



Exact speedup factors and sub-optimality for non-preemptive scheduling

Robert I. Davis¹  · Abhilash Thekkilakattil² ·
Oliver Gettings¹ · Radu Dobrin³ ·
Sasikumar Punnekkat³ · Jian-Jia Chen⁴ 

Published online: 28 October 2017

© The Author(s) 2017. This article is an open access publication

Abstract Fixed priority scheduling is used in many real-time systems; however, both preemptive and non-preemptive variants (FP-P and FP-NP) are known to be sub-optimal when compared to an optimal uniprocessor scheduling algorithm such as preemptive earliest deadline first (EDF-P). In this paper, we investigate the sub-optimality of fixed priority non-preemptive scheduling. Specifically, we derive the exact processor speed-up factor required to guarantee the feasibility under FP-NP (i.e.

Preliminary publication: This paper extends initial research into speedup factors for preemptive versus non-preemptive uniprocessor scheduling published in RTSS 2015 (Davis et al. 2015c). The main extension is the proof of the exact speedup factor required to guarantee the FP-P feasibility of any FP-NP feasible constrained-deadline task set.

✉ Robert I. Davis
rob.davis@york.ac.uk

Abhilash Thekkilakattil
abhilash.thekkilakattil@se.atlascopco.com

Oliver Gettings
oliver@gettin.gs

Radu Dobrin
radu.dobrin@mdh.se

Sasikumar Punnekkat
sasikumar.punnekkat@mdh.se

Jian-Jia Chen
jian-jia.chen@cs.uni-dortmund.de

¹ Department of Computer Science, University of York, York, UK

² AtlasCopco Industrial Tools and Solutions, Nacka, Sweden

³ Mälardalen Real-Time Research Center, Mälardalen University, Västerås, Sweden

⁴ Technische Universität Dortmund, Dortmund, Germany

schedulability assuming an optimal priority assignment) of any task set that is feasible under EDF-P. As a consequence of this work, we also derive a lower bound on the sub-optimality of non-preemptive EDF (EDF-NP). As this lower bound matches a recently published upper bound for the same quantity, it closes the exact sub-optimality for EDF-NP. It is known that neither preemptive, nor non-preemptive fixed priority scheduling dominates the other, in other words, there are task sets that are feasible on a processor of unit speed under FP-P that are not feasible under FP-NP and vice-versa. Hence comparing these two algorithms, there are non-trivial speedup factors in both directions. We derive the exact speed-up factor required to guarantee the FP-NP feasibility of any FP-P feasible task set. Further, we derive the exact speed-up factor required to guarantee FP-P feasibility of any constrained-deadline FP-NP feasible task set.

Keywords Real-time · Uniprocessor · Resource augmentation · Speedup factor · Sub-optimality · Non-preemptive scheduling · Preemptive scheduling · EDF · Fixed priority

1 Introduction

Real-time systems are prevalent in a wide variety of application areas including telecommunications, consumer electronics, aerospace systems, automotive electronics, robotics, and medical systems. The functionality of these systems is typically mapped to a set of periodic or sporadic real-time tasks, with each task giving rise to a potentially unbounded sequence of jobs. Timely execution of the tasks and their jobs is supported by the use of real-time scheduling algorithms.

Real-time scheduling algorithms for single processor systems may be classified into two main types: *fixed priority* and *dynamic priority*. Fixed priority scheduling is the de facto standard approach used in many applications. Here, a unique static priority is assigned to each task and inherited by all of its jobs. At runtime, the scheduler uses these priorities to determine which job to execute. Earliest Deadline First (EDF) is the most common example of a dynamic priority scheduling algorithm. EDF uses priorities based on the absolute deadline of each job to make scheduling decisions.

Real-time scheduling algorithms may also be classified in terms of when and if preemption is permitted. Thus we have preemptive and non-preemptive variants of both fixed priority (FP-P and FP-NP) and EDF (EDF-P and EDF-NP) scheduling.

There are a number of different ways in which the performance of real-time scheduling algorithms can be compared (Davis 2017). *Empirical techniques* typically rely on generating a large number of task sets with parameters chosen from some appropriate distributions. The performance of the scheduling algorithms are then compared by determining task set schedulability according to exact or sufficient schedulability tests and plotting a graph of the success ratio, i.e. the proportion of task sets that are deemed schedulable, at different utilization levels. More advanced approaches use a *weighted schedulability metric* (Bastoni et al. 2010) to illustrate how schedulability varies with a further parameter, for example task set cardinality, or the range of task periods. Similar comparisons may be obtained by using a simulation of each algorithm

as a necessary schedulability test, hence showing the proportion of task sets found to be definitely unschedulable due to a deadline miss in the simulation. These empirical approaches tend to focus on the average-case behaviour over large numbers of task sets rather than highlighting those task sets that are particularly difficult to schedule using one algorithm, but may be easy to schedule using another. Metrics such as *breakdown utilisation* (Lehoczky et al. 1989) and *optimality degree* (Bini and Buttazzo 2005) can also be used to examine average-case performance.

In this paper, we focus on a *theoretical method* of comparing the worst-case performance of real-time scheduling algorithms based on a *resource augmentation* metric referred to as the processor *speedup factor* (Kalyanasundaram and Pruhs 2000). Specifically, we derive bounds on the factor by which the speed of the processor needs to be increased to ensure that any task set that is feasible under some scheduling algorithm \mathcal{A} is guaranteed to be feasible under another algorithm \mathcal{B} . When \mathcal{A} is an optimal algorithm, then this speedup factor provides a measure of the *sub-optimality* of algorithm \mathcal{B} . Note, when we refer to a task set as being *feasible* under a particular scheduling algorithm, if that algorithm uses fixed priorities, then feasibility refers to the task set being schedulable assuming an optimal priority assignment.

In this paper, we use speedup factors to compare fixed priority non-preemptive scheduling (FP-NP) with both fixed priority preemptive (FP-P) and Earliest Deadline First (EDF-P) scheduling.

Our interest in FP-NP scheduling stems from the fact that in modern uniprocessor systems preemption can significantly increase overheads due to a number of factors. These include context switch costs and cache related pre-emption delays (CRPD) which have to be accounted for in both FP-P (Altmeyer et al. 2011, 2012) and EDF-P (Lunniss et al. 2013) scheduling. CRPD can have a substantial impact, increasing task execution times by as much as 33% (Bui et al. 2008). One way of reducing or eliminating CRPD is to partition the cache; however, allocating each task a cache partition, which is some fraction of the overall size of the cache, has an impact on the task's worst-case execution time (WCET) which may be significantly inflated. Such partitioning rarely improves upon schedulability compared to accounting for CRPD and allowing tasks to use the entire cache (Altmeyer et al. 2014, 2016). An alternative method which eliminates CRPD without increasing WCETs is to employ a fully non-preemptive scheduler. Non-preemptive scheduling has the additional advantage of reducing memory requirements, as well as improving the dependability of real-time systems (Short 2010). It is however well known that non-preemptive scheduling can be infeasible at low processor utilization levels due to the *long task problem* (Short 2010), where some task has a WCET greater than the deadline of another task.

When considering the theoretical optimality of uniprocessor scheduling algorithms (i.e., without accounting for overheads), then EDF-P is optimal in the sense that any task set that is feasible on a uniprocessor under some other scheduling algorithm is also feasible using EDF-P (Dertouzos 1974). As a result, EDF-P dominates other uniprocessor scheduling algorithms such as FP-P, FP-NP, and EDF-NP.

When using fixed priority scheduling, priority assignment has a significant impact on schedulability (Davis et al. 2016). For FP-P scheduling, Deadline Monotonic Priority Ordering (DMPO) is optimal for constrained-deadline task sets (Leung and Whitehead 1982). In other words, any constrained-deadline task set that is schedula-

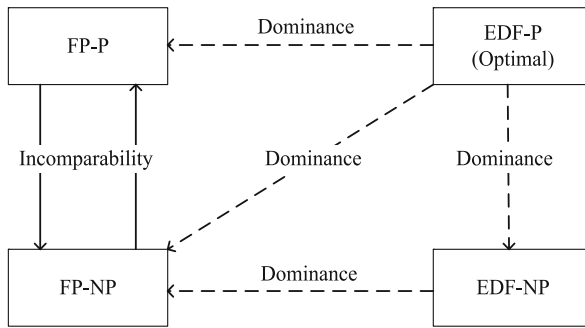


Fig. 1 Dominance relationships between fixed priority and EDF scheduling algorithms

ble under FP-P with some other priority ordering is also guaranteed to be schedulable with DMPO. DMPO is not however optimal if task deadlines are arbitrary (Lehoczký 1990) (i.e. may be larger than their periods). In that case, Audsley’s algorithm (Audsley 2001) can be used to provide an optimal priority assignment.

Within the class of non-preemptive scheduling algorithms, no *work-conserving* algorithm is optimal. This is because in general it is necessary to insert idle time to achieve a feasible schedule (George et al. 1996). EDF-NP is however *weakly* optimal in the sense that if a *work conserving non-preemptive schedule* exists for a task set, then EDF-NP can schedule it (Jeffay et al. 1991), hence EDF-NP dominates FP-NP. With FP-NP scheduling, DMPO is not optimal for constrained-deadline task sets; however, Audsley’s algorithm (Audsley 2001) can again be applied (George et al. 1996).

Comparing the preemptive and non-preemptive paradigms, EDF-P dominates EDF-NP; however, the same is not true with fixed priorities, FP-P does not dominate FP-NP. Instead, they are *incomparable*. In other words, task sets exist that are feasible under FP-NP that are not feasible under FP-P and vice-versa¹. This lack of any dominance relationship means that when fixed priorities are used, some systems are easier to schedule preemptively, while others are easier to schedule non-preemptively. (Optimality for fixed priority scheduling requires limited preemption with final non-preemptive regions (Davis and Bertogna 2012); consideration of that more complex model is however beyond the scope of this paper). Figure 1 shows the dominance relationships between preemptive and non-preemptive fixed priority and EDF scheduling algorithms.

1.1 Speedup factors

Davis et al. (2009a) derived the exact sub-optimality $S = 1/\Omega \approx 1.76$ of FP-P scheduling for constrained-deadline task sets. This exact bound complements the one for implicit-deadline task sets $S = 1/\ln(2) \approx 1.44$ that may be derived from the famous results of Liu and Layland (1973). Davis et al. (2009b) also derived upper and

¹ Task sets that are schedulable under FP-P but not under FP-NP are easily constructed, an example of a task set that is schedulable under FP-NP but not under FP-P is given in Sect. 5.1.

lower bounds of $S = 1/\Omega$ and $S = 2$ on the sub-optimality of FP-P scheduling for arbitrary-deadline task sets. Davis et al. (2015a) completed the exact characterization of the sub-optimality of FP-P scheduling by proving that the exact speedup factor required for arbitrary-deadline task sets is in fact $S = 2$. In the same paper, the authors also extended these results to the case where tasks share resources under mutual exclusion according to the stack resource policy (SRP) (Baker 1991) or the deadline floor protocol (DFP) (Burns et al. 2015), thus providing exact speedup factors comparing FP-P + SRP to EDF + SRP or EDF + DFP.

Davis et al. (2010) derived upper and lower bounds on the speedup factor required to guarantee that all task sets that are feasible under EDF-NP can be scheduled using FP-NP. These bounds are $S = 1/\Omega$ and $S = 2$ respectively for all three classes of task set (implicit, constrained and arbitrary deadline). von der Bruggen et al. (2015) proved upper bounds of $S = 1/\Omega$ for the implicit and constrained deadline cases, thus along with the prior results, showing that these values are exact. Davis et al. (2015a) also completed the exact characterization of the speedup factors required to guarantee schedulability under FP-NP of all EDF-NP feasible task sets by showing that the exact speedup factor for the arbitrary deadline case is $S = 2$ (the same as in the preemptive case for FP-P v. EDF-P).

Thekkilakattil et al. (2013, 2015) quantified the sub-optimality of EDF-NP (with respect to EDF-P), bridging between the preemptive and non-preemptive paradigms. (This result was later extended to the case of global deadline based scheduling (Thekkilakattil et al. 2014)). Abugchem et al. (2015) subsequently provided a tighter upper bound on the sub-optimality of EDF-NP.

In this paper, we focus on quantifying the sub-optimality of uniprocessor FP-NP scheduling with respect to an optimal algorithm such as EDF-P. As a consequence of this work, we also quantify the exact sub-optimality of uniprocessor EDF-NP scheduling. Further, we use the speedup factor metric to compare the performance of FP-P and FP-NP scheduling in both directions, given the lack of any dominance relation between them.

The main contributions of this paper are in determining for uniprocessor systems:

- S1** The exact speedup factor required to guarantee FP-NP feasibility of any EDF-P feasible task set (i.e. the exact *sub-optimality* of FP-NP).
- S2** The exact speedup factor required to guarantee FP-NP feasibility for any task set that is FP-P feasible.
- S3** The exact speedup factor required to guarantee EDF-NP feasibility of any EDF-P feasible task set (i.e. the exact *sub-optimality* of EDF-NP).
- S4** The exact speedup factor required to guarantee FP-P feasibility for any constrained-deadline task set that is FP-NP feasible.

Note, where we refer to the *exact sub-optimality*, or *exact speedup factor* for a non-preemptive scheduling algorithm compared to a preemptive one, then it is important to clarify precisely what we mean. Since non-preemptive scheduling suffers from the long task problem (Short 2010), whereby a task set may be trivially unschedulable because the longest execution time C_{max} of one task exceeds the shortest deadline D_{min} of another, then assuming freely determined task parameters no finite speedup factor exists. This is the case because C_{max}/D_{min} can be made arbitrarily large. Instead, in

this paper we provide exact speedup factors that are parametric in the ratio C_{max}/D_{min} , and thus hold with this minimal constraint on task parameters such that a finite speedup factor exists. We note that with further information about task set characteristics it may be possible to determine more precise speedup factors with narrower scope, i.e. more constraints on their validity. In the extreme, each individual task set effectively has a precise speedup factor which may be computed by referring to all of the parameters of its component tasks.

In this paper, as in previous work on speedup factors (Davis et al. 2009a, b, 2010, 2015a; Thekkilakattil et al. 2013) we assume that changes in processor speed have a linear effect on the time required to execute each task. Considering a uniprocessor system in more detail, our assumption is that the clock frequency may be changed and that this has a linear effect on the speed of all hardware components (processor, memory etc.) thus producing a linear scaling of execution times. Such behaviour is a reasonable approximation for simple systems.

While the results presented in this paper are mainly theoretical, they may also have practical utility in enabling system designers to quantify the maximum penalty for using FP-NP scheduling in terms of the additional processing capacity required as compared to FP-P or EDF-P. This performance penalty can then be weighed against other factors such as the additional overheads (context switch costs and CRPD) incurred by preemptive scheduling, when considering which algorithm to use. We also consider the speedup factor for FP-P scheduling versus FP-NP. This speedup factor is indicative of the increase in processor speed that may be necessary in the worst-case to maintain schedulability when making a choice to switch from using FP-NP scheduling to FP-P.

It is important to note that speedup factors are indicative only of the worst-case performance of one algorithm relative to another, and as such should only be considered for their negative implications. We note that speedup factors can lack the power to discriminate between the performance of different scheduling algorithms and schedulability tests even though their performance may be very different when viewed from the perspective of empirical evaluation (von der Bruggen et al. 2016). The interested reader is referred to recent work by Chen et al. (2017) for a full discussion of the pros and cons of using speedup factors and other resource augmentation metrics.

1.2 Organization

The rest of the paper is organized as follows: the system model is presented in Sect. 2. Section 3 recaps on the schedulability analyses for preemptive and non-preemptive EDF and fixed priority scheduling. Our main results on sub-optimality and speedup factors are presented in Sects. 4 and 5. Section 6 concludes with a summary and a discussion of open problems.

2 System model

In this section we describe the system model, terminology, and notation used in the rest of the paper.

2.1 Task model

We consider the schedulability of a set of sporadic tasks on a uniprocessor system. A task set Γ comprises a static set of n tasks $\{\tau_1, \tau_2, \dots, \tau_n\}$. Each task τ_i is characterized by its minimum inter-arrival time T_i , bounded worst-case execution time C_i , and relative deadline D_i . Deadlines may be *implicit* ($D_i = T_i$), *constrained* ($D_i \leq T_i$), or *arbitrary* (independent of the task's period). The longest execution time of any of the tasks is denoted by $C_{\max} = \max_{\tau_i \in \Gamma} C_i$. Similarly, the shortest deadline is denoted by $D_{\min} = \min_{\tau_i \in \Gamma} D_i$. In the case of fixed priority scheduling, we use $hp(i)$ and $hep(i)$ to denote respectively the set of tasks with priorities higher than, and higher than or equal to that of task τ_i (thus $hep(i)$ includes task τ_i , while $hp(i)$ does not). Similarly, we use $lp(i)$ to denote the set of tasks with priorities lower than that of task τ_i . (Note, we assume that priorities are unique). Further, we use B_i to denote the longest time for which task τ_i may be blocked by a lower priority task that is executing non-preemptively.

The utilization U_i of a task τ_i is given by $U_i = \frac{C_i}{T_i}$ and the utilization of the task set is the sum of the utilizations of the individual tasks $U = \sum_{i=1}^n U_i$.

2.2 Execution time model

To ease readability, and without loss of generality, we assume that the task set of interest is initially executing on a processor of unit speed. Accordingly, we assume that C_i represents the WCET of task τ_i on a processor of speed $S = 1$. We assume a linear relationship between execution time and processor speed. The WCET of task τ_i on a processor of speed S is therefore given by $C_i^S = C_i/S$. Conversely, the speed S required to obtain an execution time of C_i^S is given by $S = C_i/C_i^S$. This model allows us to use processor speedup factors and processor speeds interchangeably. In other words, changing the processor speed from $S = 1$ to $S = x$, is equivalent to speeding up the processor by a factor of x .

2.3 Scheduling model

In this paper, we consider four scheduling algorithms EDF-P, EDF-NP, FP-P, and FP-NP. With EDF-P, at any given time the ready task with the job that has the earliest absolute deadline is executed by the processor. Similarly, with FP-P scheduling, at any given time the processor executes the job of the ready task with the highest priority. By contrast, with EDF-NP, whenever a job is released that has an earlier absolute deadline than the currently executing job, instead of preempting the executing job the scheduler blocks the new job until the currently executing job completes. Only at that point is the ready job with the earliest absolute deadline dispatched for execution. Similarly, with FP-NP scheduling, whenever a higher priority task is released during the execution of a lower priority task τ_i , instead of preempting τ_i the scheduler blocks the higher priority task until τ_i completes its execution. Only at that point is the highest priority

ready task dispatched for execution. We note that all four scheduling algorithms are *work-conserving* and so never idle the processor when there is a task ready to execute.

2.4 Schedulability tests and priority assignment

A task set is said to be *schedulable* with respect to some scheduling algorithm and some system, if all valid sequences of jobs that may be generated by the task set can be scheduled on the system by the scheduling algorithm without any missed deadlines.

A schedulability test is referred to as *sufficient*, with respect to a scheduling algorithm and system, if all of the task sets that are deemed schedulable according to the test are in fact schedulable on the system under the scheduling algorithm. Similarly, a schedulability test is termed *necessary*, if all of the task sets that are deemed unschedulable according to the test are in fact unschedulable on the system under the scheduling algorithm. A test that is both sufficient and necessary is referred to as *exact*.

In fixed priority scheduling, a priority assignment policy P is said to be optimal with respect to some class of task sets (e.g. constrained-deadline), and some class of fixed priority scheduling algorithm (e.g. non-preemptive) if all task sets in the class that are schedulable under the scheduling algorithm using some other priority ordering policy are also schedulable using the priority assignment determined by policy P .

Audsleys Optimal Priority Assignment (OPA) algorithm (Audsley 1991, 2001) (reproduced in Algorithm 1) is an optimal priority assignment algorithm for arbitrary-deadline sporadic task sets in both the preemptive and non-preemptive case.

Algorithm 1 Optimal Priority Assignment (OPA) Algorithm

```

1: for Each priority level  $k$ , lowest first do
2:   for Each unassigned task  $\tau$  do
3:     if task  $\tau$  is schedulable at priority  $k$  with all other unassigned tasks assumed to have higher priorities
       then
4:       Assign task  $\tau$  priority  $k$ 
5:     end if
6:     break (continue outer For loop)
7:   end for
8:   return unschedulable
9: end for
10: return schedulable

```

2.5 Speedup factors

We now provide formal definitions for the terms *speedup factor*, *speedup optimal task set* and *sub-optimality* (Davis et al. 2009a, 2015a). Recall that when we use the term *feasible*, then in the case of fixed priority scheduling, we mean schedulable with an optimal priority assignment.

Definition 1 The exact **speed-up factor** of a scheduling algorithm \mathcal{A} with respect to a scheduling algorithm \mathcal{B} is defined as the *minimum* factor S , $S \geq 1$, such that any

task set that is feasible under algorithm \mathcal{B} on a processor of unit speed, is guaranteed to become feasible under algorithm \mathcal{A} on a processor that is S times faster.

Definition 2 A task set is classified as being a **speed-up optimal task set** for the comparison between scheduling algorithms \mathcal{A} and \mathcal{B} if it is feasible on a processor of unit speed under algorithm \mathcal{B} and requires the processor speed to be increased by the exact speedup factor S for the comparison between the two algorithms (see Definition 1) in order to be feasible under algorithm \mathcal{A} .

We note that for a given comparison, there are typically multiple speedup-optimal task sets. This classification is useful, since in deriving the exact speedup factors for a given comparison, we can restrict our attention to the set of speedup optimal task sets, and their properties.

Definition 3 The **sub-optimality** of a scheduling algorithm \mathcal{A} is defined by its exact speedup factor with respect to an *optimal* scheduling algorithm.

Definition 4 A scheduling algorithm is said to be **optimal** if it can schedule every task set that is feasible under some other scheduling algorithm, on a processor of equivalent speed.

The lower the value of sub-optimality for a particular scheduling algorithm (i.e. its speedup factor compared to an optimal algorithm), then the closer the algorithm is to being optimal, with a value of $S = 1$ implying optimality. We note that FP-P, FP-NP, and EDF-NP are all sub-optimal with respect to an optimal uniprocessor scheduling algorithm such as EDF-P, as illustrated in Fig. 1.

In order to derive speedup factors it is often useful to consider scaling the execution times of all tasks until the task set being considered is only just schedulable. Below, we give alternative but equivalent definitions for *speedup factor* and *speedup optimal task set* using the concept of a *critical scaling factor* (Lehoczký et al. 1989).

Definition 5 Let Ψ be some arbitrary task set, now assume that $\alpha^{\mathcal{A}}(\Psi)$ is the *critical scaling factor*, that is the maximum factor by which the execution times of all of the tasks in Ψ can be scaled, such that the task set is schedulable under algorithm \mathcal{A} on a processor of unit speed. Similarly, $\alpha^{\mathcal{B}}(\Psi)$ for algorithm \mathcal{B} . The exact **speedup factor** S for algorithm \mathcal{A} compared to algorithm \mathcal{B} is given by:

$$S = \sup_{\Psi} \left(\alpha^{\mathcal{B}}(\Psi) / \alpha^{\mathcal{A}}(\Psi) \right)$$

where Ψ ranges over all task sets.

Definition 6 Let $\alpha^{\mathcal{A}}(\Psi)$ be the critical scaling factor for task set Ψ under scheduling algorithm \mathcal{A} , and similarly $\alpha^{\mathcal{B}}(\Psi)$ under algorithm \mathcal{B} . Task set Ψ is said to be a **speedup-optimal task set** with respect to the comparison between scheduling algorithm \mathcal{A} and algorithm \mathcal{B} if $\alpha^{\mathcal{B}}(\Psi) / \alpha^{\mathcal{A}}(\Psi) = S$, where S is the exact speedup factor for the comparison between the two algorithms.

Determining the *exact* value of the speedup factor for the comparison between two algorithms \mathcal{A} and \mathcal{B} requires both necessary and sufficient conditions to be considered. To satisfy the sufficient condition, an *upper bound* on the speedup factor can be derived which represents a *sufficient* increase in processor speed to ensure feasibility under algorithm \mathcal{A} for any task set that was feasible under algorithm \mathcal{B} . To satisfy the necessary condition, a *lower bound* on the speedup factor can be determined which represents the increase in processor speed that is *necessary* to ensure feasibility under algorithm \mathcal{A} for a specific task set that was feasible under algorithm \mathcal{B} . If the upper and lower bounds on the speedup factor match, then the speedup factor has been precisely determined and is said to be *exact*.

Finally, we note that the term *speedup-optimal* can be used to describe a schedulability test.

Definition 7 A schedulability test \mathcal{T} for algorithm \mathcal{A} is said to be a **speedup-optimal schedulability test** for the comparison between algorithm \mathcal{A} and algorithm \mathcal{B} if the exact speedup factor obtained when feasibility is determined for algorithm \mathcal{A} according to test \mathcal{T} is the same as the exact speedup factor obtained when an exact schedulability test for algorithm \mathcal{A} is used.

We note that sufficient schedulability tests can be speedup-optimal, even though they are not optimal (i.e. exact) in terms of schedulability. We return to this point in the conclusions.

3 Schedulability analysis

In this section, we recapitulate schedulability analysis for fixed priority and EDF scheduling under both preemptive and non-preemptive paradigms.

3.1 Fixed priority preemptive scheduling

The schedulability of a set of arbitrary-deadline sporadic tasks under FP-P can be determined using response time analysis (Tindell et al. 1994; Lehoczky 1990). Response time analysis involves calculating the worst-case response time R_i^P of each task τ_i and comparing it to its deadline D_i . To determine schedulability, the analysis must check each job of task τ_i in the longest priority level- i busy period. This busy period starts with a critical instant corresponding to the synchronous arrival of a job of task τ_i and jobs of all higher priority tasks. Jobs of these tasks are then re-released as soon as possible. The length of the priority level- i busy period is given by the solution to the following recurrence relation:

$$A_i^P = \sum_{\forall \tau_j \in \text{hep}(i)} \left\lceil \frac{A_j^P}{T_j} \right\rceil C_j \quad (1)$$

The number of jobs of task τ_i in the busy period is given by $Q_i^P = \lceil \frac{A_i^P}{T_i} \rceil$. The completion time $W_i^P(q)$ of job q of task τ_i relative to the start of the busy period is given by the following recurrence relation:

$$W_i^P(q) = (q + 1)C_i + \sum_{\forall \tau_j \in hp(i)} \left\lceil \frac{W_i^P(q)}{T_j} \right\rceil C_j \quad (2)$$

Iteration starts with $W_i^P(q) = (q + 1)C_i$ and ends either on convergence or when $W_i^P(q) - qT_i > D_i$ in which case the job and therefore the task is unschedulable. Assuming that all Q_i^P jobs in the busy period are schedulable, then the worst-case response time of the task is given by:

$$R_i^P = \max_{q=0,1,2,\dots,Q_i^P-1} (W_i(q) - qT_i) \quad (3)$$

For task sets with constrained deadlines, only the response time of the first job in the busy period need be checked, leading to a simpler exact test (Audsley et al. 1993; Joseph and Pandya 1986), based on the following recurrence relation:

$$R_i^P = C_i + \sum_{\forall \tau_j \in hp(i)} \left\lceil \frac{R_i^P}{T_j} \right\rceil C_j \quad (4)$$

Iteration starts with $R_i^P = C_i$ and ends either on convergence or when $R_i^P > D_i$ in which case the task is unschedulable.

3.2 Fixed priority non-preemptive scheduling

Determining exact schedulability of a task τ_i under FP-NP also requires checking all of the jobs of task τ_i within a priority level- i busy period (Bril et al. 2009). In this case, the busy period starts with an interval of blocking and so its length is given by the solution to the following recurrence relation:

$$A_i^{NP} = B_i + \sum_{\forall \tau_j \in hp(i)} \left\lceil \frac{A_i^{NP}}{T_j} \right\rceil C_j \quad (5)$$

where B_i is the *blocking factor*:

$$B_i = \begin{cases} \max_{\forall \tau_k \in lp(i)} (C_k - \Delta) & i < n \\ 0 & i = n \end{cases} \quad (6)$$

and Δ is the time granularity².

The number of jobs of task τ_i in the busy period is given by $Q_i^{NP} = \lceil \frac{A_i^{NP}}{T_i} \rceil$. The start time $W_i^{NP}(q)$ of job q of task τ_i relative to the start of the busy period is given by the following recurrence relation:

² Without loss of generality, we assume that Δ is the granularity of the processor clock and that $\Delta \ll C_k$ for every task τ_k even when we increase the processor speed.

$$W_i^{NP}(q) = B_i + qC_i + \sum_{\forall \tau_j \in hp(i)} \left\lceil \frac{W_i^{NP}(q) + \Delta}{T_j} \right\rceil C_j \quad (7)$$

Iteration starts with $W_i^{NP}(q) = B_i + qC_i$ and ends either on convergence or when $W_i^{NP}(q) + C_i - qT_i > D_i$ in which case the job and therefore the task is unschedulable. Assuming that all Q_i^{NP} jobs in the busy period are schedulable, then the worst-case response time of the task is given by:

$$R_i^{NP} = \max_{q=0,1,2,\dots,Q_i^{NP}-1} (W_i^{NP}(q) + C_i - qT_i) \quad (8)$$

Note, in the above formulation we use a *ceiling* function with $+\Delta$, rather than the alternative of a *floor* function $+1$, since this assists in the proofs given later in the paper. The two formulations are however equivalent.

We make use of the following *sufficient* schedulability tests for each task τ_i under FP-NP. The first is based on a linear equation (Davis et al. 2010):

$$B_i + \sum_{\forall \tau_j \in hp(i)} \left\lceil \frac{D_i}{T_j} \right\rceil C_j \leq D_i \quad (9)$$

The second, which is only applicable to constrained-deadline task sets is based on a recurrence relation (Davis et al. 2007):

$$\begin{aligned} W_i^{NP} &= C_{max} + \sum_{\forall \tau_j \in hp(i)} \left\lceil \frac{W_i^{NP} + \Delta}{T_j} \right\rceil C_j \\ R_i^{NP} &= W_i^{NP} + C_i \end{aligned} \quad (10)$$

where W_i^{NP} is an upper bound on the longest time from release to the start of any job of task τ_i .

Finally, we also make use of a *necessary* test. This test is valid for task sets with arbitrary deadlines. It simply checks schedulability of the first job in the busy period, if this job is found to miss its deadline, then the task is unschedulable; however, if the job is found to meet its deadline, the task may or may not be schedulable, since subsequent jobs in the busy period may or may not meet their deadlines. Thus the test is *necessary* but not *sufficient*.

$$\begin{aligned} W_i^{NP} &= B_i + \sum_{\forall \tau_j \in hp(i)} \left\lceil \frac{W_i^{NP} + \Delta}{T_j} \right\rceil C_j \\ R_i^{NP} &= W_i^{NP} + C_i \end{aligned} \quad (11)$$

R_i^{NP} given by (11) provides a *lower bound response time*.

This necessary test is used later, in Sect. 5, since an upper bound on the speedup factor for FP-P v. FP-NP which is valid when a necessary test is used to determine FP-NP schedulability is also valid when an exact test is used.

3.3 Preemptive earliest deadline first scheduling

A task set is schedulable under preemptive EDF if and only if in every time interval, the total processor demand requested by the task set is no greater than the length of the interval (Baruah et al. 1990). A task set is EDF-P feasible *if and only if*:

$$\begin{aligned} \sum_{\forall \tau_i \in \Gamma} DBF_i(t) &\leq t \\ \forall t = kT_j + D_j, \forall k \in \mathbb{N}, j &\in [1, n] \\ t &\leq A_n^P \end{aligned} \quad (12)$$

where

$$DBF_i(t) = \max \left(0, 1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor \right) C_i \quad (13)$$

and A_n^P is the length of the longest busy period, given by (1) (Ripoll et al. 1996; Spuri 1996).

4 Exact sub-optimality and speedup factors

In this section, we compare the effectiveness of fixed priority non-preemptive scheduling (FP-NP) with that of preemptive scheduling; both FP-P and EDF-P. We determine the exact sub-optimality of FP-NP. Specifically, we derive the exact speedup factor **S1** required to guarantee feasibility under FP-NP of all EDF-P feasible task sets. Further, we derive the exact speedup factor **S2** required to guarantee feasibility under FP-NP of all FP-P feasible task sets. Surprisingly these two speedup factors are the same (**S1** = **S2**). We also derive an exact speedup factor for the case of FP-NP v. FP-P, when tasks have constrained deadlines. This speedup factor is smaller than the one for the arbitrary-deadline case.

We obtain the exact speedup factors by deriving upper bounds via analysis and lower bounds from example task sets and then showing that they are the same. The example task set we use to provide a lower bound for FP-NP v. EDF-P also applies to EDF-NP v. EDF-P, hence we also obtain **S3**, the exact sub-optimality of EDF-NP, since our lower bound is the same as the upper bound recently published by Abugchem et al. (2015).

Lemma 1 *An upper bound on the speedup factor required such that FP-NP, using optimal priority assignment, can schedule any arbitrary-deadline sporadic task set that is feasible under EDF-P is given by:*

$$S = 2 + \frac{C_{max}}{D_{min}}$$

Proof We show that the speedup factor in the lemma is enough to ensure schedulability under FP-NP according to the sufficient test given by (9) using DMPO, since that suffices to also prove schedulability with an exact test and optimal priority assignment.

Comparing (9) and (13) and assuming DMPO we observe the following.

$$\sum_{\forall \tau_j \in \Gamma} DBF_j(2D_i) \geq \sum_{\forall \tau_j: D_j \leq D_i} \left\lceil \frac{D_i}{T_j} \right\rceil C_j \geq \sum_{\forall \tau_j \in \text{hep}(i)} \left\lceil \frac{D_i}{T_j} \right\rceil C_j \quad (14)$$

Note in (14) $2D_i$ is a value we have chosen for convenience since the inequality is then useful later in the proof. From (9), (14), and the fact that $B_i \leq C_{\max}$ then schedulability under FP-NP is assured on a processor of speed S provided that for every task τ_i :

$$\frac{C_{\max} + \sum_{\forall \tau_k \in \Gamma} DBF_k(2D_i)}{S} \leq D_i \quad (15)$$

Since the task set is schedulable under EDF-P on a processor of unit speed, then it follows from (12) that $\sum_{\forall \tau_k \in \Gamma} DBF_k(2D_i) \leq 2D_i$. Substituting into (15) and rearranging, we have:

$$S \geq 2 + \frac{C_{\max}}{D_i}$$

Substituting D_{\min} for D_i gives an upper bound on the speed-up factor required. \square

Lemma 2 *An upper bound on the speedup factor required such that FP-NP, using optimal priority assignment, can schedule any arbitrary-deadline sporadic task set that is feasible under FP-P scheduling is given by:*

$$S = 2 + \frac{C_{\max}}{D_{\min}}$$

Proof Follows directly from Lemma 1 and the fact that EDF-P can schedule all task sets that are feasible under FP-P scheduling (Dertouzos 1974). \square

Lemma 3 *A lower bound on the speedup factor required such that FP-NP, using optimal priority assignment, can schedule any implicit, constrained, or arbitrary-deadline sporadic task set that is feasible under EDF-P (or FP-P) is given by:*

$$S = 1 + \frac{C_{\max}}{D_{\min}}$$

Proof Consider the following task set, where k is an integer and $k > 1$:

$$\begin{aligned} \tau_1: C_1 &= k - 1, D_1 = k, T_1 = k \\ \tau_2: C_2 &= k^2 + 1, D_2 = \infty, T_2 = \infty \end{aligned}$$

We note that the task set is trivially schedulable on a processor of unit speed using either EDF-P or FP-P. For the task set to be schedulable with FP-NP effectively requires that the execution time of both tasks³ (i.e. $k^2 + k$) can be accommodated within the smallest deadline $D_1 = k$.

Hence we have $S \geq (k^2 + k)/k = k + 1$. Since $\frac{C_{max}}{D_{min}} = k + \frac{1}{k}$ we obtain:

$$S \geq 1 + \frac{C_{max}}{D_{min}} - \frac{1}{k}$$

and so as $k \rightarrow \infty$ we have a lower bound of:

$$S \geq 1 + \frac{C_{max}}{D_{min}}.$$

□

Theorem 1 *The exact sub-optimality (S3) of EDF-NP, i.e., the exact speedup factor required such that EDF-NP can schedule any implicit, constrained, or arbitrary-deadline sporadic task set that is feasible under EDF-P, is given by:*

$$S = 1 + \frac{C_{max}}{D_{min}}$$

Proof Follows from a consideration of the task set in Lemma 3. For the task set to be schedulable under EDF-NP also requires that the total execution time of both tasks can be accommodated within the smallest deadline resulting in the same requirement on the speedup factor. Since the lower bound from Lemma 3 matches the upper bound given by Abugchem et al. (2015) the speedup factor is exact. □

Lemma 4 *An upper bound on the speedup factor required such that FP-NP scheduling, using optimal priority assignment, can schedule any constrained-deadline sporadic task set that is feasible under FP-P scheduling is given by:*

$$S = 1 + \frac{C_{max}}{D_{min}}$$

Note this Lemma does not apply to arbitrary-deadline tasks sets.

Proof Let Γ be a task set that is schedulable under FP-P scheduling on a processor of unit speed using DMPO, which is optimal in the constrained deadline case. We will prove that Γ is schedulable on a processor of speed S under FP-NP scheduling using the same priority ordering. We note that this ordering is not necessarily optimal for FP-NP scheduling, but nevertheless suffices to prove feasibility.

Let W_i^P be the completion time of the first job of task τ_i in the priority level- i busy period under FP-P scheduling. Since all tasks are schedulable and have constrained deadlines, then $W_i^P = R_i^P \leq D_i$. We consider two cases.

³ For ease of presentation, and since it does not affect the result, we omit the small reduction in blocking due to the time granularity $\Delta \ll 1$.

Case 1: $W_i^P \geq D_{min}$

Let $E_i^P(t)$ equate to C_i plus the maximum amount of interference from tasks of higher priority than τ_i released in an interval of length t :

$$E_i^P(t) = C_i + \sum_{\forall \tau_j \in hp(i)} \left\lceil \frac{t}{T_j} \right\rceil C_j \quad (16)$$

From (4), it follows that $E_i^P(W_i^P) = W_i^P = R_i^P$ where R_i^P is the exact response time of task τ_i under FP-P scheduling.

Let $E_i^{NP}(t)$ be the maximum amount of interference from tasks of higher priority than τ_i released in an interval of length t including any releases at the end of the interval:

$$E_i^{NP}(t) = \sum_{\forall \tau_j \in hp(i)} \left\lceil \frac{t + \Delta}{T_j} \right\rceil C_j \quad (17)$$

From the sufficient test for constrained-deadline task sets under FP-NP scheduling (10) we have $E_i^{NP}(W_i^{NP}) + C_{max} + C_i = W_i^{NP} + C_i$ where W_i^{NP} is an upper bound on the time from the release of a job of task τ_i until it starts to execute under FP-NP scheduling, and $W_i^{NP} + C_i$ is an upper bound on the task's response time.

From (16) and (17), observe that $\forall x \geq \Delta$ and $\forall t \geq x$, the following inequality holds:

$$E_i^{NP}(t - x) + C_i \leq E_i^P(t) \quad (18)$$

To ensure schedulability under FP-NP scheduling, we speed up the processor by some factor $S \geq 1$ such that the latest completion time of task τ_i under FP-NP scheduling is no greater than W_i^P the completion time under FP-P scheduling on a processor of unit speed. It follows that the start time of τ_i must be at the latest $W_i^P - \frac{C_i}{S}$. An upper bound on the interference from higher priority tasks in an interval of this length is given by $E_i^{NP}(W_i^P - \frac{C_i}{S})$. To complete task τ_i by W_i^P the processor must be able to complete execution of the interference $E_i^{NP}(W_i^P - \frac{C_i}{S})$, task τ_i itself i.e. C_i , and the maximum amount of blocking C_{max} . Schedulability under FP-NP is therefore ensured provided that:

$$\frac{C_{max} + E_i^{NP}\left(W_i^P - \frac{C_i}{S}\right) + C_i}{S} \leq W_i^P \quad (19)$$

On the faster processor of speed S the execution time of τ_i cannot be less than the time granularity ($\frac{C_i}{S} \geq \Delta$). It follows from (18) that $E_i^{NP}(W_i^P - \frac{C_i}{S}) + C_i \leq E_i^P(W_i^P)$. Since $E_i^P(W_i^P) = W_i^P$, we can substitute W_i^P into (19) in place of $E_i^{NP}(W_i^P - \frac{C_i}{S}) + C_i$ again giving an inequality which is sufficient to ensure schedulability under FP-NP. Rearranging we have:

$$S \geq 1 + \frac{C_{max}}{W_i^P} \quad (20)$$

From the assumption of this case (Case 1) $W_i^P \geq D_{min}$ and hence the task set is guaranteed to be schedulable on a processor of speed S , where:

$$S \geq 1 + \frac{C_{max}}{D_{min}} \quad (21)$$

Case 2: $W_i^P < D_{min}$

Since deadlines are constrained, there are no tasks with periods that are less than D_{min} , and so under FP-P scheduling on a processor of unit speed, we have:

$$W_i^P = C_i + \sum_{\forall j \in hp(i)} C_j \quad (22)$$

In this case, to ensure schedulability under FP-NP on a processor of speed S , we simply require that task τ_i completes before D_{min} , hence we require that:

$$\frac{C_{max} + E_i^{NP} \left(D_{min} - \frac{C_i}{S} \right) + C_i}{S} \leq D_{min} \quad (23)$$

where S is the processor speed.

Following the same logic as in Case 1, we observe that $E_i^{NP} \left(D_{min} - \frac{C_i}{S} \right) + C_i \leq E_i^P(W_i^P) = W_i^P$. Since in this case (Case 2) $W_i^P < D_{min}$ substituting into (23) and re-arranging we obtain the speed S at which the task set is guaranteed to be schedulable:

$$S \geq 1 + \frac{C_{max}}{D_{min}}. \quad (24)$$

□

Theorem 2 *The exact speedup factor required such that FP-NP, using optimal priority assignment, can schedule any implicit, or constrained-deadline sporadic task set that is feasible under FP-P scheduling is given by:*

$$S = 1 + \frac{C_{max}}{D_{min}}$$

Proof Proof follows from the lower bound given by Lemma 3 and the upper bound given by Lemma 4 which have the same value. □

Lemma 5 *A lower bound on the speedup factor required such that FP-NP scheduling, using optimal priority assignment, can schedule any arbitrary-deadline sporadic task set that is feasible under FP-P scheduling is given by:*

$$S = 2 + \frac{C_{max}}{D_{min}}$$

Proof Consider the following task set:

τ_i with $i = 1, \dots, k-1$: $C_i = 1$, $D_i = k+1$, $T_i = k$

τ_k : $C_k = 1$, $D_k = k+1$, $T_k = k+1$

τ_{k+1} : $C_{k+1} = k^2$, $D_{k+1} = \infty$, $T_{k+1} = \infty$

This task set is trivially schedulable on a processor of unit speed under FP-P. In the priority order given, then for $j = 1$ to k , task τ_j has a response time of j . Further, task τ_{k+1} executes in the one spare unit of execution time in each Least Common Multiple $k(k+1)$ of the periods of tasks τ_1 to τ_k and therefore has a worst-case response time of $k^3(k+1)$.

Under FP-NP on a processor of speed $S \geq 1$ consider the operation of Audsleys OPA algorithm, which is optimal in this case (George et al. 1996). First, task τ_{k+1} is assigned the lowest priority as it is trivially schedulable at that priority on a processor of unit speed or higher. There are then two cases to consider⁴.

Case 1 τ_k is assigned the next higher priority level above τ_{k+1} . In this case, task τ_k is subject to blocking due to task τ_{k+1} and interference (before it starts to execute) from tasks τ_1 to τ_{k-1} . Considering the critical instant for task τ_k , there are two possible scenarios which could result in the task being schedulable. In the first scenario, the first jobs of all tasks except τ_k must complete their execution strictly before the second jobs of tasks τ_1 to τ_{k-1} are released at time k . This allows task τ_k to start executing before time k , thus avoiding interference from the second job of each higher priority task. For this to happen implies the following constraint: $S > (k^2 + k - 1)/k = k + (k - 1)/k$. Further, task τ_k must also complete by time $k + 1$, which gives the weaker constraint $S \geq (k^2 + k)/(k + 1) = k$. The alternative scenario is that task τ_k does not get to start before the second jobs of tasks τ_1 to τ_{k-1} are released at time k . In this scenario, for task τ_k to be schedulable, the first job of task τ_{k+1} , the first and second jobs of tasks τ_1 to τ_{k-1} , and the first job of task τ_k must complete their execution by time $k + 1$, which leads to the constraint that $S \geq (k^2 + 2k - 1)/(k + 1) = k + (k - 1)/(k + 1)$.

Case 2 τ_{k-1} is assigned the next higher priority level above τ_{k+1} (since τ_1 to τ_{k-1} are identical this is effectively the only other option aside from Case 1 for this priority level). Considering the critical instant for task τ_{k-1} , there are two possible scenarios which could result in the task being schedulable. In the first scenario, the first jobs of all tasks except τ_{k-1} must complete their execution strictly before the second jobs of tasks τ_1 to τ_{k-2} are released at time k . This allows task τ_{k-1} to start executing before time k , thus avoiding interference from the second job of each higher priority task. As in Case 1, this implies the following constraint: $S > (k^2 + k - 1)/k = k + (k - 1)/k$. In addition task τ_{k-1} must also complete by time $k + 1$, which again gives $S \geq (k^2 + k)/(k + 1) = k$. The alternative scenario is that the first job of task τ_{k-1} does not get to start before the second jobs of tasks τ_1 to τ_{k-1} are released at time k . In this scenario, for task τ_{k-1} to be schedulable, then the first job of task τ_{k+1} , the first and second jobs of tasks τ_1 to τ_{k-2} , and the first job of task τ_{k-1} must complete their execution by time $k + 1$, which leads to the constraint $S \geq (k^2 + 2k - 2)/(k + 1) = k + (k - 2)/(k + 1)$.

Considering both Case 1 and Case 2, then the minimum speed necessary for FP-NP schedulability is $S \geq (k^2 + 2k - 2)/(k + 1) = k + (k - 2)/(k + 1)$. Since $C_{\max}/D_{\min} = k^2/(k + 1)$ we obtain:

⁴ Again, for ease of presentation, and since it does not affect the result, we omit the small reduction in blocking due to the time granularity $\Delta \ll 1$.

$$\begin{aligned}
 S &\geq \frac{C_{\max}}{D_{\min}} + \frac{k^2 + 2k - 2}{k + 1} - \frac{k^2}{k + 1} \\
 &= \frac{C_{\max}}{D_{\min}} + \frac{2k - 2}{k + 1}
 \end{aligned}$$

As $k \rightarrow \infty$ this gives a lower bound of $S = 2 + \frac{C_{\max}}{D_{\min}}$ for the speedup factor. \square

Theorem 3 *The exact speedup factor (S2) required such that FP-NP scheduling, using optimal priority assignment, can schedule any arbitrary-deadline sporadic task set that is feasible under FP-P scheduling is given by:*

$$S = 2 + \frac{C_{\max}}{D_{\min}}$$

Proof Proof follows from the lower bound given by Lemma 5 and the upper bound given by Lemma 2 which have the same value. \square

From Theorems 2 and 3, it is interesting to note that when comparing FP-NP against FP-P scheduling, then the relaxation from constrained-deadline task sets to the general case of arbitrary-deadline tasks results in an increase in the exact speedup factor required from

$$S = 1 + \frac{C_{\max}}{D_{\min}} \text{ to } 2 + \frac{C_{\max}}{D_{\min}}$$

Theorem 4 *The exact sub-optimality (S1) of FP-NP i.e. the exact speedup factor required such that FP-NP scheduling, using optimal priority assignment, can schedule any arbitrary-deadline sporadic task set that is feasible under EDF-P is given by:*

$$S = 2 + \frac{C_{\max}}{D_{\min}}$$

Proof Lemma 1 shows that the speedup factor in the theorem is a valid upper bound. Lemma 5 and the fact that EDF-P dominates FP-P shows that it is also a valid lower bound and hence exact for arbitrary-deadline task sets. \square

5 Preemptive versus non-preemptive fixed priority scheduling

Preemptive and non-preemptive fixed priority scheduling are incomparable, i.e., there are task sets that FP-P can schedule that FP-NP cannot and vice versa, hence there are non-trivial speed-up factors in both directions between these two scheduling algorithms. These speedup factors allow system designers to determine the increase in processor speed that may be necessary in the worst-case to maintain schedulability when making a choice to switch from using one of these scheduling algorithms to the other.

In this section, we derive upper and lower bounds on the processor speed-up factor that guarantees feasibility under FP-P for all constrained deadline task sets that are

feasible under FP-NP on a unit speed processor, thus deriving exact speedup factor **S4**. In Sect. 5.1 we derive a lower bound on the speed-up factor that guarantees FP-P feasibility for constrained and arbitrary deadline task sets that are FP-NP feasible. In Sect. 5.2 we derive an upper bound that is valid for arbitrary-deadline task sets. Finally in Sect. 5.3 we derive a tight upper bound that is valid for constrained-deadline task sets. This upper bound matches the lower bound from Sect. 5.1 and is thus exact in the case of constrained-deadline task sets, as noted in Sect. 5.4.

5.1 Lower bound on the speedup factor for FP-P v. FP-NP

Theorem 5 *A lower bound on the speedup factor required such that FP-P scheduling, using optimal priority assignment, can schedule any constrained- deadline sporadic task set that is feasible under FP-NP scheduling is given by:*

$$S = \sqrt{2}$$

Proof Consider the following task set scheduled on a processor of unit speed under FP-NP scheduling.

$$\begin{aligned}\tau_1: C_1 &= 2 - \sqrt{2}, D_1 = 1, T_1 = 1 \\ \tau_2: C_2 &= \sqrt{2} - 1, D_2 = \sqrt{2}, T_2 = \infty \\ \tau_3: C_3 &= \sqrt{2} - 1, D_3 = \sqrt{2}, T_3 = \infty\end{aligned}$$

This task set is schedulable with DMPO under FP-NP as evidenced by the exact schedulability test embodied in (8). The response times of the three tasks are as follows: $R_1 = 1 - \Delta$, $R_2 = \sqrt{2} - \Delta$, $R_3 = \sqrt{2}$. Note, that in each case we need only examine the response time of the first job. For task τ_1 , the priority level-1 busy period is of length 1 and so includes only one job of the task, while tasks τ_2 and τ_3 have infinite periods and so only give rise to a single job. The schedule starting with task τ_1 is illustrated in Fig. 2.

Next, consider the same task set scheduled on a processor of speed $S = \sqrt{2}$ under FP-P scheduling, again using DMPO which is optimal in this case. The scaled task execution times are now $C_1^S = \sqrt{2} - 1$, $C_2^S = (2 - \sqrt{2})/2$, and $C_3^S = (2 - \sqrt{2})/2$. The schedule is as illustrated in Fig. 3, again starting with task τ_1 . In this case, the worst-case response time of task τ_3 is 1. Further, any increase in the execution times of the tasks (i.e. by using a smaller speedup factor) would result in task τ_3 missing its deadline, due to preemption by the second job of task τ_1 which is released at time $t = 1$. Hence the speedup factor required by this task set is $S = \sqrt{2}$. \square

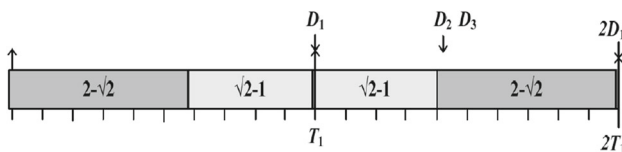
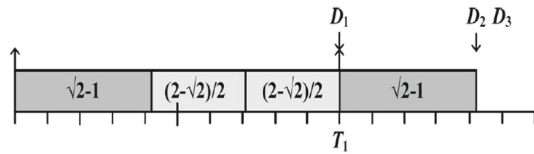


Fig. 2 Fixed priority non-preemptive schedule for the example task set in Theorem 5

Fig. 3 Fixed priority preemptive schedule for the example task set in Theorem 5



5.2 Upper bound on the speedup factor for FP-P v. FP-NP for arbitrary-deadline task sets

Theorem 6 *An upper bound on the speedup factor required such that FP-P scheduling, using optimal priority assignment, can schedule any arbitrary-deadline sporadic task set that is feasible under FP-NP scheduling is given by:*

$$S = 2$$

Proof Since EDF-P dominates FP-NP and Theorem 2 from (Davis et al. 2009a) states that an upper bound on the speed-up factor required to guarantee FP-P feasibility of any EDF-P feasible task set is $S = 2$ then such an increase in processor speed must also be sufficient to guarantee FP-P feasibility of all FP-NP feasible task sets. \square

5.3 Upper bound on the speedup factor for FP-P v. FP-NP for constrained-deadline task sets

We next derive an upper bound on the speedup factor required to guarantee feasibility under FP-P for all constrained-deadline task sets that are feasible under FP-NP. Note, the restriction to constrained-deadline task sets allows us to derive a tighter upper bound than that given in Theorem 6. (We note that any upper bound valid for constrained-deadline task sets also applies to implicit-deadline task sets since the former class includes the latter). The upper bound that we are interested in compares exact schedulability for FP-NP assuming optimal priority assignment (OPA) with exact schedulability for FP-P again assuming optimal priority assignment (DMPO in this case). Note, since DMPO is not optimal for FP-NP the two priority assignments may be different.

We use a weaker condition to derive a sound upper bound on the speedup factor. We compare schedulability according to the *necessary* test for FP-NP scheduling given by (11) assuming optimal priority assignment using Audsley's OPA algorithm against an exact test for FP-P scheduling assuming the *same priority ordering* that OPA generates for FP-NP scheduling. These relaxations can only increase the speedup factor required, and hence the bound we obtain is also a valid upper bound for the case we are ultimately interested in with exact tests and optimal priority assignment for both scheduling algorithms.

Without loss of generality, we base our proofs on the set Z of all constrained-deadline task sets where each task set $V \in Z$ has parameters such that it is deemed schedulable using the necessary test for FP-NP scheduling and the OPA algorithm for priority assignment, and scaling of the execution times of all tasks in V by any factor

> 1 (equivalent to a decrease in processor speed) would cause the task set to become unschedulable. Hence the *critical scaling factor* $\alpha^{FP-NP}(V) = 1$ with respect to the necessary test. For ease of reference later, we refer to Z as containing the *FP-NP critical task sets*. Further, also without loss of generality, we assume that the OPA algorithm checks tasks in the reverse of deadline monotonic order and therefore at each priority level from the lowest priority upwards, it assigns the schedulable task with the longest deadline.

Proof of the upper bound on the speedup factor required to guarantee feasibility under FP-P for all constrained-deadline task sets that are feasible under FP-NP requires a number of steps. Below we outline these steps and the theorems and lemmas involved.

1. We first derive a number of properties that hold for any task set that is speedup optimal for this problem. These properties are given in Theorem 7 and proved via Lemmas 6, 7, and 8. This eases the burden of proof of the upper bound, since we need only consider speedup optimal task sets with these properties in our subsequent derivation.
2. We then make a case distinction, splitting the proof of the upper bound speedup factor into two parts depending on the relative values of three factors A , B , and X . A represents the execution time of the constraining task. Note a constraining task is one that cannot have its execution time increased without a deadline being missed. B and X represent the total execution times of two different categories of interference from higher priority tasks. These categories are explained later. The values are normalised such that $X + B = 1$. The two cases considered are $A + X \leq 1$ and $A + X > 1$.
3. In the first case ($A + X \leq 1$) Theorem 8 proves an upper bound on the speedup factor, using the results of Lemmas 9 and 10.
4. In the second case ($A + X > 1$), the proof is split into two parts. Theorem 9 derives an upper bound on the speedup factor subject to the restriction that there is no task that is part of X that is re-released during the interval $[1, A + X)$. Theorem 10 covers the case where there is such a task, making use of Lemmas 11 and 12 which prove certain properties of the task set in this case.
5. Finally, Theorem 11 combines the results of Theorems 8, 9, and 10 giving an upper bound on the speedup factor that holds in all cases. Theorem 12 completes this section, showing that the speedup factor is exact for constrained-deadline task sets.

Theorem 7 *For the comparison between FP-P v. FP-NP (assuming that the necessary test and OPA algorithm are used for FP-NP, and an exact test and the same priority ordering are used for FP-P) there exists a speedup optimal task set Γ that exhibits the highest speedup factor of any constrained-deadline sporadic task set which has the following properties under FP-P scheduling.*

1. *When the execution times of all of the tasks in the task set are scaled such that it is just schedulable according to an exact test for FP-P scheduling, the lowest priority task τ_A is the only **constraining task**. (A constraining task is one that cannot have its execution time increased without missing its deadline).*
2. *Under FP-NP scheduling, with the task set scaled such that it is just schedulable according to the necessary test, then the constraining task from FP-P scheduling has its lower bound response time equal to its deadline.*

3. The task τ_L at priority level $n - 1$ has an execution time $C_L > C_A$, where C_A is the execution time of the lowest priority task.

Proof Follows from Lemmas 6, 7, and 8, given below \square

Lemma 6 *For the comparison FP-P v. FP-NP (assuming that the necessary test and OPA algorithm are used for FP-NP, and an exact test and the same priority ordering are used for FP-P), let Z be the set of FP-NP critical task sets $V \in Z$. By definition, this set contains at least one task set that is speedup-optimal with respect to the class of task sets with constrained deadlines. Let Y be a subset of Z (i.e. $Y \subseteq Z$) such that every task set in Y has a single constraining task under FP-P scheduling when scaled such that it is just schedulable according to an exact test, and that task has the lowest priority. The set Y contains at least one speedup-optimal task set.*

Proof We assume (for contradiction) that there is a task set $S \in Z \setminus Y$ (note $Z \setminus Y$ contains all of the task sets that are in Z but not in Y) that has an FP-P scaling factor $\alpha^{FP-P}(S)$ strictly smaller than that of any task set in Y . Let τ_i be the highest priority constraining task in S . Since $S \notin Y$ then $\tau_i \neq \tau_n$. We create a new task set V from S by removing all tasks of lower priority than τ_i . Since under FP-P scheduling, the response time of task τ_i and all higher priority tasks is unaffected by the removal of the lower priority tasks, the FP-P scaling factor remains the same $\alpha^{FP-P}(V) = \alpha^{FP-P}(S)$. Further, since removing tasks cannot decrease, but may increase the FP-NP scaling factor, it follows that the speedup factor required by task set V is at least as great as that required by task set S . This contradicts the assumption and so there must be at least one speedup optimal task set in Y . \square

Lemma 7 *For the comparison FP-P v. FP-NP (assuming that the necessary test and OPA algorithm are used for FP-NP, and an exact test and the same priority ordering are used for FP-P), let Z be redefined as the set Y from Lemma 6, thus every task set in Z now has a single constraining task under FP-P scheduling when scaled such that it is just schedulable according to an exact test, and that task has the lowest priority. Further, let Y be redefined as follows: $Y \subseteq Z$ such that every task set in Y is such that the constraining task under FP-P scheduling has its lower bound response time equal to its deadline when scheduled according to FP-NP scheduling. The set Y contains at least one speedup-optimal task set.*

Proof We assume (for contradiction) that there is a task set $S \in Z \setminus Y$ (note $Z \setminus Y$ contains all of the task sets that are in Z but not in Y) that has an FP-P scaling factor $\alpha^{FP-P}(S)$ strictly smaller than that of any task set in Y . We create a new task set V from S by reducing the deadline of the constraining task for FP-P scheduling to its lower bound response time determined under FP-NP scheduling (with $\alpha^{FP-NP}(S) = 1$). Since this has no effect on the scaling factor under FP-NP, $\alpha^{FP-NP}(V) = \alpha^{FP-NP}(S)$ and decreases the scaling factor for FP-P i.e. $\alpha^{FP-P}(V) < \alpha^{FP-P}(S)$, it follows that the speedup factor required by task set V is larger than that required by S . This contradicts the assumption and so there must be at least one speedup optimal task set in Y . \square

Lemma 8 *For the comparison FP-P v. FP-NP (assuming that the necessary test and OPA algorithm are used for FP-NP, and an exact test and the same priority ordering are used for FP-P), let Z be redefined as the set Y defined by Lemma 7, thus every task set in Z now has a single constraining task under FP-P scheduling when scaled such that it is just schedulable according to an exact test, and that task has the lowest priority. Further, the constraining task has a deadline equal to its lower bound response time under FP-NP scheduling. Let Y be redefined as follows: $Y \subseteq Z$ such that every task set in Y has a task τ_L at priority level $n - 1$ that has an execution time $C_L \geq C_A$, where C_A is the execution time of the lowest priority task. The set Y contains at least one speedup-optimal task set.*

Proof We assume (for contradiction) that there is a task set $S \in Z \setminus Y$ (note $Z \setminus Y$ contains all of the task sets that are in Z but not in Y) that has a speedup factor strictly greater than that of any task set in Y . By the assumption for contradiction, task set S has a task τ_L at priority level $n - 1$ that has an execution time $C_L < C_A$. (Note speedup optimal task sets must trivially have at least two tasks; otherwise the speedup factor would be 1). The lower bound response time of task τ_L is given by:

$$W_L^{NP} = C_A + \sum_{\forall \tau_j \in HP(L)} \left\lceil \frac{W_L^{NP} + \Delta}{T_j} \right\rceil C_j \quad (25)$$

where $HP(L)$ is the set of higher priority tasks excluding both τ_L and τ_A , W_L^{NP} in the solution obtained on convergence, and thus:

$$R_L^{NP} = W_L^{NP} + C_L \quad (26)$$

As τ_L is deemed schedulable by the necessary test for FP-NP, and has a constrained deadline, then $R_L^{NP} \leq D_L \leq T_L$. Comparing (25) and (26) with (33) and (34) which give the lower bound response time R_A^{NP} of task τ_A , we observe that since $C_L < C_A$ then it follows that $R_L^{NP} \geq R_A^{NP}$. From Lemma 7 we have $R_A^{NP} = D_A$ thus it follows that $T_L \geq D_L \geq R_L^{NP} \geq R_A^{NP} = D_A$, and hence task τ_A can only be subject to interference from one job of task τ_L before its deadline. We therefore form a new task set V from S by removing task τ_L and increasing the execution time of task τ_A by C_L . This has no effect on the response time of τ_A under FP-P scheduling, so $\alpha^{FP-P}(V) = \alpha^{FP-P}(S)$; however, the scaling factor permitted according to the necessary test for FP-NP scheduling either stays the same or increases hence $\alpha^{FP-NP}(V) \geq \alpha^{FP-NP}(S)$. This can be seen by considering that the start time of τ_A is reduced by at least C_L , while its execution time is increased by exactly C_L . It follows that the speedup factor required by task set V is at least as great as that required by S . Repeated application of this process, which removes a task each time, must eventually result in a task set $V \in Y$ (Note reduction to one task is not possible since such a task set cannot be speedup optimal). This contradicts the assumption and so there must be at least one speedup optimal task set in Y . \square

In the following, when considering the processor speedup factor needed to ensure schedulability under FP-P we note that it is sufficient to simply ensure schedulability

of the constraining task, since by definition, that task is the one that requires the largest scaling factor in order to be schedulable.

In the following proofs, we consider an abstract representation of the schedule for a speedup optimal task set under FP-NP and FP-P scaled such that it is just schedulable in each case, see Fig. 4. This representation reflects the *necessary* test for FP-NP scheduling given by (11), it is not a schedule as such, but rather it shows the total amount of execution from a number of sources. Execution of the constraining task τ_A is represented by A . Since we are interested in an upper bound on the speedup factor we may assume that task τ_A which constrains the scaling factor for FP-P also constrains the scaling factor for FP-NP (if any other task constrained the scaling factor in the FP-NP case, then that could only reduce the speedup factor needed). In a FP-NP schedule where task τ_A takes its lower bound response time [according to the necessary test in (11)] it is the last task to execute and starts at W_A^{NP} . Recall from Theorem 7 that all other tasks have higher priorities than the constraining task τ_A , and the priority order considered is the same for both FP-NP and FP-P. We divide execution of the higher priority tasks into two components. Execution of those higher priority tasks which are not released again between the start time W_A^{NP} of task τ_A in the FP-NP schedule and its deadline at D_A are represented by B . Execution of those higher priority tasks which are released again in this interval are represented by X . The latter tasks may cause additional interference under preemptive scheduling.

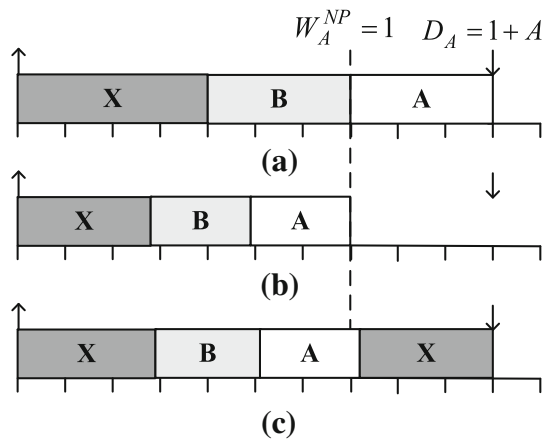
To ease understanding, we now illustrate this representation using a simple example. Consider a task set comprising three tasks with the following parameters (C, D, T) : $\tau_1 (3, 7, \infty)$, $\tau_2 (4, 7 + \epsilon, 7 + \epsilon)$, $\tau_3 (3, 10, 10)$. Task τ_3 has the lowest priority and is the constraining task; its execution is represented by A . Task τ_2 may be released again between the start time (i.e. 7) and the deadline (i.e. 10) of τ_3 in a FP-NP schedule which starts with synchronous release of all tasks, and so its execution is represented by X . Finally, task τ_2 may not be re-released in this interval and its execution is represented by B . This example corresponds to the diagram in Fig. 4a. Note in the proofs that follow X and B may comprise execution from multiple tasks, and thus Fig. 4 is not intended to show a schedule as such, but rather to show in an abstract way the total amount of execution A , B , and X from the different sources.

For ease of presentation and to simplify the formulae involved, in the following (and in Fig. 4), we normalise the execution times such that $W_A^{NP} = 1$, and thus $D_A = 1 + A$.

We can identify two scenarios in which a task set can be scaled under FP-P so that it is schedulable. Scenario (i) is illustrated in Fig. 4b. Here the task set is scaled such that all of the execution that completed by D_A under FP-NP scheduling completes by the start time of task τ_A at W_A^{NP} in that schedule. Scenario (ii) is illustrated by Fig. 4c. Here the task set is scaled such that all of the execution released by D_A completes by D_A . Since it suffices to show that the task set is schedulable in either scenario, we may compute the speedup factors required for each and then take the minimum of them as a valid bound.

In the following, we make a case distinction based on the value of $A + X$. Theorem 8, supported by Lemmas 9 and 10, covers the case where $A + X \leq 1$. Note that Lemma 9 does not depend on the value of $A + X$. Theorems 9 and 10, supported by Lemmas 11 and 12, cover the case where $A + X > 1$.

Fig. 4 Abstract representation of FP-NP schedule (a), and FP-P schedules (b) and (c)



Lemma 9 *An upper bound on the speedup factor required such that FP-P scheduling, with the same priority ordering as used by FP-NP, can schedule any constrained-deadline sporadic task set deemed schedulable according to the necessary test for FP-NP is given by:*

$$S_1 = \frac{1 + A}{1} \quad (27)$$

Proof We note from Theorem 7 that the constraining task τ_A for FP-P scheduling is at the lowest priority, and since the priority order used is the same it is also at the lowest priority under FP-NP scheduling. We now derive the speedup factor necessary for schedulability under FP-P in *scenario (i)* described above, as illustrated in Fig. 4b. Consider the interval of length $W_A^{NP} = 1$. We observe that the amount of interference considered in any interval of length t (not including blocking) is no greater in the preemptive case than in the non-preemptive case. This can be seen by comparing the summation terms in (4) and (11). Hence provided we can accommodate $X + B + A = 1 + A$ in a time interval of length $X + B = 1$ (as illustrated in Fig. 4b) then task τ_A must be schedulable under FP-P. This leads to a speedup factor of $S_1 = \frac{1+A}{1}$. \square

Lemma 10 *Subject to the constraint that $A + X \leq 1$, an upper bound on the speedup factor required such that FP-P scheduling, with the same priority ordering as used by FP-NP, can schedule any constrained-deadline sporadic task set deemed schedulable according to the necessary test for FP-NP is given by:*

$$S_2 = \frac{1 + A + X}{1 + A} \quad (28)$$

Proof We note from Lemma 6 that the constraining task τ_A for FP-P scheduling is at the lowest priority, and since the priority order used is the same it is also at the lowest priority under FP-NP scheduling. We now derive the speedup factors necessary for schedulability under FP-P in *scenario (ii)* described above, as illustrated in Fig. 4c. In this case, we are interested in a speedup factor that is sufficient such that all of the execution released in an interval of length $D_A = 1 + A$ can be completed in that

interval, thus ensuring schedulability for task τ_A under FP-P. We now make use of the constraint that $A + X \leq 1$. This implies that $A \leq 1$ and hence that the interval from the start of the busy period to W_A^{NP} is no shorter than the remainder of the interval from W_A^{NP} to D_A . Since X is the amount of interference released in the interval $[0, W_A^{NP})$ starting with synchronous release of all the tasks, then the amount of interference released in the interval $[0, D_A)$ cannot exceed $2X$. Therefore a speedup factor of $S_2 = \frac{1+A+X}{1+A}$ is sufficient to ensure schedulability of task τ_A under FP-P (as illustrated by Fig. 4c). \square

Theorem 8 *Subject to the constraint that $A + X \leq 1$, an upper bound on the speedup factor required such that FP-P scheduling, with the same priority ordering as used by FP-NP, can schedule any constrained-deadline sporadic task set deemed schedulable according to the necessary test for FP-NP is given by $S = \sqrt{2}$.*

Proof As the task set is schedulable under FP-P if it is schedulable in either scenario (i) or (ii), then the speedup factor required is given by the minimum of S_1 given by Lemma 9 and S_2 given by Lemma 10. Since S_1 is monotonically increasing in A and S_2 is monotonically decreasing in A (recall that by definition $0 \leq X \leq 1$), then the minimum is obtained when they are equal:

$$\frac{1+A}{1} = \frac{1+A+X}{1+A} \quad (29)$$

Expanding and re-arranging, we have: $X = A^2 + A$ subject to the constraint that $A + X \leq 1$. The speedup factor is therefore maximised by setting $A + X = 1$. This results in the quadratic equation $A^2 + 2A - 1 = 0$. The positive solution of which is $A = \sqrt{2} - 1$, substituting in (27) we have $S = \sqrt{2}$. \square

We now consider the case where $A + X > 1$. Theorems 9 and 10 (supported by Lemmas 11 and 12) deal with sub-cases where (i) there is no task that is part of X re-released during the interval $[1, A + X)$, and (ii) there is such a task.

Theorem 9 *Subject to the constraint that $A + X > 1$, and the restriction that there is no task that is part of X re-released during the interval $[1, A + X)$, an upper bound on the speedup factor such that FP-P scheduling, with the same priority ordering as used by FP-NP, can schedule any constrained-deadline sporadic task set deemed schedulable according to the necessary test for FP-NP is $S = \sqrt{2}$.*

Proof By the assumption in the Theorem, there is no interference released during the interval $[1, A + X)$, hence the total execution released in $[0, A + X)$ is $A + X + B = 1 + A$. Therefore speeding the processor up by the following factor is enough to ensure the schedulability of task τ_A under FP-P (as illustrated by Fig. 4b):

$$S_3 = \frac{1+A}{A+X} \quad (30)$$

Alternatively, by definition the maximum amount of interference released in the interval $[0, 1)$ from tasks that are re-released at some point in $[0, A + 1)$ is X . Since there

are no re-releases in $[1, A + X)$ and the interval $[A + X, A + 1)$ is no greater than 1 in length, then it follows that at most interference of $2X$ is released in $[0, A + 1)$. Thus if we speed the processor up such that execution of $A + X + 1$ can be accomplished in an interval of length $A + 1$, then task τ_A is also guaranteed to be schedulable under FP-P (as illustrated by Fig. 4c). Hence we have:

$$S_4 = \frac{1 + A + X}{1 + A} \quad (31)$$

Note that S_4 is the same as S_2 given in (28).

Again, as the task set is schedulable under FP-P if the processor is speeded up by a factor of either S_3 or S_4 , then the speedup factor required is given by the minimum of them. We observe that for any fixed value of X , then both S_3 and S_4 are maximised by selecting the minimum value of A . Since we have the constraint that $A + X > 1$, then we may obtain upper bounds for both scaling factors by substituting in $A = 1 - X$. Since S_3 is a monotonically decreasing function of X and S_4 is a monotonically increasing function of X , then the maximum value for the minimum of S_3 and S_4 is obtained when $S_3 = S_4$. Thus we have:

$$2 - X = \frac{2}{2 - X} \quad (32)$$

and hence either $X = 2 - \sqrt{2}$ or $X = 2 + \sqrt{2}$. Since the speedup factors S_3 and S_4 cannot be negative, we may disregard the larger value for X , thus the solution is given by $X = 2 - \sqrt{2}$, $A = \sqrt{2} - 1$, and $S = \sqrt{2}$. \square

Next, in Lemmas 11 and 12 we derive properties which are used in the proof of Theorem 10.

Lemma 11 *When schedulability is determined using the necessary test for FP-NP combined with the OPA algorithm, the two tasks assigned the lowest priorities cannot have parameters such that both $D_L > D_A$ and $C_L > C_A$ where task τ_A is at the lowest priority (n) and task τ_L is at the priority level immediately above it ($n - 1$)*

Proof We assume (for contradiction) that $D_L > D_A$ and $C_L > C_A$. Since τ_A is deemed schedulable at the lowest priority, it follows that:

$$W_A^{NP} = \sum_{\forall \tau_j \in HP(L)} \left\lceil \frac{W_A^{NP} + \Delta}{T_j} \right\rceil C_j + \left\lceil \frac{W_A^{NP} + \Delta}{T_L} \right\rceil C_L \quad (33)$$

where $HP(L)$ is the set of higher priority tasks excluding both τ_L and τ_A , and W_A^{NP} is the solution obtained on convergence.

$$R_A^{NP} = W_A^{NP} + C_A \leq D_A \quad (34)$$

Now consider what happens if the priorities of the two tasks are swapped and task τ_L is now at the lowest priority. Its lower bound response time is determined as follows:

$$W_L^{NP} = \sum_{\forall \tau_j \in HP(L)} \left\lceil \frac{W_L^{NP} + \Delta}{T_j} \right\rceil C_j + \left\lceil \frac{W_L^{NP} + \Delta}{T_A} \right\rceil C_A \quad (35)$$

with the solution W_L^{NP} obtained on convergence, and thus:

$$R_L^{NP} = W_L^{NP} + C_L \quad (36)$$

As τ_A is deemed schedulable at the lowest priority, it must be the case that from (33) $W_A^{NP} \leq D_A \leq T_A$. Considering the two fixed point iterations (35) and (33), and that $C_L > C_A$ (i.e. $C_L \geq C_A + \Delta$) and $W_A^{NP} \leq T_A$, then W_L^{NP} in (35) must converge to a value that is no greater than $W_A^{NP} - C_L + C_A$. This implies that the ceiling function in the final term of (35) is 1. It follows that $W_L^{NP} + C_L \leq W_A^{NP} + C_A$ and hence $R_L^{NP} \leq R_A^{NP}$. Since $D_L > D_A$ then task τ_L is schedulable when assigned the lowest priority. As the OPA algorithm checks tasks in reverse deadline monotonic order (i.e. task τ_L before task τ_A) the priority assignment with τ_L at a higher priority than τ_A can never be obtained; task τ_L would have been assigned the lowest priority. \square

Lemma 12 *In a speedup optimal task set scaled such that it is just schedulable according to the necessary test for FP-NP (with optimal priority assignment) that has $A + X > 1$, there cannot be an interfering task τ_H (that is part of X) that is re-released during the interval $[1, A + X)$, unless there is a task τ_L with the lowest priority with the exception of the constraining task τ_A , which is also part of X and has an execution time $C_L \geq A$.*

Proof From Theorem 7, we know that task τ_L must exist and have an execution time $C_L \geq A$. Further, since $C_L \geq A$ then from Lemma 11 we know that $D_L \leq D_A$. Since task τ_L is either part of B or part of X , we prove the Lemma by showing that τ_L is necessarily unschedulable if it is part of B . The un-schedulability of τ_L in this case contradicts the assumption in the Lemma that the task set is a *schedulable* speedup optimal task set under FP-NP.

We assume (for contradiction) that τ_L is part of B and the interfering task τ_H (that is part of X) is re-released during the interval $[1, A + X)$. Considering a FP-NP schedule starting with blocking of duration A from task τ_A , by time $A + X \geq 1$, interference of at least X has been released along with execution of duration B (that by definition does not repeat in the interval $[1, A + 1)$). The lowest priority of all of these tasks in B and X is τ_L , which is part of B , hence τ_L has not started to execute by time $A + X$. At time $t < A + X$, a further release of task τ_H occurs. Thus the lower bound response time of τ_L according to the necessary test is at least $A + X + B + C_H$ this is greater than the deadline of τ_A (recall that $D_A = A + X + B$). Since $D_L \leq D_A$ task τ_L is therefore unschedulable. \square

Theorem 10 *Subject to the constraint that $A + X > 1$, if there is a task τ_H that is part of X re-released during the interval $[1, A + X)$, an upper bound on the speedup factor*

such that FP-P scheduling, with the same priority ordering as used by FP-NP, can schedule any constrained-deadline sporadic task set deemed schedulable according to the necessary test for FP-NP, is $S = \sqrt{2}$.

Proof Lemma 12 showed that in any speedup optimal task set with $A + X > 1$ and a task τ_H that is part of X re-released during the interval $[1, A + X)$, there must be a task τ_L which has the lowest priority with the exception of τ_A that has $C_L \geq A$ and is also part of X . We now show that this constraint is sufficient to restrict the speedup factor required for schedulability under FP-P to at most $S = \sqrt{2}$.

Let δ be the amount of interference from other tasks (not including τ_A or τ_L) released in an interval of length $A + \delta$ (i.e. to the start of task τ_L when initially subject to blocking of duration A). The lower bound response time of τ_L is therefore $A + \delta + C_L$. (Note since all tasks have constrained deadlines, this is also the minimum possible period for task τ_L). In the preemptive case, we will now speed up the processor so that all of the execution released in an interval of length $A + \delta + C_L$ completes in that interval, thus ensuring the schedulability of task τ_A . (Since this response time is sufficient for τ_L to meet its deadline, it must also be sufficient for τ_A as from Lemma 11, we know that $D_L \leq D_A$). Note that neither τ_A nor τ_L are re-released in this interval, thus we are interested only in interference from other tasks.

Since δ is the maximum amount of interference (not including τ_L or τ_A) released in an interval of length $A + \delta$ starting with synchronous release of all tasks, then in any interval of length t , the amount of interference is upper bounded by:

$$\left\lceil \frac{t}{A + \delta} \right\rceil \delta \quad (37)$$

Thus the total amount of execution released in an interval of length $A + \delta + C_L$ is upper bounded by:

$$A + C_L + \delta + \left(\left\lceil \frac{A + C_L + \delta}{A + \delta} \right\rceil - 1 \right) \delta \quad (38)$$

Hence the speedup factor required to guarantee schedulability of task τ_A under FP-P is upper bounded by:

$$S_5 = \frac{A + C_L + \delta + \left(\left\lceil \frac{A + C_L + \delta}{A + \delta} \right\rceil - 1 \right) \delta}{A + C_L + \delta} \quad (39)$$

We will now employ a number of constraints to show that the speedup factor does not exceed $S = \sqrt{2}$.

We note that S_1 given by Lemma 9 is valid for all values of $A + X$, hence the task set is schedulable under FP-P with a speedup factor of either S_1 or S_5 , thus we are interested in finding the maximum value of $\min(S_1, S_5)$. In order for a speedup factor greater than $S = \sqrt{2}$ to be required, we would need a value of $A > \sqrt{2} - 1$, otherwise the speedup factor required is restricted by S_1 to be $\leq \sqrt{2}$.

Since $C_L \geq A$ and C_L is part of X , then it must be the case that δ which is the higher priority interference released in an interval of length $A + \delta$ is no greater than the higher priority interference released within the interval of length $X + B = 1$ in the worst-case schedule for A (see Fig. 4a), hence $C_L + \delta \leq 1$.

Examining (39), we observe that for any fixed values of C_L and δ , S_5 is maximised by employing the smallest possible value of A . Given that the speedup factor required ($= \min(S_1, S_5)$) is restricted by $S_1 \leq \sqrt{2}$ if $A \leq \sqrt{2} - 1$ then the speedup factor could only be greater than $\sqrt{2}$ if the value of S_5 is greater than $\sqrt{2}$ when $A = \sqrt{2} - 1$. We therefore substitute that value for A in (39) and seek the largest value of S_5 by selecting appropriate values of C_L and δ . Since $C_L + \delta \leq 1$ we observe that the largest possible value of the ceiling function in (39) is given by $\lceil (A + 1)/A \rceil$. Further, since $A = \sqrt{2} - 1$, the maximum value for the ceiling function is 4, and as $C_L > A > 0$, the minimum value of the ceiling function is 2. We now consider the three possible values (4, 3, and 2) for the ceiling function and maximise the value of S_5 via choosing appropriate values of C_L and δ in each case.

Ceiling = 4: For the ceiling function to return 4, it must be the case that $C_L > 2A + 2\delta > 2(\sqrt{2} - 1) + 2\delta$. Further, since $C_L + \delta \leq 1$ we have: $1 - \delta > 2(\sqrt{2} - 1) + 2\delta$ and therefore $\delta < (3 - 2\sqrt{2})/3$. For any fixed value of the ceiling function, then S_5 is maximised by choosing the largest possible value of δ . Hence when the ceiling function returns 4, S_5 is upper bounded by selecting $\delta = (3 - 2\sqrt{2})/3$. We therefore obtain:

$$S_5 \leq 1 + \frac{3 - 2\sqrt{2}}{\sqrt{2}} \approx 1.121 \quad (40)$$

Ceiling = 3: For the ceiling function to return 3, it must be the case that $2A + 2\delta \geq C_L > A + \delta > \sqrt{2} - 1 + \delta$. Further, since $C_L + \delta \leq 1$ we have: $1 - \delta > \sqrt{2} - 1 + \delta$ and therefore $\delta < (2 - \sqrt{2})/2$. Since for any fixed value for the ceiling function, S_5 is maximised by choosing the largest possible value of δ , S_5 is upper bounded in this case by selecting $\delta = (2 - \sqrt{2})/2$. We therefore obtain:

$$S_5 \leq 1 + \frac{2 - \sqrt{2}}{\sqrt{2}} = \sqrt{2} \quad (41)$$

Ceiling = 2: For the ceiling function to return 2, it must be the case that $A + \delta \geq C_L > A$. Further, since $C_L + \delta \leq 1$ we have: $1 - \delta > \sqrt{2} - 1$ and hence $\delta < 2 - \sqrt{2}$. Again, for any fixed value for the ceiling function, S_5 is maximised by choosing the largest possible value of δ . Hence S_5 is upper bounded in this case by selecting $\delta = 2 - \sqrt{2}$. We therefore obtain:

$$S_5 \leq 1 + \frac{2 - \sqrt{2}}{\sqrt{2}} = \sqrt{2} \quad (42)$$

We have shown, for all three possible values of the ceiling function, that the speedup factor S_5 with $A = \sqrt{2} - 1$ is upper bounded by at most $\sqrt{2}$. Since this bounds the maximum value that S_5 can take for any value of $A > \sqrt{2} - 1$ and the value of $S_1 \leq \sqrt{2}$ when $A \leq \sqrt{2} - 1$, it follows that the maximum value of $\min(S_1, S_5)$ is $\sqrt{2}$. \square

Theorem 11 *An upper bound on the speed-up factor that guarantees FP-P feasibility of all constrained-deadline FP-NP feasible task sets is given by:*

$$S = \sqrt{2}$$

Proof Follows from Theorems 8, 9, and 10, and the fact that the bound so obtained using a *necessary* test for FP-NP scheduling assuming optimal priority assignment (OPA) compared to an exact test for FP-P assuming exactly the *same priority ordering* that FP-NP uses, is also an upper bound for the general case described in the theorem. This is the case since use of a (weaker) exact rather than a necessary test for FP-NP cannot increase the speedup factor required. Similarly using a (stronger) optimal priority assignment for FP-P scheduling cannot increase the speedup factor required w.r.t. using the same priority ordering as chosen for FP-NP scheduling. \square

5.4 Exact speedup factor for FP-P v. FP-NP for constrained-deadline task sets

We now put together the results of Sects. 5.1 and 5.3 to give the exact speedup factor for FP-P v. FP-NP.

Theorem 12 *The exact speed-up factor that guarantees FP-P feasibility of all constrained-deadline FP-NP feasible task sets is given by:*

$$S = \sqrt{2}$$

Proof Follows from the upper bound given in Theorem 11 and the lower bound of the same value given in Theorem 5. \square

6 Summary and conclusions

The main contribution of this paper is the derivation of resource augmentation bounds for preemptive and non-preemptive scheduling algorithms on a uniprocessor. Our metric for measuring the relative effectiveness of these scheduling policies is a resource augmentation factor known as the processor *speedup factor*. The processor speedup factor is defined as the minimum amount by which the processor needs to be speeded up to guarantee schedulability under one algorithm (e.g. FP-NP) for any task set that is schedulable under another algorithm (e.g. EDF-P).

Specifically, we derived the following exact *sub-optimality* and *speedup factor* results:

S1 Exact sub-optimality of FP-NP for tasks with arbitrary deadlines:

$$S = 2 + \frac{C_{max}}{D_{min}}$$

For task sets with implicit or constrained-deadlines:

$$\text{Lower Bound } S = 1 + \frac{C_{\max}}{D_{\min}} \quad \text{Upper Bound } S = 2 + \frac{C_{\max}}{D_{\min}}$$

S2 Exact speedup factor required for FP-NP feasibility of any arbitrary deadline task set that is FP-P feasible:

$$S = 2 + \frac{C_{\max}}{D_{\min}}$$

For task sets with implicit or constrained deadlines:

$$S = 1 + \frac{C_{\max}}{D_{\min}}$$

S3 Exact sub-optimality of EDF-NP for implicit, constrained or arbitrary deadline task sets:

$$S = 1 + \frac{C_{\max}}{D_{\min}}$$

S4 Exact speedup factor required for FP-P feasibility of any constrained deadline task set that is FP-NP feasible.

$$S = \sqrt{2}$$

Tables 1, 2, and 3 summarise the speedup factor results for uniprocessor fixed priority and EDF scheduling algorithms. References next to the speedup factor values refer to prior results, while those without a reference were derived in this paper. Where an exact speedup factor has been derived, then only one value is given (i.e. the upper and lower bounds are equal). Further, where the same value applies to different classes of task set (e.g. with implicit, constrained, or arbitrary deadlines), then again only a single result is shown, with the cell in the table enlarged to cover the different classes.

The major remaining open problems involve tightening the upper and lower bounds where exact values are not yet known. These include determining the exact sub-optimality of FP-NP for the case of implicit and constrained deadline task sets, and determining the exact speedup factor required for FP-P feasibility of any task set that is FP-NP feasible for the implicit, and arbitrary deadline cases.

While the speedup factor results derived in this paper are mainly of interest in providing a *theoretical* comparison focusing on the worst-case behaviour of the different scheduling algorithms, these results also help by providing practical guidance to system designers. For example, the majority of real-time operating systems support fixed priority scheduling, with those mandated for automotive systems by the OSEK (OSEK/VDX 2007) and AUTOSAR (AUTOSAR 2007) standards supporting both FP-P and FP-NP scheduling. Here, it is interesting to consider the comparison between FP-P and FP-NP; even though the two scheduling policies are incomparable.

Table 1 Speedup factors for FP-P v. EDF-P and FP-NP v. EDF-NP

Task set class	FP-P v EDF-P		FP-NP v. EDF-NP	
	Lower Bound	Upper Bound	Lower Bound	Upper Bound
Implicit-deadline	$1/\ln(2) \approx 1.44269$ (Liu and Layland 1973) (Davis et al. 2009a)		$1/\Omega \approx 1.76322$ (Davis et al. 2010) (von der Bruggen et al. 2015)	
Constrained-deadline				
Arbitrary-deadline	2 (Davis et al. 2015a)		2 (Davis et al. 2015a)	

Table 2 Speedup factors for FP-NP v. EDF-P and EDF-NP v. EDF-P

Task set class	FP-NP v EDF-P		EDF-NP v. EDF-P	
	Lower Bound	Upper Bound	Lower Bound	Upper Bound
Implicit-deadline	$1 + \frac{C_{max}}{D_{min}}$	$2 + \frac{C_{max}}{D_{min}}$	$1 + \frac{C_{max}}{D_{min}}$	
Constrained-deadline				
Arbitrary-deadline				

Table 3 Speedup factors for FP-NP v. FP-P and FP-P v. FP-NP

Task set class	FP-NP v FP-P		FP-P v. FP-NP	
	Lower Bound	Upper Bound	Lower Bound	Upper Bound
Implicit-deadline	$1 + \frac{C_{max}}{D_{min}}$		≈ 1.34 (Davis et al. 2015b)	$\sqrt{2}$
Constrained-deadline			$\sqrt{2}$	
Arbitrary-deadline	$2 + \frac{C_{max}}{D_{min}}$		$\sqrt{2}$	2 (Davis et al. 2015a)

The exact speedup factor required for FP-NP feasibility of any constrained deadline task set that is FP-P feasible is $S = 1 + \frac{C_{max}}{D_{min}}$ (see Theorem 2). Thus if we have a system where the longest execution time of any task is substantially less than the shortest deadline ($C_{max} \ll D_{min}$), we can quantify the small processing speed penalty for using non-preemptive scheduling. This can then be weighed against the additional overheads (e.g. preemption costs, cache related preemption delays, support for mutually exclusive resource accesses etc.) incurred in using preemptive scheduling. When the processing speed penalty is small, other factors such as the reduced complexity

involved in accurately modelling and testing a non-preemptive system, may also favour FP-NP scheduling. When $C_{max} \gg D_{min}$ then it is clear that the penalty for using fully non-preemptive scheduling is very high (the long task problem (Short 2010)). In such cases methods that support limited preemption, effectively breaking long tasks into a set of non-preemptive regions may be preferable. A further avenue for the extension of this work is to systems that support limited preemption (Buttazzo et al. 2013), in particular including final non-preemptive regions, since that paradigm dominates both FP-P and FP-NP scheduling (Davis and Bertogna 2012).

We note that it has recently been shown (von der Bruggen et al. 2016) that the results comparing FP-P v. EDF-P, and FP-NP v. EDF-NP given in Table 1, which assume optimal priority assignment and exact schedulability tests, continue to hold when Deadline Monotonic priority assignment and simple, sufficient, linear-time schedulability tests are employed for fixed priority scheduling. Thus in these cases, in terms of the speedup-factors required, there is no penalty in using Deadline Monotonic priority assignment⁵ and simple linear-time schedulability tests. Although sufficient in terms of schedulability, these tests are *speedup-optimal* (see Definition 7) for the comparisons FP-P v. EDF-P, and FP-NP v. EDF-NP. The results in this paper show that in the arbitrary deadline case, the speedup factors for FP-NP v. EDF-NP, and FP-NP v. FP-P also hold for Deadline Monotonic priority assignment and simple linear-time schedulability tests.

Finally, we caution that it is important to understand that speedup factors are indicative only of the worst-case performance of one algorithm relative to another. We refer the interested reader to recent work (Chen et al. 2017) on the pitfalls of using speedup factors and other resource augmentation bounds for a full discussion of the pros and cons of using this metric.

Acknowledgements This work was funded in part by the EPSRC projects MCC (EP/K011626/1) and MCCps (EP/P003664/1), and the INRIA International Chair program. EPSRC Research Data Management: No new primary data was created during this study. This paper solves an open problem presented at RTSOPS (Davis et al. 2015b) and at the Dagstuhl Seminar on Scheduling 16081.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

- Abugchem F, Short M, Xu D (2015) A note on the suboptimality of nonpreemptive real-time scheduling. *IEEE Embed Syst Lett* 7(3):69–72. <https://doi.org/10.1109/LES.2015.2426657>
- Altmeyer S, Davis RI, Maiza C (2011) Cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems. In: *Real-time systems symposium (RTSS)*, pp 261–271. <https://doi.org/10.1109/RTSS.2011.31>
- Altmeyer S, Davis RI, Maiza C (2012) Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems. *Real-Time Syst* 48(5):499–526. <https://doi.org/10.1007/s11241-012-9152-2>

⁵ Recall that Deadline Monotonic priority ordering is not optimal for FP-NP, nor for FP-P in the arbitrary deadline case.

- Altmeyer S, Douma R, Lunniss W, Davis RI (2014) Evaluation of cache partitioning for hard real-time systems. In: Proceedings Euromicro conference on real-time systems (ECRTS), pp 15–26. <https://doi.org/10.1109/ECRTS.2014.11>
- Altmeyer S, Douma R, Lunniss W, Davis RI (2016) On the effectiveness of cache partitioning in hard real-time systems. *Real-Time Syst* pp 1–46. <https://doi.org/10.1007/s11241-015-9246-8>
- Audsley N (1991) Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Dept Computer Science, University of York, Technical Report YCS, p 164
- Audsley NC (2001) On priority assignment in fixed priority scheduling. *Inf Process Lett* 79(1):39–44. [https://doi.org/10.1016/S0020-0190\(00\)00165-4](https://doi.org/10.1016/S0020-0190(00)00165-4)
- Audsley N, Burns A, Richardson M, Tindell K, Wellings AJ (1993) Applying new scheduling theory to static priority pre-emptive scheduling. *Softw Eng J* 8(5):284–292
- AUTOSAR (2007) Autosar specification of operating system, v4.10. Tech. rep. <http://www.autosar.org/>
- Baker TP (1991) Stack-based scheduling for realtime processes. *Real-Time Syst* 3(1):67–99. <https://doi.org/10.1007/BF00365393>
- Baruah SK, Mok AK, Rosier LE (1990) Preemptively scheduling hard-real-time sporadic tasks on one processor. In: Proceedings real-time systems symposium (RTSS), pp 182–190. <https://doi.org/10.1109/REAL.1990.128746>
- Bastoni A, Brandenburg BB, Anderson JH (2010) Cache-related preemption and migration delays: Empirical approximation and impact on schedulability. In: Proceedings operating systems platforms for embedded real-time applications (OSPERT)
- Bini E, Buttazzo GC (2005) Measuring the performance of schedulability tests. *Real-Time Syst* 30(1–2):129–154. <https://doi.org/10.1007/s11241-005-0507-9>
- Bril RJ, Lukkien JJ, Verhaegh WF (2009) Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption. *Real-Time Syst* 42(1–3):63–119. <https://doi.org/10.1007/s11241-009-9071-z>
- Bui BD, Caccamo M, Sha L, Martinez J (2008) Impact of cache partitioning on multi-tasking real time embedded systems. In: proceedings Real-Time Computing Systems and Applications (RTCSA), pp 101–110. <https://doi.org/10.1109/RTCSA.2008.42>
- Burns A, Gutierrez M, Aldea Rivas M, Gonzalez Harbour M (2015) A deadline-floor inheritance protocol for edf scheduled embedded real-time systems with resource sharing. *IEEE Trans Comput* 64(5):1241–1253. <https://doi.org/10.1109/TC.2014.2322619>
- Buttazzo GC, Bertogna M, Yao G (2013) Limited preemptive scheduling for real-time systems. A survey. *IEEE Trans Ind Inform* 9(1):3–15. <https://doi.org/10.1109/TII.2012.2188805>
- Chen JJ, von der Bruggen G, Huang WH, Davis RI (2017) On the pitfalls of resource augmentation factors and utilization bounds in real-time scheduling. In: Proceedings Euromicro conference on real-time systems (ECRTS)
- Davis RI (2017) On the evaluation of schedulability tests for real-time scheduling algorithms. In: Proceedings international workshop on analysis tools and methodologies for embedded and real-time systems (WATERS)
- Davis RI, Bertogna M (2012) Optimal fixed priority scheduling with deferred pre-emption. In: Proceedings real-time systems symposium (RTSS), pp 39–50. <https://doi.org/10.1109/RTSS.2012.57>
- Davis RI, Burns A, Bril RJ, Lukkien JJ (2007) Controller area network (can) schedulability analysis: refuted, revisited and revised. *Real-Time Syst* 35(3):239–272. <https://doi.org/10.1007/s11241-007-9012-7>
- Davis RI, Rothvoss T, Baruah SK, Burns A (2009a) Exact quantification of the sub-optimality of uniprocessor fixed priority pre-emptive scheduling. *Real-Time Syst* 43(3):211–258. <https://doi.org/10.1007/s11241-009-9079-4>
- Davis RI, Rothvoss T, Baruah SK, Burns A (2009b) Quantifying the sub-optimality of uniprocessor fixed priority pre-emptive scheduling for sporadic tasksets with arbitrary deadlines. In: Proceedings real-time and network systems (RTNS), pp 23–31
- Davis RI, George L, Courbin P (2010) Quantifying the sub-optimality of uniprocessor fixed priority non-pre-emptive scheduling. In: Proceedings real-time and network systems (RTNS), pp 1–10
- Davis RI, Burns A, Baruah S, Rothvoss T, George L, Gettings O (2015a) Exact comparison of fixed priority and edf scheduling based on speedup factors for both pre-emptive and non-pre-emptive paradigms. *Real-Time Syst* 51(5):566–601. <https://doi.org/10.1007/s11241-015-9233-0>
- Davis RI, Gettings O, Thekkilakattil A, Dobrin R, Punnekkat S (2015b) What is the exact speedup factor for fixed priority pre-emptive versus fixed priority non-pre-emptive scheduling? In: Proceedings real-time scheduling open problems seminar (RTSOPS), pp 23–24

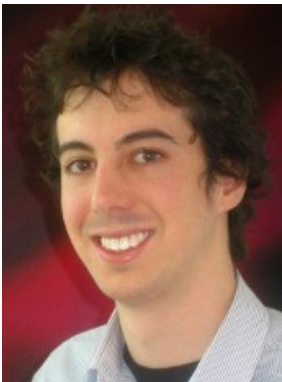
- Davis RI, Thekkilakattil A, Gettings O, Dobrin R, Punnekkat S (2015c) Quantifying the exact sub-optimality of non-preemptive scheduling. In: Proceedings real-time systems symposium (RTSS), pp 96–106. <https://doi.org/10.1109/RTSS.2015.17>
- Davis RI, Cucu-Grosjean L, Bertogna M, Burns A (2016) A review of priority assignment in real-time systems. *J Syst Archit* 65:64–82. <https://doi.org/10.1016/j.sysarc.2016.04.002>, <http://www.sciencedirect.com/science/article/pii/S1383762116300200>
- Dertouzos ML (1974) Control robotics: the procedural control of physical processes. In: Proceedings IFIP congress, pp 807–813
- George L, Rivierre N, Spuri M (1996) Preemptive and non-preemptive real-time uniprocessor scheduling. Research report, INRIA
- Jeffay K, Stanat DF, Martel CU (1991) On non-preemptive scheduling of period and sporadic tasks. In: Proceedings real-time systems symposium (RTSS), pp 129–139. <https://doi.org/10.1109/REAL.1991.160366>
- Joseph M, Pandya P (1986) Finding response times in a real-time system. *Comput J* 29(5):390–395. <https://doi.org/10.1093/comjnl/29.5.390>, <http://comjnl.oxfordjournals.org/content/29/5/390.abstract>, <http://comjnl.oxfordjournals.org/content/29/5/390.full.pdf+html>
- Kalyanasundaram B, Pruhs K (2000) Speed is as powerful as clairvoyance. *J ACM* 47(4):617–643. <https://doi.org/10.1145/347476.347479>
- Lehoczy JP (1990) Fixed priority scheduling of periodic task sets with arbitrary deadlines. In: Proceedings real-time systems symposium (RTSS), pp 201–209. <https://doi.org/10.1109/REAL.1990.128748>
- Lehoczy J, Sha L, Ding Y (1989) The rate monotonic scheduling algorithm: exact characterization and average case behavior. In: Proceedings real time systems symposium (RTSS), pp 166–171. <https://doi.org/10.1109/REAL.1989.63567>
- Leung JYT, Whitehead J (1982) On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Perform Eval* 2(4):237–250. [https://doi.org/10.1016/0166-5316\(82\)90024-4](https://doi.org/10.1016/0166-5316(82)90024-4), <http://www.sciencedirect.com/science/article/pii/0166531682900244>
- Liu CL, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. *J ACM* 20(1):46–61. <https://doi.org/10.1145/321738.321743>
- Lunniss W, Davis RI, Maiza C, Altmeyer S (2013) Integrating cache related pre-emption delay analysis into edf scheduling. In: Proceedings real-time and embedded technology and applications symposium (RTAS), pp 75–84. <https://doi.org/10.1109/RTAS.2013.6531081>
- OSEK/VDX (2007) OSEK/VDX operating system specification, version 2.2.3. Tech. rep. <http://www.irisa.fr/alf/downloads/puaut/TPNXT/images/os223.pdf>
- Ripoll I, Crespo A, Mok AK (1996) Improvement in feasibility testing for real-time tasks. *Real-Time Syst* 11(1):19–39. <https://doi.org/10.1007/BF00365519>
- Short M (2010) The case for non-preemptive, deadline-driven scheduling in real-time embedded systems. In: Proceedings of the world congress on engineering (WCE), pp 399–404
- Spuri M (1996) Analysis of deadline scheduled real-time systems. INRIA Research Report RR-2772. <https://hal.inria.fr/inria-00073920/document>
- Thekkilakattil A, Dobrin R, Punnekkat S (2013) Quantifying the sub-optimality of non-preemptive real-time scheduling. In: Proceedings Euromicro conference on real-time systems (ECRTS), pp 113–122. <https://doi.org/10.1109/ECRTS.2013.22>
- Thekkilakattil A, Baruah S, Dobrin R, Punnekkat S (2014) The global limited preemptive earliest deadline first feasibility of sporadic real-time tasks. In: Proceedings Euromicro conference on real-time systems (ECRTS), pp 301–310. <https://doi.org/10.1109/ECRTS.2014.21>
- Thekkilakattil A, Dobrin R, Punnekkat S (2015) The limited-preemptive feasibility of real-time tasks on uniprocessors. *Real-Time Syst* 51(3):247–273. <https://doi.org/10.1007/s11241-015-9222-3>
- Tindell K, Burns A, Wellings AJ (1994) An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Syst* 6(2):133–151. <https://doi.org/10.1007/BF01088593>
- von der Bruggen G, Chen JJ, Huang WH (2015) Schedulability and optimization analysis for non-preemptive static priority scheduling based on task utilisation and blocking factors. In: Proceedings Euromicro conference on real-time systems (ECRTS), pp 90–101
- von der Bruggen G, Chen JJ, Davis RI, Huang WH (2016) Exact speedup factors for linear-time schedulability tests for fixed-priority preemptive and non-preemptive scheduling. *Inf Process Lett*. <https://doi.org/10.1016/j.ipl.2016.08.001>, <http://www.sciencedirect.com/science/article/pii/S0020019016301090>



Robert I. Davis is a Senior Research Fellow in the Real-Time Systems Research Group at the University of York, UK, and an INRIA International Chair at INRIA, Paris, France. Robert received his DPhil in Computer Science from the University of York in 1995. Since then he has founded three start-up companies, all of which have succeeded in transferring real-time systems research into commercial products. Robert's research interests include the following aspects of real-time systems: scheduling algorithms and analysis for single processor, multiprocessor and networked systems; analysis of cache related preemption delays, mixed criticality systems, and probabilistic hard real-time systems.



Abhilash Thekkilakattil currently works as a test automation developer for continuous delivery at AtlasCopco Industrial Tools and Solutions, Sweden. He completed his Ph.D. in real-time scheduling in 2016. Previously, he was awarded a Master of Science degree in software engineering from Malardalen University, and a Bachelor of Technology degree in computer science and engineering from Amrita Vishwa Vidyapeetham in India. His research and development interests include various aspects of Cyber Physical Systems (CPS) and Internet of Things (IoT), with a focus on Embedded Systems and Software Engineering.



Oliver Gettings has an M.Sc. in Digital Systems Engineering (2016), an M.Sc. (by research) in Computer Science (2015), and a BEng in Computer Science (2012) from the University of York, UK. Oliver currently resides in Melbourne, Australia. His research interests include low-power reconfigurable hardware and bare metal real-time systems.



Radu Dobrin is an Associate Professor at the School of Innovation, Design and Engineering (IDT) at Mälardalen University and the head of the Computer Science and Software Engineering Division. He has a background in scheduling of dependable real-time systems and is currently involved in both research and education-oriented projects. Since 2013, Radu is also the International Coordinator at IDT. Currently, his research is focused in the area of dependable embedded real-time systems built using component-based development.



Sasikumar Punnekkat is a Professor in dependable software engineering at MDH and the leader of the Dependable Software Engineering research group. He has more than 15 years industrial experience as a scientist at the Indian Space research Organization, and was the Head of the Software test and reliability engineering. He was recipient of the prestigious Commonwealth Scholarship and was awarded Doctor of Philosophy in Computer Science by the University of York, UK in 1997 for his research on schedulability analysis of fault-tolerant systems. He was the Director of BITS-Pilani KK Birla Goa Campus during Mar 2015-Aug 2016. His research interests include multiple aspects of Real-time Systems, Dependability, and Software Engineering.



Jian-Jia Chen is a Professor at the Department of Informatics in TU Dortmund University, Germany. He was Junior Professor at the Department of Informatics in Karlsruhe Institute of Technology (KIT), Germany from May 2010 to March 2014. He received his Ph.D. degree from the Department of Computer Science and Information Engineering, National Taiwan University, Taiwan in 2006. He received his B.S. degree from the Department of Chemistry at National Taiwan University 2001. Between Jan. 2008 and April 2010, he was a postdoc researcher at ETH Zurich, Switzerland. His research interests include real-time systems, embedded systems, energy-efficient scheduling, power-aware designs, temperature-aware scheduling, and distributed computing.