Execution times and execution jitter analysis of real-time tasks under fixed-priority pre-emptive scheduling

Citation for published version (APA):

Document status and date:
Published: 01/01/2008

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 03. Sep. 2020
Execution times\footnote{The \textit{execution time} of a job of a task denotes the length of its \textit{execution interval}, i.e. its actual computation time \textit{including} pre-emptions by higher priority tasks. The \textit{execution jitter} of a task denotes the largest difference between the execution times of any of its jobs. This terminology conforms to that of \cite{2, 5}. Readers more familiar with the Int. Workshop on Worst-Case Execution Time (WCTE) Analysis, which is held in conjunction with the Euromicro Conference on Real-Time Systems (ECRTS), may experience our terminology as confusing, however.} and execution jitter analysis of real-time tasks
under fixed-priority pre-emptive scheduling

Reinder J. Bril$^1$, Gerhard Fohler$^2$, and Wim F.J. Verhaegh$^3$

1. Technische Universiteit Eindhoven (TU/e), Den Dolech 2, 5612 AZ Eindhoven, The Netherlands
2. Technische Universität Kaiserslautern, Erwin-Schrödinger-Straße, D-67663 Kaiserslautern, Germany

\texttt{r.j.bril@tue.nl, fohler@eit.uni-kl.de, wim.verhaegh@philips.com}

Abstract

In this paper, we present worst-case and best-case execution times and (absolute) execution jitter analysis of independent, periodically activated, hard real-time tasks that are executed on a single processor under fixed-priority pre-emptive scheduling (FPPS), arbitrary phasing, (absolute) activation jitter, and deadlines at most equal to (worst-case) periods minus activation jitter. We prove that the worst-case and best-case execution time of a task are equal to its worst-case and best-case response time, respectively. We present an upper bound for execution jitter of a task expressed in terms of its best-case and worst-case execution time. We briefly consider execution times and execution jitter in other settings, such as distributed multiprocessor systems with task dependencies, tasks with arbitrary deadlines, and tasks with multiple operating modes. Execution jitter analysis is particularly valuable for real-time control systems to determine the variations in sampling-actuation delays of control tasks.

1. Introduction

1.1. General context and focus

Real-time computing systems are computing systems that provide correct and \textit{timely} responses to events in their environment, where the term \textit{timely} means that the timing constraints imposed on these responses must be met. In a basic setting of such a system, we consider a set of \( n \) independent, periodically activated tasks \( \tau_1, \tau_2, \ldots, \tau_n \) that are executed on a shared resource and scheduled by means of fixed-priority pre-emptive scheduling (FPPS) \cite{12}. Each task \( \tau_i \) generates an infinite sequence of jobs \( \iota_{ik} \) with \( k \in \mathbb{Z} \). We distinguish three types of tasks based on the characteristics of the inter-activation times of its jobs, i.e. a classical \textit{periodic} task \( \tau_i \) \cite{14} with a fixed period \( T_i \) and optionally an (absolute) activation jitter \( AJ_i \), a \textit{sporadic} task \( \tau_i \) \cite{16} with a worst-case (or minimum) period \( WT_i \), and an \textit{elastic} task \( \tau_i \) \cite{6} with a worst-case period \( WT_i \) and a best-case (or maximum) period \( BT_i \), where \( WT_i < BT_i \). In this paper, we assume that the elastic coefficient of an elastic task is zero, i.e. the period of an elastic task cannot be varied by the system. We term a periodic task with an activation jitter equal to zero a \textit{strictly periodic} task. Apart from inter-activation times, each task \( \tau_i \) is characterized by a worst-case computation time \( WC_i \), a best-case computation time \( BC_i \), where \( BC_i \leq WC_i \), a phasing \( \phi_i \), and timing constraints such as a (relative) deadline. All the times \( \phi_i \) together form the phasing \( \phi \) of the tasks. We assume that an arbitrary phasing may occur. For simplicity, we assume the overhead from task scheduling and context switching to be negligible.

To analyse a real-time system, there are a number of interesting quantities, which typically come in pairs, i.e. in a best-case and a worst-case quantity. Among them are the \textit{best-case response time} \( BR_i \) and \textit{worst-case response time} \( WR_i \) of a task \( \tau_i \), being the shortest and longest interval of time from an activation of that task to the corresponding completion, respectively. When the timing constraints of a task are expressed in terms of a lower and an upper bound on its response time, then all jobs
of a task will meet these constraints when the best-case and worst-case response times do not exceed these bounds. The upper bound on the response times of a task $\tau_i$ is typically denoted by a (relative) deadline $D_i$, and the lower bound is typically (implicitly) assumed to be zero. The seminal work on response time analysis by Harter [7, 8] already covers this pair of response times for strictly periodic tasks, albeit only for deadlines equal to periods. Based on this pair, an upper bound on the worst-case (absolute) response jitter $R_I$ of $\tau_i$ can be determined. When the timing constraints on a task include an upper bound on the variation of the response times of its jobs, then the task will meet this constraint when its response jitter does not exceed this bound. Worst-case response analysis was extended by Tindell et al. to cover activation jitter, deadlines unequal to periods, and sporadic tasks in [1, 20], and to cover distributed hard real-time systems in [19]. The need for best-case response time analysis in the area of distributed systems was identified in [11, 17]. Exact best-case response times and worst-case (absolute) finalization (or completion) jitter has been presented in [3, 18], albeit for periodic tasks with deadlines at most equal to periods minus activation jitter.

Next to response times, we are interested in execution times, where the best-case execution time $BE_i$ and worst-case execution time $WE_i$ of a task $\tau_i$ are the shortest and longest interval of time from the actual start of that task to the corresponding completion, respectively. Based on this pair, an upper bound on the worst-case (absolute) execution jitter $EJ_i$ of task $\tau_i$ can be determined. When the timing constraints on a task include an upper bound on the variation of the execution times of its jobs, then the task will meet this constraint when its execution jitter does not exceed this bound. The focus of this paper is on best-case and worst-case execution times and execution jitter of tasks with deadlines at most equal to (worst-case) periods minus activation jitter. Although execution times and execution jitter do not seem as interesting as response times and response jitter, there are sensible applications of the former in the area of real-time control, as discussed in the next section.

1.2. A specific application: real-time control

Control activities in a real-time control system are typically periodic and consist of three main parts: sampling, control computation and actuation. When all three parts of a control activity are performed by a single periodic task in a computing system, we can distinguish three main types of jitter caused by fluctuations in the computation times of its jobs and by the interference of tasks with a higher priority than the task under consideration, i.e.

- **sampling jitter**: time intervals between consecutive sampling points may vary;
- **sampling-actuation jitter**: time intervals between sampling and corresponding actuation may vary;
- **actuation jitter**: time intervals between consecutive actuation points may vary.

When the jitter is not properly taken into account in the control computation, it may degrade the performance of the system and even jeopardize its stability. Several approaches are reported upon in the literature to tackle the problem of jitter, based on jitter reduction and jitter compensation. Three typical techniques to reduce jitter are described, addressed and evaluated in [4]. An example of jitter compensation can be found in [15]. Rather than reducing jitter or compensating for jitter, we focus on jitter analysis in general and on analysis of sampling-actuation jitter in particular. To this end, we assume that sampling and actuation are performed at the start and completion of a job, respectively. We now use the (absolute) execution jitter of a control task to characterize sampling-actuation jitter.

Although control tasks are typically (strictly) periodic tasks, there are situations where control tasks execute at different periods in different operating conditions [6]. In this paper, we therefore assume that control activities can also be performed by elastic and sporadic tasks, and briefly discuss their execution in multiple operating modes.

1.3. Contributions

In this paper, we define novel notions of worst-case and best-case execution times of tasks. We present worst-case and best-case execution times and (absolute) execution jitter analysis of independent, periodically activated, hard real-time tasks that are executed on a single processor under FPPS, arbitrary phasing, (absolute) activation jitter, and deadlines at most equal to (worst-case) periods minus activation jitter. Our analysis is based on a continuous scheduling model, e.g. all task parameters are taken from the real numbers. We prove that the worst-case and best-case execution time of a task are equal to its worst-case and best-case response time, respectively. We present an upper bound for the execution jitter of a task expressed in terms of its worst-case and best-case execution time, and illustrate that this bound is tight for an example task set.

We briefly discuss execution times and execution jitter in other settings. In particular, we describe how to determine best-case and worst-case execution times and execution jitter in a distributed multiprocessor system with task dependencies. We also consider tasks with deadlines larger than (worst-case) periods minus activation jitter, discuss multiple operating modes of a task, and comment on the impact of the scheduling model on our results.
1.4. Overview
The remainder of this paper is organized as follows. We start by giving a scheduling model for FPPS in Section 2. In Section 3, we briefly recapitulate response times and response jitter analysis. Execution times and execution jitter analysis are the topic of Section 4. Section 5 presents execution times and execution jitter analysis for distributed multiprocessors with task dependencies. We discuss arbitrary deadlines, multiple operating modes of a task, and the impact of the scheduling model in Section 6. Finally, we conclude the paper in Section 7.

2. A scheduling model for FPPS
This section presents a basic real-time scheduling model for FPPS. We start with basic terminology and concepts for a set of independent periodic tasks under FPPS on a single processor. We subsequently define derived notions, i.e., best-case and worst-case notions of response times and (absolute) response jitter for tasks, best-case and worst-case execution times and (absolute) execution jitter for tasks, and utilization factors for tasks and sets of tasks.

2.1. Basic notions
We assume a single processor and a set \( T \) of \( n \) independent, periodically activated tasks \( \tau_1, \tau_2, \ldots, \tau_n \) with unique, fixed priorities. At any moment in time, the processor is used to execute the highest priority task that has work pending. So, when a task \( \tau_i \) is being executed, and a release occurs for a higher priority task \( \tau_j \), then the execution of task \( \tau_i \) is preempted, and will resume when the execution of \( \tau_j \) has ended, as well as all other releases of tasks with a higher priority than \( \tau_i \) that have taken place in the meanwhile.

Each task \( \tau_i \) generates an infinite sequence of jobs \( t_{ik} \) with \( k \in \mathbb{Z} \). We distinguish three types of tasks based on the characteristics of the inter-activation times of its jobs. The inter-activation times of a periodic task \( \tau_i \) are characterized by a period \( T_i \in \mathbb{R} \) and an (absolute) activation jitter \( AJ_i \in \mathbb{R}^+ \cup \{0\} \), of a sporadic task \( \tau_i \) by a worst-case (or minimum) period \( WT_i \in \mathbb{R}^+ \) and of an elastic task \( \tau_i \) by a worst-case (or minimum) period \( WT_i \in \mathbb{R}^+ \) and a best-case (or maximum) period \( BT_i \in \mathbb{R}^+ \), where \( WT_i < BT_i \). Moreover, each task \( \tau_i \) is characterized by a worst-case computation time \( WC_i \in \mathbb{R}^+ \), a best-case computation time \( BC_i \in \mathbb{R}^+ \), where \( BC_i \leq WC_i \), a phasing \( \phi_i \in \mathbb{R} \), and a (relative) deadline \( D_i \in \mathbb{R}^+ \). The set of phasings \( \phi_i \) is termed the phasing \( \varphi \) of the task set \( T \). The deadline \( D_i \) is relative to the activations. We assume \( D_i \leq T_i - AJ_i \) for a period task and \( D_i \leq WT_i \) for sporadic and elastic tasks, since otherwise there may be too little time between successive activations to complete the task. For ease of presentation, we will use \( WT_i \) and \( BT_i \) to denote the period \( T_i \) of a periodic task \( \tau_i \), use \( AJ_i \) with a value equal to zero for a sporadic task or elastic task \( \tau_i \), and use \( BT_i \) with a value going to infinity for a sporadic task.

Note that the activations of a periodic task \( \tau_i \) do not necessarily take place strictly periodically, with period \( T_i \), but we assume they take place somewhere in an interval of length \( AJ_i \) that is repeated with period \( T_i \). The activation times \( a_{ik} \) of a periodic task \( \tau_i \) satisfy

\[
\sup_{k,l}(a_{ik}(\varphi_i) - a_{il}(\varphi_i) - (k-l)T_i) \leq AJ_i,
\]

where \( \varphi_i \) denotes the start of the interval of length \( AJ_i \) in which job zero is activated, i.e., \( \varphi_i + kT_i \leq a_{ik} \leq \varphi_i + AJ_i + kT_i \). Hence, consecutive activation times satisfy

\[
T_i - AJ_i \leq a_{i,k+1}(\varphi_i) - a_{ik}(\varphi_i) \leq T_i + AJ_i.
\]

A periodic task with activation jitter equal to zero is termed a strictly periodic task. The activation times of a sporadic or elastic task \( \tau_i \) satisfy

\[
WT_i \leq a_{i,k+1}(\varphi_i) - a_{ik}(\varphi_i) \leq BT_i,
\]

where \( \varphi_i \) now denotes the activation time \( a_{i,0} \) of job zero.

The (absolute) deadline of job \( t_{ik} \) takes place at \( d_{ik} = a_{ik} + D_i \). The (absolute) begin time \( b_{ik} \) and (absolute) finalization time \( f_{ik} \) of \( t_{ik} \) are the times at which \( \tau_i \) actually starts and ends the execution of that job, respectively. The active (or response)
interval of job $t_{ik}$ is defined as the time span between the activation time of that job and its finalization time, i.e. $[a_{ik}, f_{ik}]$. The response time $R_{ik}$ of $t_{ik}$ is defined as the length of its active interval, i.e. $R_{ik} = f_{ik} - a_{ik}$. Similarly, the start interval and the execution interval of job $t_{ik}$ are defined as the time span between the activation time and the begin time of that job and the begin time and the finalization time of that job, respectively. The (relative) start time $S_{ik}$ and the execution time $E_{ik}$ of a job $t_{ik}$ are defined as the length of its start interval and execution interval, respectively, i.e. $S_{ik} = b_{ik} - a_{ik}$ and $E_{ik} = f_{ik} - b_{ik}$. Figure 1 illustrates the above notions for an example job $t_{ik}$ of a periodic task $\tau_i$.

We assume that we do not have control over the phasing $\phi$, for instance since the tasks are released by external events, so we assume that any arbitrary phasing may occur. This assumption is common in real-time scheduling literature [10, 12, 14]. We also assume other standard basic assumptions, i.e. tasks are ready to run upon their activation and do not suspend themselves, tasks will be preempted instantaneously when a higher priority task becomes ready to run, a job of a task $\tau_i$ does not start before its previous job is completed, and the overhead of context switching and task scheduling is ignored. Finally, we assume that the deadlines are hard, i.e. each job of a task must be completed at or before its deadline.

For notational convenience, we assume that the tasks are indexed in order of decreasing priority, i.e. task $\tau_1$ has highest priority and task $\tau_n$ has lowest priority. For ease of presentation the worst-case and best-case computation times of tasks in the examples are identical, i.e. $WC_i = BC_i$, and we then simply use $C_i$.

### 2.2. Derived notions

The worst-case response time $WR_i$ and the best-case response time $BR_i$ of a task $\tau_i$ are the largest and the smallest response time of any of its jobs, respectively, i.e.

$$WR_i = \sup_{\varphi, k} R_{ik}(\varphi),$$

(4)

$$BR_i = \inf_{\varphi, k} R_{ik}(\varphi).$$

(5)

Note that the response time $R_{ik}$ has been parameterized in these equations to denote its dependency on the phasing $\varphi$. A critical instant [14] and an optimal (or favourable) instant [3, 18] of a task are defined to be (hypothetical) instants that lead to the worst-case and best-case response time for that task, respectively. The worst-case (absolute) response jitter $RI_i$ of a task is the largest difference between the response times of any of its jobs, respectively, i.e.

$$RI_i = \sup_{\varphi, k, l} (R_{ik}(\varphi) - R_{il}(\varphi)).$$

(6)

The notions derived from execution times are similar to those derived from response times. The worst-case execution time $WE_i$ and the best-case execution time $BE_i$ of a task $\tau_i$ are the largest and the smallest execution time of any of its jobs, respectively, i.e.

$$WE_i = \sup_{\varphi, k} E_{ik}(\varphi),$$

(7)

$$BE_i = \inf_{\varphi, k} E_{ik}(\varphi).$$

(8)

The worst-case (absolute) execution jitter $EI_i$ of a task is the largest difference between the execution times of any of its jobs, respectively, i.e.

$$EI_i = \sup_{\varphi, k, l} (E_{ik}(\varphi) - E_{il}(\varphi)).$$

(9)

Note that we assume arbitrary phasing for our notions of execution times and execution jitter. Conversely, [5] and [4] (implicitly) assume a specific phasing for their notions of execution jitter and input jitter, respectively. As an example, the absolute execution jitter $AEJ_i$ in [5] is defined as

$$AEJ_i = \max_k E_{ik} - \min_k E_{ik}.$$  

(10)

To analyse jitter in distributed multiprocessor systems with task dependencies [11, 17], i.e. where the finalization of a task on one processor may activate a following task on another processor, we define the worst-case (absolute) finalization jitter $FJ_i$ of a periodic task $\tau_i$ as

$$FJ_i = \sup_{\varphi, k, l} (f_{ik}(\varphi) - f_{il}(\varphi) - (k-l)T_i).$$

(11)
The \textit{(processor) utilization factor} $U^T$ of a task set $T$ is the fraction of the processor time spent on the execution of that task set [14]. The fraction of the processor time spent on executing a periodic task $\tau_i$ with a fixed computation $C_i$ is $C_i/T_i$, and is termed the \textit{utilization} $U^T_\tau$ of $\tau_i$. The \textit{cumulative utilization factor} $U^T_T$ for periodic tasks $\tau_1$ till $\tau_i$ with fixed computation times is the fraction of processor time spent on executing these tasks, and is given by

$$U^T_T = \sum_{j \leq i} U^T_\tau = \sum_{j \leq i} \frac{C_j}{T_j}. \quad (12)$$

Therefore, $U^T_T$ is equal to the cumulative utilization factor $U^T_T$ for $n$ periodic tasks.

Because we distinguish best-case and worst-case computation times for tasks in general and best-case and worst-case periods for (sporadic and) elastic tasks in particular in this paper, we get similar best-case and worst-case versions for the various notions of utilization. A \textit{necessary} schedulability condition for $T$ is now given by

$$WU^T_T \leq 1. \quad (13)$$

In the remainder of this document, we assume that task sets satisfy (13). In some cases, we are also interested in, for example, the worst-case response time as a function of the computation time or as a function of the phasing. We will therefore use a functional notation when needed, e.g. $WR_i(C)$ or $WR_i(\varphi)$.

![Timeline of $T_1$ with critical instants for all periodic tasks at time $t = 0$ and an optimal instant for $\tau_3$ at time $t = 60$. The numbers to the top right corner of the boxes denote the response time of the respective releases.](image)

\textbf{Figure 2.} Timeline of $T_1$ with critical instants for all periodic tasks at time $t = 0$ and an optimal instant for $\tau_3$ at time $t = 60$. The numbers to the top right corner of the boxes denote the response time of the respective releases.

\section*{3. Recapitulation of existing results}

Based on [1, 3, 9, 13, 14, 18, 20], we briefly recapitulate worst-case and best-case response times and response jitter and finalization jitter analysis of hard real-time tasks under FPPS. For illustration purposes, we use an example set $T_1$ of periodic tasks with characteristics as given in Table 1. Without further elaboration, we generalize existing results for periodic tasks to also cover sporadic and elastic tasks.

<table>
<thead>
<tr>
<th>Task</th>
<th>AJ</th>
<th>D</th>
<th>C</th>
<th>WR</th>
<th>BR</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>6</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>10</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>54</td>
<td>3</td>
<td>51</td>
<td>7</td>
<td>21</td>
</tr>
</tbody>
</table>

\textbf{Table 1.} Task characteristics of $T_1$ and worst-case and best-case response times of periodic tasks.

\subsection*{3.1. Worst-case response times}

A \textit{critical instant} for a task $\tau_i$ is assumed when $\tau_i$ is simultaneously released with all tasks with a higher priority than $\tau_i$, $\tau_i$ experiences a worst-case computation time at that simultaneous release, and all those tasks with a higher priority experience a worst-case computation time at that simultaneous release and subsequent releases, and experience minimal inter-activation times between consecutive releases starting from that simultaneous release. Figure 2 shows a timeline of $T_1$ with critical instants for all periodic tasks at time $t = 0$.

The \textit{worst-case response time} $WR_i$ of $\tau_i$ is given by the smallest $x \in \mathbb{R}^+$ that satisfies

$$x = WC_i + \sum_{j < i} \left[ \frac{x + AJ_i}{WT_i} \right] WC_j. \quad (14)$$
Such a smallest value exists for $\tau_i$ if and only if $W_u^{T_{i-1}} < 1$. Because we assume $W_u^T \leq 1$ and $W_c > 0$ for all tasks, $W_u^{T_{i-1}} < 1$ holds for all tasks. To calculate $W_R$, we can use an iterative procedure based on recurrence relationships, starting with a lower bound, and $\sum_{j \leq i} W_c$ is an appropriate initial value. The procedure is stopped when the same value is found for two successive iterations or when the deadline is exceeded. In the latter case, $\tau_i$ is not schedulable.

### 3.2. Best-case response times

An optimal instant is assumed when the completion of task $\tau_i$ coincides with the simultaneous release of all tasks with a higher priority than $\tau_i$, the completed job of $\tau_i$ experiences a best-case computation time, and all those tasks with a higher priority experience a best-case computation time for releases prior to that simultaneous release, and experience a maximal inter-activation time between consecutive releases prior to and ending with that simultaneous release and its previous release.

Figure 2 shows a timeline of $T_i$ with an optimal instant for task $\tau_3$ at time $t = 60$.

The best-case response time $B_R$ of a task $\tau_i$ is given by the largest $x \in \mathbb{R}^+$ that satisfies

$$x = B_C + \sum_{j < i} \left( \left\lceil \frac{x - A_{j,i}}{B_{T_i}} \right\rceil - 1 \right) B_{C,j},$$

where the notation $w^+$ stands for $\max(w, 0)$. Such a largest value exists for task $\tau_i$ if and only if $B_u^{T_{i-1}} < 1$. Because $B_u^T \leq W_u^T$ holds by definition, and we assume $W_u^T \leq 1$ and $B_C > 0$ for all tasks, the relation $B_u^{T_{i-1}} < 1$ trivially holds for all tasks. To calculate $B_R$, we can use an iterative procedure based on recurrence relationships, starting with an upper bound, and $W_R$ is an appropriate initial value. The procedure is stopped when the same value is found for two successive iterations.

Note that sporadic tasks do not contribute to the summation term in (15), because their best-case periods go to infinity.

### 3.3. Response jitter

The (absolute) response jitter $R_{J_i}$ of a task $\tau_i$ is bounded by

$$R_{J_i} \leq W_R - B_R,$$

(16)

This bound is tight for task $\tau_3$ of our example, as shown in Figure 2. In general, however, the bound in (16) on $R_{J_i}$ need not be tight, as $W_R$ and $B_R$ are not necessarily taken on for the same phasing.

### 3.4. Finalization jitter of periodic tasks

The (absolute) finalization jitter $F_{J_i}$ of a periodic task $\tau_i$ is bounded by

$$F_{J_i} \leq A_{J_i} + W_R - B_R,$$

(17)

This bound is tight for task $\tau_3$ of our example, as illustrated by Figure 2. Similar to response jitter, the bound in (17) on $F_{J_i}$ need not be tight.

From (11) and $R_{ik}(\varphi) = f_{ik}(\varphi) - a_{ik}(\varphi)$, we derive

$$F_{J_i} = \sup_{\varphi, k,l} \left( (R_{ik}(\varphi) + a_{ik}(\varphi)) - (R_{il}(\varphi) + a_{il}(\varphi)) - (k - l)T_i \right)$$

$$= \sup_{\varphi, k,l} \left( (a_{ik}(\varphi) - a_{il}(\varphi)) - (k - l)T_i + R_{ik}(\varphi) - R_{il}(\varphi) \right).$$

Hence, for a strictly periodic task $\tau_i$, the finalization jitter and response jitter are the same, i.e. $F_{J_i} = R_{J_i}$. Moreover, the bounds on the finalization jitter and the response jitter as given by (17) and (16), respectively, are also the same.

### 4. Execution times and execution jitter

This section presents theorems for the three execution notions introduced in Section 2.2, i.e. worst-case execution time in Section 4.1, best-case execution time in Section 4.2, and (absolute) execution jitter in Section 4.3. Throughout this section, we assume that all tasks are schedulable.
4.1. Worst-case execution time

The next theorem states that the worst-case execution time of a task under arbitrary phasing is equal to its worst-case response time. Firstly, we prove the following lemma.

**Lemma 1.** The worst-case execution time \( WE_i \) of a task \( \tau_i \) is at most equal to its worst-case response time \( WR_i \), i.e.

\[
WE_i \leq WR_i.
\] (18)

**Proof.** The proof follows immediately from the definition of worst-case execution time \( WE_i \), i.e.

\[
WE_i = \sup_{\varphi,k} E_{ik}(\varphi) = \sup_{\varphi,k} (R_{ik}(\varphi) - S_{ik}(\varphi)) \\
\leq \sup_{\varphi,k} R_{ik}(\varphi) - \inf_{\varphi,k} S_{ik}(\varphi) \leq \sup_{\varphi,k} WR_i(\varphi) = \{ (4) \} WR_i.
\]

\( \square \)

**Theorem 1.** The worst-case execution time \( WE_i \) of a task \( \tau_i \) is given by

\[
WE_i = WR_i.
\] (19)

**Proof.** The proof is by construction. Consider a simultaneous release of task \( \tau_i \) and all its higher priority tasks (i.e. a critical instant of \( \tau_j \)) at time \( t = 0 \). Let this release of \( \tau_i \) be denoted by job \( k \). There exists a resume time \( t_0 < f_{ik} \), such that all preemptions of tasks with a higher priority than \( \tau_i \) are completed, and the last part of job \( k \) lasting a time \( x_0 \) can be executed without preemptions between time \( t_0 \) and \( f_{ik} \); see Figure 3.

![Figure 3. Release of task \( \tau_i \) at a critical instant at time \( t = 0 \).](image)

We now move the release of task \( \tau_i \) an amount of time \( x_0/2 \) backwards in time, i.e. \( a'_k = a_k - x_0/2 = -x_0/2 \). Because the response time of job \( k \) already was the worst-case response time \( WR_i \), moving its release backwards in time cannot increase its response time. Hence, job \( k \) can immediately start executing at time \( -x_0/2 \), and the length of the start interval becomes zero, i.e. \( S'_k = b'_k - a'_k = 0 \). The response time of \( k \) cannot decrease either, because an amount \( x_0/2 \) is still to be executed in the interval \( [f_{ik} - x_0, f_{ik} - x_0/2) \), i.e. job \( k \) completes at \( f'_k = f_{ik} - x_0/2 \). As a result, the length of the execution interval equals the response time, i.e. \( E'_k = f'_k - s'_k = f_{ik} - x_0/2 - (a_k - x_0/2) = R_{ik} \). By keeping the response time of job \( k \) the same, we therefore constructed an execution interval equal to the worst-case response time, so \( WE_i \geq WR_i \); see Figure 4. Together with the relation \( WE_i \leq WR_i \), this proves the lemma. \( \square \)

![Figure 4. Release of task \( \tau_i \) at \( t = -x_0/2 \) and a simultaneous release of all higher priority tasks at time \( t = 0 \).](image)

From the proof of Theorem 1, we draw the following conclusion.

**Corollary 1.** The worst-case execution time \( WE_i \) of a task \( \tau_i \) is not assumed when that task is simultaneously released with all its higher priority tasks. Instead, it is assumed when that task \( \tau_i \) is released just before the simultaneous release of all its higher priority tasks. \( \square \)
4.2. Best-case execution time

Similar to the worst-case execution time, the next theorem states that the best-case execution time of a task under arbitrary phasing is equal to its best-case response time. First, we prove the dual of Lemma 1.

**Lemma 2.** The best-case execution time $BE_i$ of a task $\tau_i$ is at most equal to its best-case response time $BR_i$, i.e.

$$BE_i \leq BR_i. \quad (20)$$

**Proof.** The proof is similar to the proof of Lemma 1. □

**Theorem 2.** The best-case execution time $BE_i$ of a task $\tau_i$ is given by

$$BE_i = BR_i. \quad (21)$$

**Proof.** As described in [3, 18], a job of a task experiencing a best-case response time starts right upon its release. The theorem therefore immediately follows from the proof of the recursive equation for the best-case response time. □

From this theorem, we draw the following conclusion.

**Corollary 2.** The best-case execution time $BE_i$ of a task $\tau_i$ is assumed when the completion of that task coincides with the simultaneous release of all its higher priority tasks. □

4.3. Execution jitter

We derive the following bound for absolute execution jitter.

**Theorem 3.** The worst-case (absolute) execution jitter $EJ_i$ is bounded by

$$EJ_i \leq WE_i - BE_i. \quad (22)$$

**Proof.** The proof immediately follows from its definition.

$$EJ_i = \{E_i \} \sup_{\varphi,k,l} (E_{ik}(\varphi) - E_{il}(\varphi))$$

$$\leq \sup_{\varphi,k} E_{ik}(\varphi) - \inf_{\varphi,l} E_{il}(\varphi) = \{(7) \text{ and } (8)\} \{WE_i - BE_i\}.$$ □

This bound is tight for task $\tau_3$ of our example, which we will illustrate using Figure 2. Consider the job of $\tau_3$ that is released at time $t = 0$. We now move the release of that job an amount $x_0/2 = 1/2$ backwards in time, causing it to experience a worst-case execution time equal to its worst-case response time. When the next job remains released at time $t = 51$, we constructed a situation where $\tau_3$ experiences both a best-case and worst-case execution time.

Similar to response jitter, the bound in (22) on $EJ_i$ need not be tight in general, as $WE_i$ and $BE_i$ are not necessarily taken on for the same phasing. Note that the upper bounds on the response jitter and execution jitter as given by (16) and (22), respectively, are the same.

5. Distributed multiprocessor systems

In this section, we present execution times and execution jitter analysis for distributed multiprocessor systems with task dependencies, i.e. where the finalization of a task on one processor may activate a following task on another processor. We first show that a dependent task inherits the type of the task that activates it. Next, we describe how to determine execution times and execution jitter in such a system.
5.1. Type of a dependent task

The inter-activation times of a task \( \tau'_j \) that depends on a periodic task \( \tau_i \) are characterized by a period \( T'_j = T_i \) and an activation jitter \( A_{J}^\prime_j = F_{J_i} \) [3, 17]. Hence, a task that depends on a periodic task is also a periodic task. We will show that a dependent task always inherits the type of the task that activates it. To that end, we determine a lower bound for the worst-case period of a task that depends on a sporadic or elastic task, and an upper bound for the best-case period of a task that depends on an elastic task.

**Lemma 3.** The worst-case period \( WT'_j \) of a task \( \tau'_j \) that depends on a sporadic or elastic task \( \tau_i \) is bounded by

\[
WT'_j \geq WT_i + BR_i - WR_i.
\]

**Proof.** The worst-case period \( WT'_j \) of \( \tau'_j \) is the smallest inter-finalization time of two consecutive jobs of \( \tau_i \), i.e.

\[
WT'_j = \inf_{\varphi, k}(f_{i,k+1}(\varphi) - f_{i,k}(\varphi)).
\]

From (3), (4), and (5), and \( R_{ik}(\varphi) = f_{ik}(\varphi) - a_{ik}(\varphi) \), we derive

\[
WT'_j = \inf_{\varphi, k}(a_{i,k+1}(\varphi) - a_{i,k}(\varphi) + R_{i,k+1}(\varphi) - R_{i,k}(\varphi))
\]

\[
\geq WT_i + BR_i - WR_i,
\]

which proves the lemma.

Because the inter-activation times of jobs of a sporadic task have no upper bound, the inter-finalization times of jobs of a sporadic task also has no upper bound. A task that depends on a sporadic task is therefore also a sporadic task.

**Lemma 4.** The best-case period \( BT'_j \) of a task \( \tau'_j \) that depends on an elastic task \( \tau_i \) is bounded by

\[
BT'_j \leq BT_i + WR_i - BR_i.
\]

**Proof.** The proof is similar to the proof of Lemma 3.

From the former, we draw the following conclusion.

**Corollary 3.** A dependent task inherits the type of the task that activates it.

5.2. Execution times and execution jitter analysis

For a distributed multiprocessor system with task dependencies, we can use the following iterative procedure to take the effect of jitter into account, which extends the procedure described in [3, 17]. We start with inter-activation time characteristics of dependent tasks equal to the characteristics of their activating tasks, and calculate worst-case and best-case response times on each processor. We then determine the finalization jitter bound of each periodic task, as given by (17). Next, we update the estimate of the activation jitter of each task that is triggered by a periodic task, by making it equal to the finalization jitter of the triggering task, update the worst-case period of each task that is triggered by a sporadic or elastic task using the bound of (23), and update the best-case period of each task that is triggered by an elastic task using the bound of (25). With this new estimates, we then again determine the worst-case and best-case response times. We repeat this process until we obtain stable values or until either the computed response times exceed their deadlines or the worst-case periods become at most equal to zero. If we obtained stable values for the response times, these values also represent the execution times. We can subsequently determine the execution jitter bound of each task as given by (22).

During this process, the finalization jitter bounds, the worst-case response times and the best-case periods increase, and the best-case response times and worst-case periods decrease, again causing the jitter bounds to increase, etc. This monotonicity, together with the fact that the response times are bounded, and that they can only take on a finite number of values, implies termination in a finite number of steps. Furthermore, this shows that if we redetermine the worst-case and best-case response times for new estimates of the finalization jitter ad the best-case and worst-case periods, we can use their final values of the previous iteration for initialization.
6. Discussion

In Section 4, we presented execution times and jitter analysis of tasks with deadlines at most equal to (worst-case) periods minus activation jitter based on a continuous scheduling model. In this section, we first consider tasks with deadlines that exceed those values. Next, we discuss tasks with multiple operating modes. We conclude this section with a remark on the impact of the scheduling model on our results.

6.1. Arbitrary deadlines

For deadlines larger than periods minus activation jitter, the worst-case execution time and the best-case execution time can be smaller than the worst-case response time and best-case response time, respectively. We will illustrate this using an example task set \( T_2 \) consisting of two tasks with characteristics as given in Table 2. The table includes the results of the exploration of the response times and execution times. Note that the (processor) utilization factor \( U = \frac{2}{5} + \frac{4}{5} = 1 \).

<table>
<thead>
<tr>
<th>task</th>
<th>( T )</th>
<th>( AJ )</th>
<th>( D )</th>
<th>( C )</th>
<th>( WR )</th>
<th>( BR )</th>
<th>( WE )</th>
<th>( BE )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tau_1 )</td>
<td>28.4</td>
<td>7.4</td>
<td>7.4</td>
<td>8.6</td>
<td>35.4</td>
<td>10.6</td>
<td>8.6</td>
<td>6.2</td>
</tr>
<tr>
<td>( \tau_2 )</td>
<td>28.4</td>
<td>7.4</td>
<td>7</td>
<td>8.6</td>
<td>35.4</td>
<td>10.6</td>
<td>8.6</td>
<td>6.2</td>
</tr>
</tbody>
</table>

Table 2. Task characteristics of \( T_2 \) and worst-case and best-case response times and execution times.

Figure 5 shows a timeline with the executions of the tasks of \( T_2 \) in an interval of length 35, i.e. equal to the hyperperiod of the tasks. Because both tasks have a simultaneous release at time zero, that time point is a critical instant. Based on [12, 20], we therefore conclude that the job of task \( \tau_2 \) with the longest response in \([0, 35]\) experiences a worst-case response time \( WR_2 \), i.e. \( WR_2 = 8.6 \).

Figure 6 shows a timeline for \( T_2 \) with \( \phi_R = 0.4 \). For this phasing, \( \tau_2 \) experiences an optimal instant at time \( t = 35 \), corresponding with the completion of its 3rd job, i.e. \( t_{3,4} \). That job is released at time 28.4, and the best-case response time \( BR_{3,4}(0.4) \) for the relative phasing \( \phi_R = 0.4 \) is therefore equal to \( f_{2.5} - a_{2.5} = 35 - 28.4 = 6.6 \). Similar to the example with arbitrary deadlines in [18], the job experiencing the best-case response time cannot immediately start its execution upon its release, but is delayed by a previous job. In this case, the best-case response time determined by the technique in Section 3.2 yields a lower bound, being 6.2.

The worst-case and best-case response times of task \( \tau_2 \) are shown as functions of the relative phasing \( \phi_R \) in Figure 7(a). A remarkable aspect of this example is that for every relative phasing \( \phi_R \), the response jitter \( RJ_2 \) is the same, i.e. \( RJ_2 = \sup_{\phi_R} (WR_2(\phi_R) - BR_2(\phi_R)) = 1.6 \). Moreover, the response jitter bound (16) is not tight for this example, because the worst-case and best-case response times are not assumed for the same phasing, i.e. \( RJ_2 < WR_2 - BR_2 = 8.6 - 6.6 = 2.0 \).

Considering figures 5 and 6, it turns out that every execution interval of task \( \tau_2 \) is pre-empted either once or twice. For this particular example, both the worst-case and best-case execution times are independent of the relative phasing \( \phi_R \), as illustrated in Figure 7(b). As a result, the execution jitter is also constant, i.e. \( EJ_2(\phi_R) = 2.0 \). For this example, the execution jitter bound (22) is tight, i.e. \( EJ_2 = WE_2 - BE_2 = 2.0 \).
and the best-case execution time can be smaller than the worst-case response time and best-case response time, respectively. For this particular example, the worst-case execution time is equal to the smallest positive solution of (14), and the best-case execution time is equal to the largest positive solution of (15). Without formal proof, we merely state by means of the following conjectures that this is always the case.

**Conjecture 1.** The worst-case execution time of a task $\tau_i$ is given by the smallest positive solution of (14). □

**Conjecture 2.** The best-case execution time of a task $\tau_i$ is given by the largest positive solution of (15). □

We briefly sketch an argument for these conjectures. In Section 2.1, we assumed that a job of task $\tau_i$ does not start before its previous job is completed, i.e. $b_{ik} \geq f_{i,k-1}$. Hence, the execution interval of a job can be delayed, but can never be pre-empted by a previous job. The execution interval can therefore only be pre-empted by higher priority tasks. We now determine the maximum and minimum amount of pre-emptions of an execution interval by higher priority tasks. To that end, let $t' = \max(a_{ik}, f_{i,k-1})$ denote a lower bound on the begin time of $t_{ik}$, i.e. $b_{ik} \geq t'$. The interval $[t', f_{ik})$ only contains the execution of $t_{ik}$ and its pre-emptions by higher priority tasks. Because those executions of those higher priority tasks in $[t', f_{ik})$ are not influenced by $\tau_i$, both the maximum and minimum amount of pre-emptions of such an interval are independent of the deadline of $\tau_i$, i.e. we can apply the same reasoning for $D_i > T_i - A \tau_i$ as for $D_i \leq T_i - A \tau_i$. The maximum amount of pre-emption is therefore bounded from above by the largest positive solution of (14) minus the worst-case computation time $WC_i$. Hence, the worst-case execution time of a task is bounded from above by the largest positive solution of (14). For the worst-case the result now follows immediately from a proof by construction similar to the proof of Theorem 1.

Similarly, the best-case execution time of a task is bounded from above by the largest positive solution (15). Now assume that a job of $\tau_i$ has a completion that coincides with the simultaneous release of all higher priority tasks, i.e. that job experiences an optimal instant. Given such an instant, we can construct an execution interval that has a minimal pre-emption [3], and the result follows.

### 6.2. Tasks with multiple operating modes

In Section 2, we distinguished three types of tasks based on the (static) characteristics of the inter-activation times of jobs. However, there are also situations where a control task executes at different rates in different operating conditions, i.e. where a task can execute in different operating modes. Below, we briefly discuss the consequences of multiple modes for the analysis, assuming a single processor.

Consider either a sporadic or elastic task $\tau_i$ with $m_i$ operating modes $M_{i,1}, M_{i,2}, \ldots, M_{i,m_i}$. Each mode $M_{ij}$ is characterized by a period $T_{ij} \in \mathbb{R}^+$, a worst-case computation time $WC_{ij} \in \mathbb{R}^+$, a best-case computation time $BC_{ij} \in \mathbb{R}^+$, and timing constraints. The static characteristics of $\tau_i$ are derived from the mode characteristics, i.e. $WT_i = \min_{1 \leq j \leq m_i} T_{ij}$, $BT_i = \max_{1 \leq j \leq m_i} T_{ij}$ for an elastic task and goes to infinity for a sporadic task, $WC_i = \max_{1 \leq j \leq m_i} WC_{ij}$, and $BC_i = \min_{1 \leq j \leq m_i} BC_{ij}$. Note that we omit the phasing $\phi_i$, because we assume arbitrary phasing. Timing constraints are associated with each of the modes of a task, e.g. mode $M_{ij}$ of $\tau_i$ can have a relative deadline $D_{ij}$ and an upper bound $E_{ij}$ on the variation of the execution times as timing constraints. To determine whether or not these timing constraints of $\tau_i$ are met, we need to determine the worst-case response time and execution jitter of $\tau_i$ for each operating mode $M_{ij}$. To analyse
mode $M_{ij}$ for $\tau_i$, we use the characteristics of that mode and the static characteristics of all tasks with a higher priority than $\tau_i$. Hence, the mode characteristics are determining for the analysis of the task itself and the static characteristics of a task are determining for the analysis of tasks with a lower priority.

6.3. Discrete scheduling model

In Section 2, we assumed a continuous rather than a discrete scheduling model, e.g. all task parameters are taken from the real numbers. For a discrete scheduling model, all task parameters are integral numbers and tasks are released (and pre-empted) at integer values of time.

For a discrete scheduling model, our results for worst-case execution time can be pessimistic. As an example, reconsider Figure 2. The release of task $\tau_3$ at time $t=0$ experiences a worst-case response time $WR_3 = 21$. The final consecutive execution of that job has a length equal to $x_0 = 1$. Unlike the construction used in the proof of Theorem 1, we cannot move the release of that job $x_0/2 = 1/2$ backwards in time. When the release is moved a minimal amount $x_0 = 1$ backwards in time, the response time of the job is reduced to $WR_3(WC_2 - 1) + 1 = WR_3(6) + 1 = 17$. As a result, the worst-case execution time $WE_i$ for $\tau_i$ with $D_i \leq T_i - AJ_i$ can be shown to be given by

$$WE_i = \begin{cases} 1 & \text{if } WC_i = 1 \\ WR_i(WC_i - 1) + 1 & \text{otherwise}. \end{cases}$$

(26)

For $WC_i > 1$, $WE_i$ is equal to $WR_i$ if and only if

$$WR_i(WC_i) = WR_i(WC_i - 1) + 1.$$  

(27)

Our results do not change for best-case execution times.

7. Conclusion

Various types of jitter are of interest in real-time control systems. In this paper, we focussed on (absolute) execution jitter, which characterizes the maximum variation of the sampling-actuation delay among all tasks of a control task. We defined notions of worst-case and best-case execution times of tasks, and proved that the worst-case and best-case execution time of a task are equal to the worst-case and best-case response time of that task, respectively. We expressed an upper bound for the execution jitter of a task in terms of its worst-case and best-case execution times, and illustrated by means of an example task set that this bound is tight.

Our analysis assumes a continuous scheduling model with independent, periodically released, hard real-time tasks that are executed on a single processor under FPPS, and deadlines at most equal to (worst-case) periods minus activation jitter. Moreover, it distinguishes three types of tasks based on the inter-activation times of jobs, i.e. periodic tasks, sporadic tasks, and elastic tasks. We briefly discussed execution times and execution jitter in other settings. In particular, we described how to determine best-case and worst-case execution times and execution jitter in a distributed multiprocessor system with task dependencies. We showed that a dependent task inherits the type of the task that activates it. We also considered tasks with arbitrary deadlines, discussed the consequences of multiple operating modes of a task on the analysis, and commented on the impact of the scheduling model on our results.

References


