Quantifying the Sub-optimality of Uniprocessor Fixed Priority Non-Pre-emptive Scheduling

Robert I. Davis
Real-Time Systems Research Group,
Department of Computer Science,
University of York, York, UK.
rob.davis@cs.york.ac.uk

Laurent George
AOSTE Team, INRIA
Rocquencourt, BP 105,
Domaine de Voluceau, 78153,
Le Chesnay Cedex, France
lgeorge@ieee.org

Pierre Courbin
Ecole Centrale d'Electronique de Paris (ECE), LACSC,
37 Quai de Grenelle,
75015 Paris, France
courbin@ece.fr

Abstract

This paper examines the relative effectiveness of fixed priority non-pre-emptive scheduling (FP-NP) in a uniprocessor system, compared to an optimal work-conserving non-pre-emptive algorithm; Earliest Deadline First (EDF-NP). The quantitative metric used in this comparison is the processor speedup factor, defined as the factor by which processor speed needs to increase to ensure that any taskset that is schedulable according to EDF-NP can be scheduled using FP-NP scheduling. For sporadic tasksets with implicit, constrained, or arbitrary deadlines, the speedup factor is shown to be lower bounded by $1/\Omega \approx 1.76322$ *and upper bounded by 2.*

We also report the results of empirical investigations into the speedup factor required to ensure schedulability in the non-pre-emptive case.

1. Introduction

In this paper, we are interested in determining the largest factor by which the processing speed of a uniprocessor needs to be increased, to ensure that any taskset that was previously schedulable according to an optimal work-conserving (i.e. non-idling), non-pre-emptive scheduling algorithm is schedulable according to fixed priority non-pre-emptive (FP-NP) scheduling. We refer to this resource augmentation factor as the *processor speedup factor* [17].

1.1. Pre-emptive scheduling

In 1973, Liu and Layland [22] considered fixed priority pre-emptive (FP-P) scheduling of synchronous[1] tasksets comprising independent periodic tasks, with bounded execution times, and deadlines equal to their periods. We refer to such tasksets as *implicit-deadline* tasksets. Liu and Layland showed that *rate monotonic* priority ordering (RMPO) is the optimal fixed priority assignment policy for implicit-deadline tasksets, and that using rate monotonic priority ordering, FP-P can schedule any implicit-deadline taskset that has a total utilisation $U \le \ln(2) \approx 0.693$.

Liu and Layland [22] also showed that Earliest Deadline First (EDF-P) is an optimal dynamic priority pre-emptive scheduling algorithm for implicit-deadline tasksets, and that EDF-P can schedule any such taskset that has a total utilisation $U \le 1$.

In 1974, Dertouzos [9] showed that EDF-P is an optimal uniprocessor scheduling algorithm, in the sense that if a valid schedule exists for a taskset, then the schedule produced by EDF-P will also meet all deadlines.

Research into real-time scheduling during the 1980's and early 1990's focussed on lifting many of the restrictions of the Liu and Layland task model. Task arrivals were permitted to be sporadic, with known minimal inter-arrival times, (still referred to as periods), and task deadlines were permitted to be less than or equal to their periods (so called *constrained deadlines*) or less than, equal to, or greater than their periods (so called *arbitrary deadlines*).

In 1982, Leung and Whitehead [19] showed that *deadline monotonic*[2] priority ordering (DMPO) is the optimal fixed priority ordering for constrained-deadline tasksets. Exact schedulability tests for FP-P scheduling of constrained-deadline tasksets were introduced by Joseph and Pandya in 1986 [16], Lehoczky et al. in 1989 [21], and Audsley et al. in 1993 [1].

In 1990, Lehoczky [20] showed that DMPO is not optimal for tasksets with arbitrary deadlines; however, an optimal priority ordering for such tasksets can be determined in at most $n(n+1)/2$ task schedulability tests using Audsley's optimal priority assignment (OPA) algorithm[3] [1], [2]. Exact schedulability tests for tasksets with arbitrary deadlines were developed by Lehoczky [20] in 1990 and Tindell et al. [23] in 1994.

Exact EDF-P schedulability tests for both constrained and arbitrary-deadline tasksets were introduced by Baruah et al. [3], [4] in 1990.

1.2. Non-pre-emptive scheduling

In 1980, Kim and Naghibdadeh [18], and in 1991, Jeffay et al. [15], gave exact schedulability tests for implicit-deadline tasksets under Earliest Deadline First non-pre-emptive (EDF-NP) scheduling. These tests were

[1] A taskset is synchronous if all of its tasks share a common release time.

[2] *Deadline monotonic* priority ordering assigns priorities in order of task deadlines, such that the task with the shortest deadline is given the highest priority.

[3] This algorithm is optimal in the sense that it finds a schedulable priority ordering whenever such an ordering exists.

extended by George et al. [12] in 1996, to the general case of sporadic tasksets with arbitrary deadlines.

While EDF-P is an optimal uniprocessor scheduling algorithm, in the non-pre-emptive case no work-conserving algorithm is optimal. This is because in general it is necessary to insert idle time to achieve a feasible schedule. The interested reader is referred to [12] for examples of this behaviour.

In 1995, Howell and Venkatrao [14] showed that for non-concrete[4] periodic tasksets, the problem of determining a feasible non-pre-emptive schedule is NP hard. Further they showed that for sporadic tasksets, no optimal on-line inserted idle time algorithm can exist. In other words, clairvoyance is needed to determine a feasible non-pre-emptive schedule.

While no work-conserving algorithm is optimal in the strong sense that it can schedule any taskset for which a feasible non-pre-emptive schedule exists; in 1995, George et al. [13] showed that EDF-NP is optimal in the weak sense that it can schedule any taskset for which a feasible work-conserving, non-pre-emptive schedule exists. Hence we can regard EDF-NP as an optimal work-conserving, non-pre-emptive scheduling algorithm, for non-concrete tasksets.

George et al. [12] derived an exact schedulability test for fixed priority non-pre-emptive (FP-NP) scheduling of arbitrary-deadline tasksets, based on the approach of Tindell et al. [23] for the pre-emptive case. George et al. showed that unlike in the pre-emptive case, deadline monotonic priority ordering is not optimal for constrained-deadline tasksets scheduled by FP-NP. Further, they showed that Audsley's optimal priority assignment algorithm [2] is applicable, and can be used to determine an optimal priority ordering for tasksets with arbitrary-deadlines scheduled using FP-NP.

Subsequent research by Bril et al. [5] has refined exact analysis of FP-NP, correcting issues of both pessimism and optimism, and extending the schedulability tests to co-operative scheduling where each task is made up of a number of non-pre-emptive sections.

1.3. Sub-optimality and speedup factors

Combining the result of Dertouzos [9] with the results of Liu and Layland [22], shows that the processor speedup factor required to guarantee that FP-P scheduling can schedule any implicit-deadline taskset schedulable by EDF-P is $1/\ln(2) \approx 1.44270$.

In 2009, Davis et al. [7] derived the exact speedup factor for FP-P scheduling of constrained-deadline tasksets; $1/\Omega \approx 1.76322$ (where Ω is the mathematical constant defined by the transcendental equation $\ln(1/\Omega) = \Omega$, hence, $\Omega \approx 0.567143$). Also in 2009, Davis et al. [8] showed that the speedup factor for FP-P scheduling of arbitrary-deadline tasksets is lower bounded by $1/\Omega \approx 1.76322$ and upper bounded by 2. Further, if deadline monotonic priority assignment is assumed (which is not optimal for arbitrary-deadline tasksets), then the exact speedup factor required is 2.

[4] A periodic taskset is referred to as non-concrete if the times at which each task is first released are unknown.

In this paper, we derive upper and lower bounds on the speedup factor required such that any taskset that was previously schedulable according to an optimal work-conserving (i.e. non-idling) non-pre-emptive scheduling algorithm (i.e. EDF-NP) can be guaranteed to be schedulable according to fixed priority non-pre-emptive (FP-NP) scheduling. These bounds are valid for all three classes of non-concrete taskset; Implicit-deadline, constrained-deadline, and arbitrary-deadline. While these results are mainly theoretical, they also have practical utility in enabling system designers to quantify the maximum penalty for using fixed priority non-pre-emptive scheduling in terms of the additional processing capacity required. This performance penalty can then be weighed against other factors such as implementation overheads when considering which scheduling algorithm to use.

1.4. Organisation

The remainder of this paper is organised as follows. Section 2 describes the system model, notation and analysis used. Section 3 illustrates the concept of processor speedup factors via a simple example. Section 4 derives a lower bound on the processor speedup factor required for FP-NP scheduling, while Section 5 derives the corresponding upper bound. Section 6 reports on an empirical investigation into the speedup factor required for FP-NP scheduling, verifying the theoretical lower bound. Section 7 concludes with a summary of the results and suggestions for future work.

2. System model, notation, and analysis

In this section, we outline the scheduling model, notation and terminology used in the rest of the paper. We then recapitulate schedulability analysis for both FP-NP and EDF-NP scheduling.

2.1. Scheduling model, terminology and notation

In this paper, we consider the non-pre-emptive scheduling of a set of sporadic tasks (or *taskset*) on a uniprocessor.

Each taskset comprises a static set of *n* tasks ($\tau_1..\tau_n$), where *n* is a positive integer. We assume that the index *i* of task τ_i also represents the task priority used in fixed priority scheduling, hence τ_1 has the highest fixed-priority, and τ_n the lowest.

Each task τ_i is characterised by its bounded worst-case execution time C_i, minimum inter-arrival time or period T_i, and relative deadline D_i. Each task τ_i therefore gives rise to a potentially infinite sequence of invocations (or jobs), each of which has an execution time upper bounded by C_i, an arrival time at least T_i after the arrival of its previous invocation, and an absolute deadline D_i time units after its arrival.

In an *implicit-deadline* taskset, all tasks have $D_i = T_i$. In a *constrained-deadline* taskset, all tasks have $D_i \le T_i$, while in an *arbitrary-deadline* taskset, task deadlines are independent of their periods, thus each task may have a deadline that is less than, equal to, or greater than, its period. The set of arbitrary-deadline tasksets is therefore a superset of the set of constrained-deadline tasksets, which is itself a superset of the set of implicit deadline tasksets.

The *utilisation* U_i, of a task is given by its execution time divided by its period ($U_i = C_i / T_i$). The total utilisation U, of a taskset is the sum of the utilisations of all of its tasks:

$$U = \sum_{i=1}^{n} U_i \tag{1}$$

The following assumptions are made about the behaviour of the tasks:

- o The arrival times of the tasks are independent and unknown a priori (*non-concrete*), hence the tasks may share a common release time.
- o Each task is released (i.e. becomes ready to execute) as soon as it arrives.
- o The tasks are independent and so cannot block each other from executing by accessing mutually exclusive shared resources, with the exception of the processor.
- o The tasks do not voluntarily suspend themselves.

A task is said to be *ready* if it has outstanding computation awaiting execution by the processor.

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

Under EDF-NP scheduling, whenever a job completes execution, or when the processor is idle and a job becomes ready to execute, the ready job with the earliest absolute deadline is selected to execute. By contrast, under FP-NP the highest priority ready job is selected.

When a taskset is scheduled according to FP-NP, task priorities need to be assigned according to some algorithm. Audsley's Optimal Priority Assignment (OPA) algorithm [1], [2], (see Figure 1 below) provides the optimal policy for sporadic tasksets with implicit-deadlines, constrained-deadlines or arbitrary-deadlines.

Optimal Priority Assignment Algorithm

```
for each priority level k, lowest first {
    for each unassigned task τ {
        if(τ is schedulable at priority k with
all           other unassigned tasks assumed
to have           higher priorities) {
            assign τ to priority k
            break (continue outer loop)
        }
    }
    return unschedulable
}
return schedulable
```

Figure 1: OPA algorithm

A priority assignment policy P is said to be *optimal* with respect to some class of tasksets, and a fixed priority scheduling algorithm FP-X if there are no tasksets in the class that are schedulable according to FP-X using any other priority ordering policy that are not also schedulable using the priority assignment determined by policy P.

A taskset is said to be *feasible* with respect to a given system model if there exists some scheduling algorithm that can schedule all possible sequences of task activations that may be generated by the taskset on that system without missing any deadlines.

A scheduling algorithm is said to be *optimal* with respect to a system model and a tasking model if it can schedule all of the tasksets that comply with the tasking model and are feasible on the system.

We note that EDF-NP is optimal in the weak sense that it can schedule any sporadic taskset for which feasible work-conserving, non-pre-emptive schedules exist [13].

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

2.2. Schedulability analysis for FP-NP

Exact schedulability analysis for an arbitrary-deadline sporadic taskset under FP-NP was given by George et al. [12] and Bril et al. [5]. Below, we provide a simple sufficient schedulability test for FP-NP scheduling, derived from these exact tests. This sufficient test is used in the derivation of an upper bound on the speedup factor for FP-NP scheduling given in Section 5. First, we introduce the concepts of worst-case response time, priority level-i active period, and Δ-critical instant, which are fundamental to analysis of FP-NP scheduling.

For a taskset scheduled under FP-NP scheduling, the *worst-case response time* R_i of a task τ_i is given by the longest possible time from release of the task until it completes execution. Thus task τ_i is schedulable if and only if $R_i \le D_i$, and the taskset is schedulable if and only if $\forall i \quad R_i \le D_i$.

The term *priority level-i active period* refers to a continuous period of time $[t_1, t_2)$ during which tasks, of priority i or higher, that were released at the start of the active period at t_1, or during the active period but strictly before its end at t_2, are either executing or ready to execute.

A *Δ-critical instant* for a task τ_i refers to a pattern of task arrivals such that task τ_i is released simultaneously with all tasks of higher priority than i, and then subsequent releases of task τ_i and the higher priority tasks occur as early as possible given the constraints on minimum inter-arrival times. Further, some infinitesimal amount of time Δ prior to this simultaneous release, a lower priority task τ_k is released, and this task has the longest execution time of any such lower priority task. Thus the longest time that task τ_i and higher priority tasks can be blocked from executing by lower priority tasks is given by :

$$B_i = \begin{cases} \max\limits_{\forall k \in lp(i)} (C_k - \Delta) & i < n \\ 0 & i = n \end{cases} \tag{2}$$

where *lp*(*i*) is the set of tasks with priorities lower than *i*.

Bril et al. [5] showed that for FP-NP scheduling, the longest response time of a task τ_i occurs for some invocation of the task within the priority level-*i* active period starting at the Δ-*critical instant* for task τ_i. Lemma 3 in [5] states that the worst-case length of a priority level-*i* active period A_i is given by the minimum solution to the following fixed point iteration:

$$A_i^{m+1} = B_i + \sum_{\forall j \in hep(i)} \left\lceil \frac{A_i^m}{T_j} \right\rceil C_j \tag{3}$$

where *hep*(*i*) is the set of tasks with priorities higher than or equal to *i*. Iteration starts with an initial value A_i^0 guaranteed to be no larger than the minimum solution, for example $A_i^0 = C_i$, and ends when $A_i^{m+1} = A_i^m$.

From Equation (3), we can form the following simple sufficient test for FP-NP scheduling of arbitrary-deadline tasksets. Each task τ_i is schedulable provided that:

$$D_i \geq B_i + \sum_{\forall k \in hep(i)} \left\lceil \frac{D_i}{T_k} \right\rceil C_k \tag{4}$$

If Equation (4) holds, then this indicates that the solution to the fixed point iteration of Equation (3) must be $\leq D_i$. As the worst-case length of a priority level-*i* active period is then $\leq D_i$, it follows that the worst-case response time R_i of task τ_i must also be $\leq D_i$, and hence the task must be schedulable. If all of the tasks are schedulable according to Equation (4), then the taskset is schedulable. (For an exact schedulability test for FP-NP, see [5]).

2.3. Schedulability analysis for EDF-NP

Baruah et al [3], [4] gave an exact schedulability test for EDF-P based on the concept of the processor demand bound function *h*(*t*). George et al. [12] extended this test to EDF-NP via the addition of a blocking factor *B*(*t*).

$$h(t) = \sum_{i=1}^{n} \max\left(0, \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right) C_i \tag{5}$$

$$B(t) = \begin{cases} \max\limits_{\forall i: D_i > t} (C_i - \Delta) & t < \max\limits_{\forall i}(D_i) \\ 0 & t \geq \max\limits_{\forall i}(D_i) \end{cases} \tag{6}$$

George et al. [12] showed that an arbitrary-deadline taskset is schedulable under EDF-NP if and only if $U \leq 1$ and a quantity referred to as the processor LOAD is ≤ 1, where the processor LOAD is given by:

$$\text{LOAD} = \max_{\forall t} \left(\frac{h(t) + B(t)}{t} \right) \tag{7}$$

Further, if $U \leq 1$ and the value of $(h(t) + B(t))/t$ is ≤ 1 for all values of *t* in the interval $(0, L]$, (where *L* is the length of the synchronous priority level-*n* active period, given by the minimum solution to Equation (3)), then LOAD ≤ 1. Thus the only values of *t* that need to be checked are those in the interval $(0, L]$ that correspond to times when $h(t) + B(t)$ can change, i.e. $\forall i \quad t = kT_i + D_i$ for integer values of *k*. We note that recently significant developments have been made, reducing the number of values of *t* that need to be checked [11]; however, this simple description of the analysis is sufficient for the purposes of this paper. (We note that for schedulable tasksets, the maximum value of $(h(t) + B(t))/t$ does not necessarily occur in the interval $(0, L]$, but does occur in the interval $[D_{\min}, H]$ where $D_{\min}$ is the shortest task deadline and *H* is the taskset hyperperiod or Least Common Multiple of task periods [10]).

2.4. Definitions

Definition 1: Let $f^{OPT}(\Psi)$ be the lowest processor speed such that taskset Ψ is schedulable according to an optimal scheduling algorithm. Assume that $f^A(\Psi)$ is similarly the lowest processor speed that will schedule taskset Ψ using scheduling algorithm *A*. The *processor speedup factor* f^A for algorithm *A* is given by the maximum increase in processor speed required over an optimal algorithm for any taskset Ψ.

$$f^A = \max_{\forall \Psi} \left(f^A(\Psi) / f^{OPT}(\Psi) \right) \tag{8}$$

For any scheduling algorithm *A*, we have $f^A \geq 1$, with smaller values of f^A indicative of a more effective scheduling algorithm, and $f^A = 1$ implying that *A* is an optimal algorithm.

In the remainder of the paper, unless otherwise stated, when we refer to the processor speedup factor, we mean the processor speedup factor for FP-NP scheduling using an optimal priority assignment policy, as compared to EDF-NP, an optimal (in the weak sense [13]), work-conserving non-pre-emptive scheduling algorithm.

Definition 2: A taskset is said to be *speedup-optimal* if it requires the processor to be speeded up by the processor speedup factor in order to be schedulable using FP-NP scheduling. Hence for a speedup-optimal taskset Ψ, $f^A(\Psi) / f^{OPT}(\Psi) = f^A$.

Definition 3: Let *S* be some arbitrary taskset, now assume that $\alpha^{FP-NP}(S)$ is the maximum factor by which the execution times of all of the tasks in *S* can be scaled, such that the taskset is schedulable under FP-NP. Similarly, let $\alpha^{EDF-NP}(S)$ be the maximum scaling factor under EDF-NP. The speedup factor $f^{FP-NP}(S)$ for the taskset is given by:

$$f^{FP-NP}(S) = \alpha^{EDF-NP}(S) / \alpha^{FP-NP}(S) \tag{9}$$

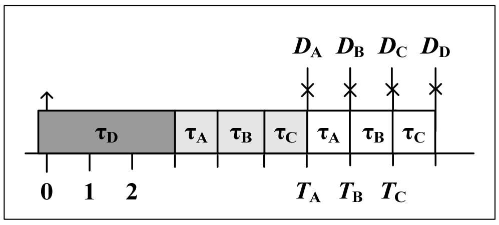
3. Example

The concept of a speedup factor for a given taskset *S* can be illustrated by means of the following example. Consider the taskset *S* comprising the tasks defined in Table 1, with priorities assigned in the order that the tasks appear in the table (i.e. τ_A has the highest priority, and τ_D the lowest).

Table 1

Task	C_i	$D_i = T_i$
τ_A	1	6
τ_B	1	7
τ_C	1	8
τ_D	$3+\Delta$	∞

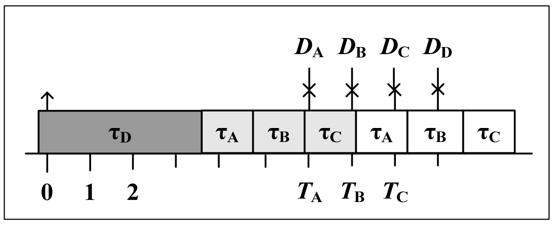
The worst-case arrival pattern for tasks τ_A, τ_B, and τ_C under FP-NP scheduling is shown in Figure 2. Note the 1st job of each task is shaded in grey, while the 2nd job of each task is un-shaded.



D_A D_B D_C D_D
τ_D τ_A τ_B τ_C τ_A τ_B τ_C
0 1 2
T_A T_B T_C

Figure 2: FP-NP schedule

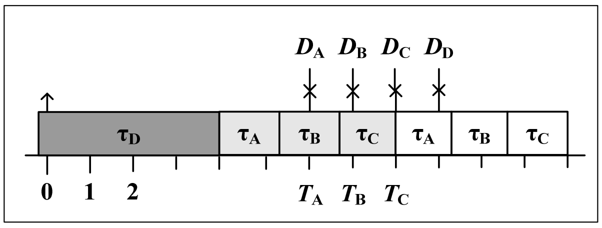
Now consider the maximum factor by which the execution times of the tasks can be scaled and the taskset remain schedulable according to FP-NP. This factor is $\alpha^{FP-NP}(S) = (6/5)^-$ (i.e. a value infinitesimally less than 6/5).



D_A D_B D_C D_D
τ_D τ_A τ_B τ_C τ_A τ_B τ_C
0 1 2
T_A T_B T_C

Figure 3: FP-NP schedule, maximal scaling

Figure 3 shows the FP-NP schedule for the scaled taskset. Scaling by any larger factor, for example, a factor equal to 6/5 would result in the first job of task τ_C being unable to start executing before the 2nd job of task τ_A is released at time $t = 6$. It would then be further delayed by the 2nd job of task τ_B, and hence fail to met its deadline at time $t = 8$. In fact, there is no priority ordering which results in taskset S, scaled by a factor of 6/5, being schedulable. This can be seen by considering the behaviour of the OPA algorithm. While task τ_D is schedulable at the lowest priority, and can therefore be assigned priority 4, none of the other tasks are schedulable at priority 3.



D_A D_B D_C D_D
τ_D τ_A τ_B τ_C τ_A τ_B τ_C
0 1 2
T_A T_B T_C

Figure 4: EDF-NP schedule, maximal scaling

With EDF-NP scheduling, the maximum scaling factor commensurate with taskset S remaining schedulable is $\alpha^{EDF-NP}(S) = 8/6$. Under EDF-NP, the first job of task τ_C has a later absolute deadline than the first jobs of tasks τ_A and τ_B, and therefore executes after those jobs and after the first job of τ_D which is released at time $t = -\Delta$. The first job of task τ_C is not however delayed by the 2nd jobs of tasks τ_A and τ_B, as these jobs have later absolute deadlines. With a scaling factor of 8/6, the first job of task τ_C just completes by its deadline (see Figure 4). Further analysis is required to prove that the scaled taskset is schedulable under EDF-NP; however, as the priority level 3 active period ends at $t = 12$, we need only check all deadlines in the interval [0, 12] to show schedulability. Note, Figure 4 shows the Δ-*critical instant* for tasks τ_A, τ_B, τ_C, and all deadlines are met in the interval [0, 12]. Further, task τ_D is trivially schedulable as it has an infinite deadline, and the taskset utilisation is less than 1.

Using Equation (9), the speedup factor for the taskset given in Table 1 is $f^{FP-NP}(S)$ = $(8/6)/(6/5)^-$ = $(40/36)^+$ = $(10/9)^+$ (i.e. a value infinitesimally larger than 10/9). In the next section, we generalise this example and show how tasksets with a similar structure but with a large number of tasks require a much larger speedup factor.

4. Lower bound speedup factor for FP-NP

In this section, we derive a lower bound on the processor speedup factor required for FP-NP scheduling using optimal priority assignment [1], [2]. This lower bound is valid for sporadic and non-concrete periodic tasksets with implicit-, constrained-, and arbitrary-deadlines. In Section 5, we derive an upper bound with the same scope.

To derive a lower bound, we need only select a single taskset and determine the required speedup factor for that taskset. The taskset S that we use is a generalisation of the taskset used as an example in Section 3. In this case, there are n tasks, with the parameters given in Table 2. Tasks τ_1 to τ_{n-1} are represented by τ_i in the first row of the table. All of these tasks have the same small execution time $\varepsilon << X$, and related periods/deadlines. Further, all of the tasks have periods equal to their deadlines, so this is an implicit-deadline taskset. Task τ_n has an execution time of $X + \Delta$, where X is a free variable that we can alter to maximise the required speedup factor.

Table 2

Task	C_i	$D_i = T_i$
τ_i	$\varepsilon = \frac{1}{(n-1)}$	$1 + X + \frac{(i-1)}{(n-1)}$
τ_n	$X + \Delta$	∞

The execution of taskset S under FP-NP is depicted in Figure 5 below. Note, jobs of task τ_{n-1} are marked with an ε.

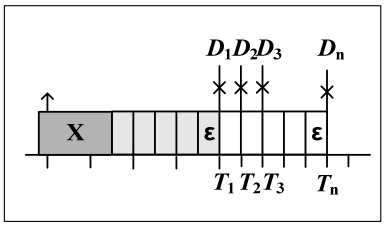


Figure 5: FP-NP schedule

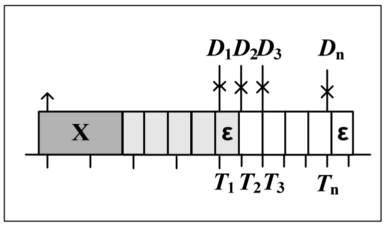


Figure 6: FP-NP schedule, maximal scaling

Lemma 1: The maximum factor $\alpha^{FP-NP}(S)$ by which the execution times of the tasks in taskset *S* (Table 2) can be scaled and the taskset remain schedulable according to FP-NP. is given by:

$$\alpha^{FP-NP}(S) = \left(\frac{1+X}{1+X-\varepsilon}\right)^{-} \underset{\varepsilon \to 0}{=} 1^{+} \tag{10}$$

Figure 6 depicts the FP-NP schedule for the scaled taskset.

Proof: Scaling by a factor equal to $(1+X)/(1+X+\varepsilon)$ would result in the first job of task τ_{n-1} being unable to start executing before the 2nd job of task τ_1 is released at time $t = 1+X$ It would then be further delayed by the 2nd jobs of tasks τ_1 to τ_{n-2}, and hence fail to met its deadline at time $t = 2+X-\varepsilon$. In fact, there is no priority ordering which results in taskset *S*, scaled by a factor of $(1+X)/(1+X+\varepsilon)$, being schedulable. This can be seen by considering the behaviour of the OPA algorithm, given the scaled taskset: Task τ_n is schedulable at the lowest priority, and can therefore be assigned that priority. However, considering the remaining tasks in turn, none of them are schedulable at priority *n*-1. Task τ_1 is not schedulable at priority *n*-1, as its 1st job would miss its deadline at $t = 1+X$. Task τ_2 is not schedulable at priority *n*-1, as its 1st job is then unable to start before the 2nd job of task τ_1 arrives, and so misses its deadline at $t = 1+X+\varepsilon$. In general, with a scaling factor of $(1+X)/(1+X+\varepsilon)$, for each task with index *i* from 2 to *n*-1, assuming that task τ_i is assigned priority *n*-1, ensures that the 1st job of task τ_i is unable to start before the 2nd job of task τ_{i-1} arrives, and so the 1st job of task τ_i misses its deadline.

By contrast, with a scaling factor of $(1+X)/(1+X+\varepsilon)^{-}$, task τ_{n-1} is schedulable at priority *n*-1, as it is able to start executing just prior to the arrival of the 2nd job of τ_1 at $t = 1+X$. Further, with this scaling factor, all of the other tasks are schedulable with priorities assigned according to their indices (i.e. in Deadline Monotonic priority order). This can be seen by checking the deadlines of all jobs up to the end of the priority level *n*-1 active period, which occurs at:

$$t^{A} = (2+X)\left(\frac{(1+X)}{1+X+\varepsilon}\right)^{-} \tag{11}$$

As $t^{A} < 2(1+X) = 2D_1$, the priority level *n*-1 active period comprises the 1st job of task τ_n and the 1st and 2nd jobs of tasks τ_1 to τ_{n-1}. All of these are schedulable (see Figure 6). The priority level-*n* active period is of similar length, and hence task τ_n is trivially schedulable given its infinite deadline □

Lemma 2: The maximum factor $\alpha^{EDF-NP}(S)$ by which the execution times of the tasks in taskset *S* (Table 2) can be scaled and the taskset remain schedulable according to EDF-NP. is given by:

$$\alpha^{EDF-NP}(S) = (1/\Omega)^{-} \tag{12}$$

Proof: There are two key conditions which limit the maximum scaling factor under EDF-FP (otherwise the taskset would become unschedulable):

1. The 1st jobs of all tasks must be complete by the deadline of task τ_{n-1}, $D_{n-1} = 2+X-\varepsilon$.
2. Utilisation of the scaled taskset must not exceed 100%.

Considering the first condition, we have:

$$\alpha^{EDF-NP}(S) \le \frac{2+X-\varepsilon}{1+X} \tag{13}$$

The utilisation of the un-scaled taskset is given by the sum of the utilisation of each task:

$$U = \sum_{i=1}^{n-1} \frac{1}{(n-1)} \frac{1}{(1+X+\dfrac{i-1}{n-1})} \tag{14}$$

The RHS of Equation (14) is recognisable as the left Riemann sum of the function 1/z, over the interval $[1+X, 2+X)$, hence:

$$U \overset{n\to\infty}{=} \int_{1+X}^{2+X} \frac{1}{z} dz = \ln\left(\frac{2+X}{1+X}\right) \tag{15}$$

Thus, considering the second condition, we have:

$$\alpha^{EDF-NP}(S) \le 1/\ln\left(\frac{2+X}{1+X}\right) \tag{16}$$

As Equation (13) is monotonically non-increasing in *X* and tends to 2 for small *X*, and Equation (16) is monotonically non-decreasing in *X* and tends to 1/ln(2) for small *X*, then the maximum value is obtained when the RHSs of Equations (13) and (16) are equal, i.e. when:

$$\alpha^{EDF-NP}(S) = \frac{2+X-\varepsilon}{1+X} = 1/\ln\left(\frac{2+X}{1+X}\right) \tag{17}$$

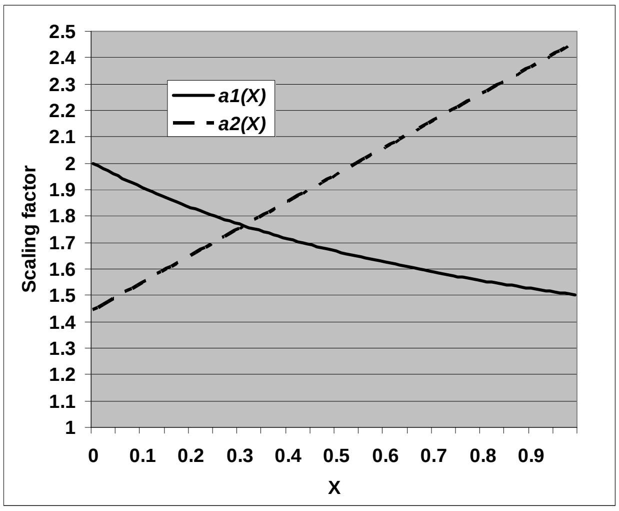


Figure 7: Constraints on the scaling factor as a function of *X*

Figure 7 plots Equations (13) and (16) (labelled *a1*(*X*) and *a2*(*X*) respectively) against *X*. As $n \to \infty$, $\varepsilon \to 0$, the solution to Equation (17) is given by the intersection of the lines plotted in Figure 7, thus $\alpha^{EDF-NP}(S) = (1/\Omega)^- \approx 1.76322$, (where Ω is the mathematical constant defined by the transcendental equation $\ln(1/\Omega) = \Omega$, hence, $\Omega \approx 0.567143$). Further,

$$X = \frac{2\Omega - 1 - \varepsilon}{1 - \Omega} \approx 0.310232 \tag{18}$$

We now show that taskset *S* (Table 2) is schedulable under EDF-NP, when scaled by a factor of $\alpha^{EDF-NP}(S) = (1/\Omega)^-$. Proof is made significantly easier by the commonality between taskset *S* and the speedup-optimal taskset *V* for the constrained-deadline case of FP-P scheduling, described in Theorem 2 of [7]. In fact, tasks τ_1 to τ_{n-1} are identical in these two tasksets, only task τ_n differs. In taskset *V*, the parameters of task τ_n are: $C_n = X$, $D_n = 2 + X$, and $T_n = \infty$, whereas in taskset *S*, the parameters of task τ_n are: $C_n = X + \Delta$, and $D_n = T_n = \infty$ Theorem 4 in [7] proves that taskset *V* is schedulable under EDF-P when scaled by a factor of $1/\Omega$. Hence for taskset *V*,

$$LOAD^P = \max_{\forall t}\left(\frac{h(t)}{t}\right) \le 1 \tag{19}$$

We make use of this result to show that taskset *S*, scaled by a factor of $(1/\Omega)^-$ is schedulable under EDF-NP. As tasks τ_1 to τ_{n-1} are identical, their contribution to the processor demand bound $h(t)$ is the same for any time *t*. We now compare the contribution from task τ_n in each case. In the pre-emptive case, (taskset V), τ_n contributes to $h(t)$ as follows:

$$h_n^P(t) = \begin{cases} 0 & 0 \le t < (2 + X) \\ X/\Omega & t \ge (2 + X) \end{cases} \tag{20}$$

whereas, in the non-pre-emptive case, (taskset *S*), τ_n contributes only to the blocking factor:

$$B(t) = (X/\Omega)^- \quad t \ge 0 \tag{21}$$

Recall that in the non-pre-emptive case, a taskset is schedulable provided that $U \le 1$ and:

$$LOAD^{NP} = \max_{\forall t}\left(\frac{h(t) + B(t)}{t}\right) \le 1 \tag{22}$$

Comparing Equations (19) and (22), and the contributions of task τ_n in each case (Equations (20) and (21)), it follows that Equation (22) holds for all values of $t \ge (2 + X)$ for taskset *S* scaled by a factor of $(1/\Omega)^-$. This is because, for all values of $t \ge (2 + X)$ the value of $(h(t) + B(t))/t$ is the same as that for taskset *V*, assuming both tasksets are scaled by the same factor. To prove the schedulability of taskset *S* scaled by a factor of $(1/\Omega)^-$, it remains only to show that $(h(t) + B(t))/t \le 1$ for all values of *t* in the interval $[0, (2 + X))$. Here, we need only check values of *t* that correspond to task deadlines. As $2 + X > 2D_1$, this amounts to checking the 1st deadline of each of the *n*-1 highest priority tasks. At each of these deadlines D_i, we have:

$$h(D_i) = \left(\frac{1}{\Omega}\right)^- \sum_{k=1}^{n-1} \max\left(0, \left(\left\lfloor \frac{D_i - D_k}{T_k} \right\rfloor + 1\right)\right) \frac{1}{n-1} \tag{23}$$

as $\forall i,k \quad D_i = T_i$ and $D_i < 2D_k$ it follows that:

$$h(D_i) = \left(\frac{1}{\Omega}\right)^- \frac{i}{n-1} \tag{24}$$

Hence the scaled taskset is schedulable provided that, $\forall i$ from 1 to *n*-1, $D_i \ge h(D_i) + B(D_i)$

$$1 + X + \frac{i-1}{n-1} \ge \left(\frac{1}{\Omega}\right)^- \left(X + \frac{i}{n-1}\right) \tag{25}$$

Substituting for $(1/\Omega)^- = (2 + X - \varepsilon)/(1 + X)$ and $\varepsilon = 1/(n-1)$, and rearranging, we have:

$$\left(1 + X + \frac{i-1}{n-1}\right)(1 + X) \ge \left(2 + X - \frac{1}{n-1}\right)\left(X + \frac{i}{n-1}\right) \tag{26}$$

which simplifies to:

$$1 + \frac{i-1}{n-1} - \frac{2i}{n-1} + \frac{i}{(n-1)^2} \ge 0 \tag{27}$$

and then to:

$$\frac{n-i-2}{n-1} + \frac{i}{(n-1)^2} \ge 0 \tag{28}$$

For $n \ge 2$, the first term in inequality (28) is non-negative $\forall i$ from 1 to *n*-2, while the second term is always positive. Further, for *i* = *n*-1, the first and second terms cancel out, thus the inequality holds $\forall i$ from 1 to *n*-1. Taskset *S* is therefore schedulable according to EDF-NP when scaled by a factor of $(1/\Omega)^-$ □

Theorem 1: A lower bound on the speedup factor required for FP-NP scheduling of an implicit-deadline taskset is:

$$f^{FP-NP} = \frac{\alpha^{EDF-NP}(S)}{\alpha^{FP-NP}(S)} = \frac{(1/\Omega)^-}{1^+} = (1/\Omega)^- \tag{29}$$

Proof: Follows from Lemmas 1 and 2 and the definition of the speedup factor □

Corollary 1: We observe that as taskset *S* is an implicit-deadline taskset, and all implicit-deadline tasksets are also constrained-deadline and arbitrary-deadline tasksets, the lower bound of Theorem 1 applies to all three classes of taskset.

It remains an open question whether or not the lower bound given in Theorem 1 is tight. While the taskset used to derive the bound is valid, whether or not it is a *speedup optimal* taskset (see Definition 2) remains to be proved / disproved. If the bound is not tight, then there exists a different taskset construction that requires a larger speedup factor.

5. Upper bound speedup factor for FP-NP

In this section, we derive an upper bound on the speedup factor required for FP-NP scheduling of arbitrary-deadline sporadic and non-concrete periodic tasksets.

Theorem 2: An upper bound on the processor speedup factor required such that FP-NP scheduling, using optimal priority assignment can schedule any arbitrary-deadline sporadic or non-concrete periodic taskset schedulable under EDF-NP according to Equation (7), is 2.

Proof: Let *S* be any taskset that is schedulable according to Equation (7) on a processor of unit speed under EDF-NP. For each task τ_k, in *S*, consider the processor demand bound and blocking factor for an interval of length $2D_k$. As taskset *S* is schedulable according to EDF-NP, it follows that:

$$\left(B(2D_k) + \sum_{i=1}^{n} \max\left(0, \left\lfloor \frac{2D_k - D_i}{T_i} \right\rfloor + 1\right) C_i \right) \le 2D_k \tag{30}$$

Next, consider taskset *S* scheduled according to FP-NP scheduling on a processor of speed 2 using Deadline Monotonic priority assignment (rather than OPA). DMPO implies that $\forall i \le k \quad D_i \le D_k$.

From Equation (30) above, assuming speed 2, and separating out the contribution from all tasks of lower or equal priority to *k* we have:

$$B(2D_k) + \sum_{i \in lep(k)} \max\left(0, \left\lfloor \frac{2D_k - D_i}{T_i} \right\rfloor + 1\right) C_i + \sum_{i \in hp(k)} \max\left(0, \left\lfloor \frac{2D_k - D_i}{T_i} \right\rfloor + 1\right) C_i \le D_k \tag{31}$$

where $lep(k)$ is the set of tasks with priorities lower than or equal to *k*. As the tasks are in DMPO, we note that all of the tasks in $lep(k)$ have deadlines $\ge D_k$.

We now consider just the first and second terms in Equation (31). Observe that the contribution to the second term from every task τ_i in $lep(k)$ with $D_i > 2D_k$ is zero. Further, there is a contribution from each task τ_i with $D_k \le D_i \le 2D_k$ of at least C_i. From the definition of $B(t)$ (Equation (6)), the definition of B_k (Equation (2)), and the fact that the tasks are in DMPO, it follows that the sum of the first two terms in Equation (31) are $\ge B_k$, the blocking factor for FP-NP scheduling:

$$\left.\begin{array}{ll} \max\limits_{\forall i: D_i > 2D_k} (C_i - \Delta) & 2D_k < \max\limits_{\forall i}(D_i) \\ 0 & 2D_k \ge \max\limits_{\forall i}(D_i) \end{array}\right\} + \sum_{\forall i: D_k \le D_i \le 2D_k} C_i \ge B_k$$

$$\text{where } B_k = \begin{cases} \max\limits_{\forall i \in lp(k)} (C_i - \Delta) & k < n \\ 0 & k = n \end{cases} \tag{32}$$

Substituting B_k for the first two terms in Equation (31) and transforming the third term by noting that $\lfloor x \rfloor + 1 \ge \lceil x \rceil$ and $\forall i \in hp(k) \quad D_i \le D_k$ we have:

$$B_k + \sum_{i \in hp(k)} \left\lceil \frac{D_k}{T_i} \right\rceil C_i \le D_k \tag{33}$$

Equation (33) is identical to Equation (4); the sufficient schedulability test for task τ_k in an arbitrary-deadline taskset *S*, scheduled under FP-NP. Repeating the above argument for each task τ_k in *S* therefore proves that the taskset is schedulable on a processor of speed 2 under FP-NP, with Deadline Monotonic priority assignment. As optimal priority assignment for FP-NP can schedule any taskset that is schedulable using FP-NP with Deadline Monotonic priority ordering □

Corollary 2: We observe that as the upper bound in Theorem 2 holds for arbitrary-deadline tasksets, it must also hold for implicit-deadline, and constrained-deadline tasksets.

6. Empirical results

In this section, we confirm by experiment the results presented in Section 4 concerning the lower bound speedup factor for FP-NP. We consider the task set proposed in Table 2 where $n-1$ tasks have worst-case execution times equal to $1/(n-1)$, and task τ_n has a worst-case execution time equal to $X + \Delta$ and an infinite period and deadline.

Our experiments indicate that a value of $X \approx 0.31$ results in the maximum speedup factor for a given number of tasks. Based on this value, we verify the lower bound given in this paper for a large number of tasks.

Taskset *S* investigated in this section is based on the task model presented in Table 2. This model is theoretical, while our empirical assumptions are as follows:

- Δ is equal to 0.001.
- The infinite values of period and deadline for task τ_n are replaced in the experiments by values such that τ_n does not interfere with other tasks.

Finally, to avoid the problem of rounding, we have used integer values, thus given that Δ is the time granularity, each task parameter $\{C_i, D_i, T_i\}$ is normalized according to Δ. For example, with three tasks and $X = 1$, Table 3 gives the task set studied.

Table 3: Example of taskset studied with $\Delta = 0.001$ and $X = 1$

Task	C_i	$D_i = T_i$
τ_1	500	2000
τ_2	500	2500
τ_3	1001	∞

In order to determine the speedup factor $f^{FP-NP}(S)$, we compute the maximum factors $\alpha^{EDF-NP}(S)$ and $\alpha^{FP-NP}(S)$ via an approach based on binary search (dichotomy). Algorithm 1 describes this approach, as used in our experiments. To check the schedulability of task set *S*, the function *checkFeasibility*() uses following exact tests:

- For EDF-NP [12]:

$$\max_{\forall t \in [0,L]} \left(\frac{h(t) + B(t)}{t} \right) \le 1 \tag{34}$$

- For FP-NP [6]:

$$\forall i, R_i = \max_{q \in [0, Q_i^{\max}]} (W_{i,q} + C_i - qT_i) < D_i \tag{35}$$

where $Q_i^{\max} = \lceil A_i / T_i \rceil - 1$, A_i is computed according to Equation (3) and $W_{i,q}$ is given by the minimum solution to the following fixed point iteration:

$$W_{i,q}^{m+1} = \left\lceil \frac{W_{i,q}^{m} + \Delta}{T_i} \right\rceil C_i + B_i \tag{36}$$

with B_i determined by Equation (2) and $W_{i,q}^{0} = qT_i$.

Finally, the speedup factor is computed via Equation (9) $f^{FP-NP}(S) = \alpha^{EDF-NP}(S) / \alpha^{FP-NP}(S)$.

```
Function $getMaxScalingFactor(\ S$ : TaskSet, $\Delta$ : Double) : Double
    [$\alpha$ must not produce a task set $S$ with utilization greater than 1]
    $\alpha_{Limit} \leftarrow \frac{1}{U}$ : Double
    $\alpha \leftarrow \min(1, \alpha_{Limit})$ : Double
    $Interval \leftarrow \frac{\alpha}{2}$ : Double
    $TestOld \leftarrow checkFeasibility(\alpha, S)$ : Boolean
    $TestNew \leftarrow TestOld$ : Boolean

    [While the interval of study is higher than the precision required or the $S$ is not schedulable at the given $\alpha$]
    While ($Interval > \Delta$ OR $testOld \neq$ true) do
        [If the $S$ is not schedulable we reduce the $\alpha$ according to the length of the current interval of study. Otherwise $\alpha$ is increased.]

        If ($testOld =$ false) then
            $\alpha \leftarrow \alpha - Interval;$
        else
            $\alpha \leftarrow \alpha + Interval;$
            [If $\alpha$ increase, we have to check the $\alpha_{Limit}$ value]
            While ($\alpha > \alpha_{Limit}$) do
                $\alpha \leftarrow \alpha - Interval;$
                $Interval \leftarrow \frac{Interval}{2};$
                $\alpha \leftarrow \alpha + Interval;$
            done
        end If
        [If the schedulability associated with the new value of $\alpha$ differs from the previous one, the interval of study is divided by two for the next iteration]
        $testNew \leftarrow checkFeasibility(\alpha, S);$
        If ($testNew \neq testOld$) then
            $Interval \leftarrow \frac{Interval}{2};$
        end If
        $testOld \leftarrow testNew;$
    done
    return $\alpha$;
End

[$checkFeasibility(\alpha, S)$ returns true if $S$ is schedulable with scaling factor $\alpha$, false otherwise]
```

Algorithm 1: Determines the maximum scaling factor of a taskset S with a precision Δ via binary search (dichotomy).

Figure 8 shows the maximum speedup factor obtained for $n = 5$ tasks. It shows the value of X for which the speedup factor was empirically found to be a maximum i.e. $X = 0.31$.

Figure 9 represents, for this optimum value $X = 0.31$, the speedup factor obtained as a function of the number of tasks (from 10 to 400 tasks). We observe that the maximum speedup factor for FP-NP tends towards 1.76322, the lower bound of $f^{FP-NP}(S)$ characterized in Section 4, as the number of tasks increases. Note that the saw tooth appearance of the curve is an artefact of the quantisation of the execution time values used in the experiment. When the number of tasks is large, this causes a noticeable quantisation of the scaling factors that can be explored.

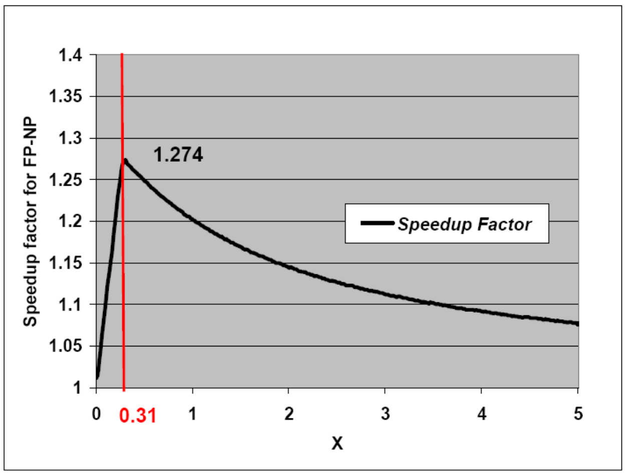


Figure 8: Constraints on the scaling factor as a function of X for $n = 5$

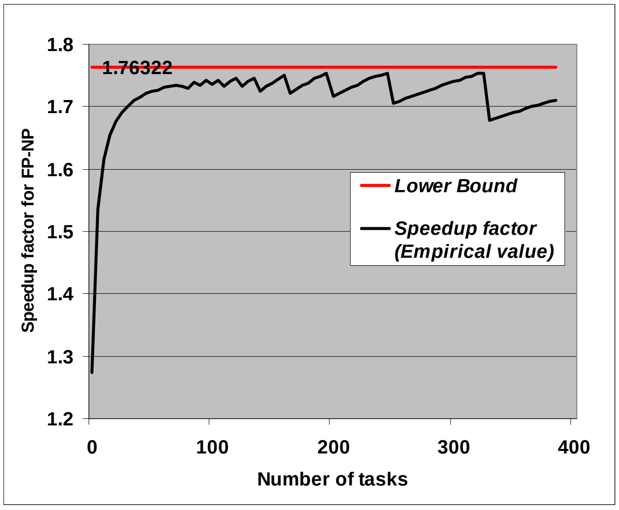


Figure 9: Speedup factor for $X = 0.31$ as a function of the number of tasks

7. Conclusions and future work

In this paper, we examined the relative effectiveness of fixed priority non-pre-emptive scheduling. Our metric for measuring the effectiveness of this scheduling algorithm is a resource augmentation factor known as the processor speedup factor. In this case, the processor speedup factor is defined as the minimum amount by which the processor needs to be speeded up so that any taskset that is schedulable by an optimal work-conserving non-pre-emptive scheduling algorithm (e.g. EDF-NP) can be guaranteed to be schedulable under FP-NP scheduling. Recall that EDF-NP is optimal in the weak sense [13], in that it can schedule all sporadic or non-concrete periodic tasksets for which a feasible non-pre-emptive, work-conserving schedule exists. It is not optimal in the strong sense as it cannot schedule all tasksets for which a feasible non-work-conserving, non-pre-emptive schedulable exists. The speedup factors derived in this paper are with respect to this weak form of optimality.

Table 4: FP scheduling speedup factors

Taskset constraints	Pre-emptive Lower Bound	Pre-emptive Upper Bound	Non-pre-emptive Lower Bound	Non-pre-emptive Upper Bound
Implicit-deadline	$1/\ln(2) \approx$ 1.44269		$(1/\Omega)^- \approx$ **1.76322**	**2**
Constrained-deadline	$1/\Omega \approx$ 1.76322		$(1/\Omega)^- \approx$ **1.76322**	**2**
Arbitrary-deadline	$1/\Omega \approx$ 1.76322	2	$(1/\Omega)^- \approx$ **1.76322**	**2**

Table 4 shows the processor speedup factor needed for fixed priority scheduling with optimal priority assignment, for both the pre-emptive case (FP-P v. EDF-P), see Davis et al. [7], [8], and for the non-pre-emptive case (FP-NP v. EDF-NP), derived in this paper.

The major contribution of this paper is in proving that the processor speedup factor for fixed priority non-pre-emptive scheduling of sporadic or non-concrete periodic tasksets with optimal priority assignment, is upper bounded by 2, and lower bounded by $(1/\Omega)^-$ = 1.76322. We note that these bounds hold for tasksets with implicit-, constrained-, and arbitrary-deadlines.

The seminal work of Liu and Layland [22] characterises the maximum performance penalty incurred when an implicit-deadline taskset is scheduled using Rate-Monotonic, fixed priority pre-emptive scheduling instead of an optimal algorithm such as EDF-P. The research in this paper provides an analogous characterisation of the maximum performance penalty incurred when tasksets are scheduled using fixed priority non-pre-emptive scheduling instead of an optimal work-conserving non-pre-emptive scheduling algorithm e.g. EDF-NP.

We note that the two cases in Table 4 where a tight bound is known correspond to the only cases where optimal priority assignment can be achieved independently of schedulability testing. In the arbitrary-deadline case for FP-P scheduling and all cases of FP-NP scheduling, Audsley's OPA algorithm is required to find the optimal priority ordering. This dependence of priority ordering on schedulability testing makes it more difficult to reason about the properties of a theoretical speedup-optimal taskset that requires the exact speedup factor to be schedulable. In these cases, the exact sub-optimality of fixed priority scheduling remains an open question.

7.1. Acknowledgements

This work was funded in part by the EPSRC project TEMPO (EP/G055548/1) and the EU funded ArtistDesign Network of Excellence. The authors would like to thank Alan Burns for his insightful comments on an earlier draft.

References

[1] Audsley N.C., "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times", *Technical Report YCS 164*, Dept. Computer Science, University of York, UK, 1991.

[2] Audsley N.C. "On priority assignment in fixed priority scheduling", Information Processing Letters, 79(1): 39-44, May 2001.

[3] Baruah S.K., Mok A.K., Rosier L.E., "Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor". *In Proc. RTSS*, pages182-190, 1990.

[4] Baruah S.K., Rosier L.E., Howell R.R., "Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic Real-Time Tasks on one Processor". *Real-Time Systems*, 2(4), pages 301-324, 1990.

[5] Bril, R.J., Lukkien, J.J., and Verhaegh, W.F., "Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred pre-emption". Real-Time Systems. 42, 1-3 (Aug. 2009), 63-119.

[6] Davis, R. I., Burns, A., "Controller area network (CAN) schedulability analysis: Refuted, revisited and revised," Real-Time Systems, vol. 35, pp. 239–272, 2007.

[7] Davis R.I., Rothvoß T., Baruah S.K., Burns A., "Exact Quantification of the Sub-optimality of Uniprocessor Fixed Priority Pre-emptive Scheduling." *Real-Time Systems*, Volume 43, Number 3, pages 211-258, November 2009.

[8] Davis, R.I., Rothvoß, T., Baruah, S.K., Burns, A., "Quantifying the Sub-optimality of Uniprocessor Fixed Priority Pre-emptive Scheduling for Sporadic Tasksets with Arbitrary Deadlines". *In proceedings of Real-Time and Network Systems (RTNS'09)*, pages 23-31, October 26-27th, 2009.

[9] Dertouzos M.L., "Control Robotics: The Procedural Control of Physical Processes". *In Proc. of the IFIP congress*, pages 807-813, 1974.

[10] Fisher, N., Baker, T. P., Baruah, S. "Algorithms for Determining the Demand-Based Load of a Sporadic Task System". In *Proceedings of the 12th IEEE international Conference on Embedded and Real-Time Computing Systems and Applications* (RTCSA), pages 135-146, 2006.

[11] George, L., Hermant, J., "A norm approach for the Partitioned EDF Scheduling of Sporadic Task Systems." In Proc. ECRTS, 2009.

[12] George, L., Rivierre, N., Spuri, M., "Preemptive and Non-Preemptive Real-Time UniProcessor Scheduling", INRIA Research Report, No. 2966, September 1996.

[13] George, L., Muhlethaler, P., Rivierre, N., "Optimality and Non-Preemptive Real-Time Scheduling Revisited," Rapport de Recherche RR-2516, INRIA, Le Chesnay Cedex, France, 1995.

[14] Howell, R.R., Venkatrao, M.K., "On non-preemptive scheduling of recurring tasks using inserted idle time", Information and computation Journal, Vol. 117, Number 1, Feb. 15, 1995.

[15] K. Jeffay, D. F. Stanat, C. U. Martel, "On Non-Preemptive Scheduling of Periodic and Sporadic Tasks", In Proc. RTSS, pages 129-139, 1991.

[16] Joseph M., Pandya P.K., "Finding Response Times in a Real-time System". *The Computer Journal*, 29(5), pages 390–395, 1986.

[17] Kalyanasundaram B., Pruhs K., "Speed is as powerful as clairvoyance". *In Proceedings of the 36th Symposium on Foundations of Computer Science*, pages 214-221, 1995.

[18] Kim, Naghibdadeh, "Prevention of task overruns in real-time non-preemptive multiprogramming systems", Proc. of Perf., Assoc. Comp. Mach., 1980, pp 267-276.

[19] Leung J.Y.-T., Whitehead J., "On the complexity of fixed-priority scheduling of periodic real-time tasks". *Performance Evaluation*, 2(4), pages 237-250, 1982.

[20] Lehoczky J., "Fixed priority scheduling of periodic task sets with arbitrary deadlines". In *Proc. RTSS*, pages 201–209, 1990.

[21] Lehoczky J.P., Sha L., Ding Y., "The rate monotonic scheduling algorithm: Exact characterization and average case behaviour". *In Proc. RTSS*, pages 166–171, 1989.

[22] Liu C.L., Layland J.W., "Scheduling algorithms for multiprogramming in a hard-real-time environment", *Journal of the ACM*, 20(1) pages 46-61, 1973.

[23] Tindell K.W., Burns A., Wellings A.J., "An extendible approach for analyzing fixed priority hard real-time tasks". *Real-Time Systems.* Volume 6, Number 2, pages 133-151, 1994.