Quantifying the Exact Sub-Optimality of Non-Preemptive Scheduling

Robert I. Davis[1], Abhilash Thekkilakattil[2], Oliver Gettings[1], Radu Dobrin[2], and Sasikumar Punnekkat[2]

[1]Real-Time Systems Research Group, University of York, UK and AOSTE team, Inria, Paris-Rocquencourt, France
[2]Mälardalen Real-Time Research Center, Mälardalen University, Sweden

***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. schedulablability 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), which since it matches a recently published upper bound gives the exact sub-optimality for EDF-NP.**

It is known that neither preemptive, nor non-preemptive fixed priority scheduling dominates the other, *i.e.,* 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 upper and lower bounds on the speed-up factor required to guarantee FP-P feasibility of any FP-NP feasible task set. Empirical evidence suggests that the lower bound may be tight, and hence equate to the exact speed-up factor in this case.

***Keywords*-real-time; uniprocessor; resource augmentation; speedup factor; sub-optimality; non-preemptive scheduling; preemptive scheduling; EDF; fixed priority.**

I. 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 defacto 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. *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 utilisation levels. More advanced approaches use a *weighted schedulability metric* [8] 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* [26] and *optimality degree* [9] 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* [25]. 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 we mean that the task set is schedulable under that algorithm with 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, pre-emption 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 [2] and EDF-P [30] scheduling. CRPD can have a substantial impact, increasing task execution times by as much as 33% [11]. 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 [3]. 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 [32]. It is however well known that non-preemptive scheduling can be infeasible at low processor utilization levels due to the *long task problem* [32], 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 [21]. 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. For FP-P scheduling, Deadline Monotonic Priority Ordering (DMPO) is optimal for constrained-deadline task sets [28]. In other words, any constrained-deadline task set that is schedulable 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 [27] (i.e. may be larger than their periods). In that case, Audsley's algorithm [5] 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 [22]. 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 [23], hence EDF-NP dominates FP-NP. With FP-NP scheduling, DMPO is not optimal for constrained-deadline task sets; however, Audsley's algorithm [5] can again be applied [22].

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-versa [39]. 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 [14]; consideration of that more complex model is however beyond the scope of this paper).

A. Speedup Factors

In 2009, Davis *et al.* [20] 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 Liu and Layland's famous results [29]. In 2009, Davis *et al.* [19] also derived upper and lower bounds of $S = 1/\Omega$ and $S = 2$ on the sub-optimality of FP-P scheduling for arbitrary-deadline task sets. In 2015, Davis *et al.* [15] 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) [6] or the Dead Floor Protocol (DFP) [12], thus providing exact speedup factors comparing FP-P + SRP to EDF + SRP or EDF + DFP.

In 2010, Davis *et al.* [17] derived upper and lower bounds on the speedup factor required to guarantee FP-NP feasibility of all EDF-NP feasible task sets. These bounds are $S = 1/\Omega$ and $S = 2$ respectively for all three classes of task set (implicit, constrained and arbitrary deadline). In 2015, von der Brüggen *et al.* [38] 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. Later in 2015, Davis *et al.* [15] also completed the exact characterization of the speedup factors required to guarantee FP-NP feasibility of 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).

In 2013, Thekkilakattil *et al.* [35][36] quantified the sub-optimality of EDF-NP (with respect to EDF-P), bridging between the preemptive and non-preemptive paradigms. (This result was subsequently extended to the case of global deadline based scheduling [34]). In 2015, Abugchem *et al.* [1] 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: Upper and lower bounds on the speedup factor required to guarantee FP-P feasibility for any 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 [32], 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 [20], [19], [17], [35], [15] 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 (such as context switch costs and CRPD) incurred by preemptive scheduling, when considering which algorithm to use.

B. Organization

The rest of the paper is organized as follows: the system model is presented in Section II. Section III 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 Sections IV and V, with the results of an empirical investigation reported in Section VI. Section VII concludes with a summary and a discussion of open problems.

II. System Model

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

A. 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 $\{\tau_1, \tau_2, ...\tau_n\}$. 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 $C_{max} = \max_{\forall \tau_i \in \Gamma} C_i$. Similarly, the shortest deadline is denoted by $D_{min} = \min_{\forall \tau_i \in \Gamma} D_i$. 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. 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 $U = \sum_{i=1}^{n} U_i$.

B. 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.

C. 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.

D. Definitions

We now provide formal definitions for the terms *speedup factor*, *speedup optimal task set* and *sub-optimality*. 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 II.1. *The **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 $\mathcal{B}$ on a processor of unit speed, is guaranteed to be feasible under $\mathcal{A}$ on a processor that is S times faster.*

Definition II.2. *A task set is said to be **speed-up optimal** for the comparison between scheduling algorithms $\mathcal{A}$ and $\mathcal{B}$ if it is feasible on a processor of unit speed under $\mathcal{B}$ and requires the processor speed to be increased by the speedup factor S in order to be feasible under $\mathcal{A}$.*

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

Definition II.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 sub-optimality of a particular scheduling algorithm, the closer it 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.

III. Schedulability Analysis

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

A. Fixed Priority Preemptive Scheduling

The schedulability of a set of arbitrary-deadline sporadic tasks under FP-P can be determined using response time analysis [37] [27]. 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:

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

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:

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

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:

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

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 [4], [24], based on the following recurrence relation:

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

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.

B. 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 [10]. 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:

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

where B_i is the *blocking factor*:

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

and Δ is the time granularity[1].

[1]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.

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:

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

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:

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

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 [17]:

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

The second, which is only applicable to constrained-deadline task sets is based on a recurrence relation [16]:

$$W_i^{NP} = C_{max} + \sum_{\forall \tau_j \in hp(i)} \left\lceil \frac{W_i^{NP} + \Delta}{T_j} \right\rceil C_j$$

$$R_i^{NP} = W_i^{NP} + C_i \quad (10)$$

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

C. 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 [7]. A task set is EDF-P feasible *if and only if*:

$$\sum_{\forall \tau_i \in \Gamma} DBF_i(t) \leq t \quad (11)$$

$$\forall t = kT_j + D_j,\ \forall k \in \mathbb{N}, j \in [1, n]$$

$$t \leq A_n^P$$

where

$$DBF_i(t) = max\left(0, 1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor\right) C_i \quad (12)$$

and A_n^P is the length of the longest busy period, given by (1) [31] [33].

IV. 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 in 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.* [1].

Lemma IV.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 (12) and assuming DMPO we observe that:

$$\sum_{\forall \tau_j \in \Gamma} DBF_j(2D_i) \geq \sum_{\forall \tau_j : D_j \leq D_i} \left\lceil \frac{D_i}{T_j} \right\rceil C_j \geq \quad (13)$$

$$\sum_{\forall \tau_j \in hep(i)} \left\lceil \frac{D_i}{T_j} \right\rceil C_j$$

From (9), (13), 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:

$$\frac{C_{max} + \sum_{\forall \tau_k \in \Gamma} DBF_k(2D_i)}{S} \leq D_i \quad (14)$$

Since the task set is schedulable under EDF-P on a processor of unit speed, then it follows from (11) that $\sum_{\forall \tau_k \in \Gamma} DBF_k(2D_i) \leq 2D_i$. Substituting into (14) and re-arranging, 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. ■

Lemma IV.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 IV.1 and the fact that EDF-P can schedule all task sets that are feasible under FP-P scheduling. [21]. ■

Lemma IV.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:
τ_1: $C_1 = k - 1$, $D_1 = k$, $T_1 = k$
τ_2: $C_2 = k^2 + 1$, $D_2 = \infty$, $T_2 = \infty$
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 tasks[2] (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 $k \to \infty$ we have a lower bound of:

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

■

Theorem IV.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 IV.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 IV.3 matches the upper bound given by Abugchem *et al.* [1] the value is exact. ■

Lemma IV.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:*

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

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

(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 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.

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

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

$$E_i^P(t) = C_i + \sum_{\forall \tau_j \in hp(i)} \left\lceil \frac{t}{T_j} \right\rceil C_j \tag{15}$$

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 execution from tasks of higher priority than τ_i released in an interval of length t including any releases at the end of the interval:

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

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

$$E_i^{NP}(t - x) + C_i \leq E_i^P(t) \tag{17}$$

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 $E_i^{NP}(W_i^P - \frac{C_i}{S})$. Schedulability under FP-NP is then ensured provided that:

$$\frac{C_{max} + E_i^{NP}(W_i^P - \frac{C_i}{S}) + C_i}{S} \leq W_i^P \tag{18}$$

This follows, since if (18) holds then the upper bound response time for task τ_i computed via (10) will be $\leq W_i^P$.

Since even on the faster processor of speed S, the execution time of τ_i cannot be less than the time granularity ($\frac{C_i}{S} \geq$

Δ), then from (17), it follows that $E_i^{NP}(W_i^P - \frac{C_i}{S}) + C_i \leq E_i^P(W_i^P)$. As $E_i^P(W_i^P) = W_i^P$, substituting into (18) and re-arranging we have:

$$S \geq 1 + \frac{C_{max}}{W_i^P} \tag{19}$$

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:

$$S \geq 1 + \frac{C_{max}}{D_{min}} \tag{20}$$

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:

$$W_i^P = C_i + \sum_{\forall j \in hp(i)} C_j \tag{21}$$

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:

$$\frac{C_{max} + E_i^{NP}(D_{min} - \frac{C_i}{S}) + C_i}{S} \leq D_{min} \tag{22}$$

where S is the processor speed.

Following the same logic as in Case 1, we observe that $E_i^{NP}(D_{min} - \frac{C_i}{S}) + C_i \leq E_i^P(W_i^P) = W_i^P$. Since in this case (Case 2) $W_i^P < D_{min}$ substituting into (22) and re-arranging we obtain the speed S at which the task set is guaranteed to be schedulable:

$$S \geq 1 + \frac{C_{max}}{D_{min}} \tag{23}$$

■

Theorem IV.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 IV.3 and the upper bound given by Lemma IV.4 which have the same value. ■

Lemma IV.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 $i = 1, \ldots, k-1$: $C_i = 1$, $D_i = k+1$, $T_i = k$
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 shown, 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 [22]. First, task τ_{k+1} is assigned as it is trivially schedulable at the lowest priority on a processor of unit speed or higher. There are then two cases to consider[3].

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 $S \geq (k^2 + 2k - 2)/(k+1) = k + (k-2)/(k+1)$.

[3]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 $\Delta \ll 1$.

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:

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

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

Theorem IV.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 IV.5 and the upper bound given by Lemma IV.2 which have the same value. ■

From Theorems IV.2 and IV.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

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

Theorem IV.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 IV.1 shows that the speedup factor in the theorem is a valid upper bound. Lemma IV.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. ■

V. Preemptive FPS vs. Non-preemptive FPS

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.

In this section, we derive upper and lower bounds on the processor speed-up factor **S4** that guarantees FP-P feasibility of FP-NP feasible task sets.

Theorem V.1. *An upper bound on the speed-up factor that guarantees FP-P feasibility of all FP-NP feasible task sets is given by:*

$$S = 2$$

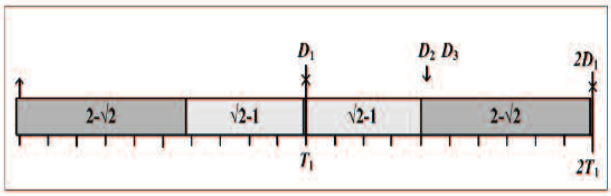

Fig. 1. Fixed Priority Non-Preemptive Schedule.

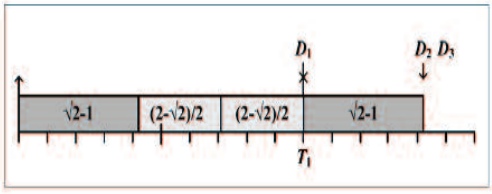

Fig. 2. Fixed Priority Preemptive Schedule.

Proof: Since EDF-P dominates FP-NP and Theorem 2 from [20] 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. ■

Theorem V.2. *A lower bound on the speed-up factor that guarantees FP-P feasibility of all FP-NP feasible task sets is given by:*

$$S = \sqrt{2}$$

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

τ_1: $C_1 = 2 - \sqrt{2}$, $D_1 = 1$, $T_1 = 1$
τ_2: $C_2 = \sqrt{2} - 1$, $D_2 = \sqrt{2}$, $T_2 = \infty$
τ_3: $C_3 = \sqrt{2} - 1$, $D_3 = \sqrt{2}$, $T_3 = \infty$

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 Figure 1.

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 Figure 2, 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 $S = \sqrt{2}$. ■

In the next section, we describe the results of an empirical

study which hints that the lower bound derived above may be tight (*i.e.,* the exact speed-up factor may be $\sqrt{2}$ rather than some larger value such as the upper bound of 2).

VI. Empirical investigation

In this section, we describe the results of an empirical investigation into the speed-up factor needed to ensure that FP-NP feasible task sets are schedulable under FP-P. This was done by using a genetic algorithm to explore the search space of task parameters. The operation of the genetic algorithm is outlined below.

First, an initial population of *N* task sets each with n tasks were created. The task utilisations were assigned according to the UUnifast [9] algorithm. Task periods were chosen in the range $[10^4, 10^7]$ according to a log uniform distribution. For task sets with constrained deadlines, deadlines were chosen according to a log uniform distribution in the range $[C, T]$ and for arbitrary deadline task sets from the range $[C, 10T]$. Computation times were determined according to $C_i = U_i T_i$. All parameters were discrete and represented using 64-bit integers. The speed-up factor for each task set was found by performing two binary searches to determine respectively, the scaling factors f^{FP-NP} and f^{FP-P} required such that that the task set was just schedulable according to FP-NP and just unschedulable according to FP-P. Audsleys algorithm was used in both cases to determine the optimal priority assignment, along with exact schedulability tests for arbitrary deadline task sets under FP-NP (8), and FP-P (3). The speed-up factor for the task set was then given by $S = \frac{f^{FP-P}}{f^{FP-NP}}$. This approach ensured that any lack of precision in the computed speed-up factor caused by imprecision in the binary searches could only result in a small underestimate (and no overestimate) of the precise speed-up factor for the specific task set. The precision of the binary searches was 0.01% (*i.e.,* the termination condition was such that the low and high values for the scaling factor were within 0.01% of each other).

Crossover type	1-point
Crossover probability	0.5
Mutation	20% of C, D, or T
Mutation probability	0.6
Parent selection	Tournament
Survival selection	Tournament on combined old and new population
Population size	20,000
Tournament size	50
Generations	400

TABLE I
Experimental parameters

The computed speed-up factor for each task set was used as the fitness function in the genetic algorithm. The genetic algorithm operated as follows. First a population of *N* random task sets each with n tasks were created, as described above. The fitness (speed-up factor) of each of these task sets was evaluated, with the maximum speed-up factor of any task set in the population recorded at this and each subsequent stage. A new generation of *N* child task sets was then produced. Parent task sets were selected from the existing population using a tournament selection process. These parents produced children via crossover and mutation operations, which occurred with specified probabilities. First copies were made of the parent task sets, which became the children. If crossover occurred, then a random position in the list of tasks was chosen and the child task sets split and recombined at that point. (The head of one being joined to the tail of the other). Each child task set was then potentially subject to mutation. If mutation occurred, then a single parameter *C*, *D*, or *T* was selected at random, and increased or decreased by a random value in the range $[0, 20\%]$. The parameters were then repaired as necessary to ensure that any constraints on task deadlines continued to be met. For example, with constrained deadlines, if the period of a task was decreased below its deadline, then the deadline was adjusted to be equal to the period. (Repairs were also made to avoid parameters going out of range).

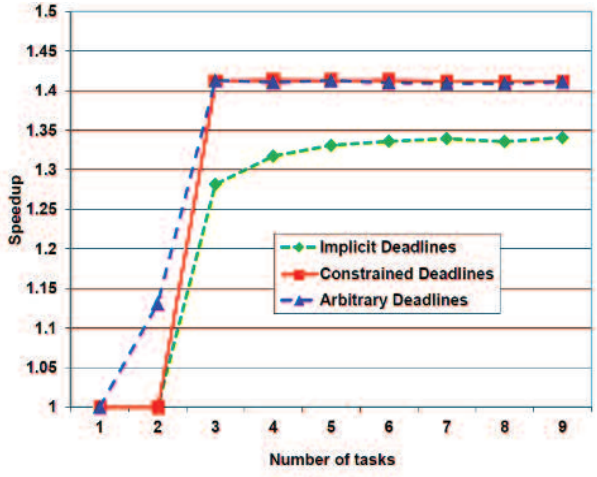

Fig. 3. Empirical results

Once *N* child nodes had been produced, then their fitness was evaluated, and a further tournament selection used to reduce the overall population (parents and children) to *N* task sets. This overall process was then repeated for further generations, with task sets with higher speed-up factors more likely to survive and produce offspring. The tournament selection process involved random selection of 50 task sets with the one with the highest speed-up factor selected for the next part of the process. Here a large tournament size increases selection pressure, but reduces diversity in the population. The parameters enumerated in Table I were found empirically to be effective for this problem.

Figure 3 shows the results obtained for task sets with implicit, constrained and arbitrary deadlines. These results were produced using the genetic algorithm described above. 400 generations of a population of $20,000$ task sets, *i.e.,* 8 million task sets were generated for each task set cardinality and deadline type. We note that with both implicit and constrained deadlines, task sets of cardinality two have a

speed-up factor of 1. This is because with FP-NP both tasks have the same worst-case response time which must be less than the smaller of their deadlines, and hence periods. This response time is the same as that of the lowest priority task under FP-P, hence all task sets of cardinality two that are schedulable under FP-NP are also schedulable under FP-P, implying a speed-up factor of 1.

For three or more tasks, then with constrained or arbitrary deadlines, the maximum speed-up factor found by the genetic algorithm is very close to $\sqrt{2} = 1.414213562$. In fact the values range from 1.4118 to 1.4139 (constrained deadlines) and from 1.4089 to 1.4128 (arbitrary deadlines) for task sets of cardinality 3 to 10. With implicit-deadline task sets, the largest speed-up factor found was somewhat lower at 1.3405.

The fact that the maximum value found empirically (1.4139) is very close to but does not exceed $\sqrt{2} = 1.414213562$ gives credence to the hypothesis that the theoretical lower bound (of $\sqrt{2}$) on the speed-up factor is the exact value. It remains an interesting open question whether or not this is the case [18].

VII. 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. Specifically, we derived the following *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:

Lower Bound $S = 1 + \frac{C_{max}}{D_{min}}$ 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: Speedup factor required for FP-P feasibility of any constrained or arbitrary deadline task set that is FP-NP feasible.

Lower Bound $S = \sqrt{2}$ Upper Bound $S = 2$

A summary of the results derived in this paper (underlined), together with the state-of-the-art for *arbitrary deadline* task sets is presented in Figure 4. (The dashed arrows on the figure represent dominance relationships, where the exact speed-up factor in the reverse direction is 1).

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, constrained 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 provide practical guidance. For example, the majority of real-time operating systems support fixed priority scheduling, with those mandated for automotive systems by the OSEK and AUTOSAR 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. 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 IV.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; as well as other considerations such as the additional complexity involved in accurately modelling and testing a preemptive system. 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 [32]), 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 [13], in particular including final non-preemptive regions, since that paradigm dominates both FP-P and FP-NP scheduling [14].

Acknowledgments

This work was funded in part by the EPSRC project MCC (EP/K011626/1) and the Inria International Chair program. EPSRC Research Data Management: No new primary data was created during this study.

References

[1] Fathi Abugchem, Michael Short, and Donglai Xu. A note on the sub-optimality of non-preemptive real-time scheduling. *Embedded Systems Letters, IEEE*, PP(99):1–1, 2015.

[2] Sebastian Altmeyer, Robert I. Davis, and Claire Maiza. Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems. *Real-Time Systems*, 48(5):499–526, 2012.

[3] Sebastian Altmeyer, Roeland Douma, Will Lunniss, and Robert I. Davis. Evaluation of cache partitioning for hard real-time systems. In *proceedings Euromicro Conference on Real-Time Systems (ECRTS)*, pages 15–26, July 2014.

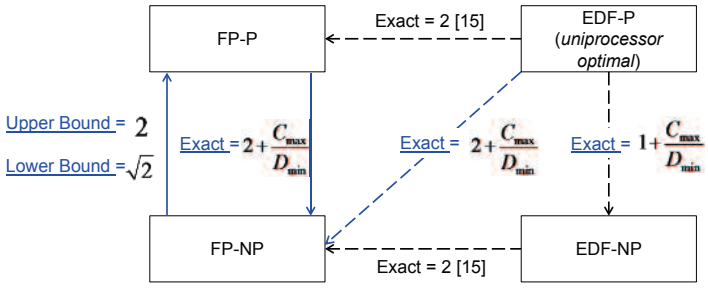

Fig. 4. Summary of speed-up factors for arbitrary deadline task sets.

[4] Neil Audsley, Alan Burns, Mike Richardson, Ken Tindell, and Andrew J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, Sep 1993.

[5] Neil. C. Audsley. On priority asignment in fixed priority scheduling. *Information Processing Letters*, 79(1):39–44, May 2001.

[6] Ted P. Baker. Stack-based scheduling for realtime processes. *Real-Time Systems*, 3(1):67–99, April 1991.

[7] Sanjoy K. Baruah, Aloysius K. Mok, and Louis E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *proceedings Real-Time Systems Symposium (RTSS)*, pages 182–190, Dec 1990.

[8] Andrea Bastoni, Bjrn B. Brandenburg, and James H. Anderson. Cache-related preemption and migration delays: Empirical approximation and impact on schedulability. In *proceedings Operating Systems Platforms for Embedded Real-Time applications (OSPERT)*, July 2010.

[9] Enrico Bini and Giorgio C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[10] Reinder J. Bril, Johan J. Lukkien, and Wim F.J. Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption. *Real-Time Systems*, 42(1-3):63–119, 2009.

[11] Bach D. Bui, Marco Caccamo, Lui Sha, and Joseph Martinez. Impact of cache partitioning on multi-tasking real time embedded systems. In *proceedings Real-Time Computing Systems and Applications (RTCSA)*, pages 101–110, Aug 2008.

[12] Alan Burns, Marina Gutierrez, Mario Aldea Rivas, and Michael Gonzalez Harbour. A deadline-floor inheritance protocol for edf scheduled embedded real-time systems with resource sharing. *IEEE Transactions on Computers*, 64(5):1241–1253, May 2015.

[13] Giorgio C. Buttazzo, Marko Bertogna, and Gang Yao. Limited preemptive scheduling for real-time systems. a survey. *IEEE Transactions on Industrial Informatics*, 9(1):3–15, Feb 2013.

[14] Robert I. Davis and Marko Bertogna. Optimal fixed priority scheduling with deferred pre-emption. In *proceedings Real-Time Systems Symposium (RTSS)*, pages 39–50, Dec 2012.

[15] Robert I. Davis, Alan Burns, Sanjoy Baruah, Thomas Rothvoss, Laurent George, and Oliver Gettings. Exact comparison of fixed priority and edf scheduling based on speedup factors for both pre-emptive and non-pre-emptive paradigms. *Real-Time Systems*, 51(5):566–601, 2015.

[16] Robert I. Davis, Alan Burns, Reinder J. Bril, and Johan J. Lukkien. Controller area network (can) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, 2007.

[17] Robert I. Davis, Laurent George, and Pierre Courbin. Quantifying the sub-optimality of uniprocessor fixed priority non-pre-emptive scheduling. In *proceedings Real-Time and Network Systems (RTNS)*, pages 1–10, 2010.

[18] Robert I. Davis, Oliver Gettings, Abhilash Thekkilakattil, Radu Dobrin, and Sasikumar Punnekkat. 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)*, pages 23–24, 2015.

[19] Robert I. Davis, Thomas Rothvoss, Sanjoy K. Baruah, and Alan Burns. 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)*, pages 23–31.

[20] Robert I. Davis, Thomas Rothvoss, Sanjoy K. Baruah, and Alan Burns. Exact quantification of the sub-optimality ofuniprocessor fixed priority pre-emptive scheduling. *Real-Time Systems*, 43(3):211–258, 2009.

[21] Michael L. Dertouzos. Control robotics: The procedural control of physical processes. In *proceedings IFIP Congress*, pages 807–813.

[22] Laurent George, Nicolas Rivierre, and Marco Spuri. Preemptive and Non-Preemptive Real-Time UniProcessor Scheduling. Research report, INRIA, 1996.

[23] Kevin Jeffay, Donald F. Stanat, and Charles U. Martel. On non-preemptive scheduling of period and sporadic tasks. In *proceedings Real-Time Systems Symposium (RTSS)*, pages 129–139, Dec 1991.

[24] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

[25] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *Journal of ACM*, 47(4):617–643, July 2000.

[26] John Lehoczky, Lui Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *proceedings Real Time Systems Symposium (RTSS)*, pages 166–171, Dec 1989.

[27] John P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *proceedings Real-Time Systems Symposium (RTSS)*, pages 201–209, Dec 1990.

[28] Joseph Y.-T. Leung and Jennifer Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237 – 250, 1982.

[29] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *The Journal of ACM*, 20(1):46–61, January 1973.

[30] Will Lunniss, Robert I. Davis, Claire Maiza, and Sebastian Altmeyer. Integrating cache related pre-emption delay analysis into edf scheduling. In *proceedings Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 75–84, April 2013.

[31] Ismael Ripoll, Alfons Crespo, and Aloysius K. Mok. Improvement in feasibility testing for real-time tasks. *Real-Time Systems*, 11(1):19–39, 1996.

[32] Michael Short. The case for non-preemptive, deadline-driven scheduling in real-time embedded systems. In *proceedings of the World Congress on Engineering (WCE)*, pages 399–404, 2010.

[33] Marco Spuri. Analysis of deadline scheduled real-time systems. *Inria Research Report RR-2772*, 1996.

[34] Abhilash Thekkilakattil, Sanjoy Baruah, Radu Dobrin, and Sasikumar Punnekkat. The global limited preemptive earliest deadline first feasibility of sporadic real-time tasks. In *proceedings Euromicro Conference on Real-Time Systems (ECRTS)*, pages 301–310, July 2014.

[35] Abhilash Thekkilakattil, Radu Dobrin, and Sasikumar Punnekkat. Quantifying the sub-optimality of non-preemptive real-time scheduling. In *proceedings Euromicro Conference on Real-Time Systems (ECRTS)*, pages 113–122, July 2013.

[36] Abhilash Thekkilakattil, Radu Dobrin, and Sasikumar Punnekkat. The limited-preemptive feasibility of real-time tasks on uniprocessors. *Real-Time Systems*, 51(3):247–273, 2015.

[37] Ken Tindell, Alan Burns, and Andrew J. Wellings. An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems*, 6(2), 1994.

[38] Georg van der Bruggen, Jian-Jia Chen, and Wen-Hung Huang. 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)*, pages 90–101, July 2015.

[39] Gang Yao, Giorgio C. Buttazzo, and Marko Bertogna. Bounding the maximum length of non-preemptive regions under fixed priority scheduling. In *proceedings Real-Time Computing Systems and Applications (RTCSA)*, pages 351–360, Aug 2009.