Exact Quantification of the Sub-optimality of Uniprocessor Fixed Priority Pre-emptive Scheduling.

Robert I. Davis, Thomas Rothvoß, Sanjoy K. Baruah, and Alan Burns

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

Thomas Rothvoß
Ecole Polytechnique Federale de Lausanne, Institute of Mathematics, Station 8 - Bâtiment MA, CH-1015 Lausanne, Switzerland.
Email: thomas.rothvoss@efpl.ch

Sanjoy K. Baruah
Department of Computer Science, University of North Carolina, Chapel Hill, NC 27599-317, Carolina, USA.
Email: baruah@cs.unc.edu

Alan Burns
Real-Time Systems Research Group, Department of Computer Science, University of York, York, UK.
Email: alan.burns@cs.york.ac.uk

Keywords: Real-time Uniprocessor Fixed priority pre-emptive scheduling Earliest Deadline First (EDF) Sub-optimality Processor speedup factor Constrained deadline Implicit deadline Omega constant

Abstract

This paper examines the relative effectiveness of fixed priority pre-emptive scheduling in a uniprocessor system, compared to an optimal algorithm such as Earliest Deadline First (EDF).

The quantitative metric used in this comparison is the processor speedup factor, equivalent to the factor by which processor speed needs to increase to ensure that any taskset that is schedulable according to an optimal scheduling algorithm can be scheduled using fixed priority pre-emptive scheduling.

For constrained-deadline tasksets where all task deadlines are less than or equal to their periods, the maximum value for the processor speedup factor is shown to be $1/\Omega \approx 1.76322$, (where Ω *is the mathematical constant defined by the transcendental equation* $\ln(1/\Omega) = \Omega$, *hence,* $\Omega \approx 0.567143$). *Further, for implicit-deadline tasksets where all task deadlines are equal to their periods, the maximum value for the processor speedup factor is shown to be* $1/\ln(2) \approx 1.44270$. *The derivation of this latter result provides an alternative proof of the well-known Liu and Layland result.*

1. Introduction

In this paper, we are interested in determining the largest factor by which the processing speed of a uniprocessor would need to be increased, such that any taskset, that was previously schedulable according to an optimal scheduling algorithm, could be guaranteed to be schedulable according to fixed priority pre-emptive scheduling, assuming an optimal priority assignment policy. We refer to this resource augmentation factor as the *processor speedup factor* (Kalyanasundaram and Pruhs, 1995).

Analysis of fixed priority pre-emptive scheduling effectively began with Fineberg and Serlin (1967) who considered priority assignment for two independent periodic tasks with deadlines equal to their periods and bounded execution times. They noted that if the task with the shorter period is assigned the higher priority, then the taskset is guaranteed to be schedulable provided that its total utilisation[1] $U \leq 2(\sqrt{2}-1) \approx 82.8\%$.

The above result was generalised by Liu and Layland

[1] The utilisation of a task is defined as its execution time divided by its period. The utilisation of a taskset is the sum of the utilisations of its tasks.

(1973) who considered the pre-emptive scheduling of synchronous[2] 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 (1973) showed that *rate monotonic* (RM) priority ordering is the optimal fixed priority assignment policy for implicit-deadline tasksets, and that using rate monotonic priority ordering, fixed priority pre-emptive scheduling can schedule any implicit-deadline taskset with a total utilisation $U \leq \ln(2) \approx 69.3\%$.

Liu and Layland (1973) also showed that Earliest Deadline First (EDF) is an optimal dynamic priority scheduling algorithm for implicit-deadline tasksets, and that EDF can schedule any such taskset with a total utilisation $U \leq 1$.

Dertouzos (1974) showed that EDF is in fact an optimal pre-emptive uniprocessor scheduling algorithm, in the sense that if a schedule exists for a taskset, then the schedule produced by EDF will also be feasible.

Combining the result of Dertouzos (1974) with the results of Liu and Layland (1973) for both EDF and fixed priority pre-emptive scheduling, we can see that the processor speedup factor required for guaranteed that fixed priority pre-emptive scheduling can schedule any feasible implicit-deadline taskset is $1/\ln(2) \approx 1.44270$.

In the 1980's, and early 1990's research into real-time scheduling focussed on *constrained-deadline* tasksets; synchronous tasksets comprising independent sporadic tasks with bounded execution times, known minimal inter-arrival times or periods, and deadlines constrained to be less than or equal to their periods.

Leung and Whitehead (1982) showed that *deadline monotonic*[3] (DM) priority ordering is the optimal fixed priority ordering for constrained-deadline tasksets. Exact fixed priority schedulability tests for constrained-deadline tasksets were introduced by Joseph and Pandya (1986), Lehoczky et al. (1989), and Audsley et al. (1993). Exact EDF schedulability tests for constrained-deadline tasksets were introduced by Baruah et al. (1990a, 1990b).

Recently, Baruah and Burns (2008) showed that the processor speedup factor required for fixed priority pre-emptive scheduling of constrained-deadline tasksets is upper-bounded by 2 and lower-bounded by 1.5.

In this paper, we prove that the exact processor speedup factor required for fixed priority pre-emptive scheduling of constrained-deadline tasksets is $1/\Omega \approx 1.76322$ (where Ω is the mathematical constant defined by the transcendental equation $\ln(1/\Omega) = \Omega$, hence, $\Omega \approx 0.567143$).

The significance of our main result is to provide a bound, analogous to the seminal schedulability result of Liu and Layland (1973) ($U \leq \ln(2) \approx 69.3\%$), that applies to constrained-deadline rather than implicit-deadline tasksets.

An exact condition for the schedulability of a constrained-deadline taskset under an optimal pre-emptive uniprocessor scheduling algorithm, such as EDF (Dertouzos, 1974), is that a quantity referred to as the processor LOAD (see Section 2.4) does not exceed the capacity of the processor (i.e. $\text{LOAD} \leq 1$) (Baruah et al. 1990a, 1990b).

The processor speedup factor derived in this paper shows that every constrained-deadline taskset with $\text{LOAD} \leq \Omega \approx 0.567143$ is guaranteed to be schedulable according to fixed priority pre-emptive scheduling using deadline-monotonic priority assignment.

While the results presented in this paper are mainly theoretical, they also have practical utility in enabling system designers to quantify the maximum penalty for using fixed priority 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.

1.1. Related work on average case sub-optimality

This paper examines the sub-optimality of fixed priority pre-emptive scheduling in the worst-case, other research has examined its behaviour in the average-case.

Lehoczky et al. (1989) introduced the *breakdown utilisation* metric: A taskset is randomly generated, and then all task execution times are scaled until a deadline is just missed. The utilisation of the scaled taskset gives the breakdown utilisation. Lehoczky et al. (1989) showed that the average breakdown utilisation, for implicit-deadline tasksets of large cardinality under fixed priority pre-emptive scheduling is approximately 88%, corresponding to a penalty of approximately 12% of processing capacity with respect to an optimal algorithm such as EDF.

Bini and Buttazzo (2005) showed that breakdown utilisation suffers from a bias which tends to penalise fixed priority scheduling by favouring tasksets where the utilisation of individual tasks is similar. Bini and Buttazzo (2005) introduced the *optimality degree* metric, defined as the number of tasksets in a given domain that are schedulable according to some algorithm *A* divided by the number that are schedulable according to an optimal algorithm. Using this metric, they showed that the penalty for using fixed priority-pre-emptive scheduling for implicit-deadline tasksets is typically

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

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

significantly lower than that assumed by determining the average breakdown utilisation.

1.2. Organisation

The remainder of this paper is organised as follows. Section 2 describes the system model and notation used, recapitulates exact schedulability analysis for both fixed priority and EDF scheduling, and provides a number of key definitions. Section 3 derives the structure and parameters of a speedup-optimal taskset (defined in Section 2) for the class of tasksets with constrained deadlines. Section 4 derives the exact processor speedup factor required for constrained-deadline tasksets of arbitrary cardinality, under fixed priority pre-emptive scheduling. Appendix A complements Section 4 by providing an upper bound on the processor speedup factor for tasksets of cardinality *n* which improves upon the general result for arbitrary *n*. Section 5 extends the results of Sections 3 and 4 to implicit-deadline tasksets, providing an alternative proof of the seminal results of Fineberg and Serlin (1967) and Liu and Layland (1973). Finally, Section 6 concludes with a summary of the results.

2. Scheduling model and schedulability analysis

In this section, we outline the scheduling model, notation and terminology used in the rest of the paper. We then recapitulate the exact schedulability analysis for both fixed priority pre-emptive scheduling and EDF scheduling. Finally, we provide a number of definitions that are used in subsequent analysis and illustrate the fundamental concepts with an example.

2.1. Scheduling model, terminology and notation

In this paper, we consider the pre-emptive scheduling of a set of 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 pre-emptive 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, 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 a *constrained-deadline* taskset, all tasks have $D_i \le T_i$, while in an *implicit-deadline* taskset, all tasks have $D_i = T_i$.

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 utilisation of all of its tasks.

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

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

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

A taskset is said to be *schedulable* with respect to some scheduling algorithm and some system, if anu sequence of invocations generated by the taskset can be scheduled on the system by the scheduling algorithm without any deadlines being missed.

Under Earliest Deadline First (EDF) scheduling, at any given time, the ready task (invocation) with the earliest absolute deadline is executed by the processor. In contrast, under fixed priority pre-emptive scheduling, at any given time, the highest priority ready task is executed by the processor.

We assume that when a taskset is scheduled according to fixed priorities, these priorities are assigned according to deadline-monotonic priority ordering, as deadline-monotonic is known to be the optimal priority ordering for both constrained-deadline (Leung and Whitehead, 1982) and implicit-deadline tasksets (Liu and Layland, 1973).

We note that deadline-monotonic is the optimal priority ordering in the sense that there are no constrained-deadline tasksets that are schedulable according to fixed priority pre-emptive scheduling using any other priority ordering that are not also schedulable using deadline-monotonic priority ordering. This does not however mean that deadline-monotonic fixed priority pre-emptive scheduling is an optimal scheduling algorithm for such tasksets. (The definition of scheduling algorithm optimality is given below).

2.2. Feasibility and optimality

A taskset is said to be *feasible* with respect to a given system model if there exists some scheduling algorithm that can schedule the taskset on that system without missing any deadlines. Note, in this paper, we are primarily interested in a reference system model that consists of a pre-emptive uniprocessor with unit

processing speed.

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 is known to be an optimal pre-emptive uniprocessor scheduling algorithm for constrained-deadline tasksets compliant with the tasking model described in Section 2.1 (Dertouzos, 1974). Least Laxity First is another such optimal algorithm (Mok, 1983).

2.3. Exact schedulability analysis for FPPS

In this section, we give a brief summary of Response Time Analysis (Audsley et al., 1993) used to provide an exact schedulability test for fixed priority pre-emptive scheduling on a uniprocessor. First, we introduce the concepts of worst-case response time, critical instant, and busy periods, which are fundamental to this form of analysis.

For a given taskset scheduled under fixed priority pre-emptive scheduling, the *worst-case response time* R_i of 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 $\forall i \quad R_i \le D_i$.

A *critical instant* for task τ_i, refers to a time at which task τ_i is released, and the pattern of releases of other tasks in the taskset is such that task τ_i exhibits its worst-case response time (Liu and Layland, 1973). Under fixed priority pre-emptive scheduling, for independent tasks with constrained-deadlines, a critical instant occurs when all of the tasks are released simultaneously, and then subsequent task releases occur as early as possible.

The term *priority level-i busy period* refers to a period of time $[t_1, t_2)$ during which the processor is busy executing computation at priority *i* or higher, that was released at the start of the busy period at t_1, or during the busy period but strictly before its end at t_2.

Under fixed priority pre-emptive scheduling, the worst-case response time R_i of a constrained-deadline task τ_i corresponds to the length of the longest priority level-*i busy period*, which starts at a critical instant.

The busy period comprises two components, the execution time C_i of the task itself, and so called *interference*, equal to the time for which task τ_i is prevented from executing by higher priority tasks. The length of the busy period w_i, can be computed using the following fixed point iteration (Audsley et al., 1993), with the summation term giving the interference due to the set of higher priority tasks *hp*(*i*).

$$w_i^{m+1} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i^m}{T_j} \right\rceil C_j \qquad (1)$$

Iteration starts with an initial value w_i^0, typically $w_i^0 = C_i$, and ends when either $w_i^{m+1} = w_i^m$ in which case the worst-case response time R_i, is given by w_i^{m+1}, or when $w_i^{m+1} > D_i$ in which case the task is unschedulable. The fixed point iteration is guaranteed to converge provided that the overall taskset utilisation is less than or equal to 1.

Response Time Analysis, as embodied in Equation (1), thus provides an exact schedulability test for constrained-deadline tasksets scheduled under fixed priority pre-emptive scheduling.

In subsequent analysis, we also make use of the concept of a *priority level-i idle period.* This is defined as an interval of time $[t_3, t_4)$ of length greater than zero, during which no tasks are ready to execute at priority *i* or higher strictly before the end of the idle period at t_4.

2.4. Exact schedulability analysis for EDF

The schedulability of a constrained-deadline taskset under EDF can be determined via the processor demand bound function *h*(*t*) given below:

$$h(t) = \sum_{i=1}^{n} \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right) C_i \qquad (2)$$

Baruah et al (1990a, 1990b) showed that a taskset is schedulable under EDF if and only if a quantity referred to as the processor LOAD is ≤ 1 where the processor LOAD is defined as follows:

$$\text{LOAD} = \max_{\forall t} \left(\frac{h(t)}{t} \right) \qquad (3)$$

Further, they showed that the maximum value of $h(t)/t$ occurs for some value of *t* in the interval $(0, L)$, where *L* is defined as follows, thus limiting the number of values of *t* that need to be checked to determine schedulability.

$$L = \max\left(D_1, D_2, \ldots D_n, \max_{\forall i} \left((T_i - D_i) \frac{U}{1-U} \right) \right)$$

Significant developments have been made, extending the scope of the schedulability tests given in Equations (1) and (2); however, these basic forms are sufficient for the purposes of this paper.

2.5. Definitions

Definition 1: Let Ψ be a taskset that is feasible (i.e. schedulable according to an optimal scheduling algorithm) on a processor of speed 1. Now assume that $f(\Psi)$ is the lowest speed of any similar processor that will schedule taskset Ψ using scheduling algorithm *A*. The *processor speedup factor* f^A for scheduling algorithm *A* is given by the maximum processor speed required to schedule any such taskset Ψ.

$$f^A = \max_{\forall \Psi} \left(f(\Psi) \right)$$

By definition, $f^A \geq 1$, with smaller values 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 fixed priority pre-emptive scheduling using an optimal priority assignment policy.

In our analysis rather than consider the problem of determining the processor speedup factor from the perspective of speeding up a processor to guarantee that a taskset that is already schedulable under an optimal algorithm becomes schedulable under fixed priority pre-emptive scheduling. Rather, we consider the equivalent problem of slowing down a processor and hence scaling up the execution times of a set of tasks that are only just schedulable according to fixed priority pre-emptive scheduling, until they are only just schedulable according to an optimal algorithm.

For a taskset *S* schedulable according to fixed priorities, let α_S ($\alpha_S \geq 1$) be the largest factor by which all of the execution times of the tasks in *S* can be scaled and the taskset remain schedulable under fixed priority pre-emptive scheduling. We refer to α_S as the *fixed priority scaling factor*. Similarly, let f_S ($f_S \geq 1$) be the largest factor by which all of the execution times of the tasks in *S* can be scaled and the taskset remain schedulable under EDF scheduling. We refer to f_S as the *EDF scaling factor*.

We now give an alternative but equivalent definition of the processor speedup factor. (In Appendix B, we prove that Definition 2 is equivalent to Definition 1).

Definition 2: The *processor speedup factor* for fixed priority pre-emptive scheduling is the maximum scaling factor by which the execution times of a set of tasks, that are only just schedulable according to fixed priority pre-emptive scheduling (i.e. with $\alpha_S = 1$), can be increased, and yet the taskset remain schedulable according to an optimal scheduling algorithm (e.g. EDF).

Corollary 1: The processor speedup factor for fixed priority pre-emptive scheduling is equal to the largest EDF scaling factor f_S of any taskset with a fixed priority scaling factor $\alpha_S = 1$.

Definition 3: A taskset is said to be *speedup-optimal* if it has the largest EDF scaling factor f_S of any taskset that is only just schedulable according to fixed priority pre-emptive scheduling ($\alpha_S = 1$).

Corollary 2: The processor speedup factor is equal to the EDF scaling factor f_S of a speedup-optimal taskset.

We note that the value of the *processor speedup factor* and the parameters of *speedup-optimal* tasksets depend on the class of tasksets considered. For example the class of implicit-deadline tasksets of cardinality two has a smaller processor speedup factor than the class of constrained-deadline tasksets with arbitrary cardinality, and different speedup-optimal tasksets. In the remainder of the paper, when the terms *processor speedup factor* and *speedup-optimal taskset* are used, the class of tasksets considered is explicitly stated only when this is not readily apparent from the context.

Definition 4: A *constraining task* is defined as a task that cannot have its execution time increased without missing its deadline, and hence the taskset becoming unschedulable.

Corollary 3: Under fixed priority pre-emptive scheduling, for a constraining task τ_i, the interval $[0, D_i)$, starting with a critical instant at *t*=0, where all tasks are released simultaneously and then subsequently released as early as possible, contains no priority level-*i* idle time.

Corollary 4: Any taskset *S* with $\alpha_S = 1$, has at least one constraining task.

2.6. Example

The concepts introduced in this section can be illustrated by means of an example. Consider an implicit-deadline taskset *S* comprising the two tasks defined in Table 1. The parameters of these tasks appear to have some rather unusual values; however, this is because they have been chosen so that the taskset is speedup-optimal with respect to the class of implicit-deadline tasksets with cardinality two. The total utilisation of the taskset is $2(\sqrt{2}-1) \approx 0.828427$.

Table 1

Task	C_i	$D_i = T_i$	U_i	R_i
τ_1	1	$1+\sqrt{2}$	$\sqrt{2}-1$	1
τ_2	$\sqrt{2}$	$2+\sqrt{2}$	$\sqrt{2}-1$	$1+\sqrt{2}$

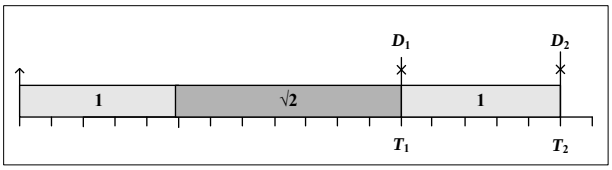

Figure 1

Figure 1 illustrates the execution of the tasks under fixed priority pre-emptive scheduling, starting at a critical instant. Note that although the worst-case response time of task τ_2 is significantly less than its

deadline D_2, τ_2 is a constraining task; since any increase in its execution time will cause its deadline to be missed.

Taskset *S* has a fixed priority scaling factor $\alpha_S = 1$, as increasing the execution times of the tasks by any factor greater than 1 would result in the taskset becoming unschedulable.

As EDF can schedule any implicit deadline taskset with utilisation no greater than 100%, we can scale the execution times of tasks τ_1 and τ_2 by a factor of $1/(2(\sqrt{2}-1))$, and the resulting taskset, with 100% utilisation, will be just schedulable under EDF. The EDF scaling factor f_S for taskset *S* is therefore $1/(2(\sqrt{2}-1)) \approx 1.207107$.

We note that the parameters of the tasks in taskset *S* have been carefully selected so that the utilisation of this taskset matches the Fineberg and Serlin (1967) utilisation bound of $2(\sqrt{2}-1)$ for implicit-deadline tasksets of cardinality two. As all tasksets with cardinality two and utilisation less than or equal to this bound are known to be schedulable, any taskset with utilisation strictly less than the bound must have a fixed priority scaling factor that is strictly greater than 1. Hence taskset *S* has the minimum utilisation of any taskset that is only just schedulable according to fixed priority pre-emptive scheduling ($\alpha_S = 1$). Taskset *S* therefore exhibits the largest EDF scaling factor f_S of any taskset that has $\alpha_S = 1$, and hence is a speedup-optimal taskset.

The processor speedup factor is equal to the EDF scaling factor f_S of a speedup-optimal taskset, hence the processor speedup factor for fixed priority pre-emptive scheduling of implicit deadline tasksets of cardinality two is $1/(2(\sqrt{2}-1)) \approx 1.207107$.

Note, in Section 5 we prove this result independently without using the utilisation bound of Fineberg and Serlin (1967).

3. Speedup-optimal tasksets

In this section, we derive the structure and parameters of speedup-optimal tasksets for the class of tasksets with constrained deadlines ($D_i \le T_i$).

Before considering tasksets of arbitrary cardinality, we first present results for tasksets comprising just two tasks. The derivation of this result provides the intuition for the general case.

Theorem 1 describes the parameters of a taskset that is speedup-optimal with respect to all constrained-deadline tasksets of cardinality two.

Theorem 2 describes the parameters of a taskset that is speedup-optimal with respect to all constrained-deadline tasksets.

The proofs of Theorems 1 and 2 rely on Lemmas 1-8. The basic method and intuition behind the proofs of Lemmas 1-8 is given after the Theorems, this followed by the Lemmas and their specific proofs.

Note that in the various discussions, theorems, lemmas, and proofs in this and subsequent sections, fixed priority pre-emptive scheduling should be assumed unless otherwise stated.

Theorem 1: For constrained-deadline tasksets of cardinality two, there is a speedup-optimal taskset *V*, with $\alpha_V = 1$, which has the following parameters:

$\tau_1 : C_1 = 1,\ D_1 = T_1 = 1 + X$

$\tau_2 : C_2 = X,\ D_2 = 2 + X,\ T_2 = \infty$

Where $X \ge 0$ is some as yet unknown value for the execution time of τ_2.

Note that the execution time of τ_1 in Theorem 1 has been normalised to 1 and the task periods and deadlines adjusted accordingly[4]. There is one free variable in the taskset parameters, that is *X*, the execution time of task τ_2. This taskset is illustrated in Figure 2.

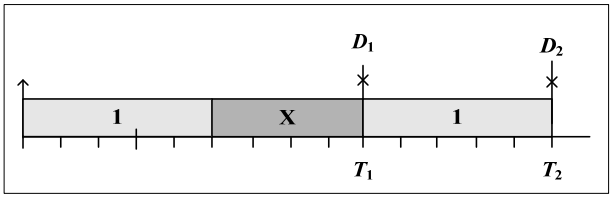

Figure 2

Proof: Proof follows directly from Lemmas 1 to 8, specifically:

- τ_2 must be a constraining task, with the longest deadline and the lowest priority (Lemma 1).
- τ_2 must have an infinite period (Lemma 2).
- $t = D_2$ must be the start of an idle period (Lemma 3).
- $T_1 < D_2$ (Lemma 4)
- $D_1 = T_1$ (Lemma 5).
- $T_1 > D_2 / 2$ (Lemma 6).
- Following a critical instant, τ_2 must execute continuously from when it first starts execution until it completes (Lemma 7).
- The parameters of task τ_1 must comply with $D_1 = T_1 = C_1 + C_2 = 1 + X$ (Lemma 8).

□

Theorem 2: For constrained-deadline tasksets with arbitrary cardinality, there is a speedup-optimal taskset *V*, with $\alpha_V = 1$, which has the following parameters:

Taskset *V* has an infinite number of tasks and is expressed as the limit of the following taskset as $n \to \infty$.

$\forall i \ne n \quad D_i = T_i = 1 + X + (i-1)/(n-1)$

[4] All of the task parameters can be scaled linearly without changing the fundamental properties of the taskset.

$$C_i = 1/(n-1)$$

$$C_n = X \ , \ D_n = 2 + X \ , \ T_n = \infty \qquad (12)$$

Where $X \geq 0$ is some as yet unknown value for the execution time of τ_n.

Note that in Theorem 2, the total higher priority task execution time has again been normalised to 1 and the task periods and deadlines adjusted accordingly. There is one free variable in the taskset parameters, that is *X*, the execution time of task τ_n. This taskset is illustrated in Figure 3.

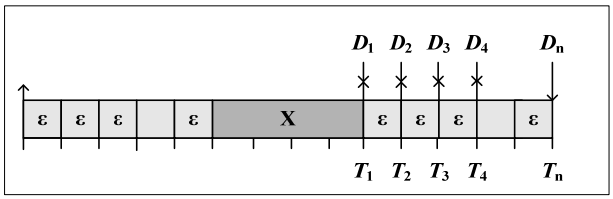

Figure 3

Proof: Proof follows directly from Lemmas 1 to 9, specifically:

- τ_n must be a constraining task, with the longest deadline and the lowest priority (Lemma 1).
- τ_n must have an infinite period (Lemma 2).
- $t = D_n$ must be the start of an idle period (Lemma 3).
- $\forall i \neq n \quad T_i < D_n$ (Lemma 4)
- $\forall i \neq n \quad D_i = T_i$ (Lemma 5).
- $\forall i \neq n \quad T_i > D_n / 2$ (Lemma 6).
- Following a critical instant, τ_n must execute continuously from when it first starts execution until it completes (Lemma 7).
- The task parameters must comply with the following equation (Lemma 8):

$$\forall i \neq n \quad D_i = T_i = \sum_{\forall j} C_j + \sum_{\forall j \in hp(i)} C_j$$

Finally, applying Lemma 9 repeatedly shows that slicing the total amount of higher priority task execution time into an infinite number of tasks each with an infinitesimal execution time leads to a speedup-optimal taskset[5] □

3.1. Method and intuition

The problem of determining the parameters of a speedup-optimal taskset for the class of tasksets with constrained-deadlines is solved by breaking it down into a series of basic steps corresponding to Lemmas 1 to 8.

At each step, we start with a set *Z* of tasksets, where *Z* is known to contain at least one speedup-optimal taskset. We then place a condition on the task parameters which selects a subset *Y* of the tasksets in *Z* ($Y \subseteq Z$). We then prove that for every taskset *S* that is in *Z* but is not a member of *Y* ($S \in Z \cap Y^c$), there is a taskset $V \in Y$ that has an EDF scaling factor f_V at least as large as that of taskset *S* ($f_V \geq f_S$). Hence we show that the reduced set *Y* also contains at least one speedup-optimal taskset.

The majority of our proofs are by contradiction. They work in the following way: First we assume (for contradiction) that there is a taskset $S \in Z \cap Y^c$ with an EDF scaling factor strictly larger than that of any taskset in *Y*. We then show that this cannot be the case, by transforming taskset *S* into another taskset $V \in Y$ which we show has an EDF scaling factor at least as large as that of *S* ($f_V \geq f_S$). This contradicts the original assumption. It follows that there are no tasksets in $Z \cap Y^c$ that have an EDF scaling factor strictly greater than the maximum EDF scaling factor of any task in *Y*. As Z was known to contain at least one speedup-optimal taskset (with the maximum EDF scaling factor), then it follows that there must be at least one taskset in Y that has the maximum EDF scaling factor, and so is a speedup-optimal taskset, which proves the Lemma.

Lemma 1 starts with *Z* representing the set of all tasksets that are just schedulable according to fixed priority pre-emptive scheduling. By definition, this set contains at least one speedup-optimal taskset. Once Lemma 1 is proven, we know that the subset $Y \subseteq Z$ contains at least one speedup-optimal taskset.

Each subsequent lemma starts by setting *Z* equal to the subset *Y* defined by the previous lemma; hence multiple conditions are applied resulting in a constrained subset that is known to contain at least one speedup-optimal taskset. At the end of Lemmas 1-8, this allows for just one free variable *X* among the taskset parameters, aside from the number of tasks which is addressed by Lemma 9. By virtue of Lemmas 1-9, Theorems 1 and 2 define speedup-optimal tasksets sufficiently well for subsequent analysis of the exact processor speedup factor in Section 4

In a number of the lemmas, we make use of the concept of a *sustainable change*. A change to the parameters of the tasks in a schedulable taskset is said to be *sustainable* (Baruah and Burns, 2006) if following that change the taskset is guaranteed to remain schedulable. Baruah and Burns (2006) showed that under both EDF and fixed priority pre-emptive scheduling, increases in task periods, increases in task deadlines, and decreases in task execution times are all sustainable changes.

[5] Strictly speaking, a speedup optimal taskset does not exist for the case of tasksets with arbitrary cardinality, in the same way that a largest integer does not exist for the set of integers. The concept of a speedup optimal taskset is however extremely useful in understanding processor speedup factors. In Theorem 2 we therefore express the speedup optimal taskset in terms of the limit of a taskset as its cardinality approaches infinity, and refer to it in subsequent text as if it does exist. We note that in practice, all real-time systems have a finite number of tasks, and speedup optimal tasksets exists for any class of tasksets with cardinality limited by a finite value.

3.2. Lemmas 1-9

The following lemmas are applicable to the class of tasksets with constrained deadlines.

Lemma 1: Let *Z* be the set of all (constrained-deadline) tasksets that are just schedulable according to fixed priority pre-emptive scheduling ($\alpha_S = 1$). By definition, this set contains at least one taskset that is speedup-optimal with respect to the class of tasksets with constrained deadlines. Let $Y \subseteq Z$ such that every taskset in *Y* has a single constraining task, and that task has the lowest priority, and hence the longest deadline. The set *Y* contains at least one speedup-optimal taskset.

Proof: We assume (for contradiction) that there is a taskset $S \in Z \cap Y^c$ that has an EDF scaling factor f_S strictly greater than that of any taskset in *Y*.

By Corollary 4, taskset *S* contains at least one constraining task. Let task τ_i $(i \neq n)$ be the highest priority constraining task in S. (Note, τ_n cannot be the highest priority constraining task in *S*, otherwise *S* would be a member of *Y*). We now create a new taskset *V* by removing all tasks of lower priority than *i* from *S*. As the lowest priority task in *V* is a constraining task, $\alpha_V = 1$, and hence $V \in Y$. Further, the tasks in *V* are a subset of the tasks in *S*. Removing a task is equivalent to decreasing its execution time to zero, and decreasing the execution time of any task is a sustainable change under both fixed priority and EDF scheduling. Taskset *V*, scaled by a factor of f_S is therefore schedulable under EDF, hence $f_V \geq f_S$. This contradicts our original assumption, hence there are no tasksets in $Z \cap Y^c$ that have an EDF scaling factor strictly greater than the maximum EDF scaling factor of any taskset in *Y*, and so there must be at least one speedup-optimal taskset in Y
□

Lemma 2: Let *Z* be the set *Y* defined by Lemma 1, and *Y* be redefined as follows: $Y \subseteq Z$ such that every taskset in *Y* has a lowest priority task τ_n with an infinite period. The set *Y* contains at least one speedup-optimal taskset.

Proof: We assume (for contradiction) that there is a taskset $S \in Z \cap Y^c$ that has an EDF scaling factor f_S strictly greater than that of any taskset in *Y*.

We now create a new taskset *V* from taskset *S*, by increasing the period of task τ_n to infinity. Taskset *V* has $\alpha_V = 1$, as schedulability of τ_n under fixed priority pre-emptive scheduling is independent of its period T_n provided that $D_n \leq T_n$ (see Equation (1)), hence $V \in Y$.

Increasing the period of any task is a sustainable change under fixed priority and EDF scheduling and so taskset *V*, scaled by a factor of f_S is schedulable under EDF, hence $f_V \geq f_S$. This contradicts our original assumption, hence there are no tasksets in $Z \cap Y^c$ that have an EDF scaling factor strictly greater than the maximum EDF scaling factor of any taskset in *Y*, and so there must be at least one speedup-optimal taskset in Y
□

Lemma 3: Let *Z* be the set *Y* defined by Lemma 2, and *Y* be redefined as follows: $Y \subseteq Z$ such that every taskset in *Y* has a priority level-(*n*-1) idle period starting at time $t = D_n$, following a critical instant at time *t* = 0, when all of the tasks are released simultaneously, and are then released again as early as possible. Stated otherwise, all of the task execution released in the interval $[0, D_n)$ is completed by D_n, and no tasks of priority higher than *n* are released at time D_n. The set *Y* contains at least one speedup-optimal taskset.

Proof: We assume (for contradiction) that there is a taskset $S \in Z \cap Y^c$ that has an EDF scaling factor f_S strictly greater than that of any taskset in *Y*.

For taskset *S*, with a critical instant at time *t* = 0, time $t = D_n$ is not the start of a priority level-(*n*-1) idle period, otherwise *S* would be a member of *Y*. Let the next such idle period start at some later time $t = D'_n > D_n$. We now create a new taskset *V* from taskset *S* by increasing the deadline of τ_n to D'_n. We refer to the modified task as τ'_n. As τ'_n is a constraining task, taskset *V* has $\alpha_V = 1$ and hence $V \in Y$.

Increasing the deadline of any task is a sustainable change under fixed priority and EDF scheduling and so taskset *V*, scaled by a factor of f_S is schedulable under EDF, hence $f_V \geq f_S$. This contradicts our original assumption, hence there are no tasksets in $Z \cap Y^c$ that have an EDF scaling factor strictly greater than the maximum EDF scaling factor of any taskset in *Y*, and so there must be at least one speedup-optimal taskset in Y
□

Lemma 4: Let *Z* be the set *Y* defined by Lemma 3, and *Y* be redefined as follows: $Y \subseteq Z$ such that every taskset in *Y* has task periods such that $\forall i \neq n \quad T_i < D_n$. The set *Y* contains at least one speedup-optimal taskset.

Proof: We assume (for contradiction) that there is a taskset $S \in Z \cap Y^c$ that has an EDF scaling factor f_S strictly greater than that of any taskset in *Y*.

Let τ_i be a task in taskset *S* that has $T_i \geq D_n$. We now create a new taskset *V* from taskset *S* by removing each such task τ_i with $T_i \geq D_n$, and increasing the execution time of τ_n by C_i (to form task τ'_n). As there was only one invocation of each task τ_i in the interval $[0, D_n)$, the same amount of computation remains in this interval, hence τ'_n is a constraining task and $\alpha_V = 1$, hence $V \in Y$.

As τ'_n has an infinite period (due to the constraints placed on the tasksets in set *Z* by Lemma 2) and a

deadline not less than that of τ_i (due to the constraints placed on the tasksets in the set *Z* by Lemma 1), then the processor demand function $h(t)$ for taskset *V* is never larger than that for taskset *S*. Taskset *V* scaled by a factor of f_S is therefore schedulable under EDF, hence $f_V \geq f_S$. This contradicts our original assumption, hence there are no tasksets in $Z \cap Y^c$ that have an EDF scaling factor strictly greater than the maximum EDF scaling factor of any taskset in *Y*, and so there must be at least one speedup-optimal taskset in Y □

Lemma 5: Let *Z* be the set *Y* defined by Lemma 4, and *Y* be redefined as follows: $Y \subseteq Z$ such that every taskset in *Y* has task periods and deadlines such that $\forall i \neq n \quad D_i = T_i$. The set *Y* contains at least one speedup-optimal taskset.

Proof: We assume (for contradiction) that there is a taskset $S \in Z \cap Y^c$ that has an EDF scaling factor f_S strictly greater than that of any taskset in *Y*.

Let τ_i be a task in taskset *S* that has $D_i < T_i$. We now create a new taskset *V* from taskset *S* by increasing the deadline of each such task τ_i, to form task τ_i' with $D_i' = T_i$. The total execution time in $[0, D_n)$ remains the same, and so τ_n remains a constraining task and so $\alpha_V = 1$, hence $V \in Y$.

Increasing the deadline of any task is a sustainable change under fixed priority and EDF scheduling and so taskset *V*, scaled by a factor of f_S is schedulable under EDF, hence $f_V \geq f_S$. This contradicts our original assumption, hence there are no tasksets in $Z \cap Y^c$ that have an EDF scaling factor strictly greater than the maximum EDF scaling factor of any taskset in *Y*, and so there must be at least one speedup-optimal taskset in Y □

We note that the transformation detailed in the proof of Lemma 5 may result in changes to the order of task deadlines with respect to task priority. We assume that if this is the case, then the task priorities are altered so that they are once again in deadline monotonic priority order. We note that this does not affect taskset schedulability as deadline monotonic priority ordering is known to be optimal (Leung and Whitehead, 1982), and the taskset remains schedulable with its original priority ordering. Further, Lemma 4, shows that $\forall i \neq n \quad T_i < D_n$, hence after the above transformation, $\forall i \neq n \quad D_i < D_n$, so task τ_n remains the lowest priority and constraining task.

Lemma 6: Let *Z* be the set *Y* defined by Lemma 5, and *Y* be redefined as follows: $Y \subseteq Z$ such that every taskset in *Y* has task periods such that $\forall i \neq n \quad T_i > D_n / 2$. The set *Y* contains at least one speedup-optimal taskset.

Proof: We assume (for contradiction) that there is a taskset $S \in Z \cap Y^c$ that has an EDF scaling factor f_S strictly greater than that of any taskset in *Y*.

Let τ_i be a task in *S* that has $T_i < D_n / 2$. We now create a new taskset *V* from taskset *S* by transforming the parameters of each such task τ_i (to form task τ_i') as follows. $C_i' = mC_i$, $D_i' = T_i' = mT_i$ where $m = \lfloor D_n / T_i \rfloor$. As $T_i < D_n / 2$ we have $D_n \geq T_i' \geq D_n / 2$.

We note that this transformation may result in changes to the order of task deadlines with respect to task priority. We assume that if this is the case, then the task priorities are altered so that the tasks are once again in deadline monotonic priority order. Note that τ_n remains the lowest priority task.

Following the above transformation, the amount of execution time released by τ_i' in the interval $[0, D_n)$ cannot be less than that released by τ_i, hence taskset *V* has $\alpha_V \leq 1$. (We consider the fact that taskset *V* may now be unschedulable according to fixed priority scheduling (i.e. $\alpha_V < 1$) later in the proof).

Considering EDF scheduling, the contribution to the processor demand function from each original task τ_i (with $D_i = T_i$) is given by:

$$h_i(t) = \left\lfloor \frac{t}{T_i} \right\rfloor C_i \tag{4}$$

Similarly, the contribution to the processor demand function from each transformed task τ_i' is given by:

$$h_i'(t) = \left\lfloor \frac{t}{mT_i} \right\rfloor mC_i \tag{5}$$

Equations (4) and (5) are both monotonically non-decreasing functions of *t*.

$h_i(t)$ is zero for $t < T_i$, and only increases in value at times $t = gT_i$, for integer values of *g*. At these times, $h_i(t) = gC_i$.

$h_i'(t)$ is zero for $t < mT_i$, and only increases in value at times $t = kmT_i$, for integer values of *k*. At these times, $h_i'(t) = kmC_i$.

Now substituting the times at which Equation (5) increases into Equation (4), we find that at these times, $h_i(t) = kmC_i$, hence $\forall t \quad h_i(t) \geq h_i'(t)$.

As the processor demand function $h(t)$ for taskset *V* is never larger than that for taskset *S*, taskset *V*, scaled by a factor of f_S is schedulable under EDF, hence $f_V \geq f_S$.

We now further transform taskset *V* ensuring that it is just schedulable according to fixed priority pre-emptive scheduling, and a member of the set *Y*. We achieve this by applying the following steps 1-6 repeatedly until the parameters of the taskset cease to change on step 6:

1. Reduce all task execution times by the same scaling factor until taskset *V* is just schedulable according to fixed priority pre-emptive

scheduling. We now have $\alpha_V = 1$. Reducing task execution times is a sustainable change, and so cannot decrease the taskset's EDF scaling factor f_V.

2. Remove all tasks of lower priority than the highest priority constraining task (Lemma 1).
3. Give the lowest priority task τ_n, the longest possible (e.g. infinite) period (Lemma 2).
4. Increase the deadline of task τ_n until there is a priority level-(*n-1*) idle period starting at D_n (Lemma 3).
5. Remove any task τ_i with $T_i \geq D_n$, and add its execution time to that of the lowest priority task (Lemma 4).
6. Transform the parameters of any task τ_i with $T_i < D_n / 2$, as described in the 2nd paragraph of this proof, and re-assign task priorities in deadline monotonic priority order.

Note that step 6 can only change the taskset parameters if a task was removed in step 2, and so the number of times that the sequence of six steps can repeat is limited by the cardinality of the original taskset.

Once the repeated transformation is complete, then taskset *V* complies with the constraints imposed by Lemmas 1-6 and has $\alpha_V = 1$, hence $V \in Y$.

As none of the above six steps can decrease the EDF scaling factor of taskset *V*, we have $f_V \geq f_S$. This contradicts our original assumption, hence there are no tasksets in $Z \cap Y^c$ that have an EDF scaling factor strictly greater than the maximum EDF scaling factor of any taskset in *Y*, and so there must be at least one speedup-optimal taskset in Y □

Corollary 3: There is a speedup-optimal taskset where, following a critical instant, all tasks with priorities greater than *n* execute exactly twice in the interval $[0, D_n)$. This follows directly from the fact that all tasksets in the set *Y* defined by Lemma 6 have this property.

Lemma 7: Let *Z* be the set *Y* defined by Lemma 6, and *Y* be redefined as follows: $Y \subseteq Z$ such that every taskset in *Y*, has task τ_n executing continuously from when it first starts execution until it completes, without pre-emption by any higher priority task τ_i (assuming the tasks are released at a critical instant). The set *Y* contains at least one speedup-optimal taskset.

Proof: We assume (for contradiction) that there is a taskset $S \in Z \cap Y^c$ that has an EDF scaling factor f_S strictly greater than that of any taskset in *Y*.

Corollary 3 shows that any task τ_i ($i \neq n$) in taskset *S* executes exactly twice in the interval $[0, D_n)$.

We now construct a new taskset *V* from taskset *S*, initially, we make *V* a copy of *S*, then we apply the following transformation repeatedly until there are no tasks in *V* whose second invocation is released prior to the completion of τ_n.

Transformation: Let τ_i ($i \neq n$) be a task in *V* that initially pre-empts τ_n at time T_i, and that an amount of execution time $c_n \neq 0$, of task τ_n remains at this time. We increase both the period and deadline of τ_i by c_n (to form task τ_i'), hence $D_i' = T_i' = T_i + c_n$. We note that there is no idle time in the interval $[0, T_i')$ as any time in this interval that is not now taken up processing τ_i' will instead be used to execute τ_n (or another task of higher priority than *n*).

Each time the transformation is applied, the total execution time in $[0, D_n)$ remains the same, it is just re-ordered, hence τ_n remains a constraining task and α_V = 1.

Repeated application of the above transformation until there are no tasks whose second invocation is released prior to the completion of τ_n results in a taskset *V* where τ_n is not pre-empted following a critical instant, and $\alpha_V = 1$, hence $V \in Y$.

Increasing the deadline or period of any task is a sustainable change under fixed priority and EDF scheduling and so taskset *V*, scaled by a factor of f_S is schedulable under EDF, hence $f_V \geq f_S$. This contradicts our original assumption, hence there are no tasksets in $Z \cap Y^c$ that have an EDF scaling factor strictly greater than the maximum EDF scaling factor of any taskset in *Y*, and so there must be at least one speedup-optimal taskset in Y □

We note that the transformation detailed in the proof of Lemma 7 may also result in changes to the order of task deadlines with respect to task priority. Again, we assume that if this is the case, then the task priorities are altered so that they are once again in deadline monotonic priority order. We note again that this does not affect taskset schedulability as deadline monotonic priority ordering is optimal, and the taskset remains schedulable with its original priority ordering. Further, Lemma 4, shows that $\forall i \neq n \quad T_i < D_n$, hence after the above transformation, $\forall i \neq n \quad D_i < D_n$, so task τ_n remains the lowest priority and constraining task.

We further note that the transformation detailed in the proof of Lemma 7 may, in some cases, need to be applied more than once per higher priority task, before the final state with no second invocations prior to the completion of τ_n is reached.

Lemma 8: Let *Z* be the set *Y* defined by Lemma 7, and *Y* be redefined as follows: $Y \subseteq Z$ such that every taskset in *Y*, has task parameters related according to Equation (6).

$$\forall i \neq n \quad D_i = T_i = \sum_{\forall j} C_j + \sum_{\forall j \in hp(i)} C_j \qquad (6)$$

The set *Y* contains at least one speedup-optimal taskset.

Proof: We assume (for contradiction) that there is a taskset $S \in Z \cap Y^c$ that has an EDF scaling factor f_S strictly greater than that of any taskset in *Y*.

By Lemma 1, as τ_n is a constraining task, taskset *S*, has no idle time in the interval $[0, D_n)$. By Lemma 7, assuming priorities in deadline monotonic order, the period and deadline of τ_1 are equal to the completion time of task τ_n.

$$D_1 = \sum_{\forall j} C_j \tag{7}$$

For i = 2..(n-1), as there is no idle time in the interval, then task τ_i must have a period (and deadline) less than or equal to the completion time of the second invocation of τ_{i-1}. This completion time is given by:

$$\sum_{\forall j} C_j + \sum_{\forall j \in hp(i)} C_j \tag{8}$$

Let τ_i be a task in taskset *S* where the period and deadline of τ_i are less than this completion time.

We now create a new taskset *V* from taskset *S* by increasing the period and deadline of each such task τ_i, to the completion time of task τ_{i-1}. Following this transformation, the total execution time in $[0, D_n)$ remains the same, hence τ_n remains a constraining task and $\alpha_V = 1$, hence $V \in Y$.

Increasing the deadline or period of any task is a sustainable change under fixed priority and EDF scheduling and so taskset *V*, scaled by a factor of f_S is schedulable under EDF, hence $f_V \geq f_S$. This contradicts our original assumption, hence there are no tasksets in $Z \cap Y^c$ that have an EDF scaling factor strictly greater than the maximum EDF scaling factor of any taskset in *Y*, and so there must be at least one speedup-optimal taskset in Y □

Lemma 9: Let the set *Y* be as defined by Lemma 8. For a taskset $S \in Y$, splitting a task τ_i ($i \neq n$) with parameters C_i, $D_i = T_i$, into two new tasks, τ_i' and τ_i'' with parameters C_i' (where C_i' is any arbitrary non-zero value that is less than C_i), $D_i' = T_i' = D_i$ and $C_i'' = C_i - C_i'$, $D_i'' = T_i'' = D_i + C_i'$ results in a new taskset $V \in Y$ with a speedup factor at least as large as that for taskset *S* (i.e. $f_V \geq f_S$).

Proof: The execution of tasks τ_i' and τ_i'' (from taskset *V*) exactly replaces that of task τ_i (from taskset *S*) in the interval $[0, D_n)$ starting from a critical instant, hence τ_n remains a constraining task and $\alpha_V = 1$.

We prove the Lemma by showing that at any arbitrary time *t*, the processor demand bound function for taskset *V* is no greater than that for taskset *S*. The demand bound functions are identical save for the contributions from tasks τ_i, τ_i' and τ_i''. We therefore only consider the contributions from these tasks.

The contribution from tasks τ_i' and τ_i'' is given by:

$$h_i'(t) + h_i''(t) = \left(1 + \left\lfloor \frac{t - T_i}{T_i} \right\rfloor\right) C_i' + \left(1 + \left\lfloor \frac{t - (T_i + C_i')}{T_i + C_i'} \right\rfloor\right)(C_i - C') \tag{9}$$

The contribution from task τ_i is given by:

$$h_i(t) = \left(1 + \left\lfloor \frac{t - T_i}{T_i} \right\rfloor\right) C_i = \left(1 + \left\lfloor \frac{t - T_i}{T_i} \right\rfloor\right) C_i' + \left(1 + \left\lfloor \frac{t - T_i}{T_i} \right\rfloor\right)(C_i - C') \tag{10}$$

For positive values of C_i', then:

$$\forall t \quad \left\lfloor \frac{t - T_i}{T_i} \right\rfloor \geq \left\lfloor \frac{t - (T_i + C_i')}{T_i + C_i'} \right\rfloor \tag{11}$$

Hence $h_i'(t) + h_i''(t) \leq h_i(t)$ and so taskset *V* is schedulable with an EDF scaling factor at least as large as that for taskset *S* (i.e. $f_V \geq f_S$) □

4. Processor speedup factor for constrained-deadline tasksets

In this section, we derive the processor speedup factor for constrained-deadline tasksets under fixed priority pre-emptive scheduling. We do this first for tasksets of cardinality two and then for the general case of cardinality *n*.

In each case, the basic approach we use is as follows:

- First, we derive an upper bound on the maximum EDF scaling factor of the appropriate speedup-optimal taskset (defined in either Theorem 1 or Theorem 2), assuming any arbitrary value for the execution time *X*, of the lowest priority task. As part of this derivation, we determine the value of *X* that results in this upper bound.
- Second, we prove that the maximum EDF scaling factor is in fact equal to the upper bound. We do this by showing that the speedup-optimal taskset, characterised by the previously obtained value of *X*, is schedulable according to EDF when all task execution times are scaled by the upper bound. This shows that the bound is tight.
- Finally, the processor speedup factor for fixed priority pre-emptive scheduling is equal to the maximum EDF scaling factor (for any value of *X*) for the speedup-optimal taskset. So the processor speedup factor is equal to our tight

upper bound.

Theorem 3: For a constrained-deadline taskset of cardinality two, the processor speedup factor for fixed priority pre-emptive scheduling is $\sqrt{2} \approx 1.414214$.

Proof: We prove the theorem by determining the maximum EDF scaling factor for the taskset *V* described in Theorem 1, for any value of *X*.

Three constraints on the EDF scheduling of the taskset described in Theorem 1, after it is scaled by a factor *f* are:

(i) Task τ_2 with execution time *fX* must complete by its deadline at 2+*X* (subject to interference of *f* from task τ_1).

(ii) The second invocation of task τ_1 must complete by its deadline at 2*X*+2.

(iii) The total utilisation of task τ_1 must be less than or equal to 1 (The utilisation of task τ_2 is effectively zero as it has an infinite period).

Constraint (i), leads to the following equation bounding the EDF scaling factor as a function of *X*:

$$f1(X) = \frac{2+X}{1+X} \tag{13}$$

Constraint (ii), leads to the following equation, also bounding the EDF scaling factor as a function of *X*:

$$f2(X) = \frac{2X+2}{2+X} \tag{14}$$

Constraint (iii), again leads to an equation bounding the EDF scaling factor:

$$f3(X) = 1+X \tag{15}$$

As $f3(X) \geq f2(X)$ for all values of *X*, we may disregard $f3(X)$ as $f2(X)$ provides a tighter bound.

Figure 4 illustrates the three functions bounding the EDF scaling factor for two tasks.

Equation (13) is a continuous non-increasing function of *X* with a maximum value of *f*1(0) = 2. Equation (14) is a continuous non-decreasing function of *X* with a minimum value of *f*2(0) =1. Hence, the intersection of these two functions determines an upper bound on the maximum EDF scaling factor *f*. We have:

$$\frac{2+X}{1+X} = \frac{2X+2}{2+X}$$
$$(2+X)^2 = (2X+2)(1+X)$$
$$X^2 + 4X + 4 = 2X^2 + 4X + 2$$
$$X^2 = 2$$
$$X = \sqrt{2}$$
$$f2(\sqrt{2}) = \frac{2\sqrt{2}+2}{2+\sqrt{2}} = \frac{\sqrt{2}(2\sqrt{2}+2)}{2\sqrt{2}+2} = \sqrt{2} \tag{16}$$

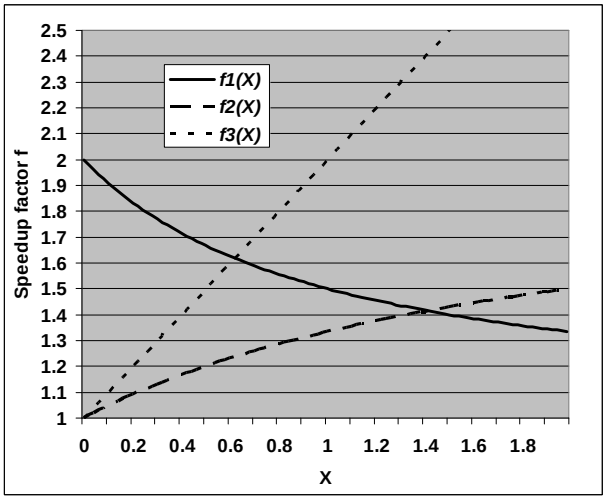

Figure 4: Constraints on the speedup factor for two tasks

To show that the maximum EDF scaling factor is equal to this upper bound, we must show that the taskset is schedulable, with scaled parameters, under EDF.

The scaled taskset parameters are as follows:

$$C_1 = \sqrt{2},\ D_1 = 1+\sqrt{2},\ T_1 = 1+\sqrt{2}$$
$$C_2 = 2,\ D_2 = 2+\sqrt{2},\ T_2 = \infty$$

The taskset is schedulable provided that $\forall t \quad h(t)/t \leq 1$ (See Equation (2)). From Equation (2), we note that the maximum value of $h(t)/t$ occurs at the deadline of some invocation of a task. It is therefore sufficient to check that $h(t)/t \leq 1$ at the deadlines of all task invocations.

For task τ_2, there is only one deadline to consider, $D_2 = 2+\sqrt{2}$, and:

$$\frac{h(2+\sqrt{2})}{2+\sqrt{2}} = \frac{C_2 + C_1}{2+\sqrt{2}} = \frac{2+\sqrt{2}}{2+\sqrt{2}} = 1 \tag{17}$$

For task τ_1, there are deadlines to consider at times $t = kD_1 = k(1+\sqrt{2})$, where k is a positive integer. For $k = 1$, we have:

$$\frac{h(1+\sqrt{2})}{1+\sqrt{2}} = \frac{C_1}{1+\sqrt{2}} = \frac{\sqrt{2}}{1+\sqrt{2}} < 1 \tag{18}$$

Further, for $k \geq 2$, we have:

$$\frac{h(k(1+\sqrt{2}))}{k(1+\sqrt{2})} = \frac{kC_1 + C_2}{k(1+\sqrt{2})} = \frac{k\sqrt{2}+2}{k(1+\sqrt{2})} \overset{k\geq 2}{\leq} \frac{k(1+\sqrt{2})}{k(1+\sqrt{2})} = 1 \tag{19}$$

Hence the taskset is schedulable. The maximum EDF scaling factor, for any value of *X*, is therefore $\sqrt{2}$.

For a constrained-deadline taskset of cardinality two, the processor speedup factor for fixed priority pre-emptive scheduling is equal to the maximum EDF scaling factor (for any value of *X*) for the speedup-optimal taskset described Theorem 1 □

Corollary 5: The maximum EDF scaling factor and hence the processor speedup factor for a constrained-deadline taskset of cardinality two is achieved for the taskset described in Theorem 1, with a value of $X = \sqrt{2}$.

Next we derive the processor speedup factor for the general case of *n* tasks. We do this by reference to the speedup-optimal taskset described in Theorem 2; however, first we prove the following Lemma.

Lemma 10: The total utilisation of the higher priority tasks τ_1 to τ_{n-1} in taskset *V* described in Theorem 2 is given by:

$$U^V = \ln\left(\frac{2+X}{1+X}\right) \tag{20}$$

Proof: Given that the execution time of each higher priority task is 1/(*n*-1) and the period of task τ_i is $(1+X+(i-1)/(n-1))$, the total utilisation of tasks τ_1 to τ_{n-1} described in Theorem 2, is given by:

$$U^V = \lim_{n-1\to\infty}\left(\frac{1}{(n-1)}\sum_{i=1}^{n-1}\frac{1}{(1+X+(i-1)/(n-1))}\right) \tag{21}$$

Substituting $k = n$-1 gives:

$$U^V = \lim_{k\to\infty}\left(\frac{1}{k}\sum_{i=1}^{k}\frac{1}{(1+X+(i-1)/k)}\right) \tag{22}$$

Equation (22) is recognisable as the Riemann sum[6] of the function $y = 1/z$ over the partition [(1+*X*), (1+X)+1]. The start of each of the *k* intervals in the Riemann sum is at $1+X+(i-1)/k$ (for i = 1 to k), and the width of each interval is 1/*k*.

The limit as $k \to \infty$ of the Riemann sum is simply the integral over the partition so:

$$U^V = \int_{1+X}^{2+X}\frac{1}{z}dz = \ln\left(\frac{2+X}{1+X}\right) \tag{23}$$

□

Theorem 4: For a constrained-deadline taskset of arbitrary cardinality, the processor speedup factor for fixed priority pre-emptive scheduling is $1/\Omega \approx 1.76322$ (Where Ω is the mathematical constant defined by the transcendental equation $\ln(1/\Omega) = \Omega$. Hence, $\Omega \approx 0.567143$).

Proof: We prove the theorem by determining the maximum EDF scaling factor for the taskset *V* described in Theorem 2, for any value of *X*.

Two constraints on the EDF scheduling of taskset *V*, after it is scaled by a factor *f* are:

(i) τ_n with execution time *fX* must complete by 2+*X* (subject also to interference in total of *f* from tasks τ_1 to τ_{n-1}).

(ii) The total utilisation of tasks τ_1 to τ_{n-1} must be less than or equal to 1. (The utilisation of task τ_n is effectively zero as it has an infinite period).

Constraint (i), leads to the following equation bounding the EDF scaling factor as a function of *X*.

$$f1(X) = \frac{2+X}{1+X} \tag{24}$$

Constraint (ii), leads via Lemma 10, to the following equation which bounds the EDF scaling factor as a function of *X* (assuming $U = 1$):

$$f2(X) = \frac{1}{\ln\left(\frac{2+X}{1+X}\right)} \tag{25}$$

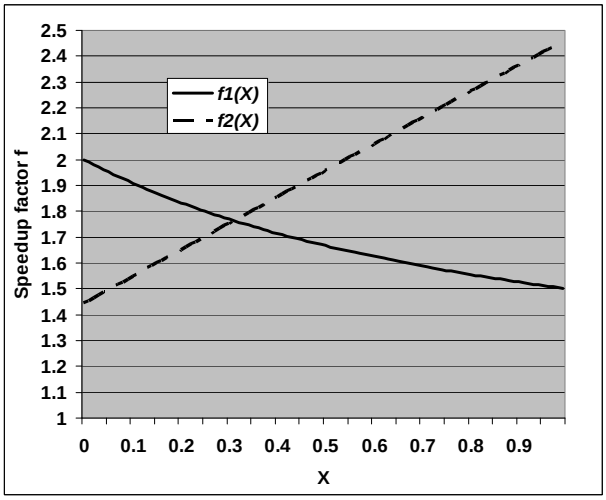

Figure 5: Constraints on the speedup factor for *n* tasks with D≤T

Equation (24) is a continuous non-increasing function of *X* with a maximum value of $f1(0) = 2$. Equation (25) is a continuous non-decreasing function of *X* with a minimum value of $f2(0) = 1/\ln(2) \approx 1.44270$. Hence, the intersection of the two functions determines an upper bound on the EDF scaling factor *f*.

Figure 5 illustrates the two functions bounding the EDF scaling factor, plotted against values of *X*.

The intersection of the two functions is given by:

$$f2(X) = \frac{1}{\ln\left(\frac{2+X}{1+X}\right)} = \frac{2+X}{1+X} = f1(X) \tag{26}$$

Hence we need to find the value of *X* such that:

[6] Technically it is the 'left Riemann sum' as the function is approximated by its value at the left end point of each interval.

$$\ln\left(\frac{2+X}{1+X}\right)=\left(\frac{1}{\left(\frac{2+X}{1+X}\right)}\right) \tag{27}$$

Which can be re-written as:

$$\ln\left(\frac{1}{\left(\frac{1+X}{2+X}\right)}\right)=\left(\frac{1+X}{2+X}\right) \tag{28}$$

Noticing the similarity between Equation (28) and that defining the mathematical constant Ω ($\ln(1/\Omega)=\Omega$), we have:

$$\frac{1+X}{2+X}=\Omega \tag{29}$$

$$\frac{2+X}{1+X}=\frac{1}{\Omega} \tag{30}$$

and hence

$$X=\frac{2\Omega-1}{1-\Omega} \tag{31}$$

Therefore

$$f2\left(\frac{2\Omega-1}{1-\Omega}\right)=f1\left(\frac{2\Omega-1}{1-\Omega}\right)=\frac{1}{\Omega} \tag{32}$$

To show that the maximum EDF scaling factor is equal to this upper bound, we must show that the taskset given in Theorem 2 is schedulable under EDF, for the value of *X* given by Equation (31) and with execution times scaled by a factor of $f=1/\Omega$.

To prove that this taskset is schedulable, we show that its processor demand bound function $h(t)$, is such that $\forall t \quad h(t)/t \le 1$. Our proof proceeds as follows:

- First, we represent $h(t)$ by an infinite series of piecewise linear functions, the *k*th of which corresponds to the processor demand from the *k*th invocations of all the higher priority tasks ($i \ne n$).
- Next, using these piecewise linear functions, we show that the maximum value of $h(t)/t$ must occur for some value of $t=k(2+X)$, where *k* is an integer. We use the discrete function *H*(*k*) to denote the values of $h(t)/t$ at these maxima.
- Finally, we show that $\forall k \ge 1 \quad H(k) \le 1$, hence proving that $\forall t \quad h(t)/t \le 1$.

We first consider the contribution $h'(t)$, to the processor demand bound function $h(t)$, from all the higher priority tasks ($\forall i \ne n$):

$$h'(t)=\lim_{n-1\to\infty}\sum_{i=1}^{n-1}\frac{f}{n-1}\left\lfloor\frac{t}{1+X+(i-1)/(n-1)}\right\rfloor \tag{33}$$

In the limit as $n-1\to\infty$, we can represent this contribution as the sum of an infinite series of piecewise linear functions $g'(k,t)$ for $k=1..\infty$, where the *k*th function represents the contribution from the *k*th invocations of all the higher priority tasks ($i \ne n$).

$$h'(t)=\sum_{k=1}^{\infty}g'(k,t) \tag{34}$$

where:

$$g'(k,t)=\begin{cases}0 & 0\le t<k(1+X)\\ \frac{f}{k}(t-k(1+X)) & k(1+X)\le t<k(2+X)\\ f & t\ge k(2+X)\end{cases} \tag{35}$$

The contribution from task τ_n is similarly represented by a piecewise linear function:

$$g''(t)=\begin{cases}0 & 0\le t<(2+X)\\ fX & t\ge(2+X)\end{cases} \tag{36}$$

The processor demand bound function $h(t)$, is therefore:

$$h(t)=g''(t)+\sum_{k=1}^{\infty}g'(k,t) \tag{37}$$

Figure 6 illustrates $h(t)/t$ plotted against $t/(2+X)$.

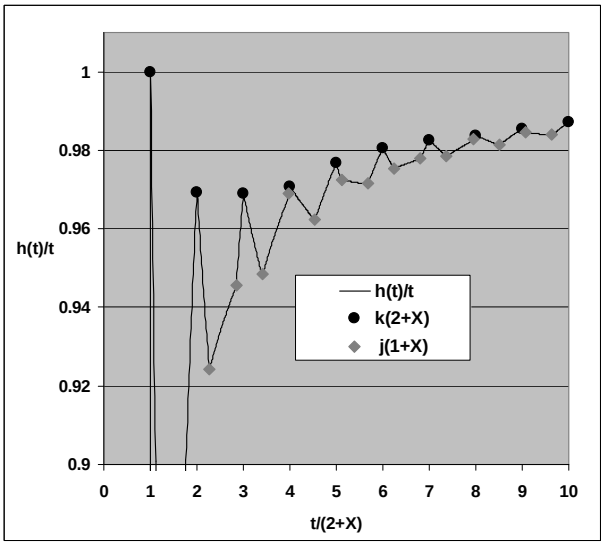

Figure 6: *h(t)/t* for the speedup-optimal taskset

As $h(t)$ is composed from the piecewise linear functions $g'(k,t)$ and $g''(t)$, $y(t)=h(t)/t$ is itself a piecewise continuous function, which is differentiable at all values of *t*, with the exception of the start and end points of the pieces, given by $j(1+X)$, and $k(2+X)$, where *j* and *k* are integers.

Within each piece, $y(t)$ is differentiable with respect to *t*. From Equations (35), (36) and (37):

$$\frac{dy}{dt}=\frac{c_p}{t^2} \tag{38}$$

where c_p is a constant for the particular piece. Note that c_p can be computed from Equation (37), and could take positive, zero or negative values for a given piece; however, the actual values for different pieces are

irrelevant, what is important is the form of the first derivative.

Equation (38) indicates that there are no turning points (maxima or minima where $dy/dt = 0$) within each piece, unless $c_p = 0$, in which case all points within the piece and its end points are potentially maxima or minima that have the same value. Thus, to find the maximum value of the function $h(t)/t$, we need only consider the end points of the pieces.

Further, from Equation (35), we observe that at points $j(1+X)$, the slope of $h(t)/t$ increases, as the jth invocations begin to contribute to $h(t)$, whilst at points $k(2+X)$, the slope of $h(t)/t$ decreases, as the kth invocations cease to contribute to $h(t)$. Thus for $h(t)/t$, maxima (but not minima) can occur at $t = k(2+X)$. Similarly, minima (but not maxima) can occur at $t = j(1+X)$. These maxima and minima are depicted in Figure 6.

As the maximum value of $h(t)/t$ must occur for some value of $t = k(2+X)$, we need only consider $h(t)/t$ at these specific values of t. We use the discrete function $H(k)$ to denote the value of $h(t)$ at $t = k(2+X)$, for $k = 1..\infty$.

$$H(k) = \frac{h(k(2+X))}{k(2+X)} \tag{39}$$

We now use this function to prove that the taskset is schedulable, by showing that:

$$\forall k \geq 1 \quad H(k) \leq 1 \tag{40}$$

Proof of the inequality in equation (40) is in three Lemmas:

1. Lemma 11 shows that as $k \to \infty$, $H(k) \to 1$.
2. Lemma 12 shows that $\forall k \geq 6$, $H(k)$ is a monotonic non-decreasing function of k, with $H(k+1) \geq H(k)$.
3. Lemma 13 shows that $\forall k \leq 6$, $H(k) \leq 1$

Lemma 11: As $k \to \infty$, $H(k) \to 1$.

Proof: For $k \geq 2$, and by reference to Equations (35) and (36), we can separate the processor demand in the interval [0, $k(2+X)$) into 3 components:

(i) $g''(k(2+X)) = fX$

(ii) $g'(j, k(2+X))$ for invocations j=1 to k of all the higher priority tasks τ_i ($i \neq n$).

$$\sum_{j=1}^{k} g'(j, k(2+X)) = fk$$

(iii) $g'(j, k(2+X))$ for invocations $j = k+1$ to ∞ of the higher priority tasks τ_i ($i \neq n$). The index of the final invocations that can contribute to the processor demand in the interval [0, $k(2+X)$), is given by $\lfloor k(2+X)/(1+X) \rfloor$, hence:

$$\sum_{j=k+1}^{\infty} g'(j, k(2+X)) = f \sum_{j=k+1}^{\left\lfloor \frac{k(2+X)}{(1+X)} \right\rfloor} \frac{k(2+X) - j(1+X)}{j}$$

We can therefore write $H(k)$ as follows:

$$H(k) = \frac{f(k+X)}{k(2+X)} + \frac{f}{k(2+X)} \sum_{j=k+1}^{\left\lfloor \frac{k(2+X)}{(1+X)} \right\rfloor} \frac{k(2+X) - j(1+X)}{j} \tag{41}$$

Noting that $(2+X)/(1+X) = f$, we have:

$$\begin{aligned} H(k) &= \frac{(k+X)}{k(1+X)} + \frac{1}{k} \sum_{j=k+1}^{\lfloor fk \rfloor} \frac{(fk - j)}{j} \\ &= \frac{(k+X)}{k(1+X)} - \frac{\lfloor fk \rfloor - (k+1) + 1}{k} + f \sum_{j=k+1}^{\lfloor fk \rfloor} \frac{1}{j} \\ &= \frac{(k+X)}{k(1+X)} - \frac{\lfloor fk \rfloor}{k} + 1 + f \sum_{j=k+1}^{\lfloor fk \rfloor} \frac{1}{j} \\ &= \frac{k(2+X)+X}{k(1+X)} - \frac{\lfloor fk \rfloor}{k} + f \sum_{j=k+1}^{\lfloor fk \rfloor} \frac{1}{j} \end{aligned} \tag{42}$$

Substituting, $X = (2\Omega - 1)/(1-\Omega)$, $1 + X = \Omega/(1-\Omega)$, $2 + X = 1/(1-\Omega)$, and $f = 1/\Omega$ in Equation (42) gives:

$$\begin{aligned} H(k) &= \frac{k + 2\Omega - 1}{k\Omega} - \frac{\lfloor k/\Omega \rfloor}{k} + \frac{1}{\Omega} \sum_{j=k+1}^{\lfloor k/\Omega \rfloor} \frac{1}{j} \\ &= \frac{1}{\Omega} + \frac{2}{k} - \frac{1}{k\Omega} - \frac{\lfloor k/\Omega \rfloor}{k} + \frac{1}{\Omega} \sum_{j=k+1}^{\lfloor k/\Omega \rfloor} \frac{1}{j} \end{aligned} \tag{43}$$

Note that Equation (43) is only valid for $k \geq 2$, as for $k = 1$, $k + 1 > \lfloor k/\Omega \rfloor$.

As $1/z$ is a positive decreasing function, the summation term in Equation (43) is bounded by the following integral, (given that $\lfloor k/\Omega \rfloor \leq k/\Omega$).

$$\frac{1}{\Omega} \sum_{j=k+1}^{\lfloor k/\Omega \rfloor} \frac{1}{j} \leq \frac{1}{\Omega} \int_{j=k}^{k/\Omega} \frac{1}{z} dz = \frac{1}{\Omega} \ln\left(\frac{(k/\Omega)}{k} \right) = \frac{1}{\Omega} \ln\left(\frac{1}{\Omega} \right) = 1 \tag{44}$$

As $k \to \infty$, $\lfloor k/\Omega \rfloor \to k/\Omega$ hence we have:

$$H(k) \overset{k \to \infty}{=} \frac{1}{\Omega} + \frac{2}{k} - \frac{1}{k\Omega} - \frac{1}{\Omega} + 1 = 1 \tag{45}$$

□

Lemma 12: $\forall k \geq 6$, $H(k)$ is a monotonic non-decreasing function, with $H(k+1) \geq H(k)$.

Proof: As $1/\Omega \approx 1.76322$, there are two distinct cases to consider:

Case 1: $\lfloor (k+1)/\Omega \rfloor = \lfloor k/\Omega \rfloor + 1$

Case 2: $\lfloor (k+1)/\Omega \rfloor = \lfloor k/\Omega \rfloor + 2$

Case 1: $\lfloor (k+1)/\Omega \rfloor = \lfloor k/\Omega \rfloor + 1$:

From Equation (43) we have:

$$H(k+1)-H(k)=\frac{2}{(k+1)}-\frac{1}{(k+1)\Omega}-\frac{\lfloor k/\Omega\rfloor+1}{(k+1)}$$
$$+\frac{1}{\Omega}\sum_{j=k+2}^{\lfloor k/\Omega\rfloor+1}\frac{1}{j}-\frac{2}{k}+\frac{1}{k\Omega}+\frac{\lfloor k/\Omega\rfloor}{k}-\frac{1}{\Omega}\sum_{j=k+1}^{\lfloor k/\Omega\rfloor}\frac{1}{j}$$
$$=-\frac{2}{k(k+1)}+\frac{1}{k(k+1)\Omega}+\frac{\lfloor k/\Omega\rfloor-k}{k(k+1)}$$
$$+\frac{1}{\Omega(\lfloor k/\Omega\rfloor+1)}-\frac{1}{\Omega(k+1)} \quad (46)$$

As we are interested only in showing that $H(k+1)-H(k)\geq 0$, we are free to multiply the expression in Equation (46) by any positive quantity. Multiplying by $\Omega k(k+1)(\lfloor k/\Omega\rfloor+1)$ gives:

$$(\lfloor k/\Omega\rfloor+1)(1-2\Omega-k(1+\Omega)+\Omega\lfloor k/\Omega\rfloor)+k(k+1) \quad (47)$$

Substituting $\lfloor k/\Omega\rfloor=(k/\Omega)-\varepsilon$, where ε is a value in the range $0\leq\varepsilon<1$, gives:

$$((k/\Omega)-\varepsilon+1)(1-2\Omega-k\Omega-\Omega\varepsilon)+k(k+1) \quad (48)$$

For ease of reference, we refer to the expression in Equation (48) as $y(\varepsilon)$. Expanding, simplifying and re-arranging, we have:

$$y(\varepsilon)=k/\Omega-2k-k^2-k\varepsilon-\varepsilon+2\Omega\varepsilon+\Omega k\varepsilon+\Omega\varepsilon^2$$
$$+1-2\Omega-k\Omega-\Omega\varepsilon+k^2+k$$
$$=\Omega\varepsilon^2+(\Omega-k+\Omega k-1)\varepsilon+k(1/\Omega-1-\Omega)+1-2\Omega \quad (49)$$

To find the minimum / maximum value of $y(\varepsilon)$ for any possible value of ε, we differentiate with respect to ε:

$$\frac{dy}{d\varepsilon}=2\Omega\varepsilon+(k+1)(\Omega-1) \quad (50)$$

The turning point (minimum / maximum value) of $y(\varepsilon)$ occurs when $dy/d\varepsilon=0$, i.e. for a value of ε given by:

$$\varepsilon=\frac{(k+1)(1-\Omega)}{2\Omega} \quad (51)$$

For $k\geq 2$, it follows that the turning point occurs for $\varepsilon>1$, which is outside the permitted range of values for ε. Hence the maximum / minimum values of $y(\varepsilon)$ must occur for the maximum and minimum permitted values of ε.

We now show that the range of values that ε can take is constrained by the fact that $\lfloor(k+1)/\Omega\rfloor=\lfloor k/\Omega\rfloor+1$.

Assuming that $\varepsilon\geq 2-(1/\Omega)$, then:

$$\frac{k}{\Omega}=\left\lfloor\frac{k}{\Omega}\right\rfloor+\varepsilon\geq\left\lfloor\frac{k}{\Omega}\right\rfloor+2-\frac{1}{\Omega} \quad (52)$$

and so

$$\frac{k+1}{\Omega}\geq\left\lfloor\frac{k}{\Omega}\right\rfloor+2 \quad (53)$$

which implies that $\lfloor(k+1)/\Omega\rfloor=\lfloor k/\Omega\rfloor+2$. This contradicts the assumption that $\lfloor(k+1)/\Omega\rfloor=\lfloor k/\Omega\rfloor+1$, and so ε is constrained (for this, Case 1) to be in the range $0\leq\varepsilon<2-(1/\Omega)$.

We now evaluate $y(\varepsilon)$ for the minimum and maximum possible values of ε.

For $\varepsilon=0$, we have:

$$y(0)=k(1/\Omega-1-\Omega)+1-2\Omega \quad (54)$$

So $y(0)\geq 0$ provided that:

$$k\geq\frac{2\Omega-1}{1/\Omega-(1+\Omega)}\approx 0.684876 \quad (55)$$

Hence $\forall k\geq 2$, $y(0)\geq 0$.

For $\varepsilon=2-(1/\Omega)$, from Equation (49), we have:

$$y\big(2-(1/\Omega)\big)=\left(2\Omega-1-2k+\frac{k}{\Omega}+2\Omega k-k-2+\frac{1}{\Omega}\right)$$
$$+\Omega\left(2-\frac{1}{\Omega}\right)^2+\left(\frac{k}{\Omega}-k-\Omega k+1-2\Omega\right) \quad (56)$$

Simplifying:

$$y\big(2-(1/\Omega)\big)=\Omega\left(2-\frac{1}{\Omega}\right)^2+k\left(\frac{2}{\Omega}-4+\Omega\right)-\left(2-\frac{1}{\Omega}\right) \quad (57)$$

So $y\big(2-(1/\Omega)\big)\geq 0$ provided that:

$$k\geq\frac{\left(2-\frac{1}{\Omega}\right)-\Omega\left(2-\frac{1}{\Omega}\right)^2}{\left(\frac{2}{\Omega}-4+\Omega\right)}\approx 2.190228 \quad (58)$$

Hence $\forall k\geq 3$, $y\big(2-(1/\Omega)\big)\geq 0$.

To summarise,

(i) In this case, the range of potential values for ε is limited to $0\leq\varepsilon<2-(1/\Omega)\approx 0.236778$.

(ii) Equation (51) shows that provided $k\geq 2$, $y(\varepsilon)$ has no turning points in the range $0\leq\varepsilon<1$, and so the maximum and minimum values of $y(\varepsilon)$ must occur for the maximum and minimum permitted values of ε.

(iii) Equations (55) and (58) show that for $k\geq 3$, the maximum and minimum values of $y(\varepsilon)$ are both positive.

We conclude that for $k\geq 3$, $y(\varepsilon)$ is always positive and hence for $k\geq 3$ and $\lfloor(k+1)/\Omega\rfloor=\lfloor k/\Omega\rfloor+1$, $H(k+1)\geq H(k)$.

Case 2: $\lfloor(k+1)/\Omega\rfloor=\lfloor k/\Omega\rfloor+2$:

From Equation (43) we have:

$$H(k+1)-H(k)=\frac{2}{(k+1)}-\frac{1}{(k+1)\Omega}-\frac{\lfloor k/\Omega\rfloor+2}{(k+1)}$$
$$+\frac{1}{\Omega}\sum_{j=k+2}^{\lfloor k/\Omega\rfloor+2}\frac{1}{j}-\frac{2}{k}+\frac{1}{k\Omega}+\frac{\lfloor k/\Omega\rfloor}{k}-\frac{1}{\Omega}\sum_{j=k+1}^{\lfloor k/\Omega\rfloor}\frac{1}{j}$$

$$= -\frac{2}{k(k+1)} + \frac{1}{k(k+1)\Omega} + \frac{\lfloor k/\Omega \rfloor - 2k}{k(k+1)} + \frac{1}{\Omega(\lfloor k/\Omega \rfloor + 2)} + \frac{1}{\Omega(\lfloor k/\Omega \rfloor + 1)} - \frac{1}{\Omega(k+1)} \quad (59)$$

As we are interested only in showing that $H(k+1) - H(k) \geq 0$, we are free to multiply the expression in Equation (59) by any positive quantity. Multiplying by $\Omega k(k+1)(\lfloor k/\Omega \rfloor + 1)(\lfloor k/\Omega \rfloor + 2)$ gives:

$$(\lfloor k/\Omega \rfloor + 2)(\lfloor k/\Omega \rfloor + 1)(1 - 2\Omega - k(1+2\Omega) + \Omega\lfloor k/\Omega \rfloor) + k(k+1)(\lfloor k/\Omega \rfloor + 1) + k(k+1)(\lfloor k/\Omega \rfloor + 2) \quad (60)$$

Substituting $\lfloor k/\Omega \rfloor = (k/\Omega) - \varepsilon$, where ε is a value in the range $0 \leq \varepsilon < 1$, gives:

$$\left(\frac{k}{\Omega} - \varepsilon + 2\right)\left(\frac{k}{\Omega} - \varepsilon + 1\right)(1 - 2\Omega - 2\Omega k - \Omega\varepsilon) + k(k+1)\left(\frac{k}{\Omega} - \varepsilon + 1\right) + k(k+1)\left(\frac{k}{\Omega} - \varepsilon + 2\right) \quad (61)$$

Expanding we have:

$$\frac{k^2}{\Omega^2} - \frac{2k^2}{\Omega} - \frac{2k^3}{\Omega} - \frac{\varepsilon k^2}{\Omega} - \frac{2\varepsilon k}{\Omega} + 4\varepsilon k + 4\varepsilon k^2 + 2\varepsilon^2 k + \frac{3k}{\Omega} - 6k - 6k^2 - 3\varepsilon k - 3\varepsilon + 6\Omega\varepsilon + 6\Omega\varepsilon k + 3\Omega\varepsilon^2 + \varepsilon^2 - 2\Omega\varepsilon^2 - 2\Omega\varepsilon^2 k - \Omega\varepsilon^3 + 2 - 4\Omega - 4\Omega k - 2\Omega\varepsilon + \frac{2k^3}{\Omega} + \frac{2k^2}{\Omega} - 2\varepsilon k^2 - 2\varepsilon k + 3k^2 + 3k \quad (62)$$

For ease of reference, we refer to the expression in Equation (62) as the function $y(\varepsilon)$. Simplifying and re-arranging, we have:

$$y(\varepsilon) = (-\Omega)\varepsilon^3 + (2k + \Omega + 1 - 2\Omega k)\varepsilon^2 + \left(-\frac{k^2}{\Omega} - \frac{2k}{\Omega} - k + 2k^2 - 3 + 4\Omega + 6\Omega k\right)\varepsilon + \frac{k^2}{\Omega^2} + \frac{3k}{\Omega} - 3k - 3k^2 + 2 - 4\Omega - 4\Omega k \quad (63)$$

To find the minimum / maximum value of $y(\varepsilon)$ for any possible value of ε, we differentiate with respect to ε:

$$\frac{dy}{d\varepsilon} = -3\Omega\varepsilon^2 + 2(2k + \Omega + 1 - 2\Omega k)\varepsilon + \left(-\frac{k^2}{\Omega} - \frac{2k}{\Omega} - k + 2k^2 - 3 + 4\Omega + 6\Omega k\right) \quad (64)$$

The turning points (minimum / maximum values) of $y(\varepsilon)$ occur when $dy/d\varepsilon = 0$, i.e. for the values of ε given by the solutions to a quadratic equation, formed from the expression in Equation (64).

The solutions to a quadratic equation of the form $a\varepsilon^2 + b\varepsilon + c = 0$ are given by:

$$\varepsilon = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad (65)$$

We are interested in the case where the turning points of $y(\varepsilon)$ occur outside of the permitted range of values for ε ($0 \leq \varepsilon < 1$). From Equation (65), we can see that this is the case provided that $(-b/2a) > 1/2$ and $-4ac \geq 0$, as the two solutions are then less than zero and greater than one respectively. Now as

$$\frac{-b}{2a} = \frac{4k(1-\Omega) + 2(1+\Omega)}{6\Omega} \quad (66)$$

$(-b/2a) > 1/2$ provided that:

$$k \geq \frac{3\Omega - 2(1+\Omega)}{4(1-\Omega)} \approx -0.827557 \quad (67)$$

Further,

$$-4ac = 12\Omega\left(-\frac{k^2}{\Omega} - \frac{2k}{\Omega} - k + 2k^2 - 3 + 4\Omega + 6\Omega k\right) \quad (68)$$

So $-4ac \geq 0$ provided that:

$$\left(2 - \frac{1}{\Omega}\right)k^2 + \left(6\Omega - 1 - \frac{2}{\Omega}\right)k - 3 + 4\Omega \geq 0 \quad (69)$$

Evaluating the coefficients in Equation (69), we have:

$$1.611439k^2 - 7.646811k - 4.977885 \geq 0 \quad (70)$$

Solving for *k*, gives:

$$k = 2.372665 \pm 2.952733 \quad (71)$$

Therefore, $\forall k \geq 6$, the inequality in Equation (70) holds and the turning points of $y(\varepsilon)$ occur outside of the permitted range of values of ε. Thus $\forall k \geq 6$ the maximum / minimum values of $y(\varepsilon)$ must occur for the maximum and minimum permitted values of ε.

From Equation (63), for $\varepsilon = 0$, we have:

$$y(0) = \left(\frac{1}{\Omega^2} - 3\right)k^2 + \left(\frac{3}{\Omega} - 3 - 4\Omega\right)k + 2 - 4\Omega \quad (72)$$

Evaluating the coefficients in Equation (72), gives:

$$0.1089547k^2 - 0.02109534k - 0.26857316 \geq 0 \quad (73)$$

Hence, $\forall k \geq 2$, $y(0) \geq 0$.

From Equation (63), for $\varepsilon = 1$, we have:

$$y(1) = -\Omega + 2k + \Omega + 1 - 2\Omega k - \frac{k^2}{\Omega} - \frac{2k}{\Omega} - k + 2k^2 - 3 + 4\Omega + 6\Omega k + \frac{k^2}{\Omega^2} + \frac{3k}{\Omega} - 3k - 3k^2 + 2 - 4\Omega - 4\Omega k \quad (74)$$

Simplifying Equation (74) gives:

$$y(1) = \left(\frac{1}{\Omega^2} - \frac{1}{\Omega} - 1\right)k^2 + \left(\frac{1}{\Omega} - 2\right)k \quad (75)$$

Evaluating the coefficients in Equation (75), gives:

$$y(1) = 0.34573192k^2 - 0.23677716k \quad (76)$$

Hence, $\forall k \geq 2$, $y(1) \geq 0$. (From Equation (76), $y(1)$ is positive provided that $k \geq 1$; however, our analysis is only valid for $k \geq 2$).

To summarise,

(i) In this case, the range of potential values for ε is bounded by $0 \leq \varepsilon < 1$.

(ii) Equations (64) to (71) show that provided $k \geq 6$, $y(\varepsilon)$ has no turning points in the range $0 \leq \varepsilon < 1$, and so the maximum and minimum values of $y(\varepsilon)$ must occur for the maximum and minimum values of ε.

(iii) Equations (73) and (76) show that for $k \geq 2$, the values of $y(0)$ and $y(1)$ are both positive.

We conclude that for $k \geq 6$, $y(\varepsilon)$ is always positive and hence for $k \geq 6$ and $\lfloor (k+1)/\Omega \rfloor = \lfloor k/\Omega \rfloor + 2$, $H(k+1) \geq H(k)$.

Combining the results for Case 1 and Case 2, shows that $\forall k \geq 6$, $H(k+1) \geq H(k)$ □

Lemma 13: $\forall k \leq 6$, $H(k) \leq 1$.

Proof: Recall that $H(k)$ is defined by Equation (39) as the value of the processor load $h(t)/t$ at discrete points in time given by $t = k(2+X)$, for $k = 1..\infty$, where $h(t)$ is the processor demand bound function given by Equation (34).

First we consider the case where $k = 1$. For $k = 1$, we have:

$$H(k) = \frac{f(1+X)}{(2+X)} = 1 \quad (77)$$

Table 2 below gives the computed values of $H(k)$, for $k = 1..6$.

Table 2

k	$H(k)$
1	1
2	0.969352362
3	0.968932165
4	0.970821141
5	0.976740571
6	0.980546791

□

Lemma 11 showed that as $k \to \infty$, $H(k) \to 1$, Lemma 12 showed that $\forall k \geq 6$, $H(k)$ is a monotonic non-decreasing function of k, and Lemma 13 showed that $\forall k \leq 6$, $H(k) \leq 1$. It follows that $\forall k \geq 1 \quad H(k) \leq 1$.

As the maximum points of the function $h(t)/t$ are given by the values of $H(k)$, we conclude that:

$$\forall t \quad \frac{h(t)}{t} \leq 1 \quad (78)$$

and hence that the taskset in Theorem 2, with the value of X defined by Equation (31), is schedulable under EDF with all execution times scaled by a factor of $1/\Omega$. Given the constraints expressed in Equations (24) and (25), the maximum EDF scaling factor, for any value of X, is therefore $1/\Omega$.

For a constrained-deadline taskset of arbitrary cardinality, the processor speedup factor for fixed priority pre-emptive scheduling is equal to the maximum EDF scaling factor (for any value of X) for the speedup-optimal taskset described in Theorem 2 □

Corollary 6: The maximum EDF scaling factor and hence the processor speedup factor for a constrained-deadline taskset of any cardinality is achieved for the speedup-optimal taskset described in Theorem 2, with an execution time for the lowest priority task of $X = (2\Omega - 1)/(1 - \Omega) \approx 0.312333$.

5. Processor speedup factor for implicit-deadline tasksets

In this section, we extend the results of Sections 3 and 4 to implicit-deadline tasksets ($\forall i \quad D_i = T_i$).

Before considering tasksets of arbitrary cardinality, we first present results for tasksets comprising just two tasks. The derivation of this result again provides the intuition for the general case.

Theorem 5: For the class of tasksets with implicit deadlines and cardinality two, there is a speedup-optimal taskset V, which has the following parameters:

Taskset V is identical to the taskset described in Theorem 1 with the exception that the period of task τ_n, rather than being infinite, is equal to its deadline.

Proof: As the class of implicit-deadline tasksets is a subset of the class of constrained-deadline tasksets, proof follows directly from the proof of Theorem 1, noting that Lemma 2 does not apply and instead we have the constraint that $T_n = D_n$ □

Theorem 6: For an implicit-deadline taskset of cardinality two, the processor speedup factor for fixed priority pre-emptive scheduling is $1/2(\sqrt{2}-1) \approx 1.207107$.

Proof: We prove the theorem by determining the maximum EDF scaling factor for the taskset V described in Theorem 5, for any value of X.

As taskset V is an implicit-deadline taskset, a sufficient and necessary condition for schedulability under EDF, is that the total utilisation of the taskset must

be less than or equal to 1, after the task execution times are scaled by a factor of *f*.

The total utilisation of the tasks is given by:

$$\sum_{i=1}^{2} U_i = \frac{1}{1+X} + \frac{X}{2+X} \tag{79}$$

Therefore, assuming total utilisation $U = 1$, we have the following equation for the maximum EDF scaling factor as a function of *X*.

$$f(X) = \frac{1}{\left(\frac{1}{1+X} + \frac{X}{2+X}\right)}$$

$$= \frac{(1+X)(2+X)}{(2+X)+(1+X)X} = \frac{X^2+3X+2}{X^2+2X+2} \tag{80}$$

Figure 7 plots the maximum EDF scaling factor $f(X)$, against values of *X*.

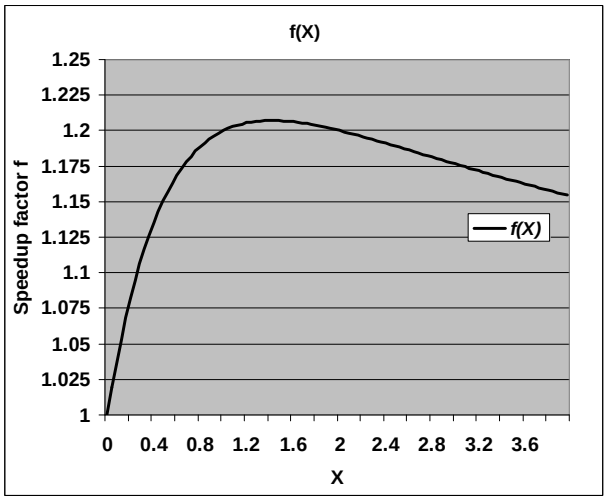

Figure 7: Speedup factor for *2* tasks with D=T

Equation (80), is a continuous function of *X*, with maximum / minimum values where the first derivative with respect to *X* is zero.

$$\frac{df}{dX} = \frac{-X^2+2}{(X^2+2X+2)^2} \tag{81}$$

Hence the maximum value occurs for $X = \sqrt{2}$, resulting in a maximum EDF scaling factor of:

$$f(\sqrt{2}) = \frac{4+3\sqrt{2}}{4+2\sqrt{2}} = \frac{1}{2(\sqrt{2}-1)} \approx 1.207107 \tag{82}$$

For an implicit-deadline taskset of cardinality two, the processor speedup factor for fixed priority pre-emptive scheduling is equal to the maximum EDF scaling factor (for any value of *X*) for the speedup-optimal taskset described in Theorem 5 □

Corollary 7: As EDF is known to schedule any implicit-deadline taskset, provided that $U \le 1$, Theorem 6 shows that fixed priority pre-emptive scheduling can schedule any implicit-deadline taskset of cardinality two, provided that $U \le 2(\sqrt{2}-1) \approx 0.828427$, in agreement with, and providing a diverse proof of, the result of Fineberg and Serlin (1967).

We note that this speedup-optimal taskset for the implicit-deadline case and cardinality two, is the one used as an illustrative example in Section 2.6.

Theorem 7: For the class of implicit-deadline tasksets with arbitrary cardinality, there is a speedup-optimal taskset *V*, which has the following parameters:

Taskset *V* is identical to the taskset described in Theorem 2 with the exception that the period of task τ_n, rather than being infinite, is equal to its deadline.

Proof: As the class of implicit-deadline tasksets is a subset of the class of constrained-deadline tasksets, proof follows directly from the proof of Theorem 2, noting that Lemma 2 does not apply and instead we have the constraint that $T_n = D_n$ □

Theorem 8: For an implicit-deadline taskset of arbitrary cardinality, the processor speedup factor for fixed priority pre-emptive scheduling is $1/\ln(2) \approx 1.44270$.

Proof: We prove the theorem by determining the maximum EDF scaling factor for the taskset *V* described in Theorem 7, for any value of *X*.

As taskset *V* is an implicit-deadline taskset, a sufficient and necessary condition for schedulability under EDF, is that the total utilisation of the taskset must be less than or equal to 1, after the task execution times are scaled by a factor of *f*.

From Lemma 10, the total utilisation of the tasks $\tau_1..\tau_{n-1}$ is given by:

$$\sum_{i=1}^{n-1} U_i = \ln\left(\frac{2+X}{1+X}\right) \tag{83}$$

Further, the utilisation of task τ_n is:

$$U_n = \frac{X}{2+X} \tag{84}$$

Therefore, assuming total utilisation $U = 1$, we have the following equation for the maximum EDF scaling factor as a function of *X*.

$$f(X) = \frac{1}{\left(\ln\left(\frac{2+X}{1+X}\right) + \frac{X}{2+X}\right)} \tag{85}$$

Figure 8 plots the maximum EDF scaling factor, against values of *X*.

Equation (85), is a continuous non-increasing function of *X*, with a maximum value $f(0) = 1/\ln(2)$. The maximum EDF scaling factor is therefore $1/\ln(2) \approx 1.44270$, which occurs as the execution time *X* of the lowest priority task tends to zero.

For an implicit-deadline taskset of arbitrary cardinality, the processor speedup factor for fixed priority pre-emptive scheduling is equal to the maximum EDF scaling factor (for any value of X) for the speedup-optimal taskset described in Theorem 7 □

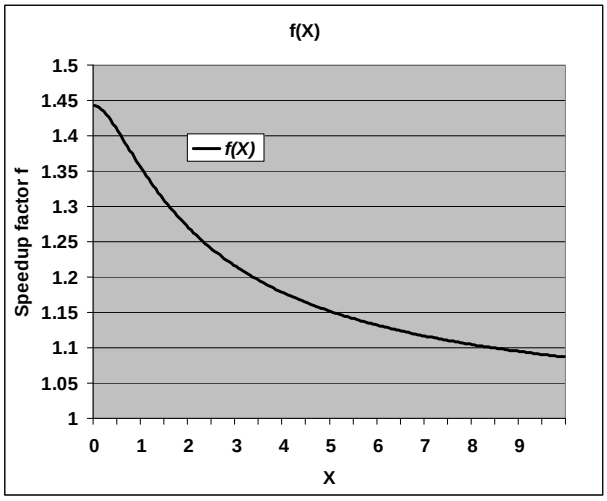

Figure 8: Speedup factor for *n* tasks with D=T

Corollary 8: As EDF is known to schedule any implicit-deadline taskset, provided that $U \le 1$, Theorem 7 shows that fixed priority pre-emptive scheduling can schedule any implicit-deadline taskset, provided that $U \le \ln(2) \approx 0.693147$, in agreement with, and providing a diverse proof of, the well known result of Liu and Layland (1973).

6. Summary and Conclusions

In this paper, we have examined the relative effectiveness of fixed priority pre-emptive scheduling for sporadic / periodic tasks with constrained deadlines ($D_i \le T_i$). Our metric for measuring the effectiveness of this scheduling policy is a resource augmentation factor known as the processor speedup factor.

The processor speedup factor is defined as the maximum amount by which the execution time of all tasks in a taskset that is only just schedulable under fixed priority pre-emptive scheduling can be scaled up and the taskset remain feasible (i.e. schedulable under an optimal algorithm such as EDF).

An alternate and equivalent definition of the processor speedup factor is the maximum amount by which the processor needs to be speeded up so that any taskset that is feasible (i.e. schedulable by an optimal algorithm such as EDF) can be guaranteed to be schedulable under fixed priority pre-emptive scheduling.

The major contributions of this paper are as follows:

- Deriving the structure and parameters of a *speedup-optimal* taskset that provides a tight bound on the processor speedup factor for constrained-deadline tasksets.
- Proving that the processor speedup factor for constrained-deadline tasksets of cardinality two, is $\sqrt{2} \approx 1.414214$.
- Proving that the processor speedup factor for constrained-deadline tasksets of arbitrary cardinality, is $1/\Omega \approx 1.76323$.
- Deriving, in Appendix A, an upper bound on the processor speedup factor for small n which improves upon the general result for constrained-deadline tasksets of arbitrary cardinality.
- Proving that the processor speedup factor for implicit-deadline tasksets of cardinality two is $1/2(\sqrt{2}-1) \approx 1.207107$. A result that provides a diverse proof of one of the earliest published results on fixed priority schedulability analysis, the sufficient schedulability test $U \le 2(\sqrt{2}-1)$ for two tasks by Fineberg and Serlin (1967).
- Proving that the processor speedup factor for implicit-deadline tasksets of arbitrary cardinality is $1/\ln(2) \approx 1.44270$. A result that is in agreement with, and provides a diverse proof of, the well known sufficient schedulability test $U \le \ln(2)$ of Liu and Layland (1973).

The seminal work of Liu and Layland (1973) 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.

The research in this paper provides an analogous characterisation of the maximum performance penalty incurred when constrained-deadline tasksets are scheduled using deadline-monotonic, fixed priority pre-emptive scheduling instead of an optimal algorithm such as EDF. Table 3 summarises the extent of these performance penalties.

Table 3

	Optimal (e.g. EDF)	Fixed Priority	Speedup factor
Implicit-deadline	$U \le 1$	$U \le \ln(2)$ ≈ 0.693147	$1/\ln(2)$ ≈ 1.44270
Constrained-deadline	$\text{LOAD} \le 1$	$\text{LOAD} \le \Omega$ ≈ 0.567143	$1/\Omega$ ≈ 1.76323

Note that although in this paper, we have made numerous references to EDF as an example of an optimal pre-emptive uniprocessor scheduling algorithm, and made use of results about EDF in our proofs, our results are valid with respect to *any* optimal pre-emptive uniprocessor scheduling algorithm, for example Least Laxity First (Mok, 1983). This is because all such optimal algorithms can by definition schedule exactly the same set of tasksets: all those that are feasible.

In conclusion, this paper provides for the first time, a tight bound on the sub-optimality of fixed priority pre-emptive scheduling for uniprocessor systems with constrained-deadlines.

6.1. Future work

In future, we intend to investigate the sub-optimality of fixed priority pre-emptive scheduling with respect to *arbitrary-deadline* tasksets, where task deadlines may be less than, equal to, or greater than their periods.

To the best of our knowledge, no research has yet been done to characterise the average-case sub-optimality of fixed priority pre-emptive scheduling for constrained-deadline tasksets. This is also an interesting area for future research.

6.2. Acknowledgements

This work was funded in part by the EU Frescor, eMuCo and Jeopard projects.

Appendix A: Processor speedup factor for constrained-deadline tasksets with cardinality *n*

In this appendix, we provide an upper bound on the processor speedup factor for fixed priority pre-emptive scheduling of constrained-deadline tasksets comprising a small number of tasks.

Theorem A.1: For the class of tasksets with constrained deadlines and cardinality *n*, there is a speedup-optimal taskset *V*, with $\alpha_V = 1$, which has the following parameters:

$$\forall i \neq n \quad D_i = T_i = \sum_{\forall j \in hp(n)} C_j + C_n + \sum_{\forall j \in hp(i)} C_j$$

$$D_n = 1,\ T_n = \infty \qquad \text{(A.1)}$$

Note that in Theorem A.1, the deadline of task τ_n has been normalised to 1 and the other task periods and deadlines adjusted accordingly. Further, the task execution times are free variables, subject to the constraint:

$$2 \sum_{\forall j \in hp(n)} C_j + C_n = 1 \qquad \text{(A.2)}$$

Proof: Proof follows directly from Lemmas 1 to 8, specifically:

- τ_n must be a constraining task, with the longest deadline and the lowest priority (Lemma 1).
- τ_n must have an infinite period (Lemma 2).
- $t = D_n$ must be the start of an idle period (Lemma 3).
- $\forall i \neq n \quad T_i < D_n$ (Lemma 4)
- $\forall i \neq n \quad D_i = T_i$ (Lemma 5).
- $\forall i \neq n \quad T_i > D_n / 2$ (Lemma 6).
- Following a critical instant, τ_n must execute continuously from when it first starts execution until it completes (Lemma 7).
- The task parameters must comply with the following equation (Lemma 8):

$$\forall i \neq n \quad D_i = T_i = \sum_{\forall j} C_j + \sum_{\forall j \in hp(i)} C_j$$

□

Lemma A.1: The following inequality holds for any implicit-deadline taskset *W*, with cardinality $n > 2$ that is not schedulable according to fixed priority pre-emptive scheduling:

$$\sum_{\forall j \in hp(n)} U_j > (n-1)\left(\sqrt[(n-1)]{\frac{2}{1+U_n}} - 1 \right) \qquad \text{(A.3)}$$

Proof: As taskset *W* is unschedulable, it cannot be schedulable according the Hyperbolic bound (Bini et al., 2003). Hence:

$$\prod_{\forall j} \left(1 + U_j\right) > 2 \qquad \text{(A.4)}$$

Consider the arithmetic and geometric means of the series of values $(1+U_j)$, $\forall j \in hp(n)$. By the inequality of arithmetic and geometric means, we have:

$$\sum_{\forall j \in hp(n)} \left(\frac{1+U_j}{n-1} \right) \geq \sqrt[(n-1)]{\prod_{\forall j \in hp(n)} \left(1+U_j\right)} \qquad \text{(A.5)}$$

Substituting for the Hyperbolic bound divided by $1+U_n$, into Equation (A.5) gives:

$$\sum_{\forall j \in hp(n)} \left(\frac{1+U_j}{n-1} \right) > \sqrt[(n-1)]{\frac{2}{1+U_n}} \qquad \text{(A.6)}$$

As:

$$\sum_{\forall j \in hp(n)} U_j = (n-1)\left(\sum_{\forall j \in hp(n)} \frac{1+U_j}{(n-1)} - 1 \right) \qquad \text{(A.7)}$$

it follows that:

$$\sum_{\forall j \in hp(n)} U_j > (n-1)\left(\sqrt[(n-1)]{\frac{2}{1+U_n}} - 1 \right) \qquad \text{(A.8)}$$

□

Theorem A.2: For a constrained-deadline taskset, of cardinality $n \geq 2$, an upper bound on the processor speedup factor for fixed priority pre-emptive scheduling is given by $f^{UB}(n)$, where $f^{UB}(n)$ is defined as follows:

$$f^{UB}(n) = 1 / L(n)$$

Where $L(n)$ forms the solution to the following equality, where C_n may take any non-negative real value:

$$L(n) = \left(\frac{1+C_n}{2} \right) = (n-1)\left(\sqrt[(n-1)]{\frac{2}{1+C_n}} - 1 \right) \qquad \text{(A.9)}$$

Proof: Let taskset V' be formed from taskset V in Theorem A.1 by an infinitesimal increase in the execution time of task τ_n, with no changes to any of the other task parameters. Note that taskset V' is unschedulable according to fixed priority pre-emptive scheduling as τ_n misses its deadline. From Equation (A.2) which holds for taskset *V*, for taskset V' we have:

$$2 \sum_{\forall j \in hp(n)} C_j + C_n > 1 \qquad \text{(A.10)}$$

We now consider the total utilisation *U* of taskset V'. Observe that if we set the period of task τ_n equal to its deadline ($T_n = D_n = 1$), then this transforms V' into an implicit deadline taskset *W*, which is unschedulable according to fixed priority pre-emptive scheduling, as taskset V' is already unschedulable with $\tau_n = \infty$.

From Lemma A.1, as $T_n = 1$ for the implicit-deadline taskset *W*, we have:

$$\sum_{\forall j \in hp(n)} U_j > (n-1)\left(\sqrt[(n-1)]{\frac{2}{1+C_n}} - 1 \right) \qquad \text{(A.11)}$$

As $T_n = \infty$ in taskset V', $U_n = 0$, and so the LHS of Equation (A.11) corresponds to the total utilisation *U* of taskset V'. Hence we have:

$$U > (n-1)\left(\sqrt[(n-1)]{\frac{2}{1+C_n}} - 1 \right) \qquad \text{(A.12)}$$

Now consider taskset V' scheduled according to EDF. Taskset V' is schedulable according to EDF if and only if:

$$\max_{\forall t}\left(\frac{h(t)}{t} \right) \le 1 \qquad \text{(A.13)}$$

where $h(t)$ is the processor demand bound function defined by Equation (2).
Now,

$$\max_{\forall t}\left(\frac{h(t)}{t} \right) \ge \max\left(\frac{h(1)}{1}, \frac{h(\infty)}{\infty} \right) = \max\left(h(1), U\right) \qquad \text{(A.14)}$$

From Equation (2):

$$h(1) = \sum_{\forall j \in hp(n)} C_j + C_n \qquad \text{(A.15)}$$

Substituting for the summation term from Equation (A.10) gives:

$$h(1) > \frac{1+C_n}{2} \qquad \text{(A.16)}$$

Substituting the total utilisation *U* from Equation (A.12) and the processor demand bound function $h(1)$ from Equation (A.16) into Equation (A.14) gives:

$$\max_{\forall t}\left(\frac{h(t)}{t} \right) > \max\left(\left(\frac{1+C_n}{2} \right), (n-1)\left(\sqrt[(n-1)]{\frac{2}{1+C_n}} - 1 \right) \right) \qquad \text{(A.17)}$$

For any given value of $n \ge 2$, the left hand term within the max expression, is a monotonically increasing function of C_n, whilst the right hand term is a monotonically decreasing function of C_n. Hence the minimum value $L(n)$, of the max expression is achieved when these two terms are equal.

$$L(n) = \left(\frac{1+C_n}{2} \right) = (n-1)\left(\sqrt[(n-1)]{\frac{2}{1+C_n}} - 1 \right) \qquad \text{(A.18)}$$

Let us assume that taskset V', which we know is unschedulable according to fixed priority pre-emptive scheduling, was obtained from a taskset S' by applying a processor speedup factor $f \ge f^{UB}(n) = 1/L(n)$, such that the execution times of the tasks in taskset S' are f times those of the corresponding tasks in taskset V'. Equation (A.19) shows that taskset S' cannot be schedulable according to EDF, as:

$$\max_{\forall t}\left(\frac{h(t)}{t} \right) > \frac{f}{L(n)} \ge 1 \qquad \text{(A.19)}$$

Hence using a processor speedup factor $f \ge f^{UB}(n)$, it is not possible to obtain a taskset V' that is unschedulable according to fixed priority pre-emptive scheduling, from a taskset S' that is schedulable according to EDF.

Recall that taskset V' is related to the speedup-optimal taskset *V* of Theorem A.1 via an infinitesimal increase in the execution time of task τ_n. We conclude that there can be no tasksets, that are schedulable according to EDF that are unschedulable according to fixed priorities, when a processor speedup factor of $f \ge f^{UB}(n)$ is applied. $f^{UB}(n)$ is therefore an upper bound on the processor speedup factor for constrained-deadline tasksets of cardinality *n*, which is *sufficient* to ensure that any such taskset that is schedulable according to an optimal scheduling policy (e.g. EDF) remains schedulable according to fixed priority pre-emptive scheduling □

We observe that $f^{UB}(n)$ is an upper bound, as opposed to an exact value, due to the fact that the Hyperbolic bound used in its derivation is a sufficient but not necessary schedulability test, and also due to the inequality in Equation (A.14).

Table 4 and

Table 5 give the values of $L(n)$ and $f^{UB}(n)$ as a function of *n*, computed by numerical approximation. Note the values of $f^{UB}(n)$ have been rounded up, to ensure sufficiency.

Table 4

n	$L(n)$	$f^{UB}(n)$	$f^{LB}(n)$
2	0.618034	1.619	1.414
3	0.594313	1.683	1.587
4	0.585670	1.708	1.656
5	0.581198	1.721	1.684
6	0.578464	1.729	1.704
7	0.576621	1.735	1.718
8	0.575294	1.739	1.723
9	0.574293	1.742	1.730

Table 5

n	$L(n)$	$f^{UB}(n)$	$f^{LB}(n)$
2	0.618034	1.619	1.414
4	0.585670	1.708	1.656
8	0.575294	1.739	1.723
16	0.570987	1.752	1.749
32	0.569012	1.758	1.757
64	0.568065	1.761	1.759
128	0.567601	1.762	1.761
256	0.567371	1.763	1.762
∞	Ω	$1/\Omega$	$1/\Omega$

To complement the upper bounds given by Theorem A.2, Table 4 and

Table 5 also give lower bounds $f^{LB}(n)$ on the processor speedup factor. These lower bounds are *necessary* to schedule tasksets whose parameters comply with Theorem A.1.

The lower bounds were found by searching for tasksets of the desired cardinality that comply with Theorem A.1 and remain schedulable according to EDF when their execution times are scaled up by the maximum possible scaling factor *f*. The search for these tasksets, and hence the lower bounds, involved iterating over:

(i) A range of possible values for the EDF scaling factor *f* from 1.4 to 1.77 in steps of 0.001.

(ii) A range of possible values for C_n as a proportion of the total execution time of the higher priority tasks τ_1 to τ_{n-1}, from 0.001 to 2.0.

(iii) (a) Arithmetic progressions in the values of C_1 to C_{n-1}, with a range of increments from 0.0 to 1.0 times C_1 in steps of 0.001 times C_1.

(b) Geometric progressions in the values of C_1 to C_{n-1}, with multiplying factors from 1.0 to 1.5 in steps of 0.001.

The lower bounds in Table 4 and

Table 5 are effectively rounded down to ensure necessity. Note that the types of taskset used to determine these lower bounds are only known to be speedup-optimal for $n = 2$ and $n = \infty$.

Figure 9 plots the upper and lower bounds for small values of *n*, from Table 4. The difference between the upper and lower bounds is less than 1% for tasksets of cardinality of 7 or more.

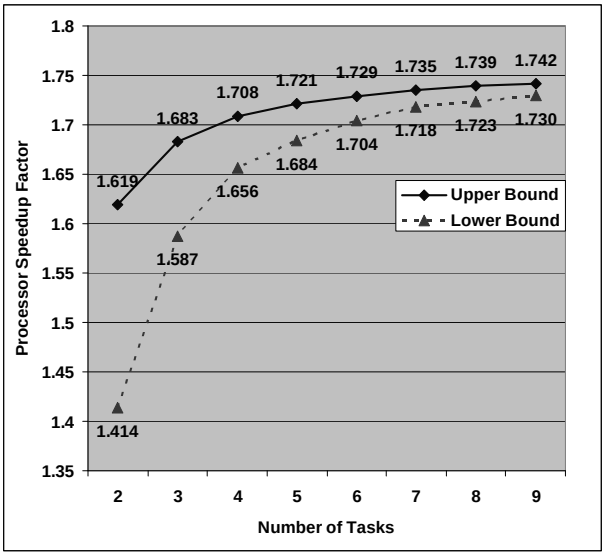

Figure 9: Upper and lower bounds on the Speedup factor for *n* tasks with D≤T

Similarly, Figure 10 plots the upper and lower bounds from

Table 5, for values of *n* from 2 to 256. The difference between the upper and lower bounds is less than 0.1% for tasksets of cardinality of 32 or more.

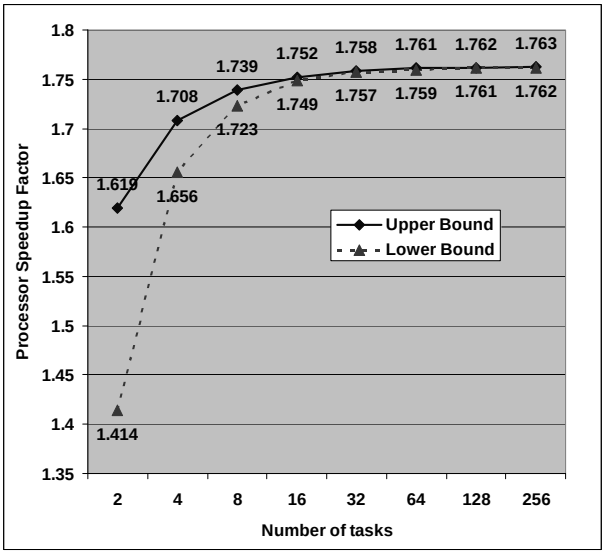

Figure 10: Upper and lower bounds on the Speedup factor for *n* tasks with D≤T

Theorem A.3: For a constrained-deadline taskset, of cardinality *n*, the upper bound $f^{UB}(n)$ on the processor speedup factor for fixed priority pre-emptive scheduling tends to $1/\Omega$ as *n* tends to infinity.

Proof: Follows the same logic as the proof of Theorem A.2. From Equation (A.12), we have:

$$1+\frac{U}{(n-1)} > \sqrt[(n-1)]{\frac{2}{1+C_n}} = e^{\frac{1}{(n-1)}\ln\left(\frac{2}{1+C_n}\right)} \quad \text{(A.20)}$$

Hence:

$$\ln\left(1+\frac{U}{(n-1)}\right) > \frac{1}{(n-1)}\ln\left(\frac{2}{1+C_n}\right) \quad \text{(A.21)}$$

From the Maclaurin expansion of $\ln(1+x)$, as $x \to 0$, $\ln(1+x) \to x$, hence as $n \to \infty$, we have:

$$U > \ln\left(\frac{2}{1+C_n}\right) \quad \text{(A.22)}$$

We observe that Equation (A.22) holds for any value of $n \geq 2$ and $0 < U \leq 1$, as $x > \ln(1+x)$ for $0 < x \leq 1$.

Substituting for *U* from Equation (A.22) into Equation (A.14) yields the following, in place of Equation (A.17):

$$\max_{\forall t}\left(\frac{h(t)}{t}\right) > \max\left(\left(\frac{1+C_n}{2}\right), \ln\left(\frac{2}{1+C_n}\right)\right) \quad \text{(A.23)}$$

As in Equation (A.17), the left hand term within the max expression of Equation (A.23), is a monotonically increasing function of C_n, whilst the right hand term is a monotonically decreasing function of C_n. Hence the minimum value $L(\infty)$, of the max expression is achieved when these two terms are equal.

$$L(\infty) = \left(\frac{1+C_n}{2}\right) = \ln\left(\frac{2}{1+C_n}\right) \quad \text{(A.24)}$$

This occurs for a value of $L(\infty) = \Omega$. (Where Ω is the mathematical constant defined by $\ln(1/\Omega) = \Omega$. Hence, $\Omega \approx 0.567143$). Finally, $f^{UB}(\infty) = 1/L(\infty) = 1/\Omega$ □

In summary, to guarantee schedulability of any constrained-deadline taskset of cardinality *n* under fixed priority pre-emptive scheduling it is *sufficient* to use a speedup factor equal to the upper bound values $f^{LB}(n)$ given in Table 4 and
Table 5. For small *n*, this represents an improvement over using the exact value for arbitrary *n* ($1/\Omega \approx 1.763223$).

Appendix B:

In this appendix, we prove that the two alternate definitions for the processor speedup factor given in Section 2.5 are equivalent.

Theorem B.1: Definition 2 is equivalent to Definition 1 with fixed priority pre-emptive scheduling as scheduling algorithm *A*.

According to Definition 1, let $f(\Psi)$ be the minimum speedup factor required to make a feasible taskset Ψ schedulable according to fixed priority pre-emptive scheduling.

Lemma B.1: Let *Z* be the set of all feasible tasksets. Note, by Definition 1, *Z* contains at least one taskset that requires the (maximum) processor speedup-factor f^A to be schedulable under fixed priority pre-emptive scheduling. Let $Y \subseteq Z$ be the set of all feasible tasksets that cannot have their task execution times scaled by any factor >1 without becoming infeasible. The set *Y* contains at least one taskset that requires the (maximum) processor speedup-factor f^A.

Proof: We assume (for contradiction) that there is a taskset Ψ that is a member of *Z*, but not a member of *Y* ($\Psi \in Z \cap Y^c$) that has a speedup factor $f(\Psi)$ strictly greater than that of any taskset which is only just feasible, (and therefore a member of *Y*). We note that Ψ can have its execution times scaled by a factor >1 and remain feasible, otherwise it would be a member of *Y*. We now construct a new taskset Φ from taskset Ψ by scaling all of the execution times of the tasks in Ψ by a factor >1, such that the resulting taskset is only just feasible, hence $\Phi \in Y$. From Equation (1), increasing the execution time of tasks cannot make a taskset schedulable under fixed priority pre-emptive scheduling. Hence the speedup factor required by taskset Φ must be at least as large as that required by taskset Ψ (i.e. $f(\Phi) \geq f(\Psi)$) which contradicts the original assumption. Hence there are no tasksets in $Z \cap Y^c$ that require a speedup factor exceeding the maximum speedup factor required by any taskset in *Y*. As there was, by definition, at least one taskset in *Z* that required the maximum processor speedup factor, then there must also be at least one such taskset in *Y* □

Proof (of Theorem B.1): Lemma B.1 shows that there is at least one taskset that is only just schedulable according to an optimal algorithm such as EDF and requires the maximum processor speedup factor, according to Definition 1 to be schedulable under fixed priority pre-emptive scheduling. Let Φ be such a taskset.

Increasing processor speed by a factor f is equivalent to reducing task execution times by the same factor. By definition of the speedup factor $f(\Theta)$,

reducing the execution times of the tasks in each feasible taskset Θ by $f(\Theta)$, transforms that taskset into one that is only just schedulable according to fixed priority pre-emptive scheduling. It follows that to find the maximum processor speedup factor, it suffices to determine the maximum scaling factor by which the execution times of tasks in any taskset *S*, that is only just schedulable according to fixed priority pre-emptive scheduling (i.e. with $\alpha_S = 1$), can be increased, and the taskset remain schedulable according to an optimal scheduling algorithm (e.g. EDF). This is Definition 2 □

References

Audsley N.C., Burns A., Richardson M., Wellings A.J. (1993) "Applying new Scheduling Theory to Static Priority Pre-emptive Scheduling". Software Engineering Journal, 8(5) pp. 284-292.

Baker T.P., (1991). "Stack-based Scheduling of Real-Time Processes." Real-Time Systems Journal (3)1, pp. 67-100.

Baruah S., Burns A. (2006) "Sustainable Scheduling Analysis". In Proceedings of the IEEE Real-Time Systems Symposium, pp. 159-168.

Baruah S., Burns A. (2008) "Quantifying the sub-optimality of uniprocessor fixed priority scheduling." In Proceedings of the IEEE International conference on Real-Time and Network Systems pp. 89-95.

Baruah S.K., Mok A.K., Rosier L.E. (1990b) "Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor". In Proceedings of the IEEE Real-Time System Symposium, pp.182-190.

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

E. Bini, G.C. Buttazzo (2005), Measuring the Performance of Schedulability Tests, Real-Time Systems 30 (1-2), pp. 129-154.

Bini E., Buttazzo G.C., Buttazzo G.M. (2003) "Rate Monotonic Scheduling: The Hyperbolic Bound". IEEE Transactions on Computers, 52(7):933–942.

Dertouzos M.L. (1974) "Control Robotics: The Procedural Control of Physical Processes". In Proceedings of the IFIP congress, pp.807-813.

Fineberg M.S., Serlin O., "Multiprogramming for hybrid computation", In *proceedings AFIPS Fall Joint Computing Conference*, pp 1-13, 1967

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

Kalyanasundaram B., Pruhs K. (1995), "Speed is as powerful as clairvoyance". In Proceedings of the 36th Symposium on Foundations of Computer Science, pp. 214--221.

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

Lehoczky J.P., Sha L., Ding Y. (1989) "The rate monotonic scheduling algorithm: Exact characterization and average case behaviour". In Proceedings of the IEEE Real-Time Systems Symposium, pp. 166–171.

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

Mok A.K., (1983) "Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment," *Ph.D. Thesis*, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Zuhily A., Burns A. (2007) "Optimality of (D-J)-monotonic Priority Assignment". Information Processing Letters. Number 103, pp. 247-250.