# Towards best-case response times of real-time tasks under fixed-priority scheduling with deferred preemption

Reinder J. Bril

*Technische Universiteit Eindhoven (TU/e),*
*Den Dolech 2, 5600 AZ Eindhoven,*
*The Netherlands*
*r.j.bril@tue.nl*

Wim F.J. Verhaegh

*Philips Research Laboratories,*
*Prof. Holstlaan 4, 5656 AA Eindhoven,*
*The Netherlands*
*wim.verhaegh@philips.com*

## Abstract

*In this paper, we present lower bounds for best-case response times of periodic tasks under fixed-priority scheduling with deferred preemption (FPDS) and arbitrary phasing. Our analysis is based on a dedicated conjecture for a* $\Delta$*-optimal instant, and uses the notion of* best-case occupied time. *We briefly compare best-case analysis and worst-case analysis for FPDS and arbitrary phasing.*

## 1 Introduction

Based on the seminal paper of Liu and Layland [8], many results have been achieved in the area of 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, 7, 10]. 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 [3], and are denoted by fixed-priority scheduling with deferred preemption (FPDS) in the remainder of this paper.

Worst-case response time analysis of periodic real-time tasks under FPDS and arbitrary phasing has been addressed in a number of papers [3, 7, 2]. Best-case response time analysis received considerably less attention, and is the topic of this paper. It is based on a dedicated theorem for a so-called $\Delta$-optimal instant, and uses a notion which we term *best-case occupied time*. For space considerations, we only discuss results; proofs will appear elsewhere. Application of best-case response times, e.g. to analyze jitter in distributed multiprocessor systems, also falls outside the scope of this paper.

This paper is organized as follows. Section 2 describes best-case analysis of periodic tasks under FPPS and arbitrary phasing. For completeness reasons, we start with a recapitulation of best-case response times. We subsequently address best-case occupied times. In Section 3, we present our results for FPDS and arbitrary phasing. In Section 4, we briefly compare the approaches for best-case analysis and worst-case analysis.

## 2 Best-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 (*best-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 all $i$. A release of a task is also termed a job. The release of task $\tau_i$ at time $\varphi_i \in \mathbb{R}$ with $0 \leq \varphi_i < T_i$ serves as a reference release. Time $\varphi_i$ is also termed the *phasing* of task $\tau_i$, and $\varphi = (\varphi_1, \ldots, \varphi_n)$ is called the phasing of the task set $\Gamma$. We assume that we do not have control over the phasing $\varphi$, 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, 8]. We also assume other standard basic assumptions [8], i.e. tasks are ready to run at the start of each period and do no suspend themselves, tasks will be preempted instantaneously when a higher priority task becomes ready to run, a job of 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 tasks are given in order of decreasing priority, i.e. task $\tau_1$ has highest priority and task $\tau_n$ has lowest priority.

## 2.2 Recapitulation of best-case response times

The best-case response time of a task is the length of the *shortest* interval from a task's release till its completion. To determine best-case response times under arbitrary phasing, it suffices to consider only so-called *optimal* (or *favourable*) *instants* [9, 1]. For FPPS, an optimal instant for task $\tau_i$ is given by a point in time in which the *completion* of $\tau_i$ coincides with the simultaneous release of all higher priority tasks. From this notion of optimal instants, it has been derived that the best-case response time $BR_i$ of a task $\tau_i$ is given by the *largest* $x \in \mathbb{R}^+$ that satisfies

$$x = C_i + \sum_{j<i} \left( \left\lceil \frac{x}{T_j} \right\rceil - 1 \right) C_j. \qquad (1)$$

Assuming a optimal instant at time 0, the factor $\left( \left\lceil \frac{x}{T_j} \right\rceil - 1 \right)$ in (1) gives the best-case number of preemptions that an execution of task $\tau_i$ suffers from task $\tau_j$ in an interval $(-x, 0)$. In order to calculate the best-case response times, we can use the following iterative procedure based on recurrence relationships. The procedure starts with an upper bound. When the worst-case response time $WR_i$ of $\tau_i$ is known, we can use it as initial value.

$$\begin{aligned} BR_i^{(0)} &= WR_i \\ BR_i^{(k+1)} &= C_i + \sum_{j<i} \left( \left\lceil \frac{BR_i^{(k)}}{T_j} \right\rceil - 1 \right) C_j \end{aligned}$$

The procedure is stopped when the same value is found for two successive iterations of $k$, yielding the largest solution of the recursive equation, i.e. the best-case response time of $\tau_i$. Termination of the procedure is ensured by the fact that the sequence $BR_i^{(k)}$ is bounded (from below by $C_i$, and from above by $WR_i$) and non-increasing, and that different values for successive iterations differ at least $\min_{j<i} C_j$. Table 1 provides an example with characteristics of a task set $\Gamma$ consisting of three tasks, and the response times under FPPS. We used the superscript P to denote FPPS in Table 1. Similarly, we will use superscript D later to denote FPDS.

Figure 1 shows a timeline with the executions of these three tasks and an optimal instant at time zero for task $\tau_3$. Note that for the construction of the best-case response time of task $\tau_3$ using a timeline:

- we first draw the releases and executions of the higher priority tasks *before* their simultaneous release, and
- we subsequently draw the execution of $\tau_3$ *backwards* in time.

| | $T_i$ | $D_i$ | $C_i$ | $WR_i^{\mathrm{P}}$ | $BR_i^{\mathrm{P}}$ | $WR_i^{\mathrm{D}}$ | $BR_i^{\mathrm{D}}$ |
|---|---|---|---|---|---|---|---|
| $\tau_1$ | 5 | 4 | 2 | 2 | 2 | 4 | 2 |
| $\tau_2$ | 7 | 7 | 1 + 2 | 5 | 3 | 7 | 3 |
| $\tau_3$ | 30 | 30 | 2 + 2 | 28 | 16 | 21 | 9 |

**Table 1. Task characteristics and response times under FPPS and FPDS.**

## 2.3 Best-case occupied time

The best-case occupied time of a task is closely related to its best-case response time. The best-case occupied time $BO_i$ of a task $\tau_i$ is defined as the length of the shortest 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 that the same job could start an additional bit of computation. Because that additional bit of computation extends $BR_i$ *backwards* in time (see Figure 1), $BO_i$ of task $\tau_i$ is equal to $BR_i$ extended with all aligning *preceding* executions of higher priority tasks.

Given this above definition, we may simply derive the best-case occupied time $BO_3$ for $\tau_3$ from Figure 1. $BO_3$ extends $BR_3$ with the aligning preceding executions of both $\tau_1$ and $\tau_2$, i.e. $BO_3 = BR_3 + C_1 + C_2 = 21$.

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

$$x = C_i + \sum_{j<i} \left\lfloor \frac{x}{T_j} \right\rfloor C_j. \qquad (2)$$

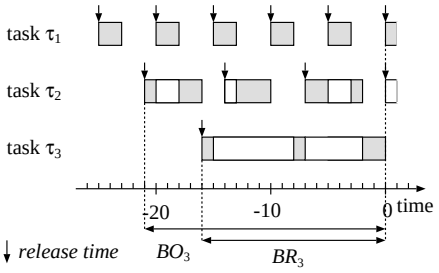


**Figure 1. Timeline under FPPS with an optimal instant at time zero for task $\tau_3$.**

Assuming a optimal instant at time 0, the factor $\left\lfloor \frac{x}{T_j} \right\rfloor$ in (2) gives the best-case number of preemptions that an execution of task $\tau_i$ suffers from task $\tau_j$ in an interval $[-x, 0)$. Similarly to best-case response times, we can use an iterative procedure based on recurrence relationships to calculate best-case occupied times. When the worst-case response time $WR_i$ of $\tau_i$ is known, we can use it as initial value.

$$\begin{aligned} BO_i^{(0)} &= WR_i \\ BO_i^{(k+1)} &= C_i + \sum_{j<i} \left\lfloor \frac{BO_i^{(k)}}{T_j} \right\rfloor C_j \end{aligned}$$

The procedure is stopped when the same value is found for two successive iterations of $k$, yielding the largest solution of the recursive equation. The iterative procedure is ensured to terminate for the same reasons as the termination of the procedure for the best-case response time.

Unlike the best-case response time, the best-case occupied time is well defined for a computation time of zero. In this case, the best-case occupied time is equal to the *best-case start time*, i.e. the length of the shortest interval from a release of a task till the start of its execution. For our model, the best-case start time is equal to zero for all tasks.

# 3 Best-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 $C_i = \sum_{j=i}^{m_i} C_{i,j}$. 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.

To illustrate the model, consider the task characteristics of Table 1, i.e. let $m_1 = 1$, $m_2 = 2$ with $C_{2,1} = 1$ and $C_{2,2} = 2$, and $m_3 = 2$ with $C_{3,1} = C_{3,2} = 2$ for FPDS. As illustrated in [2], task set $\Gamma$ is schedulable under FPDS, i.e. $WR_i^{\mathrm{D}} \leq D_i$ for all tasks $\tau_i$ of $\Gamma$. Figure 2 shows a timeline with the executions of these three tasks under FPDS with a simultaneous release at time zero. Note that both task $\tau_1$ and task $\tau_2$ have releases with a response time equal to their computation time in this figure, e.g. at time 0 and 7, respectively. Hence, $BR_1^{\mathrm{D}} = C_1 = 2$ and $BR_2^{\mathrm{D}} = C_2 = 3$ for our example.

In the remainder, we will parameterize $BR_i$ and $BO_i$ of task $\tau_i$ with $C_i$ when needed.

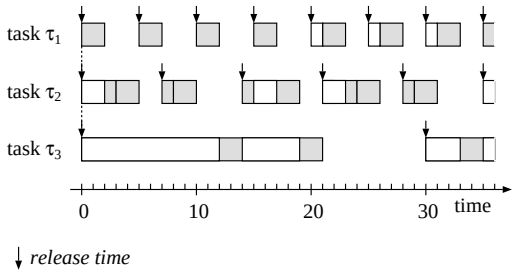


**Figure 2. Timeline under FPDS with a simultaneous release of all tasks at time zero.**

## 3.2 Best-case response times

The non-preemptive nature of subjobs may cause blocking of a task by lower priority tasks under FPDS. Whereas we are interested in the *maximum* blocking of task $\tau_i$ by lower priority tasks for worst-case response times [1], we are interested in the *minimum* blocking of task $\tau_i$ for best-case response times. In this paper, we make the safe, i.e. pessimistic, assumption that the minimum blocking equals zero, which gives rise to a lower bound for the best-case response time.

The highest priority task $\tau_1$ experiences a minimum blocking equal to zero for a simultaneous release of all tasks. Hence, the best-case response time of $\tau_1$ under FPDS and arbitrary phasing is equal to its computation time, i.e.

$$BR_1^{\mathrm{D}} = C_1. \tag{3}$$

To determine best-case response times under FPDS and arbitrary phasing for a lower priority task $\tau_i$ (with $1 < i \leq n$), we revisit optimal instants. We can determine lower bounds for best-case response times of lower priority tasks using the following conjecture, which we merely postulate in this paper.

**Conjecture 1** *A $\Delta$-optimal instant of a lower priority task $\tau_i$ (with $1 < i \leq n$) under FPDS and arbitrary phasing occurs when the final sub-job of $\tau_i$ starts a (sufficiently small) finite time $\Delta > 0$ before the simultaneous release of all tasks with a higher priority than $\tau_i$.*

Based on this conjecture, we now determine a lower bound for the best-case reponse time of $\tau_i$. Task $\tau_i$ can only be preempted at subjob boundaries by higher priority tasks. As mentioned above, we assume that the minimum blocking of task $\tau_i$ is equal to zero. Hence, we may ignore tasks with a lower priority than task $\tau_i$ when determining a lower bound for the best-case response time of $\tau_i$. The non-preemptive nature of the subjobs of $\tau_i$ 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 minimal amount of time spent on executions of all but the final subjob of $\tau_i$ including the (deferred) preemptions of higher priority tasks is given by $BR_i^{\mathrm{P}}(C_i - F_i)$. According to Conjecture 1, the final subjob of $\tau_i$ has to start a (sufficiently small) finite $\Delta > 0$ *before* the simultaneous release of its higher priority tasks. Hence, the minimal amount of time spent from the release of $\tau_i$ on executions of $\tau_i$ and its higher priority tasks till the simultaneous release of those higher priority tasks is given by $BR_i^{\mathrm{P}}(C_i - F_i + \Delta)$. This latter value is equal to $BO_i^{\mathrm{P}}(C_i - F_i)$ *plus* a finite time $\Delta > 0$. A lower bound for the best-case response time $BR_i^{\mathrm{D}}$ of $\tau_i$ is therefore given by

$$BR_i^{\mathrm{D}} \geq BO_i^{\mathrm{P}}(C_i - F_i) + F_i. \quad (4)$$

### 3.3 An example

To illustrate the analysis, consider the task characteristics of Table 1. As shown in [2], task set $\Gamma$ is schedulable under FPDS, i.e. $WR_i^{\mathrm{D}} \leq D_i$ for all three tasks $\tau_i$ of $\Gamma$. Figure 3 shows a timeline with the executions of these three tasks and a $\Delta$-optimal instant at time zero for task $\tau_3$.

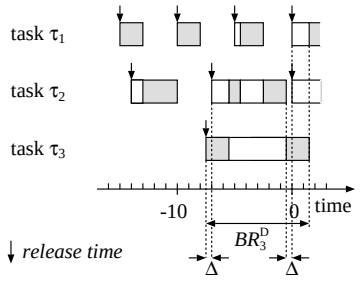


**Figure 3. Timeline under FPDS with a $\Delta$-optimal instant at time zero for task $\tau_3$.**

Using (4) yields $BR_3^{\mathrm{D}} \geq BO_3^{\mathrm{P}}(C_3 - F_3) + F_3 = BO_3^{\mathrm{P}}(2) + 2 = 7 + 2 = 9$. Note that this lower bound of 9 for $BR_3^{\mathrm{D}}$ of task $\tau_3$ is larger than its computation time $C_3$, which equals 4, and is smaller than $BR_3^{\mathrm{P}}$, which equals 16.

## 4 Discussion

In this section we briefly compare the approaches for best-case analysis and worst-case analysis for FPDS under arbitrary phasing.

Both worst-case response times and best-case response times under FPDS can be expressed in terms of response times and occupied times under FPPS. Worst-case analysis for FPDS is based on a so-called $\varepsilon$-critical instant [2], where $\varepsilon$ is an infinitesimal time larger than zero. Such an instant is a supremum for all but the lowest priority task, i.e. that instant can not be assumed. Conversely, best-case analysis for FPDS is based on a $\Delta$-optimal instant, where $\Delta$ is a (sufficiently small) finite time larger than zero. Therefore, a $\Delta$-optimal instant can be assumed.

## Acknowledgements

We thank Jan H.M. Korst and Clemens C. Wüst for their comments on a previous version of this paper.

## References

[1] R. Bril, E. Steffens, and W. Verhaegh. Best-case response times and jitter analysis of real-time tasks. *Journal of Scheduling*, 7(2):133–147, March 2004.

[2] R. Bril, W. Verhaegh, and J. Lukkien. Exact worst-case response times of real-time tasks under fixed-priority scheduling with deferred preemption. In *Proc. Work-in-Progress (WiP) session of the 16$^{th}$ Euromicro Conf. on Real-Time Systems (ECRTS)*, pages 57–60, June 2004.

[3] A. Burns. Preemptive priority based scheduling: An appropriate engineering approach. In S. Son, editor, *Advances in Real-Time Systems*, pages 225–248. Prentice-Hall, 1994.

[4] A. Burns and A. Wellings. Restricted tasking models. In *Proc. 8$^{th}$ Int. Real-Time Ada Workshop*, pages 27–32, 1997.

[5] R. Gopalakrishnan and G. Parulkar. Bringing real-time scheduling theory and practice closer for multimedia computing. In *Proc. ACM Sigmetrics Conf. on Measurement & Modeling of Computer Systems*, pages 1–12, May 1996.

[6] M. Klein, T. Ralya, B. Pollak, R. Obenza, and M. G. Harbour. *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*. Kluwer Academic Publishers, 1993.

[7] S. Lee, C.-G. Lee, M. Lee, S. Min, and C.-S. Kim. Limited preemptible scheduling to embrace cache memory in real-time systems. In *Proc. ACM Sigplan Workshop on Languages, Compilers and Tools for Embedded Systems (LCTES)*, pages 51–64, June 1998.

[8] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[9] O. Redell and M. Sanfridson. Exact best-case response time analysis of fixed priority scheduled tasks. In *Proc. 14$^{th}$ Euromicro Conf. on Real Time Systems (ECRTS)*, pages 165–172, June 2002.

[10] J. Simonson and J. Patel. Use of preferred preemption points in cache-based real-time systems. In *Proc. IEEE International Computer Performance and Dependability Symposium (IPDS'95)*, pages 316–325, April 1995.