22^{nd} Real-Time Systems Symposium, pages 49-58, RTSS'01, December 2001, London, UK.

ranging over the instances of a over a given scheduling interval.

Let a be an activity consisting of, for example, two tasks τ_i and τ_j, with outputs of τ_i serving as inputs for τ_j. In this case, a is a pair $\langle \tau_i, \tau_j \rangle$. Note that τ_i and τ_j may not necessarily have the same rate [2], except when they are dependent. In Figure 3-a, τ_i and τ_j have two different rates and, in order to be more up-to-date when executing τ_j, we choose the most recent instance of τ_i for the last value $RT_3(a)$. What is of interest then is the elapsed time, within a given scheduling period, between the most recent execution start date of an instance of τ_i and the execution end date of its succeeding instance of τ_j; see $RT_k(a)$ for $1 \leq k \leq 3$ in Figure 3-a. On the other hand, with respect to cohesion jitter, what is of interest is the elapsed time between the start and end dates of execution of a given task instance; see $RT_k(\tau_i)$ for $1 \leq k \leq 4$ in Figure 3-b.

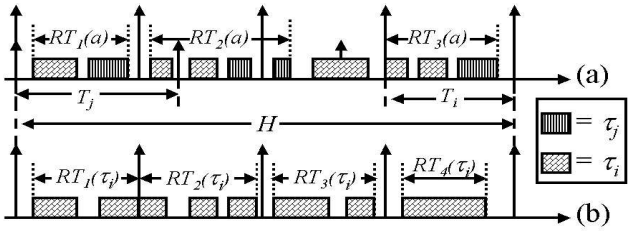

Figure 3. (a) : End-to-end jitter for an activity ($a =< \tau_i, \tau_j >$) - (b) : cohesion jitter for a task.

In defining end-to-end jitter, it is necessary to compare RT_k values defined above against a mean value, which could be computed either from RT_k values themselves or, alternatively, using the data collected by monitoring the automation, for example, using a fitted response time curve which makes the system stable. Let us denote whatever the chosen value by $\overline{RT}$. The end-to-end jitter can then be considered in terms of the difference: $\Delta_k = \frac{|RT_k - \overline{RT}|}{\overline{RT}} 100\ \%$. The latter is similar to (1) and leads to the following definition:

DEFINITION 2: *Given the response times $RT_k(a)$ and $RT_k(\tau_i)$ as introduced above, the mean end-to-end jitter is defined as a percentage by: $J_{Mean}(a) = \frac{1}{N_a} \sum_{k=1}^{N_a} \frac{|RT_k(a) - \overline{RT}|}{\overline{RT}} 100\%$, N_a being the the number of instances of the activity a. Analogously, the mean cohesion jitter as a percentage is defined by $J_{Mean,c}(\tau_i) = \frac{1}{N_i} \sum_{k=1}^{N_i} \frac{|RT_k(\tau_i) - C_i|}{C_i} 100\%$, N_i being the number of instances of the task τ_i executed during H.*

The above are illustrated in Figures 3-a and 3-b for $N_a = 3$ and $N_i = 4$. Our focus in Section 3 and 4 is on regularity jitter.

3. Approach to Controlling Regularity Jitter

Let R and NR be two sets of periodic tasks with n_R and n_{NR} elements respectively and let their union characterize a real-time task configuration. The tasks τ_R in the first set are required to be regular in term of execution, whereas the tasks τ_{NR} in the second set are not subject to such a requirement. Thus, the distinction between the two task sets is based on whether or not the periodicity is to be observed strictly. As was mentioned earlier, task execution irregularities arise due to the way scheduling algorithms work and do not affect other constraints such as deadlines. Task regularity constraints may be required in the case of sampling tasks and control tasks.

Execution windows of a task τ_i correspond to intervals of the form $[|r_{i,j}; e_{i,j}|] (j \in \mathbb{N}^*)$. We assume that the periods are not alterable, since they usually come from process requirements. We also assume that the worst case execution characteristics of each task is known and fixed, and that redefinition of release dates $r_{i,1}$ would not lead to redefinition of any relative deadlines (D_i) (offset free systems [12]). Obviously, even where it is permitted, any redefinition of D_i must result in a smaller value than its original value.

3.1. General scheme

The jitter control technique presented here is based on the idea that if two regular tasks have a non empty intersection of their execution windows, then one of them will be exposed to regularity or cohesion jitter. Let us illustrate this idea with an example involving a configuration of three tasks (τ_1, τ_2, τ_3), which are to be scheduled with external priority ($Prio_i$); see table 1. Figure 4 shows the scheduling sequence where it can be noted that τ_2 is preempted by τ_1, and τ_3 is preempted by both τ_1 and τ_2. Equivalently, the intersection of task execution windows are effectively not empty, resulting in jitter on τ_2 and τ_3. An interesting solution would be to cancel jitter by desynchronizing the release dates. Indeed, the triplet $(r_{1,1}, r_{2,1}, r_{3,1}) = (1, 0, 6)$ results in an empty intersection of execution windows, eliminating the jitter completely; see Figure 5.

Table 1. Example: Task characteristics.

	$r_{i,1}$	C_i	D_i	T_i	$Prio_i$
τ_1	0	3	16	16	3
τ_2	0	1	4	4	2
τ_3	0	2	8	8	1

Our technique is a generalization of the above idea and consists basically of adjusting execution windows of regular tasks through redefinition of release dates and relative deadlines, but always maintaining the required inter–relationships between tasks. Thus, the approach to jitter

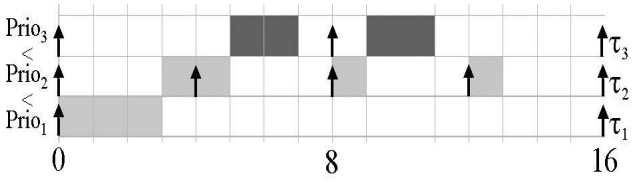

Figure 4. τ_2 preemption by τ_1, τ_3 preemption by τ_1 and τ_2 : jitter on τ_2 and τ_3.

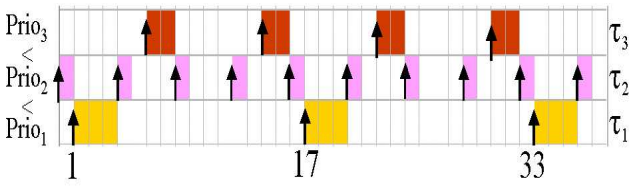

Figure 5. Scheduling sequence with no jitter.

cancellation and jitter bounding can be summarized in four steps:

Step 1: release date desynchronization,

Step 2: maintenance of the precedence relation,

Step 3: restriction of execution windows to a certain bound,

Step 4: veri£cation of schedulability through tests.

This above strategy can be accomplished with two different effects: either by completely eliminating regularity jitter (i.e. by forbidding preemption of R tasks by any NR task) or by bounding regularity jitter (i.e. by permitting preemption to a certain extent). This relies on three separate tools. The £rst tool searches for release dates for regular tasks so that each one of the tasks completes after a certain amount of time following its activation without causing an execution overlap. The second tool veri£es the maintenance of the precedence constraints on tasks. The third tool performs the rede£nition of relative deadlines in order to achieve the required task priorities.

3.2. Release date desynchronization

Let $\{\tau_1, ..., \tau_n\}$ be a set of task to be desynchronized. For every task τ_i, we de£ne a time value W_i, $W_i \in \mathbb{N}^*$ and $0 < C_i \leq W_i \leq D_i$, such that by W_i the execution of the corresponding instance of τ_i is fully completed; see Figure 6. The problem of desynchronization of execution windows [9] consists then of a constraint satisfaction problem, namely, of £nding a solution $(r_{1,1}, \ldots, r_{n,1})$ satisfying the inequalities:

$$(\forall\, i, j \in 1 \cdots n;\ i < j)$$
$$r_{i,1} - r_{j,1} \neq 0 \, mod\, (T_i \wedge T_j)$$
$$\left(\begin{array}{c} r_{i,1} - r_{j,1} \neq 1 \, mod\, (T_i \wedge T_j) \\ r_{i,1} - r_{j,1} \neq 2 \, mod\, (T_i \wedge T_j) \\ \vdots \qquad\qquad \vdots \\ r_{i,1} - r_{j,1} \neq (W_j - 1) \, mod\, (T_i \wedge T_j) \end{array} \right) \quad (2)$$
$$and$$
$$\left(\begin{array}{c} r_{j,1} - r_{i,1} \neq 1 \, mod\, (T_i \wedge T_j) \\ r_{j,1} - r_{i,1} \neq 2 \, mod\, (T_i \wedge T_j) \\ \vdots \qquad\qquad \vdots \\ r_{j,1} - r_{i,1} \neq (W_i - 1) \, mod\, (T_i \wedge T_j) \end{array} \right)$$

where $\wedge$ denotes the greatest common divisor (GCD) of its operands, and mod the modulo symbol. These inequalities follow from the *Generalized Chinese Remainder Theorem* [12, 16]. This is illustrated in the £gure 7, where $r_{i,k}$ ($k \in \mathbb{N}^*$) is £xed and $r_{j,k}$ is to be de£ned at a location such that it would cause no overlap with other jobs from R tasks.

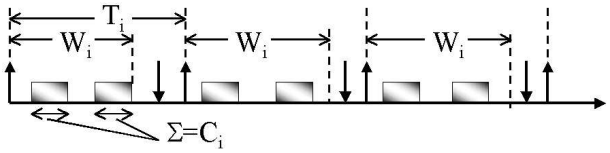

Figure 6. The W_i parameter used in task desynchronization.

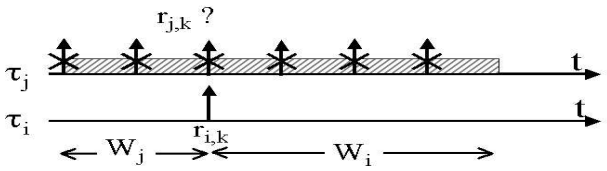

Figure 7. Illustrative diagram for task desynchonisation.

A necessary condition for a solution to (2) is that the periods are NOT pairwise relatively prime. Let us attempt a proof of this claim by contradiction, assuming its negation and showing that this would make the £rst inequality inconsistent:

proof: Suppose that: $(\forall\, i, j \in 1 \cdots n;\ i < j)(\exists\, r_{i,1}, r_{j,1} \in \mathbb{N}^*)\ r_{i,1} - r_{j,1} \neq 0 \, mod\, (T_i \wedge T_j)$ with periods pairwise relatively prime. For p, q and m in $\mathbb{N}^*$, the de£-

nition of congruence is given by $p = q\ mod(m) \Leftrightarrow (\exists\, k \in \mathbb{Z}^*)\ p - q = k \cdot m$. In others words, $p \neq q\ mod(m) \Leftrightarrow (\forall\, k \in \mathbb{Z}^*)\ p - q \neq k \cdot m$. In the £rst inequality of (2), pairs such as $\langle r_{i,1}, r_{j,1} \rangle$ are supposed to be known. Let d_{ij} denote the difference $r_{i,1} - r_{j,1}$. It is necessarily the case that d_{ij} is in $\mathbb{Z}^*$. The fact that $r_{i,1} - r_{j,1} = d_{ij}$ ensures the existence of k in $\mathbb{Z}^*$, such that $r_{i,1} - r_{j,1} = k \cdot 1$. Hence, $r_{i,1} \neq r_{j,1}\, mod(1)$ is false, leading to a contradiction. As a consequence we conclude that our original assumption is false, resulting in the following condition: periods T_i and T_j must not be pairwise relatively prime. □

If it is possible to £nd $(r_{1,1}, ..., r_{n,1})$ subject to above conditions, and if the execution window of every task τ_i $(1 \leq i \leq n)$ is equal to W_i, then the execution windows are guaranteed not to overlap. For the task con£guration of Figure 5, the above conditions are satis£ed for $W_1 = 3$, $W_2 = 1$ and $W_3 = 2$. In this case, for $i = 1$ and $j = 2$ the inequalities (2) take the form:

$$r_{1,1} - r_{2,1} \neq 0\, mod\,(4) \quad and \quad \left(\begin{array}{l} r_{2,1} - r_{1,1} \neq 1\, mod\,(4) \\ r_{2,1} - r_{1,1} \neq 2\, mod\,(4) \end{array} \right) \tag{3}$$

Solving the problem stated in (2) can be quite complex, more so as n increases. Any precise analytical technique is likely to be of limited value and restricted to a particular set of conditions. Therefore, generally numerical techniques [9] are to be preferred. Though it is exponential, the problem complexity is not a critical factor in real time applications with a relatively small number of regular tasks, and with a relatively narrow time interval (W_i). Indeed, the regular tasks involved in this problem are mostly input–output tasks, usually with a maximum execution time (C_i) much smaller than the period (T_i) and with a choice of W_i parameter close to C_i in order to reduce jitter. As with other task parameters, $r_{i,1}$s are assumed to be non-negative integers [12]. Characteristics of applications dealt with by the authors comply with these assumptions.

3.3. Precedence Relation

One of the important constraints on an activity is the right execution order. Generally, tasks involved in an activity are dependent on one another and have the same period T, which also happens to be the activity period. In this work we rely on this as an assumption, and choose conditions that impose minimal restrictions on task temporal attributes. According to [4], in order to ensure precedence of τ_i over τ_j, it is necessary to impose that

$$\begin{array}{lll} & r_{i,1} \leq r_{j,1} & Prio_i > Prio_j \\ or & r_{i,1} < r_{j,1} & Prio_i \geq Prio_j \end{array} . \tag{4}$$

Obviously, these conditions depend on the choice of the scheduling algorithm. For example, in DM the priority ordering $Prio_i > Prio_j$ corresponds to $D_i < D_j$.

3.4. Priority Rede£nition

Let us examine here how to rede£ne relative deadlines so that tasks in R will always enjoy higher priorities compared to those in NR. Again, this would depend on the chosen scheduling algorithm. In DM, where tasks with the shorter deadlines will have higher priorities, the condition that needs to be veri£ed is

$$(\forall \tau_i \in R)\ C_i \leq D_i^* \leq \min_{\tau_j \in NR}(D_i; D_j - 1), \tag{5}$$

where D_i^* denotes the new relative deadline to be considered for task τ_i. In EDF, priorities are dynamic and, therefore, it is dif£cult to foresee the actual runtime priorities of tasks. However, there is a condition which enables the imposition of the required priority *a priori*, namely

$$(\forall \tau_i \in R)\ D_i^* = C_i, \tag{6}$$

according to which every task τ_i will have the highest priority whenever it requires, and acquires, the processor.

4. Elimination of Regularity Jitter

This section examines £rst the results obtained for independent task con£gurations and then, using an illustrative example, the case of dependent task con£gurations.

4.1. The Case of Independent Tasks

As the authors of [10] underline, controlling jitter is quite a complex problem in the presence of task preemptions. Thus, our approach is to ensure that no regular task is ever preempted, thus achieving the kind of scheduling sequence illustrated in Figure 8. Let us study the approach using an example [9]. Consider the task con£guration with characteristics given in Table 2 and with two of them $\langle \tau_{Acq_1}, \tau_{Acq_2} \rangle$ having an execution regularity constraint. Scheduling using DM, as well as EDF, leads to a sequence with non–negligible jitter on regular tasks (see table 2).

Now let us modify the task temporal attributes.

STEP 1: Apply the desynchronization scheme described in Section 3.2 on both regular tasks with $W_i = C_i$. According to an enumeration solution technique, we obtain a set of 377 pairs £tting this requirement. For example, one solution is the pair $(r_{Acq_1,1}, r_{Acq_2,1}) = (0, 2)$.

STEP 3: To guaranty execution of regular tasks in R with no preemption, it is necessary to make tasks in R have