Exact worst-case response times of real-time tasks under fixed-priority scheduling
Bril, R.J.; Verhaegh, W.F.J.; Lukkien, J.J.

Published in:

Published: 01/01/2004

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 author’s 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

Citation for published version (APA):
Exact worst-case response times of real-time tasks under fixed-priority scheduling with deferred preemption

Reinder J. Bril\textsuperscript{1,2}, Wim F.J. Verhaegh\textsuperscript{2}, and Johan J. Lukkien\textsuperscript{1}

\textsuperscript{1}Technische Universiteit Eindhoven (TU/e), Den Dolech 2, 5600 AZ Eindhoven, The Netherlands
\textsuperscript{2}Philips Research Laboratories, Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands
\{r.j.bril,j.j.lukkien\}@tue.nl, \{reinder.bril,wim.verhaegh\}@philips.com

Abstract

In this paper, we present equations to determine the exact worst-case response times of periodic tasks under fixed-priority scheduling with deferred preemption (FPDS) and arbitrary phasing. We show that the worst-case response time analysis is not uniform for all tasks. Our exact analysis is based on a dedicated conjecture for an ε-critical instant, and uses the notion of worst-case occupied time.

1 Introduction

Based on the seminal paper of Liu and Layland [10], many results have been achieved in the area of worst-case analysis for fixed-priority preemptive scheduling (FPPS). Arbitrary preemption of real-time tasks has a number of drawbacks, though. In particular in systems using cache memory, e.g. to bridge the speed gap between processors and main memory, arbitrary preemptions induce additional cache flushes and reloads. As a consequence, system performance and predictability are degraded, complicating system design, analysis and testing [4, 5, 8, 11]. Although fixed-priority non-preemptive scheduling (FPNS) may resolve these problems, it generally leads to reduced schedulability compared to FPPS. Therefore, alternative scheduling schemes have been proposed between the extremes of arbitrary preemption and no preemption. These schemes are also known as deferred preemption or co-operative scheduling [2], and are denoted by fixed-priority scheduling with deferred preemption (FPDS) in the remainder of this paper.

2 Worst-case analysis for FPPS

2.1 Basic model

We assume a single processor and a set $\Gamma$ of $n$ periodic, independent tasks $\tau_1, \tau_2, \ldots, \tau_n$. Each task $\tau_i$ is characterized by a (release) period $T_i \in \mathbb{R}^+$, a (worst-case) computation time $C_i \in \mathbb{R}^+$, and a (relative) deadline $D_i \in \mathbb{R}^+$. In this paper, we assume that a task’s deadline does not exceed its period, i.e. $D_i \leq T_i$ for each $i$. A release of a task is also termed a job. The release of task $\tau_i$ at time $\phi_i \in \mathbb{R}$ with $0 \leq \phi_i < T_i$ serves as a reference release. Time $\phi_i$ is also termed the phasing of task $\tau_i$, and $\phi = (\phi_1, \ldots, \phi_n)$ is called the phasing of the task set $\Gamma$. 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 [6, 7, 10]. We also assume other standard basic assumptions [10], i.e. tasks are ready to run at the start of each period and do not suspend themselves, tasks will be preempted instantaneously when a higher priority task becomes ready to run, a job of task

on a dedicated theorem for an ε-critical instant, and uses a notion that has already been used implicitly in [12] to determine slack, and which we term worst-case occupied time. For space considerations, we only discuss results; proofs will appear elsewhere.

This paper is organized as follows. Section 2 describes worst-case analysis of periodic tasks under FPPS and arbitrary phasing. For completeness reasons, we start with a recapitulation of worst-case response times. We subsequently address worst-case occupied times. In Section 3, we present our results for FPDS and arbitrary phasing. In Section 4, we briefly compare our results with those presented in the literature, and show that the application of existing results yields values that are either too optimistic or too pessimistic in specific situations.
\[ x = C_i + \sum_{j < i} \left\lfloor \frac{x}{T_j} \right\rfloor C_j. \] (1)

Assuming a critical instant at time zero, the factor \( \left\lfloor \frac{x}{T_j} \right\rfloor \) in (1) gives the worst-case number of preemptions that an execution of task \( \tau_i \) suffers from task \( \tau_j \) in an interval \([0,x]\). To calculate worst-case response times, we can use an iterative procedure based on recurrence relationships [1].

Table 1 provides an example with characteristics of a task set \( \Gamma \) consisting of three tasks, and the worst-case response times under FPPS. We used the superscript \( P \) to denote FPPS in Table 1. Similarly, we will use superscripts \( D \) and \( N \) later to denote FPDS and FPNS, respectively.

Figure 1 shows a timeline with the executions of these three tasks and a critical instant at time zero.

### 2.3 Worst-case occupied time

The worst-case occupied time of a task is closely related to its worst-case response time. The worst-case occupied time \( O_i \) of a task \( \tau_i \) is defined as the length of the longest possible interval from a release of that task during which the processor is occupied with the execution of \( C_i \) of that job and executions of higher priority tasks till the moment in time that the same job could start an additional bit of computation. Hence, \( O_i \) includes preemptions and aligning successive executions of higher priority tasks. For \( D_i \leq T_i \), the worst-case occupied time of \( \tau_i \) is the worst-case response time of that task extended with all aligning successive executions of higher priority tasks. Note that we only consider a single job of \( \tau_i \) for the worst-case occupied time, making our notion different from the notion of busy period [9].

Given this above definition, we may simply derive the worst-case occupied times for \( \Gamma \) from Figure 1. For the highest priority task \( \tau_1 \), \( O_1 \) equals \( R_1 \), i.e., \( O_1 = 2 \). For task \( \tau_2 \), \( O_2 \) extends \( R_2 \) with the aligning execution of task \( \tau_1 \), i.e., \( O_2 = R_2 + C_1 = 7 \). Note that the aligning job of task \( \tau_2 \) itself at time seven is not included. Finally, \( O_3 \) of task \( \tau_3 \) extends \( R_3 \) with the aligning executions of both \( \tau_1 \) and \( \tau_2 \), i.e., \( O_3 = R_3 + C_1 + C_2 = 33 \). Note that \( O_3 \) of task \( \tau_3 \) is larger than its deadline \( D_3 \) and its period \( T_3 \).

To determine worst-case occupied times under arbitrary phasing, we can use a similar approach as for worst-case response times. However, rather than considering releases of higher priority tasks in an interval \([0,x]\), we need to consider releases of those tasks in an interval \([0,x]\), i.e., including those at time \( x \). As already explained in [12], this can be done by using the floor rather than the ceiling function, and adding a term \( 1 \), i.e., the worst-case occupied time \( O_i \) of a task \( \tau_i \) is given by the smallest \( x \in \mathbb{R}^+ \) that satisfies

\[ x = C_i + \sum_{j < i} \left( \frac{x}{T_j} \right) + 1 C_j. \] (2)

Similarly to worst-case response times, we can use an iterative procedure based on recurrence relationships to calculate worst-case occupied times.

Unlike the worst-case response time, the worst-case response time is well defined for a computation time of zero.

In this case, the worst-case occupied time is equal to the worst-case start time, i.e., the length of the longest interval from a release of a task till the start of its execution.

For our leading example, the worst-case start time \( S^P_1 \) of task \( \tau_1 \) equals \( 0 \), \( S^P_2 = 2 \), and \( S^P_3 = 12 \); cf. Figure 1.

### 2.4 Concluding remarks

In the remainder, we will parameterize \( R_i \) and \( O_i \) of task \( \tau_i \) with \( C_i \) when needed. As illustrated in [7], blocking can
be taken into account when calculating $R_i$ of task $\tau_i$ by incorporating a worst-case blocking term $B_i$ in $C_i$.

3 Worst-case analysis for FPDS

3.1 Refined model

For FPDS, we need to refine our basic model of Section 2.1. Each job of task $\tau_i$ is now assumed to consist of $m(i)$ subjobs. The $j$th subjob of $\tau_i$ is characterized by a computation time $C_{i,j} \in \mathbb{R}^+$, where $C_i = \sum_{j=1}^{m(i)} C_{i,j}$. We assume that subjobs are non-preemptable. Hence, tasks can only be preempted at subjob boundaries, i.e. at so-called preemption points. For convenience, we will use the term $F_i$ to denote the computation time $C_{i,m(i)}$ of the final subjob of $\tau_i$. Note that when $m(i) = 1$ for all $i$, we have FPNS as special case.

3.2 Worst-case response times

The non-preemptive nature of subjobs may cause blocking of a task by at most one lower priority task under FPDS. The maximum blocking of task $\tau_i$ by a lower priority task is equal to the longest computation time of any subjob of a task with a priority lower than task $\tau_i$, which is given by

$$B_i = \max_{j>1} \max_{1 \leq k \leq m(j)} C_{i,k}. \quad (3)$$

To determine worst-case response times under FPDS and arbitrary phasing, we have to revisit critical instants. In this paper, we merely postulate the following conjecture.

Conjecture 1 An $\varepsilon$-critical instant of a task $\tau_i$ under FPDS and arbitrary phasing occurs when that task is released simultaneously with all tasks with a higher priority than $\tau_i$, and the subjob with the longest computation time of all lower priority tasks starts an infinitesimal time $\varepsilon > 0$ before that simultaneous release.

From this conjecture we conclude that a critical instant for FPDS is a supremum for all but the lowest priority task, i.e. that instant cannot be assumed.

For the analysis, we consider three cases: the highest priority task $\tau_1$, the lowest priority task $\tau_n$, and a medium priority task $\tau_i$ (with $1 < i < n$).

Task $\tau_1$ may be blocked, but is never preempted. The worst-case response time $R^D_1$ of task $\tau_1$ therefore includes a term $B_1$, i.e.

$$R^D_1 = B_1 + C_1. \quad (4)$$

Note that $B_1 + C_1$ is a supremum, i.e. that value cannot be assumed, but it can be approximated arbitrarily closely. Further note that this latter equation may also be written as $R^D_1 = R^D_1(B_1 + C_1)$, or $R^D_1 = R^D_1(B_1 + C_1 - F_1) + F_1$. Because $R^D_1 = O^D_1$, the equation may even be written as $R^D_1 = O^D_1(B_1 + C_1)$, or $R^D_1 = O^D_1(B_1 + C_1 - F_1) + F_1$.

Task $\tau_n$ may be preempted (at subjob boundaries), but is never blocked. The worst-case response time $R^D_n$ of task $\tau_n$ can hence be found by calculating the worst-case start time of the final subjob, and adding its computation time $F_n$. The non-preemptive nature of the other subjobs of $\tau_n$ may result in deferred preemptions by higher priority tasks. Although that has an influence on the order of the executions of the subjobs of tasks, it does not influence the total amount of time spent on those executions. The amount of time spent on executions of all but the final subjob of $\tau_n$ including the (deferred) preemptions of higher priority tasks is given by $R^D_n(C_n - F_n)$. The final subjob of $\tau_n$ may subsequently start after the aligning successive executions of higher priority tasks have completed. Hence, the worst-case start time of the final subjob of task $\tau_n$ is given by $O^D_n(C_n - F_n)$, and we arrive at the following equation for $R^D_n$.

$$R^D_n = O^D_n(C_n - F_n) + F_n \quad (5)$$

Note that $O^D_n(C_n - F_n) + F_n$ is a maximum, i.e. that value can be assumed. Further note that for $m(n) = 1$, we get $O^D_n(C_n - F_n) = O^D_n(0)$, which is equal to the worst-case start time $S^D_n$ of task $\tau_n$.

Task $\tau_i$ with $1 < i < n$, may be both preempted at subjob boundaries by higher priority tasks and blocked by a lower priority task. Similarly to task $\tau_n$, the worst-case response time $R^D_i$ of $\tau_i$ can be found by calculating the worst-case start time of the final subjob, and by subsequently adding its computation time $F_i$. Similarly to $\tau_n$, the non-preemptive nature of the other subjobs of $\tau_i$ has no influence on the worst-case start time of the final subjob. At first hand, it therefore looks as if the same reasoning applies as for $\tau_n$, and that we can calculate the worst-case start time by means of $O^D_i(B_i + C_i - F_i)$. However, the blocking subjob of the lower priority tasks has to start an infinitesimal time $\varepsilon > 0$ before the simultaneous release of $\tau_i$ and its higher priority tasks. Hence, the amount of time spent from the release of $\tau_i$ on executions of the blocking subjob and all but the final subjob of $\tau_i$, including the (deferred) preemptions of higher priority tasks is given by $O^D_i(B_i - \varepsilon + C_i - F_i)$. This latter value is equal to $R^D_i(B_i + C_i - F_i)$ minus an infinitesimal time $\varepsilon > 0$. It is exactly this infinitesimal difference, which approaches zero, that allows the final subjob of $\tau_i$ to start executing, and that defers potential additional preemptions from higher priority tasks at time $R^D_i(B_i + C_i - F_i)$. The worst-case response time $R^D_i$ of $\tau_i$ is therefore given by

$$R^D_i = R^D_i(B_i + C_i - F_i) + F_i. \quad (6)$$

Note that $R^D_i(B_i + C_i - F_i) + F_i$ is also a supremum.

As mentioned above, we may rewrite (4) to $R^D_1 =
O^P_i(B_1 + C_1 - F_1) + F_i as well as to \( R^D_1 = R^P_1 (B_1 + C_1 - F_1) + F_1 \). Hence, (4) is similar to both (5) and (6).

### 3.3 An example

To illustrate the equations, consider the task characteristics of Table 1. For FPNS, the set is not schedulable because the worst-case response time \( \tilde{R}_1 \) of task \( \tau_1 \) is equal to \( B_1 + C_1 = 4 + 2 = 6 \), which exceeds \( D_1 \).

For FPDS, let \( m(1) = 1 \), \( m(2) = 2 \) with \( C_{2,1} = 1 \) and \( C_{2,2} = 2 \), and \( m(3) = 2 \) with \( C_{3,1} = C_{3,2} = 2 \); see Table 1. Using the equations above yields \( R^P_1 = B_1 + C_1 = 2 + 2 = 4 \), \( R^D_1 = R^P_2 (B_2 + C_2 - F_2) + F_2 = R^P_2 (2 + 3 - 2) + 2 = 7 \), and \( R^D_1 = O^P_3 (C_3 - F_3) + F_3 = O^P_3 (2) + 2 = 21 \). Hence, by splitting both task \( \tau_2 \) and task \( \tau_3 \) into two non-preemptive subjobs, the task set becomes schedulable under FPDS.

### 4 Related Work

We briefly compare our results with those presented in the literature. The schedulability test in [5] is based on utilization bounds, and is therefore typically pessimistic. The worst-case response time analysis in [8] is based on a single equation, i.e., it is uniform for all tasks. The blocking effect of (partially) non-preemptive lower priority tasks has been covered in that analysis, but the effect of the non-preemptive nature of the final subjob is not taken into account. The analysis is therefore pessimistic.

The results presented in [2, 3, 4] are very similar to ours. Unlike our approach, their approach is uniform for all tasks. Using our notation, the worst-case response time \( R^D_1 \) under FPDS and arbitrary phasing presented in [2, 4] is given by

\[
\tilde{R}_1 = R^P_1 (B_i + C_i - (F_i - \Delta)) + (F_i - \Delta). \tag{7}
\]

According to [4], \( \Delta \) is an arbitrary small positive value needed to ensure that the final subjob has actually started. Hence, when task \( \tau_i \) has consumed \( C_i - (F_i - \Delta) \), the final subjob has (just) started. When \( \Delta \) approaches zero, we may rewrite (7) to

\[
\tilde{R}_D = O^P_i (B_i + C_i - F_i) + F_i.
\]

This result is identical to ours for the highest and lowest priority tasks, but differs from ours for intermediate tasks. For the example presented above, our analysis yields \( \tilde{R}_D = 7 \), whereas the analysis presented in [4] yields \( \tilde{R}_D = 9 \). In the example, this latter result is too pessimistic; because \( \tilde{R}_D \) exceeds \( D_1 \), the task set would incorrectly be considered non-schedulable. This difference between \( R^D_2 \) and \( \tilde{R}_D \) can be traced back to Conjecture 1. The analysis in [4] does not take into account that \( \tau_i \) can only be blocked by a subjob of a lower priority task if that subjob also starts an amount of time \( \Delta \) before the simultaneous release of \( \tau_i \) and all tasks with a higher priority than \( \tau_i \). When this aspect is be taken into account in the analysis of [4], e.g., when \( B_i \) is replaced by \( B_i - \Delta \) in (7), their result becomes identical to ours.

In the scheduling analysis review presented in [3], the worst-case response time \( \tilde{R}_D \) ignores the term \( \Delta \), i.e.,

\[
\tilde{R}_D = R^P_i (B_i + C_i - F_i) + F_i.
\]

This result is identical to ours, except for the lowest priority task. For the example presented above, this result in \( \tilde{R}_3 = R^P_3 (C_3 - F_3) + F_3 = 16 \), which is too optimistic.

### References


