Quantifying the Sub-optimality of Uniprocessor Fixed Priority Pre-emptive Scheduling for Sporadic Tasksets with Arbitrary Deadlines

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

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

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

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

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, defined as 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 implicit-deadline tasksets, the speedup factor is $1/\ln(2) \approx 1.44270$. *For constrained-deadline tasksets, the speedup factor is* $1/\Omega \approx 1.76322$. *In this paper, we show that for arbitrary-deadline tasksets, the speedup factor is lower bounded by* $1/\Omega \approx 1.76322$ *and upper bounded by 2. Further, when deadline monotonic priority assignment is used, we show that the speedup factor is exactly 2.*

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 feasible taskset (that was previously schedulable according to an optimal scheduling algorithm) could be guaranteed to be schedulable according to fixed priority pre-emptive scheduling. We refer to this resource augmentation factor as the *processor speedup factor* [14].

In 1973, Liu and Layland [18] considered fixed priority pre-emptive scheduling of synchronous[1] tasksets comprising independent periodic tasks, with bounded execution times, and deadlines equal to their periods. We refer to such tasksets as *implicit-deadline* tasksets. Liu and Layland showed that *rate monotonic* priority ordering (RMPO) is the optimal fixed priority assignment policy for implicit-deadline tasksets, and that using rate monotonic priority ordering, fixed priority pre-emptive scheduling can schedule any implicit-deadline taskset with a total utilisation $U \leq \ln(2) \approx 0.693$.

Liu and Layland 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 $U \leq 1$.

In 1974, Dertouzos [11] showed that EDF is in fact an optimal pre-emptive uniprocessor scheduling algorithm, in the sense that if a valid schedule exists for a taskset, then the schedule produced by EDF will also meet all deadlines.

Combining the result of Dertouzos [11] with the results of Liu and Layland [18] for both EDF and fixed priority pre-emptive scheduling, we can see that the processor speedup factor required to guarantee that fixed priority pre-emptive scheduling can schedule any feasible implicit-deadline taskset is $1/\ln(2) \approx 1.44270$.

Research into real-time scheduling during the 1980's and early 1990's focussed on lifting many of the restrictions of the Liu and Layland task model. Task arrivals were permitted to be sporadic, with known

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

minimal inter-arrival times, (still referred to as periods), and task deadlines were permitted to be less than or equal to their periods (so called *constrained deadlines*) or less than, equal to, or greater than their periods (so called *arbitrary deadlines*).

In 1982, Leung and Whitehead [15] showed that *deadline monotonic*[2] priority ordering (DMPO) 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 in 1986 [13], Lehoczky et al. in 1989 [17], and Audsley et al. in 1993 [1].

In 1990, Lehoczky [16] showed that deadline monotonic priority ordering is not optimal for tasksets with arbitrary deadlines; however, an optimal priority ordering for such tasksets can be determined, in at most $n(n+1)/2$ task schedulability tests, using Audsley's optimal priority assignment algorithm[3] [1].

Exact schedulability tests for tasksets with arbitrary deadlines were developed by Lehoczky [16] in 1990 and Tindell et al. in 1994 [20].

Exact EDF schedulability tests for both constrained and arbitrary-deadline tasksets were introduced by Baruah et al. in 1990 [6], [7].

In 2008, Baruah and Burns [5] showed that the processor speedup factor for constrained-deadline tasksets is lower bounded by 1.5 and upper bounded by 2. In 2009, Davis et al. [10] derived the exact speedup factor for constrained-deadline tasksets; $1/\Omega \approx 1.76322$ (where Ω is the mathematical constant defined by the transcendental equation $\ln(1/\Omega) = \Omega$, hence, $\Omega \approx 0.567143$).

In this paper, we derive the speedup factor for fixed priority pre-emptive scheduling of arbitrary-deadline tasksets. We are able to give an exact speedup factor when deadline monotonic priority assignment is used, and upper and lower bounds assuming an optimal priority assignment.

It is known that an exact condition for the schedulability of a constrained or arbitrary-deadline taskset under an optimal pre-emptive uniprocessor scheduling algorithm, such as EDF [11], is that a quantity referred to as the processor LOAD (see Section 2.3) does not exceed the capacity of the processor (i.e. $\text{LOAD} \leq 1$) [6], [7].

The processor speedup factor derived in this paper shows that every arbitrary-deadline taskset with $\text{LOAD} \leq 0.5$ is guaranteed to be schedulable according to fixed priority pre-emptive scheduling using either deadline-monotonic priority assignment or an optimal priority assignment algorithm.

This result complements the earlier results of Davis et al. [10] 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; and the seminal result of Liu and Layland [18] ($U \leq \ln(2) \approx 0.693$), that applies to implicit-deadline tasksets.

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.

In 1989, Lehoczky et al. [17] 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. 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.

In 2005, Bini and Buttazzo [8] 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 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 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, and recapitulates exact schedulability analysis for both fixed priority and EDF scheduling. Section 3 illustrates the processor speedup factor via a simple example. Section 4

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

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

derives the processor speedup factor required for arbitrary-deadline tasksets under fixed priority pre-emptive scheduling. Section 5 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.

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

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

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \quad (1)$$

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 all possible sequences of task invocations (or jobs) that may be generated by the taskset can be scheduled on the system by the scheduling algorithm without any deadlines being missed.

Under 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.

When a taskset is scheduled according to fixed priorities, task priorities need to be assigned according to some algorithm. Optimal priority assignment algorithms are known for implicit-deadline [18], constrained-deadline [15], and arbitrary-deadline [1] tasksets.

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

A taskset is said to be *feasible* with respect to a given system model if there exists some scheduling algorithm that can schedule all possible sequences of task activations that may be generated by the taskset on that system without missing any deadlines. 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 tasksets compliant with the tasking model described in this section [11]. Least Laxity First is another such optimal algorithm [19].

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

algorithm. A schedulability test that is both sufficient and necessary is referred to as *exact*.

2.2. Schedulability analysis for fixed priority pre-emptive scheduling

In this section, we give a brief summary of Response Time Analysis [2] used to provide an exact schedulability test for fixed priority pre-emptive scheduling of constrained-deadline tasksets. We then recapitulate on response time analysis for arbitrary-deadline tasksets.

First, we introduce the concepts of worst-case response time, synchronous arrival sequence, and busy periods, which are fundamental to response time 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 *synchronous arrival sequence* refers to a pattern of arrival such that all tasks arrive simultaneously, and then subsequently as early as possible given the constraints on minimum inter-arrival times.

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.

The synchronous arrival sequence generates the longest possible priority level-*i* busy period. For constrained-deadline tasksets, the length w_i of this busy period corresponds directly to the worst-case response time of task τ_i. In the remainder of this paper, when we refer to a priority level-*i* busy period, we mean the longest such busy period. Further, when it is clear which priority level is referred to we use the more concise term, busy period.

The busy period comprises two components, the execution time 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.

For constrained-deadline tasksets, the length of the busy period w_i, can be computed using the following fixed point iteration [2], with the summation term giving the interference due to the set of higher priority tasks *hp*(*i*).

$$w_i^{m+1} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i^m}{T_j} \right\rceil C_j \qquad (2)$$

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.

Equation (2) gives an exact schedulability test for the fixed priority pre-emptive scheduling of constrained-deadline tasksets with any fixed priority ordering.

For arbitrary-deadline tasksets, execution of one invocation of a task may not necessarily be complete before the next invocation is released. Hence a number of invocations of task τ_i may be present within the longest priority level-*i* busy period, with earlier invocations delaying the execution of later ones. In general it is therefore necessary to compute the response times of all invocations within the busy period in order to determine the worst-case response time [20].

The length of the busy period $w_i(q)$, starting at the simultaneous arrival of all tasks and extending until the completion of the *q*th invocation of τ_i (where $q = 0$ is the first invocation) is given by the fixed point iteration:

$$w_i^{n+1}(q) = (q+1)C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i^n(q)}{T_j} \right\rceil C_j \qquad (3)$$

Iteration starts with an initial value $w_i^0(q)$, typically $w_i^0(q) = (q+1)C_i$, and ends when either $w_i^{n+1}(q) = w_i^n(q)$ in which case the worst-case response time $R_i(q)$, of invocation q, is given by $w_i^{n+1}(q) - qT_i$ or when $w_i^{n+1}(q) - qT_i > D_i$ in which case invocation q is unschedulable.

Invocation q can only impinge upon the execution of subsequent invocations if its completion occurs after their release. Hence, response times need to be calculated for invocations q=0,1,2,3… until an invocation q is found that completes at or before the earliest possible release of the next invocation q+1, i.e. where: $w_i(q) \le (q+1)T_i$. The worst-case response time of task τ_i is then given by:

$$R_i = \max_{\forall q}(w_i(q) - qT_i) \qquad (4)$$

Again, the task is schedulable provided that $R_i \le D_i$.

Equations (3) and (4) give an exact schedulability test for the fixed priority pre-emptive scheduling of arbitrary-deadline tasksets with any fixed priority ordering.

The exact schedulability test given by Equations (3) and (4) potentially requires the examination of a large number of invocations of the task of interest.

A simpler sufficient schedulability test for a task τ_i in an arbitrary-deadline taskset can be derived by considering the maximum amount of task execution at priority i and higher released within an interval of length D_i starting with simultaneous arrival of all tasks. If all of this execution can be completed by D_i, then this indicates that the length of the longest priority level-*i* busy period is at most D_i, and hence that all invocations of τ_i released in that busy period meet their deadlines, and so τ_i is schedulable. This sufficient schedulability test is given by

Equation (5):

$$\sum_{\forall j=hep(i)} \left\lceil \frac{D_i}{T_j} \right\rceil C_j \le D_i \quad (5)$$

Where *hep*(*i*) is the set of tasks with priorities higher than or equal to *i*.

2.3. Exact schedulability analysis for EDF

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

$$h(t) = \sum_{i=1}^{n} \max\left(0, \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right) C_i \quad (6)$$

Baruah et al [6], [7] 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 given by:

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

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.

$$L = \max\left(D_1, D_2, \ldots D_n, \max_{\forall i}\left((T_i - D_i)\frac{U}{1-U}\right)\right) \quad (8)$$

The only values of *t* that need to be checked in the interval $(0, L]$ are those where the processor LOAD can change, i.e. $\forall i \quad t = kT_i + D_i$ for integer values of *k*.

Significant developments have been made, extending the scope of the schedulability tests for both fixed priority pre-emptive scheduling and EDF; however, these basic forms are sufficient for the purposes of this paper.

2.4. 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.

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

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

In the remainder of the paper, unless otherwise stated, when we refer to the processor speedup factor, we mean the processor speedup factor for fixed priority pre-emptive scheduling using an optimal priority assignment policy.

Definition 2: A taskset is said to be *speedup-optimal* if it requires the processor to be speeded up by the processor speedup factor in order to be schedulable under fixed priority pre-emptive scheduling. Hence for a speedup-optimal taskset Ψ, $f(\Psi) = f^A$.

3. Example

The concept of processor speedup factor defined in the previous section can be illustrated by means of an example.

Consider the arbitrary-deadline taskset *S* comprising the two tasks defined in Table 1. The parameters of these tasks appear to have some unusual values; however, this is because they have been chosen so that the taskset is just schedulable according to EDF, yet requires a speedup factor of 1.8 in order to be schedulable according to fixed priority pre-emptive scheduling, with priorities ordered via deadline monotonic priority assignment.

Table 1

Task	C_i	T_i	D_i
τ_1	1.8	2	16
τ_2	14.4	∞	17

We now show that taskset *S* is schedulable according to EDF

Under EDF scheduling, the processor demand bound function $h(t)$ for taskset *S* is the sum of the processor demand bound functions $h(t, \tau_1)$ and $h(t, \tau_2)$ for tasks τ_1 and τ_2 respectively, where $h(t, \tau_i)$ is the processor demand bound at time *t* for a single task τ_i, given below:

$$h(t, \tau_i) = \max\left(0, \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right) C_i \quad (9)$$

Thus:

$$h(t, \tau_1) = \left\{ \begin{matrix} 0 & t < 16 \\ \left\lfloor \frac{t-16}{2} \right\rfloor 1.8 + 1.8 & t \ge 16 \end{matrix} \right\} \quad (10)$$

as $\lfloor x/y \rfloor \le x/y$, we have:

$$h(t, \tau_1) \le \left\{ \begin{matrix} 0 & t < 16 \\ \frac{1.8(t-16)}{2} + 1.8 & t \ge 16 \end{matrix} \right\} \quad (11)$$

Similarly, the processor demand bound function for task τ_2 is:

$$h(t,\tau_2)=\begin{cases}0 & t<17\\ 14.4 & t\geq 17\end{cases} \qquad (12)$$

Recall that any arbitrary-deadline taskset is schedulable according to EDF, provided that:

$$\text{LOAD}=\max_{\forall t}\left(\frac{h(t)}{t}\right)\leq 1 \qquad (13)$$

Now, given the following:

(i) The value of $h(t)/t$ at times $t=16$, $t=17$, and $t=18$ are 1.8, 16.2 and 18 respectively.

(ii) From Equations (11) and (12), an upper bound on the value of $h(t)/t$ at time $t=18$ is 18.

(iii) From Equation (11), the rate of increase of the upper bound on $h(t)/t$ for $t\geq 18$ is 0.9.

Hence, the maximum value of $h(t)/t$ occurs at time $t=18$. The processor LOAD of taskset *S* is therefore 1, indicating that the taskset is just schedulable according to EDF.

We now consider the schedulability of taskset *S* when scheduled according to fixed priority pre-emptive scheduling, using deadline monotonic priority assignment, on a processor that has been speeded up by a factor of 1.8. The parameters of the taskset on this faster processor are given in Table 2. We refer to this taskset as *V*.

Table 2

Task	C_i	T_i	D_i
τ_1	1	2	16
τ_2	8	∞	17

Figure 1 illustrates the execution of taskset *V* under fixed priority pre-emptive scheduling, assuming a synchronous arrival sequence.

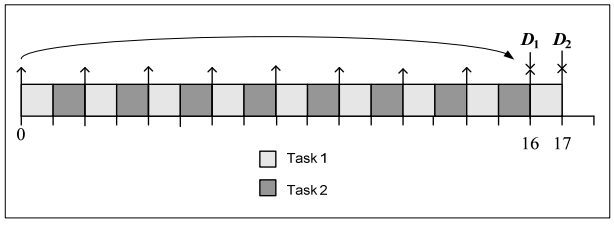

Figure 1

We note that the worst-case response time of task τ_1 is 1 and that of task τ_2 is 16. Taskset *V* is only just schedulable under fixed priority pre-emptive scheduling, using deadline monotonic priority assignment. Any reduction in processor speed would result in the taskset being unschedulable. The processor speedup factor required is therefore 1.8.

4. Processor speedup factor for arbitrary-deadline tasksets

In this section, we derive the exact processor speedup factor required for the (non-optimal) case where deadline monotonic priority ordering is used in conjunction with arbitrary-deadline tasksets. Further, we provide upper and lower bounds on the processor speedup factor required for the general case where an optimal priority assignment algorithm [1] is used to determine task priorities.

4.1. Arbitrary-deadline tasksets with deadline Monotonic priority ordering

Initially, we consider the case of arbitrary-deadline tasksets where task priorities are assigned in deadline monotonic priority order (DMPO). Recall that DMPO is not optimal in this case [16]; nevertheless, fixed priority pre-emptive scheduling using DMPO is a simple combination of scheduling algorithm and priority assignment policy that is used in many real-time systems. We now derive an exact processor speedup factor for this combination.

Lemma 1: An upper bound on the processor speedup factor for fixed priority pre-emptive scheduling of arbitrary-deadline tasksets using deadline monotonic priority assignment is 2.

Proof: Let *S* be any taskset that is schedulable on a processor of unit speed according to an optimal scheduling policy such as EDF.

For each task τ_k, in *S*, consider the processor demand bound during an interval of length $2D_k$. As taskset *S* is schedulable according to EDF, it follows that:

$$s\sum_{i=1}^{n}\max\left(0,\left\lfloor\frac{2D_k-D_i}{T_i}\right\rfloor+1\right)C_i\leq 2D_k \qquad (14)$$

Where *s* = 1 is the speed of the processor.

Next, consider taskset *S* scheduled according to fixed priority pre-emptive scheduling on a processor of speed *s* = 2 using deadline monotonic priority assignment. DMPO implies that $\forall i\leq k \quad D_i\leq D_k$.

From Equation (14) above, assuming speed *s* = 2, and discarding the contribution from all tasks of lower priority than *k* we have:

$$\sum_{i=1}^{k}\max\left(0,\left\lfloor\frac{2D_k-D_i}{T_i}\right\rfloor+1\right)C_i\leq D_k \qquad (15)$$

As $\lfloor x\rfloor+1\geq\lceil x\rceil$ and $\forall i\leq k \quad D_i\leq D_k$ then:

$$\sum_{i=1}^{k}\left\lceil\frac{D_k}{T_i}\right\rceil C_i\leq D_k \qquad (16)$$

Equation (16) is recognisable as the sufficient schedulability test for task τ_k in an arbitrary-deadline taskset *S*, scheduled under fixed priority pre-emptive scheduling (see Equation (4) in Section 2.2). Repeating the above argument for each task τ_k in *S* proves that the taskset is schedulable on a processor of speed 2 under fixed priority pre-emptive scheduling using deadline monotonic priority assignment □

Theorem 1: An exact bound on the processor speedup factor for fixed priority pre-emptive scheduling of arbitrary-deadline tasksets using deadline monotonic priority ordering is 2.

Proof: Consider taskset *V* with the following parameters on a processor of speed f :

τ_1 : $C_1 = 1/2k$, $T_1 = 1/k$, $D_1 = 1$

τ_2 : $C_2 = 1/2$, $T_2 = \infty$, $D_2 = 1 + 1/2k$

where *k* is an integer, and task τ_1 has a higher priority than task τ_2 i.e. deadline monotonic priority ordering. The execution of taskset *V* under fixed priority pre-emptive scheduling is illustrated in Figure 2. (Note the similarity to the taskset used as an example in Section 3).

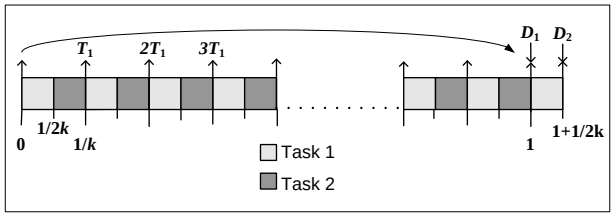

Figure 2

We observe that with fixed priority pre-emptive scheduling, any increase in the execution time of either task will cause task τ_2 to miss its first deadline following simultaneous release of the two tasks.

We now consider the execution of taskset *V* under EDF on a processor of unit speed. Let taskset *S* be formed from taskset *V* by increasing the execution times of tasks τ_1 and τ_2 by a scaling factor f to form tasks τ_1' and τ_2', thus accounting for the reduction in processor speed.

We observe that $f = 2$ is an upper bound on the maximum scaling factor that could possibly result in a schedulable taskset under EDF as this scaling factor results in task τ_1' having a utilisation of 100%.

Under EDF scheduling, the processor demand bound function $h(t)$ for taskset *S* is the sum of the processor demand bound functions $h(t, \tau_1')$ and $h(t, \tau_2')$ for tasks τ_1' and τ_2' respectively.

$$h(t, \tau_1') = \left\{ \begin{matrix} 0 & t < 1 \\ \left\lfloor \dfrac{t - 1 + (1/k)}{(1/k)} \right\rfloor \dfrac{f}{2k} & t \geq 1 \end{matrix} \right\} \qquad (17)$$

as $\lfloor x/y \rfloor \leq x/y$, we have the following upper bound:

$$h(t, \tau_1') \leq \left\{ \begin{matrix} 0 & t < 1 \\ \dfrac{f(t-1)}{2} + \dfrac{f}{2k} & t \geq 1 \end{matrix} \right\} \qquad (18)$$

Similarly, the processor demand bound function for task τ_2' is:

$$h(t, \tau_2') = \left\{ \begin{matrix} 0 & t < 1 + (1/2k) \\ f/2 & t \geq 1 + (1/2k) \end{matrix} \right\} \qquad (19)$$

Recall that any arbitrary-deadline taskset is schedulable according to EDF, provided that:

$$\text{LOAD} = \max_{\forall t} \left(\frac{h(t)}{t} \right) \leq 1 \qquad (20)$$

Now, given the following:

(i) The value of $h(t)/t$ at time $t = 1$ is $f/2k$.

(ii) An upper bound, from Equations (18) and (19), on the value of $h(t)/t$ at time $t = 1 + (1/2k)$ is:

$$\frac{h(1 + (1/2k))}{(1 + (1/2k))} \leq \frac{(f/2) + ((1 + (1/2k)) - 1)(f/2) + (f/2k)}{(1 + (1/2k))}$$

$$= \frac{f(k + (3/2))}{2(k + (1/2))} \qquad (21)$$

(iii) The rate of increase of the upper bound on $h(t)/t$ for $t > 1 + (1/2k)$ is $f/2$ (from Equation (18)).

Then for values of $f \leq 2$, the maximum value of the upper bound on $h(t)/t$ occurs at time $t = 1 + 1/2k$, therefore:

$$\max_{\forall t} \left(\frac{h(t)}{t} \right) = \frac{f(k + (3/2))}{2(k + (1/2))} \overset{\lim_{k \to \infty}}{=} \frac{f}{2} \qquad (22)$$

From Equation (22), the minimum value for the processor LOAD is achieved in the limit as $k \to \infty$, and this value is $f/2$. From Equation (22), for $k = \infty$, taskset *V* is schedulable according to EDF when its task execution times are scaled up by a factor of $f = 2$ to form taskset *S*. Hence taskset *S* requires a processor speedup factor of 2 in order to be schedulable under fixed priority pre-emptive scheduling with deadline monotonic priority ordering. As the processor speedup factor for fixed priority pre-emptive scheduling of arbitrary-deadline tasksets using deadline monotonic priority ordering is also upper bounded by 2 (Lemma 1), the exact processor speedup factor is 2 □

Corollary 1: Taskset *S* defined in the proof of Theorem 1 (with $k = \infty$), is a speedup-optimal taskset for fixed priority pre-emptive scheduling of arbitrary-deadline tasksets using deadline monotonic priority ordering.

It is interesting to note that the speedup-optimal taskset (requiring the largest speedup factor), includes a task τ_1 , with a deadline much larger than its infinitesimal period,

and a task τ_2, with a deadline much smaller than its infinite period.

Theorem 2: An upper bound on the processor speedup factor for fixed priority pre-emptive scheduling of arbitrary-deadline tasksets using an optimal priority assignment algorithm is 2.

Proof: Follows directly from the fact that using an optimal priority assignment algorithm, fixed priority pre-emptive scheduling can schedule any taskset that is schedulable using deadline monotonic priority ordering. Hence the processor speedup factor required can be no greater with optimal priority assignment than the exact processor speedup factor given by Theorem 1 for deadline monotonic priority ordering □

Theorem 3: A lower bound on the processor speedup factor for fixed priority pre-emptive scheduling of arbitrary-deadline tasksets using an optimal priority assignment algorithm is $1/\Omega = 1.76322$.

Proof: Follows directly from the fact that the set of arbitrary-deadline tasksets is a superset of the set of constrained-deadline tasksets, and the proof given by Davis et al. [10] that the exact speedup factor required for constrained-deadline tasksets is $1/\Omega$ □

5. Summary and conclusions

In this paper, we have examined the relative effectiveness of fixed priority pre-emptive scheduling for tasksets with arbitrary deadlines. Our metric for measuring the effectiveness of this scheduling algorithm is a resource augmentation factor known as the processor speedup factor.

The processor speedup factor is defined as the minimum amount by which the processor needs to be speeded up 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.

Table 3 shows the processor speedup factor needed for fixed priority pre-emptive scheduling given the different taskset classifications (implicit-, constrained-, and arbitrary-deadline) and different priority assignment policies. In Table 3, when a single value is shown for both the upper and lower bounds, this implies that the bounds are the same and the value is exact. (Note the results shown are for tasksets of arbitrary cardinality).

Table 3: Fixed priority pre-emptive scheduling processor speedup factors

Taskset constraints [Priority ordering]	Lower Bound	Upper Bound
Implicit-deadline [Optimal (RMPO)]	$1/\ln(2) =$ **1.44269**	
Constrained-deadline [Optimal (DMPO)]	$1/\Omega =$ **1.76322**	
Arbitrary-deadline [Not optimal (DMPO)]	**2**	
Arbitrary-deadline [Optimal algorithm]	$1/\Omega =$ **1.76322**	**2**

In conclusion, the major contributions of this paper are as follows:

- Proving that the exact processor speedup factor for fixed priority pre-emptive scheduling of arbitrary-deadline tasksets with priorities assigned according to deadline monotonic priority assignment is 2.
- Proving that the processor speedup factor for fixed priority pre-emptive scheduling of arbitrary-deadline tasksets with priorities assigned according to Audsley's optimal priority assignment algorithm, is upper bounded by 2 and lower bounded by $1/\Omega = 1.76322$.

The seminal work of Liu and Layland [18] 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 arbitrary-deadline tasksets are scheduled using fixed priority pre-emptive scheduling instead of an optimal algorithm such as EDF. Table 4 summarises the maximum extent of these performance penalties, when deadline monotonic priority assignment is used.

Table 4: Sub-optimality of fixed priority pre-emptive scheduling using deadline monotonic priority assignment

	Optimal (e.g. EDF)	Fixed Priority (DMPO)	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
Arbitrary-deadline	$\text{LOAD} \le 1$	$\text{LOAD} \le 0.5$	2

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 [19]. 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, bounds on the sub-optimality of fixed priority pre-emptive scheduling for uniprocessor systems with arbitrary-deadlines

Future work

Although this paper provides upper and lower bounds, the exact sub-optimality of fixed priority pre-emptive scheduling with respect to *arbitrary-deadline* tasksets assuming optimal priority assignment remains an open question. 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 arbitrary-deadline tasksets. This is also an interesting area for future research.

Acknowledgements

This work was funded in part by the EU FP7 projects Jeopard (project number 216682) and eMuCo (project number 216378).

References

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

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

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

[4] Baruah S., Burns A. "Sustainable Scheduling Analysis". *In Proceedings of the IEEE Real-Time Systems Symposium*, pages 159-168, 2006.

[5] Baruah S., Burns A., "Quantifying the sub-optimality of uniprocessor fixed priority scheduling." *In Proceedings of the IEEE International conference on Real-Time and Network Systems*, pages 89-95, 2008.

[6] Baruah S.K., Mok A.K., Rosier L.E., "Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor". *In Proceedings of the IEEE Real-Time System Symposium*, pages182-190, 1990.

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

[8] Bini E., Buttazzo G.C., "Measuring the Performance of Schedulability Tests", *Real-Time Systems* 30 (1-2), pages 129-154, 2005.

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

[10] Davis R.I., Rothvoß T., Baruah S.K., Burns A., "Exact Quantification of the Sub-optimality of Uniprocessor Fixed Priority Pre-emptive Scheduling." *Real-Time Systems to appear* 2009.

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

[12] Fineberg M.S., Serlin O., "Multiprogramming for hybrid computation". *In Proceedings of AFIPS Fall Joint Computing Conference*, pages 1-13, 1967.

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

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

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

[16] Lehoczky J., "Fixed priority scheduling of periodic task sets with arbitrary deadlines". In *Proceedings 11th IEEE Real-Time Systems Symposium*, pages 201–209, 1990.

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

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

[19] Mok A.K., "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, 1983.

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

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