A Review of Fixed Priority and EDF Scheduling for Hard Real-Time Uniprocessor Systems

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

ABSTRACT

This paper was written to accompany a talk at the ETR Summer School in Toulouse 2013. It provides a background and primer to scheduling and schedulability analysis for hard real-time single processor systems. The paper focuses on the two main scheduling algorithms used: Fixed Priority and Earliest Deadline First. The paper has two broad aims: Firstly to provide a guide to the fundamental results for these real-time scheduling algorithms. Secondly to provide a brief review of extensions aimed at (i) limiting the effects and overheads of pre-emption and (ii) accounting for the effects of pre-emptions; specifically cache related pre-emption delays. The paper also briefly looks back at success stories in real-time scheduling, and forwards at the current hot topics in this research area.

Categories and Subject Descriptors

C.3 [Special-Purpose and Application-Based Systems]: *Real-time and embedded systems*.

General Terms

Algorithms, Performance, Theory, Verification.

Keywords

Real-time; scheduling; schedulability analysis; fixed priority; EDF; single processor; uniprocessor; survey; limited pre-emption; cache-related pre-emption delays.

1 INTRODUCTION

This paper was written to accompany a talk at the ETR Summer School in Toulouse 2013. It aims to provide a brief guide and review of scheduling and schedulability analysis for hard real-time uniprocessor systems. This is a broad field which has been the subject of extensive research, starting in the 1970's and 1980's, gathering pace in the 1990's through to the present day.

In this paper, we necessarily cannot give full coverage of the field; instead, we seek to provide a primer covering the fundamental results, complemented by a brief review highlighting two interesting topics or themes that build upon these foundations. For the interested reader, we provide references to further reading in the form of books, surveys, and the original sources. We hope that readers will be curious to find out exactly how some of the techniques mentioned, but not covered in detail, actually work and so go on to read the cited literature.

2 ADDITIONAL READING

Burns and Wellings Book *"Real-Time Systems and Programing Languages: Ada 95, Real Time Java and Real Time Posix"* [19] provides a useful introduction to real-time scheduling. For an excellent description of early work, the interested reader is also referred to two surveys: *"Fixed Priority Scheduling An Historical Perspective"* [6], and *"Real-Time Scheduling Theory A Historical Perspective"* [62], both published in *Real-Time Systems* which is the main journal covering this field. An up-to-date source of further reading can be found on the IEEE Technical Committee on Real-Time Systems (TCRTS) website via the list of seminal papers recommended for education (see http://tcrts.org/education/seminal-papers/).

3 BACKGROUND

Real-time systems are characterised not only by the need for functional correctness, but also the need for temporal or timing correctness. Real-time systems continually monitor and respond to stimuli from the environment and the physical systems that they control. In order for such systems to behave correctly, they must not only execute the correct computations, but also do so within predefined time constraints. These time constraints are typically expressed in terms of end-to-end *deadlines* on the elapsed time between a stimuli and the corresponding response. Applications in real-time systems may be classified as *hard* real-time, where failure to meet a deadline constitutes a failure of the application; or *soft* real-time, where latency in excess of the deadline leads only to a degraded quality of service. Today, hard real-time systems are found in many diverse application areas including; automotive electronics, avionics, space systems, medical systems, household automation, and robotics.

Many real-time systems are composed of *tasks*. Tasks are software components (programs) that execute in response to some stimulus and output a response. Often the stimulus is periodic (e.g. *time-driven*) thus a *periodic* task might execute once every 50ms. Tasks can also be *event-driven* from some stimulus that does not necessarily occur with a fixed period, but instead has some minimum inter-arrival time, for example an interrupt that occurs from a sensor triggered on each cycle or rotation of an engine. Such tasks are referred to as *sporadic*. Each instance or execution of a task is referred to as a *job* of that task. Thus each task can give rise to a potentially unbounded sequence of jobs.

Time constraints in a real-time system typically put a limit on the maximum permitted *response time* from the arrival of the event (or timer tick) that subsequently leads to the release of a job of a task by the operating system, until the job completes its execution and outputs its response. This time limit is the *relative deadline* (or just *deadline*) of the task. The arrival time of the triggering event or timer tick plus the relative deadline of the task gives the *absolute deadline* of the job. More complex behaviours may be implemented such that the time constraint is from the stimulus for one task, through to the response of a different task. This time interval is referred to as the *end-to-end response time* and it must meet the corresponding *end-to-end deadline* if the system is to operate correctly.

In a single processor system, the processor can only run a single task at any given time, hence there is an important choice to be made as to which job to run when there is more than one job that is ready to execute. The scheduling policy employed determines this choice, which is then implemented by the scheduler, which is part of the real-time operating system.

Scheduling policies may be classified according to whether they make scheduling decisions *online* (depending on the current state of the active jobs) or *offline* (using a pre-computed table). The latter approach is referred to as *static cyclic scheduling*. Static cyclic scheduling was used in early real-time systems, and is still used today in very simple systems, and also in many safety critical systems (e.g. aircraft flight control systems). Static cyclic scheduling lacks flexibility and has a number of other disadvantages stemming from the use of pre-computed tables. For these reasons in has been supplanted in many real-time systems by online scheduling policies. In this paper, we focus soley on online scheduling and in particular two scheduling policies that both use priorities to determine which job to execute.

1. Fixed Priority (FP) scheduling
2. Earliest Deadline First (EDF) scheduling

With fixed priority scheduling, each task is assigned a static priority which is then inherited by all jobs of that task. By contrast, with EDF scheduling the priority of each job is derived from its absolute deadline, such that the job with the earliest absolute deadline has the highest priority. Fixed priority scheduling has fixed task priorities, whereas EDF has fixed job priorities. Other scheduling algorithms such as Shortest Remaining Processing Time First (SRPT) [40], [64] and Least Laxity First (LLF) [60] have dynamic job priorities that change as each job executes.

Scheduling policies are classified as *pre-emptive*, if they allow an already running job to be suspended so that another (higher priority) job may execute, or *non-pre-emptive* if they do not. Thus there are pre-emptive and non-pre-emptive variants of both fixed priority and EDF scheduling.

In hard real-time systems, timing correctness (meeting all deadlines) is an essential aspect of correct system behaviour, hence it is important to be able to ensure before the system is deployed that all deadlines will be met during operation[1] – assuming that the system behaves as expected (i.e. no faults, correct functional behaviour etc.).

Schedulability analysis is the name given to a mathematical approach used to determine offline if deadlines can be guaranteed to be met at runtime.

Schedulability analysis requires information about:

(i) The possible scenarios and patterns of arrivals of the jobs of each task.

(ii) The execution behaviour of the tasks, in particular, an upper bound on the longest time a job of each task could take to execute non-pre-emptively, referred to as the worst-case execution time (WCET), and any interactions between the tasks (e.g. mutually exclusive access to shared resources).

(iii) The scheduling policy used (e.g. fixed priority) and any modifications to this (e.g. when jobs may temporarily execute at a higher priority, due to non-pre-emptive behaviour, or when accessing shared resources etc.).

In this paper, we will assume that an upper bound on the WCET of each task is known. We note that WCET analysis is an active field of research, and the problem of determining an accurate estimate of the WCET is a difficult one, particularly for processors with hardware acceleration features such as pipelines and caches [71].

We now introduce some terminology that is used in research focused on scheduling and schedulability analysis. We define what is meant by *schedulable* and *feasible* task sets, *optimal* scheduling algorithms and *optimal* priority assignment policies, and different types of schedulability test, including *exact* ones.

We say that a task set is *schedulable* with respect to some scheduling algorithm, if all valid sequences of jobs that may be generated by the task set can be scheduled by the scheduling algorithm without any missed deadlines. Further, we say a task set is *feasible* if there exists some scheduling algorithm that can schedule all possible sequences of jobs that may be generated by the task set without missing any deadlines. A scheduling algorithm is said to be *optimal* with respect to a task model if it can schedule all of the feasible task sets that comply with the model.

A schedulability test is said to be *sufficient*, with respect to a scheduling algorithm, if all of the task sets that are deemed schedulable according to the test are in fact schedulable under the scheduling algorithm. Similarly, a schedulability test is said to be *necessary*, if all of the task sets that are deemed unschedulable according to the test are in fact unschedulable under the scheduling algorithm. A schedulability test that is both sufficient and necessary is referred to as *exact*. Thus our aim in schedulability analysis research is to find efficient, exact schedulability tests; however, there is typically a trade-off between the computational complexity of the tests and their effectiveness. Hence sometimes we need to use efficient tests that only give sufficient results rather than exact tests that take too long to compute.

In fixed priority scheduling, a priority assignment policy P is said to be *optimal* with respect to some class of task sets (e.g. those with arbitrary deadlines), and some class of fixed priority scheduling algorithm (e.g. pre-emptive) if there are no task sets in the class that are schedulable under the scheduling

[1] In practice often what is required is an assurance that there is only a very small probability that deadlines will be missed. Probabilistic analysis which seeks to address this more refined view is the topic of another talk at ETR 2013.

algorithm using any other priority ordering policy that are not also schedulable using the priority assignment determined by policy *P*. Using an optimal priority assignment policy we can schedule any task set that is schedulable using any other priority assignment policy; hence in analysis of fixed priority systems, we always aim to find and use optimal priority assignment policies.

The rest of the paper is organised as follows. In Section 4, we outline the fundamental research into schedulability analysis for single processor systems using fixed priority and EDF scheduling. In Section 5 we provide a formal tasking model and describe general forms of schedulability analysis for both fixed priority and EDF scheduling of sporadic task sets. These are the building blocks that researchers working in this area need to be aware of. In Section 6, we provide a guide to research along two themes that build upon the fundamental schedulability analysis research: limiting pre-emption, and accounting for its costs. Section 7 concludes the paper with a brief look at success stories for real-time schedulability analysis and hot topics of future research.

4 FIXED PRIORITY AND EDF SCHEDULING

In this section, we provide an overview of the fundamental work on schedulability analysis for single processor systems using fixed priority and EDF scheduling.

4.1 Pre-emptive scheduling

In 1973, Liu and Layland [56] considered fixed priority pre-emptive scheduling of *synchronous*[2] task sets comprising independent periodic tasks, with bounded worst-case execution times, and deadlines equal to their periods (so called *implicit-deadlines*). Liu and Layland showed that *rate monotonic* priority ordering, where tasks are assigned priorities in order of their periods, with a shorter period implying a higher priority, is the optimal priority assignment policy for such task sets. They further proved that that using rate monotonic priority ordering, fixed priority pre-emptive scheduling can schedule any implicit-deadline task set that has a total utilisation $U \leq \ln(2) \approx 0.693$, where the utilisation of a task is its worst-case execution time divided by its period, and the total utilisation of the task set is simply the sum of the individual task utilisations. Liu and Layland [56] also showed that that EDF can schedule any task set that has a total utilisation $U \leq 1$.

In 1974, Dertouzos [41] showed that pre-emptive EDF is an optimal single processor scheduling algorithm, in the sense that if a valid schedule exists for a task set, then the schedule produced by EDF will also meet all deadlines. Least Laxity First [60] is another such optimal algorithm.

Research into real-time scheduling during the 1980's and early 1990's focused on lifting many of the restrictions of the Liu and Layland task model. Task arrivals were permitted to be sporadic, with known minimal inter-arrival times, (still slightly confusingly referred to in the literature 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 [53] showed that *deadline monotonic* priority ordering, where tasks are assigned priorities in order of their deadlines, with a shorter deadline implying a higher priority, is the optimal priority ordering for constrained-deadline task sets. Exact schedulability tests for pre-emptive fixed priority scheduling of constrained-deadline task sets were introduced by Joseph and Pandya in 1986 [48], Lehoczky et al. in 1989 [55], and Audsley et al. in 1993 [5].

In 1990, Lehoczky [54] showed that deadline monotonic priority ordering is not optimal for task sets with arbitrary deadlines; however, an optimal priority ordering can be determined in this case, in at most $n(n+1)/2$ task schedulability tests using Audsley's optimal priority assignment (OPA) algorithm[3] [4], [7].

Exact schedulability tests for task sets with arbitrary deadlines were developed by Lehoczky [54] in 1990, and Tindell et al. [67] in 1994.

The release jitter problem was first address by Audsley et al. [5] in 1993 for constrained deadline task sets, and later by Tindell et al. [67] for the arbitrary deadline case. Here, jobs of a task may follow a periodic or sporadic arrival pattern, but do not necessarily become ready to execute immediately, instead their release may be delayed by up to a maximum time referred to as the *Release Jitter* of the task. This is a typical behaviour in real systems where there may be some latency in responding to the event that triggers task execution. Zuhily and Burns [75] showed that for constrained deadline task sets with release jitter, Deadline minus Jitter monotonic priority ordering is optimal.

Exact schedulability tests for constrained and arbitrary-deadline task sets scheduled using pre-emptive EDF were introduced by Baruah et al. [10], [11] in 1990. Subsequently, exact tests for EDF have been developed by George and Hermant [43] and Zhang et al. [73], [74] that are more efficient in practice.

4.2 Resource sharing

In pre-emptive systems, mechanisms are required to ensure that jobs access shared resources in mutual exclusion, otherwise data corruption, or erroneous behaviour of hardware devices could ensue. If simple semaphores are used to serialise resource access, then unbounded *priority inversion* can occur. Priority inversion is the name given to the situation where a high priority job attempting to lock a resource has to wait not only for the low priority job that currently holds the resource, but also any jobs of intermediate priority that may pre-empt it. A number of concurrency control protocols have been developed which avoid this problem. These include the Priority Inheritance Protocol (PIP) [63] and the Priority Ceiling Protocol (PCP) [63] originally developed for fixed priority scheduling and the Stack Resource Policy (SRP) [8] introduced by Baker in 1991, which is applicable to both fixed priority and EDF scheduling. SRP is the most effective protocol and has been adopted by the OSEK and Autosar operating system standards.

The Stack Resource Policy [8] associates a pre-emption level with each resource. In EDF (FP) scheduling, this pre-emption

[2] A task set is referred to as synchronous if all of the tasks can share a common release time.

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

level corresponds to the shortest relative deadline (highest priority) of any task that accesses the resource. At run-time, when a job locks a resource, its current priority is saved (on a stack), and it continues to execute at the higher of its previous priority and the pre-emption level of the resource. On unlocking the resource the job returns to its previous priority (obtained from the stack). Thus SRP allows resource locking to be properly nested.

The Stack Resource Policy [8] has a number of important properties:

- Once a job starts executing it never has to wait to access a resource. This means that with SRP, resource accesses add no additional context switches, and single stack execution is possible saving memory.
- No job *j* can be delayed from executing by lower pre-emption level jobs for longer than the maximum single resource access time of any such lower priority job, where the resource is shared with other jobs of the same or a higher pre-emption level than job *j*.
- The system is deadlock free.

These properties strictly bound the *blocking time* for which higher priority jobs can be delayed by the execution of lower priority jobs.

Baker [8] initially provided a sufficient schedulability test for EDF and SRP. Later, in 1996 Spuri [65] modified the exact test for EDF scheduling to account for resource locking under SRP. Exact tests for fixed priority scheduling with either PCP or SRP where introduced by Audsley et al. in 1993 [5] for constrained deadline task sets, and Tindell et al. [67] in 1994 for arbitrary deadline task sets.

4.3 Non-pre-emptive scheduling

In 1980, Kim and Naghibdadeh [50], and in 1991, Jeffay et al. [47], gave exact schedulability tests for implicit-deadline task sets under non-pre-emptive EDF scheduling. These tests were extended by George et al. [44] in 1996, to the general case of sporadic task sets with arbitrary deadlines.

While pre-emptive EDF is an optimal single processor scheduling algorithm, in the non-pre-emptive case no *work-conserving* algorithm is optimal. (A work-conserving scheduling algorithm is one that does not idle the processor when there are tasks ready to be executed). This is because in general, with non-pre-emptive scheduling, it is necessary to insert idle time to achieve a feasible schedule. The interested reader is referred to [44] for examples of this behaviour. In 1995, Howell and Venkatrao [46] showed that for non-concrete strictly periodic task sets, where the times at which each task may be first released are unknown, the problem of determining a feasible non-pre-emptive schedule is NP hard. Further, they showed that for sporadic task sets, no optimal on-line inserted idle time algorithm can exist. In other words, clairvoyance is needed to determine a feasible non-pre-emptive schedule.

While no work-conserving algorithm is optimal in the strong sense that it can schedule any task set for which a feasible non-pre-emptive schedule exists; in 1995, George et al. [45] showed that non-pre-emptive EDF is optimal in the weak sense that it can schedule any task set for which a feasible work-conserving, non-pre-emptive schedule exists.

For fixed priority non-pre-emptive scheduling of arbitrary-deadline task sets, George et al. [44] derived an exact schedulability test based on the approach of Tindell et al. [67] for the pre-emptive case. George et al. showed that unlike in the pre-emptive case, deadline monotonic priority ordering is not optimal for constrained-deadline task sets scheduled non-pre-emptively. Further, they showed that Audsley's optimal priority assignment algorithm [7] is applicable, and can be used to determine an optimal priority ordering in this case.

Subsequent research by Bril et al. [17] has refined exact analysis of non-pre-emptive fixed priority scheduling, correcting issues of both pessimism and optimism, and extending the schedulability tests to co-operative scheduling where each task is made up of a number of non-pre-emptive regions. There are various approaches that allow limited pre-emption and thus represent a compromise between fully pre-emptive and fully non-pre-emptive scheduling. We return to these in Section 6.

5 SCHEDULABILITY TESTS

In this section we first describe the sporadic task model, which is commonly used in real-time scheduling research, and forms the basis for the schedulability analysis that we then recapitulate. We also provide details of the defacto standard notation sometimes referred to as *Burns' Standard Notation* [32], which is used in this area.

5.1 Task model and notation

We consider the scheduling of a set of sporadic tasks (or *task set*) on a single processor. Each task set comprises a static set of *n* tasks ($\tau_1..\tau_n$), where *n* is a positive integer. Without loss of generality, we assume that the index *i* of task τ_i also represents the task priority used in fixed priority scheduling, hence τ_1 has the highest fixed-priority, and τ_n the lowest.

We use the notation $hp(i)$ (and $lp(i)$) to mean the set of tasks with priorities higher than (lower than) *i*, and the notation $hep(i)$ (and $lep(i)$) to mean the set of tasks with priorities higher than or equal to (lower than or equal to) *i*.

Each task τ_i is characterized by its bounded worst-case execution time C_i, minimum inter-arrival time or period T_i, relative deadline D_i, and release jitter J_i. Each task τ_i therefore gives rise to a potentially unbounded sequence of *jobs*, each of which has an execution time upper bounded by C_i, an arrival time at least T_i after the arrival of the previous job of the same task, and an absolute deadline that is D_i after its arrival. Due to release jitter, release of each job may be delayed for a variable amount of time from 0 to J_i after its arrival.

In an *implicit-deadline* task set, all tasks have $D_i = T_i$. In a *constrained-deadline* task set, all tasks have $D_i \le T_i$, while in an *arbitrary-deadline* task set, task deadlines are independent of their periods. The set of arbitrary-deadline task sets is therefore a superset of the set of constrained-deadline task sets, which is itself a superset of the set of implicit deadline task sets.

The *worst-case response time* R_i of a task τ_i is given by the longest possible time from the arrival of the task until it completes execution. Thus task τ_i is schedulable if and only if $R_i \le D_i$, and a task set is schedulable if and only if $\forall i \quad R_i \le D_i$. The *utilization* U_i, of a task is given by its execution time divided by its period ($U_i = C_i / T_i$). The total utilization *U*, of a task set is the sum of the utilizations 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 unknown a priori, hence the tasks may share a common release time.
- The tasks may access mutually exclusive shared resources according to the Stack Resource Policy [8]. Thus each task τ_i may be subject to a maximum amount of *blocking* B_i that equates to the longest time that a job of a task of lower priority (pre-emption level) may lock a resource that is shared with tasks of the same or higher priority (pre-emption level) than task τ_i.
- The tasks are otherwise independent, and do not voluntarily suspend themselves.

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

Under pre-emptive EDF scheduling, at any given time the ready job with the earliest absolute deadline is selected for execution by the processor. In contrast, under pre-emptive fixed priority scheduling, at any given time the earliest released unfinished job of the highest priority ready task is selected for execution.

Similarly, under non-pre-emptive EDF, at any given time when there is no currently executing job then the ready job with the earliest absolute deadline is selected for execution by the processor. Under non-pre-emptive fixed priority scheduling, at any given time when there is no currently executing job then the earliest released unfinished job of the highest priority ready task is selected for execution.

5.2 Analysis for pre-emptive FP scheduling

We now give a summary of Response Time Analysis [5] used to provide an exact schedulability test for fixed priority pre-emptive scheduling of constrained-deadline task sets. We then recapitulate the somewhat more complex response time analysis for arbitrary-deadline task sets [67].

First, we introduce the concepts of a *critical instant*, and *busy periods*, which are fundamental to response time analysis.

The *critical instant* for a task τ_i is defined as a point in time when the arrival of a job of that task will have the longest possible response time, given any valid pattern of arrivals and execution behaviour of the tasks in the task set.

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

Under fixed priority pre-emptive scheduling, the critical instant for a task τ_i occurs within the longest priority level-*i* busy period, which is associated with an arrival sequence such that the first jobs of task τ_i and all higher priority tasks are released simultaneously after their maximum permitted release jitter, and subsequent jobs of the same tasks are released as soon as possible thereafter. For tasks with arbitrary deadlines and large release jitter, this means that more than one job of the task may be released at the same time. In this case, we assume that jobs of the same task are nevertheless released in order of their arrival, and that a later arriving job cannot overtake an earlier arrival and so be released first. Further, we assume that later arriving jobs of the same task effectively have lower priority, and are executed after earlier arrivals. Finally, we assume that just prior to the initial simultaneous release, a lower priority task locks the resource resulting in the maximum blocking time B_i.

For a constrained deadline task τ_i, the worst-case response time R_i corresponds to its release jitter plus the length of the priority level-*i* busy period described above. The busy period comprises three components, (i) the blocking time B_i, (ii) the execution time of the task itself C_i, and (iii) so called *interference*, equal to the time for which task τ_i is prevented from executing by all higher priority tasks. The length of the busy period w_i, can be computed using the following fixed point iteration [5], with the summation term giving the interference due to the set of higher priority tasks *hp*(*i*).

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

Iteration starts with an initial value w_i^0, typically $w_i^0 = B_i + C_i$, and ends when either $w_i^{m+1} = w_i^m$ in which case the worst-case response time R_i is given by $w_i^{m+1} + J_i$, or when $w_i^{m+1} > D_i - J_i$ in which case the task is unschedulable. The fixed point iteration is guaranteed to converge provided that the overall task set utilisation is less than or equal to 1. We note that iteration may be speeded up using the techniques described in [33].

Equation (1) provides an exact schedulability test for the fixed priority pre-emptive scheduling of constrained-deadline task sets with *any* fixed priority ordering.

For fixed priority pre-emptively scheduled systems, where task deadlines are *arbitrary*, execution of one job of a task may not necessarily be completed before the next job is released. Hence a number of jobs of task τ_i may be present within a priority level-*i* busy period, with earlier jobs delaying the execution of later ones. In this case, the critical instant and hence the worst-case response time for task τ_i occurs for some job released within the priority level-*i* busy period corresponding to the sequence of releases previously described, but not necessarily the first job. The length L_i of this longest priority level-*i* busy period can be found via the following fixed point iteration:

$$L_i^{m+1} = B_i + \sum_{\forall j \in hep(i)} \left\lceil \frac{L_i^m + J_j}{T_j} \right\rceil C_j \qquad (2)$$

Iteration starts with an initial value guaranteed to be no larger than the minimum solution, for example $L_i^0 = C_i$, and ends when $L_i^{m+1} = L_i^m$. The number of jobs Q_i of task τ_i in the busy period is given by:

$$Q_i = \left\lceil \frac{L_i + J_i}{T_i} \right\rceil \qquad (3)$$

In general it is necessary to compute the response times of all jobs of a task τ_i within the longest priority level-*i* busy period in order to determine the task's worst-case response time. The *completion time* $W_{i,q}$ of the qth job (where $q = 0$ is the first job) of task τ_i, with respect to the start of the busy period, is given by the following fixed point iteration:

$$w_{i,q}^{m+1} = B_i + (q+1)C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_{i,q}^m + J_j}{T_j} \right\rceil C_j \quad (4)$$

Iteration starts with an initial value $w_{i,q}^0$, typically $w_{i,q}^0 = B_i + (q+1)C_i$, and ends when either $w_{i,q}^{m+1} = w_{i,q}^m$ in which case, $W_{i,q} = w_{i,q}^{m+1}$ or when $w_{i,q}^{m+1} - qT_i > D_i - J_i$ in which case job q, and hence task τ_i is unschedulable.

To find the worst-case response time of task τ_i, completion times $W_{i,q}$ need to be calculated for jobs $q = 0,1,2,3,...Q_i - 1$. The worst-case response time of task τ_i is then given by:

$$R_i = \max_{\forall q=0,1,2...Q_i-1} (W_{i,q}^p - qT_i + J_i) \quad (5)$$

Task τ_i is schedulable provided that $R_i \le D_i$.

Equations (4) and (5) give an exact schedulability test for the pre-emptive fixed priority scheduling of arbitrary-deadline task sets with any specified priority ordering.

5.3 Analysis for non-pre-emptive FP scheduling

We now recapitulate Response Time Analysis used to provide an exact schedulability test for fixed priority non-pre-emptive scheduling of arbitrary deadline task sets. This analysis has similarities to the pre-emptive case, but also some key differences.

George et al. [44] and Bril et al. [17] showed that for fixed priority non-pre-emptive scheduling, the longest response time of a job of task τ_i occurs for some job of that task within a priority level-*i* busy period starting with the simultaneous release of jobs of task τ_i and all higher priority tasks that have been subject to the maximum release jitter, with subsequent jobs of those tasks released as soon as possible. Further, the minimum possible amount of time Δ prior to the initial simultaneous release, a lower priority task τ_k is released, and this task has the longest execution time of any such lower priority task. (Note that in the following, we assume discrete time and so $\Delta = 1$).

The worst-case length of the priority level-*i* busy period is again given by the minimum solution to (2), with B_i defined to be the longest time that task τ_i and any higher priority tasks can be blocked from executing by the non-pre-emptive execution of a lower priority task:

$$B_i = \max_{\forall k \in lp(i)} (C_k - 1) \quad (6)$$

Note in (6) the maximum of an empty set is assumed to be zero. The number of jobs Q_i of task τ_i in the active period is given by:

$$Q_i = \left\lceil \frac{L_i + J_i}{T_i} \right\rceil \quad (7)$$

The *start time* $W_{i,q}$ of the *q*th job (where $q = 0$ is the first job) of task τ_i measured with respected to the initial simultaneous release is given by the minimum solution to the following fixed point iteration:

$$w_{i,q}^{m+1} = B_i + qC_i + \sum_{\forall j \in hp(i)} \left(\left\lfloor \frac{w_{i,q}^m}{T_j} \right\rfloor + 1 \right) C_j \quad (8)$$

Iteration starts with an initial value $w_{i,q}^0$, typically $w_{i,q}^0 = B_i + qC_i$, and ends when either $w_{i,q}^{m+1} = w_{i,q}^m$ in which case $W_{i,q} = w_{i,q}^{m+1}$, or when: $w_{i,q}^{m+1} + C_i - qT_i > D_i - J_i$ in which case job q, and hence task τ_i is unschedulable.

To find the worst-case response time, the start times $W_{i,q}$ need to be calculated for jobs $q = 0,1,2,3,...Q_i - 1$. The worst-case response time of task τ_i is then given by:

$$R_i = \max_{\forall q=0,1,2...Q_i^{NP}-1} (W_{i,q}^P + C_i - qT_i + J_i) \quad (9)$$

Task τ_i is schedulable provided that $R_i \le D_i$.

Equations (8) and (9) give an exact schedulability test for non-pre-emptive fixed priority scheduling of arbitrary-deadline task sets with any specified priority ordering.

The exact analysis for both pre-emptive and non-pre-emptive fixed priority scheduling is pseudo-polynomial in complexity. In practice, the techniques described above based on fixed point iteration are fast enough to be used to determine task set schedulability, and even to perform sensitivity analysis by varying task parameters and determining if the task set remains schedulable. In cases where a significant trade-off between accuracy and speed is required, then Davis and Burns [35] provide response time upper bounds for both pre-emptive and non-pre-emptive cases that are linear in complexity and can be computed in $O(n)$ time for all n tasks in a task set.

5.4 Priority assignment

The schedulability tests given above for both fixed priority pre-emptive and non-pre-emptive scheduling are agnostic as far as the priority assignment policy is concerned; they work with any given assignment. However, priority assignment can have a very large effect on schedulability as shown by Davis et al. [30], hence it is important whenever possible to use an optimal priority assignment policy.

Audsley's Optimal Priority Assignment (OPA) algorithm [4], [7], reproduced below, provides an optimal priority assignment for arbitrary-deadline sporadic task sets with respect to both pre-emptive and non-pre-emptive fixed priority scheduling.

This greedy approach requires at most $n(n+1)/2$ task schedulability tests to determine a schedulable priority ordering whenever such an ordering exists. This compares favourably with the $n!$ tests that would otherwise be required, assuming a brute force approach that checks every possible combination of priorities.

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

Algorithm 1: Optimal Priority Assignment Algorithm

While it provides an optimal priority assignment, the OPA algorithm has one significant drawback: it does not make any choice about priority ordering other than to ensure schedulability. Thus using the OPA algorithm the priority assignment obtained can leave the task set of the edge of un-schedulability, with any small increase in the execution time of a task resulting in a deadline miss. Davis and Burns addressed

this issue with the Robust Priority Assignment algorithm (RPA) [34]. This algorithm is based on Audsley's OPA algorithm; however, it makes use of a scaling factor α which is used to characterise additional interference. This additional interference may take a very general form provided that it is monotonically increasing with the priority level, and the time interval considered. For example the scaling factor α might characterise an increase in the execution time or one, or all of the tasks, or of some additional interference at the highest priority level. Using an appropriate choice of additional interference, the RPA algorithm provides a priority assignment that is not only optimal, but also as robust as possible to increases in interference. The RPA algorithm is shown below.

```
for each priority level k, lowest first {
    for each unassigned task τ {
         binary search for the largest value of α
         for which task τ is schedulable at
         priority k
    }
    if no tasks are schedulable at priority k {
         return unschedulable
    }
    else {
         assign the schedulable task that tolerates
         the max α at priority k to  priority k
    }
}
return schedulable
```

Algorithm 2: Robust Priority Assignment Algorithm

For more information on priority assignment, the interested reader is directed to the presentation that accompanied the keynote talk *"Getting ones priorities right"* at RTNS 2012 [26].

5.5 Analysis for pre-emptive EDF

We now recapitulate schedulability analysis for arbitrary-deadline task sets under pre-emptive EDF. In the analysis of EDF scheduling, we use the concept of *processor demand. bound*. This is defined for an interval of length *t*, as the maximum possible processor demand from jobs with both their release times and their deadlines in the interval. A necessary condition for schedulability is that for any interval of length *t*, the processor demand bound must not exceed the length of the interval. Otherwise the processor could not possibly complete all of the jobs that are released in the interval and also need to finish within the interval.

Exact tests for EDF were first given by Baruah et al [10], [11]. Below, we provide extended versions, including blocking and release jitter due to Spuri [65] and Zhang and Burns [74].

The processor demand bound function $h(t)$ is given by:

$$h(t) = \sum_{i=1}^{n} \max\left(0, \left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor + 1\right) C_i \qquad (10)$$

A task set is schedulable under pre-emptive EDF if and only if $U \leq 1$ and the processor $LOAD$ is ≤ 1 where the processor $LOAD$ is given by:

$$LOAD = \max_{\forall t}\left(\frac{h(t) + b(t)}{t}\right) \qquad (11)$$

where resource access is accounted for by the blocking term:

$$b(t) = \max\left(C_{a,k} \mid D_a - J_a > t, D_k - J_k \leq t\right) \qquad (12)$$

In which $C_{a,k}$ is the maximum amount of time that a task τ_a may spend accessing a resource also used by some task τ_k with a higher pre-emption level. (Note that the pre-emption level corresponds to $D_k - J_k$).

If $LOAD \leq 1$ for all values of t in the interval $(0, L]$, where L is defined as follows, then the task set is schedulable.

$$L = \max((D_1 - T_1 - J_1), ...(D_{n,} - T_{n,} - J_{n,}), \frac{\sum_{\forall i}(T_i + J_i - D_i)U_i}{1 - U}) \qquad (13)$$

Note the length L of the interval that needs to be examined is also limited to the length of the longest priority level-n busy period, which may be computed using (2) and should be used instead of (13) in the case that $U = 1$. The only values of t that need to be checked in the interval $(0, L]$ are those where the $LOAD$ can change, i.e. $\forall i \quad t = kT_i + D_i - J_i$ for integer values of k. However, there can still be a very large number of such values that need to be checked.

Zhang and Burns [73], [74] used the fact that $h(t) + b(t)$ is monotonically non-decreasing with respect to t as the basis for a highly efficient exact schedulability test for pre-emptive EDF, called Quick Processor-demand Analysis (QPA). The QPA algorithm is reproduced below. In practice, this algorithm typically checks a far small set of values in order to ascertain schedulability than the classical method.

***QPA schedulability test for pre-emptive EDF*:** An arbitrary deadline sporadic task set is schedulable under pre-emptive EDF scheduling if and only if $U \leq 1$ and the result of the following iterative algorithm is $h(t) + b(t) \leq DJ_{\min}$, where $DJ_{\min}$ is the smallest deadline minus jitter of any of the tasks, and d_i is an absolute task deadline, i.e. $\forall i \quad d_i = kT_i + D_i - J_i$ for integer values of k.

```
1   t = max{d_i | d_i ≤ L}
2   while( h(t) + b(t) ≤ t ∧ h(t) + b(t) > DJ_min ) {
3       if( h(t) + b(t) < t )
4           { t = h(t) + b(t) }
5       else
6           { t = max{d_i | d_i < t} }
7   }
8   if( h(t) + b(t) ≤ DJ_min ) ⇒ task set is schedulable
9   else ⇒ task set is unschedulable
```

Algorithm 3: QPA for EDF

5.6 Analysis for non-pre-emptive EDF

George et al. [44] extended the schedulability test for pre-emptive EDF to the non-pre-emptive case via the use of a revised blocking factor $b(t)$.

$$b(t) = \max_{\forall i: D_i - J_i > t}(C_i - 1) \qquad (14)$$

Note in (14) the maximum of an empty set is assumed to be zero.

Effectively, non-pre-emptive EDF scheduling may be considered a special case of resource sharing in pre-emptive EDF and thus the QPA schedulability analysis (Algorithm 3) may also be used in this case [74].

5.7 Comparisons between FP and EDF

Fixed priority and EDF scheduling are in some respects competing techniques. A theoretical comparison between the effectiveness of fixed priority and EDF scheduling has been derived by considering the *speedup factor*: the maximum processor speed necessary so that fixed priority scheduling can schedule any task set that is schedulable on a processor of unit speed under EDF. This gives a measure of the worst-case sub-optimality of fixed priority scheduling.

Liu and Layland's utilisation bounds [56] effectively show that the speedup factor required to guarantee that pre-emptive fixed priority scheduling can schedule any implicit-deadline task set schedulable by EDF is $1/\ln(2) \approx 1.44270$. Davis et al. [37] showed that the exact speedup factor for the constrained deadline case is $1/\Omega \approx 1.76322$ (where Ω is the mathematical constant defined by the transcendental equation $\ln(1/\Omega) = \Omega$, hence, $\Omega \approx 0.567143$). Further, for the arbitrary deadline case [38], the speedup factor is lower bounded by $1/\Omega$ and upper bounded by 2. These upper and lower bounds also hold for a comparison between non-pre-emptive fixed priority and non-pre-emptive EDF scheduling [39]. It remains an interesting open problem what the exact speedup factors are in these cases.

The average case sub-optimality of pre-emptive fixed priority scheduling has also been investigated by Lehoczky et al. [55] who used task set average *breakdown utilisation* as a metric. The breakdown utilisation is obtained by scaling task execution times until the task set is only just schedulable and then recording the utilisation. For large implicit-deadline task sets they found that the breakdown utilisation for is approx. 88%, corresponding to a penalty of approximately 12% in processing capacity compared to pre-emptive EDF. Bini and Buttazzo [16] later showed that breakdown utilisation suffers from a bias which tends to penalise fixed priority scheduling by favouring task sets where the utilisation of individual tasks is similar. They introduced an *optimality degree* metric which gives a fairer comparison and shows that the average penalty for using fixed priority-pre-emptive scheduling is significantly smaller.

In the paper "*Rate Monotonic vs. EDF: Judgment Day*", Buttazzo [23] provided a comprehensive comparison and review of the advantages and disadvantages of fixed priority versus EDF scheduling policies; looking at factors such as implementation complexity, runtime overhead, number of pre-emptions, robustness during overload, jitter on task execution, and responsiveness when handling aperiodic (soft real-time) tasks. While EDF has many advantages, it is notable that to this day, the overwhelming majority of commercial Real-Time Operating Systems implement fixed priority rather than EDF scheduling. This is done for reasons of simplicity and lower overheads: in particular, the number of priority levels is often limited allowing a bit-map representation of the ready queue to be used, resulting in $O(1)$ queuing and de-queuing operations.

6 BUILDING ON THE FUNDAMENTALS

In the previous section, we recapitulated schedulability analysis for fixed priority and EDF, pre-emptive and non-pre-emptive scheduling. In the majority of theoretical work on real-time scheduling, the overheads of pre-emption are assumed to be negligible. Indeed this assumption is made in the analysis presented so far. In real systems; however, the overheads of pre-emption can be large and have a significant impact on system schedulability.

There are two main threads of research that address this problem. Firstly, work that seeks to minimise the number of pre-emptions via the use of fixed or floating non-pre-emptive regions (e.g. co-operative scheduling), or via pre-emption thresholds, and non-pre-emption groups, where tasks compete for the processor at a different priority to the one at which they actually execute. Secondly, work that seeks to account for and reduce the cost of pre-emptions, in particular cache related pre-emption delays, via the integration of CRPD analysis into schedulability tests, and via manipulating task layouts and cache partitioning to improve schedulability.

6.1 Limiting pre-emption

Pre-emptive and non-pre-emptive scheduling can be considered as two extremes. It may be argued that fully pre-emptive scheduling allows too many pre-emptions, which can have a high cost if permitted at any arbitrary point. By contrast, non-pre-emptive scheduling permits no pre-emptions, and can have a severe effect on schedulability when some tasks have much shorter deadlines than others. Indeed, with just two tasks one of which has an execution time that is longer than the other's deadline, then non-pre-emptive scheduling is simply not viable.

Limited pre-emption scheduling is the general term given to a collection of different scheduling policies aimed at reducing the number of pre-emptions, and hence their overall cost, while ensuring that the additional blocking introduced does not make the system unschedulable.

The first approach is *deferred pre-emption*. Two different models of scheduling with deferred pre-emption have been developed in the literature. In both cases, non-pre-emptive regions are used to limit pre-emption. This is particularly effective towards the end of a task's execution where a *final non-pre-emptive region* may prevent any further context switches and consequent pre-emption related overheads before the task completes.

In the *fixed* model, introduced by Burns in 1994 [20] for fixed priority scheduling, the location of each non-pre-emptive region is statically determined prior to execution. Pre-emption is then only permitted at pre-defined locations in the code of each task, referred to as *pre-emption points*. This method is also referred to as *co-operative scheduling*, as the tasks co-operate, providing re-scheduling / pre-emption points to improve schedulability. Exact schedulability analysis for co-operative fixed priority scheduling was derived by Bril et al. in 2009 [17]. Since then, Bertogna et al. have integrated pre-emption costs into analysis of the fixed model, considering both fixed [13] and variable [14] pre-emption costs, for both EDF and fixed scheduling. Bertogna et al. [15], also derived a method of computing the optimal length of the final non-pre-emptive region of each task in order to maximize schedulability for a given priority assignment. Finally, in 2012, Davis and Bertogna [29] introduced an optimal algorithm that is able to find a schedulable combination of priority assignment and final-non-pre-emptive region lengths whenever such a schedulable combination exists. This algorithm improves upon the schedulability that can be obtained by either fully pre-emptive or fully non-pre-emptive fixed priority scheduling.

In the *floating* model of non-pre-emptive regions, introduced by Baruah in 2005 [12] for EDF scheduling, an upper bound is given on the length of the longest non-pre-emptive region of each task. However, the location of each non-pre-emptive region is not known a priori and may vary at run-time, for example under the control of the operating system. Analysis for the floating model under fixed priority scheduling was given by Yao et al. [72].

Alternative approaches to limiting pre-emption include *Pre-emption Thresholds* (FPTS) [69], [61] and *Non-pre-emption Groups* [28] in which each task has a base priority at which it competes for the processor, and a *threshold* priority at which it executes, thus limiting pre-emption to those tasks with a base priority higher than the threshold. In [69], [61] Saksena and Wang attempted to derive an integrated approach to priority and pre-emption threshold assignment, but did not succeed in finding an optimal algorithm with less than exponential complexity. Research by Bril et al. [18] in 2012 combines the ideas of deferred pre-emption and pre-emption thresholds, generalising both into a single scheme with pre-emption thresholds between a set of sub-jobs which execute non-pre-emptively.

For further information on limited pre-emption scheduling the reader is referred to the survey in [22].

6.2 Accounting for pre-emption costs

There is a runtime overhead when the processor switches from the execution of one job to another. There are various components that make up this overhead. These include:

- The cost of scheduler operations (e.g. run-queue manipulation, timer interrupts etc.).
- The cost of the context switch from one task to another (saving and restoring registers etc.).
- For processors with cache, the *cache related pre-emption delay* (CRPD).

With careful RTOS design, the first two components can often be tightly bounded and either included in the execution time of each task, or modelled as virtual high priority tasks. Further, the additional blocking effects caused by the operating system can also be modelled and included in the analysis – see for example the work of Whitham et al. [70]. Cache related pre-emption delays can however be a substantial proportion of a tasks' execution time, and are highly dependent on both the pre-empting task and the pre-empted task. Cache is fast local memory, which is used to store instructions (and data) improving access times and hence bridging the large gap that exists between the speed of main memory and the speed of high performance processors. Memory requests that can be serviced by the cache can be significantly faster (10 to 100 times faster in some cases) than accesses that have to go to main memory. However, cache is typically much smaller than main memory and so not all of the instructions and data needed by all tasks can be held in cache at the same time.

Analysis of cache related pre-emption delays uses the concept of *useful cache blocks* (UCBs) and *evicting cache blocks* (ECBs) based on the work by Lee et al. [52]. The ECBs for a task are the cache blocks that may be loaded into cache by the task during its execution. Whereas, UCBs are blocks that are not only loaded into cache by the task, but are certain to be reused before potentially being evicted by the task, not counting evictions from pre-empting tasks. If a UCB is evicted by a pre-empting task, then an additional delay may be introduced if the UCB needs to be re-loaded when it otherwise would not have been. This is termed a cache related pre-emption delay.

Depending on the approach used, CRPD analysis combines information about the UCBs belonging to the pre-empted task(s) with the ECBs of the pre-empting task(s). Using this information, the total number of cache blocks that are evicted, which must then be reloaded after pre-emption, can be calculated and combined with the cost of reloading a block to give the CRPD.

A number of approaches have been developed for calculating the CRPD under pre-emptive fixed priority scheduling. These include the UCB-Only approach of Lee et al. [52], which considers only the pre-empted task(s), and the ECB-Only approach of Busquets et al. [21] which considers only the pre-empting task. Approaches that consider both pre-empted and pre-empting tasks include the UCB-Union approach of Tan and Mooney [66] and the ECB-Union approach of Altmeyer *et al.* [1] and their multi-set extensions [3].

Until recently, there has been significantly less work in developing CRPD analysis for pre-emptive EDF scheduling. In 2007, Ju et al. [49] considered the intersection of the pre-empted task's UCBs with the pre-empting task's ECBs; however, this method has significant pessimism as each pair of tasks is considered separately. In 2013, Lunniss et al. [58] adapted the state-of-the-art approaches [1], [3] for calculating CRPD for fixed priority scheduling to work with EDF.

Cache related pre-emption delays can have a significant effect on schedulability. They can vary substantially depending on how the code of tasks is laid out in memory and hence how it is mapped to the cache, thus providing scope for optimising schedulability via manipulating the position of tasks in memory [59].

One way of eliminating CRPDs is to partition the use of cache so that each task has its own cache partition, hence ensuring that the ECBs of a pre-empting task do not overlap with the UCBs of any pre-empted task. However, such partitioning comes at a price. The smaller cache partition allocated to each task increases the non-pre-emptive WCET of the task. Recent work by Altmeyer at el. [2] has investigated this trade-off concluding that in many cases, even optimal cache partitioning is less effective than sharing the cache and accounting for CRPD via integrated analysis [1], [3].

7 SUMMARY AND CONCLUSIONS

In this paper, we have provided a guide to the fundamental schedulability analysis results for pre-emptive and non-pre-emptive, fixed priority and EDF, scheduling of sporadic task sets. We have also briefly covered extensions to this work that relax the rather unrealistic assumption that scheduling and pre-emption costs are negligible. In real systems, particularly those running on processors with cache, the costs of pre-emption can be relatively high when compared with the non-pre-emptive WCETs of the tasks. Thus techniques for limiting pre-emption are needed, as well as analysis that accounts for the cost of pre-emption overheads.

7.1 Looking back: success stories

Despite the advent of multicore processors and substantial research into multiprocessor scheduling, research into real-time scheduling on single core systems still thrives. One of the hardest challenges in this area is to progress from simple

models of the system and tasks to more complex models that capture the important behaviours of real systems and applications.

A good example of where this has happened and a major success story for research into schedulability analysis is Controller Area Network. This is a simple broadcast network used to connect Electronic Control Units in automotive systems. (CAN is used in the vast majority of cars manufactured today, with sales of CAN enabled microprocessors approaching 1 billion per year). From the perspective of analysing message response times, the broadcast behaviour of CAN means that its analysis resembles that of a single processor with fixed priority non-pre-emptive scheduling. Precise schedulability analysis has been derived for Controller Area Networks [36] that makes certain assumptions in terms of the behaviour of the nodes connected to the bus (perfect priority queues); however, in practice these assumptions may not always hold. Recent work has sort to address particular features of the hardware such as non-abortable transmit requests [51], and the fact that FIFO queuing policies are often used [30], [31]. These extensions make the analysis more flexible and better suited to commercial use.

7.2 Looking ahead: research directions

Within the theme of single processor real-time scheduling, hot research topics today include:

(i) The integration of schedulability analysis, and WCET analysis, and the development of integrated techniques such as limited pre-emption to improve schedulability (see Section 6).

(ii) Scheduling of Mixed Criticality Systems based on fixed priority [68], [9] and EDF [42].

(iii) Probabilistic approaches to schedulability analysis [57] and WCET analysis [25], [24], and their integration [27].

8 ACKNOWLEDGEMENTS

This work was funded in part by the EPSRC project MCC (EP/K011626/1). The author would like to thank Will Lunniss for his help with the background material on CRPD analysis.

9 REFERENCES

[1] S. Altmeyer, R.I. Davis, C. Maiza “Cache related Pre-emption Delay aware response time analysis for fixed priority pre-emptive systems”. In proceedings IEEE Real-Time Systems Symposium (RTSS), pages 261-271, 2011.

[2] S. Altmeyer, Roeland Douma, W. Lunniss, R.I. Davis, “Evaluation of Cache Partitioning for Hard Real-Time Systems”. Under submission to IEEE Real-Time Systems Symposium (RTSS) 2013.

[3] S. Altmeyer, R.I. Davis, C. Maiza “Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems”. Real-Time Systems, Volume 48, Issue 5, pages 499-526, Sept 2012

[4] N.C. Audsley, "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.

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

[6] N.C. Audsley, A. Burns, R.I. Davis, K. W. Tindell and A. J. Wellings, “Fixed Priority scheduling an Historical perspective”. Real-Time Systems 8(3), pages 173-198. 1995.

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

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

[9] S.K. Baruah, A. Burns, R.I. Davis “Response Time Analysis for Mixed Criticality Systems”. In proceedings 32nd IEEE Real-Time Systems Symposium (RTSS'11), pages 34-43, Nov 29th - Dec 2nd, 2011.

[10] S.K. Baruah, A.K. Mok, L.E. Rosier, “Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor”. In proceedings IEEE Real-Time Systems Symposium (RTSS), pages182-190, 1990.

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

[12] S.K. Baruah. “The limited-preemption uniprocessor scheduling of sporadic task systems”. In Proceedings Euromicro Conference on Real-Time Systems (ECRTS), pages 137–144, 2005.

[13] M. Bertogna, G. Buttazzo, M. Marinoni, G. Yao, F. Esposito, M. Caccamo. "Preemption points placement for sporadic task sets", In proceedings Euromicro Conference on Real-Time Systems (ECRTS), 2010.

[14] M. Bertogna, O. Xhani, M. Marinoni, F. Esposito, G. Buttazzo. "Optimal Selection of Preemption Points to Minimize Preemption Overhead", In proceedings Euromicro Conference on Real-Time Systems (ECRTS), 2011.

[15] M. Bertogna, G. Buttazzo, G. Yao. "Improving Feasibility of Fixed Priority Tasks using Non-Preemptive Regions", In proceedings Real-Time Systems Symposium (RTSS), 2011.

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

[17] R.J. Bril, J.J Lukkien, W.F. Verhaegh, “Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred pre-emption”. Real-Time Systems. 42, 1-3, pages 63-119, 2009.

[18] R.J. Bril, M.M.H.P. van den Heuvel, U. Keskin, J.J. Lukkien, “Generalized fixed-priority scheduling with limited preemptions”, In proceedings Euromicro Conference on Real-Time Systems (ECRTS), pages 209-220, 2012.

[19] A. Burns, A.J. Wellings, “Real-Time Systems and Programing Languages: Ada 95, Real Time Java and Real Time Posix”, Addison Wesley, 2001.

[20] A. Burns, “Preemptive priority based scheduling: An appropriate engineering approach”. S. Son, editor, Advances in Real-Time Systems, pages 225–248, 1994.

[21] J. V. Busquets-Mataix, J. J. Serrano, R. Ors, P. Gil, and A. Wellings, "Adding Instruction Cache Effect to Schedulability Analysis of Preemptive Real-Time Systems". In proceedings IEEE Real-Time Technology and Applications Symposium (RTAS), pages. 204-212, 1996.

[22] G.C. Buttazzo, M. Bertogna, G. Yao. "Limited Preemptive Scheduling for Real-Time Systems: A Survey". IEEE Transactions on Industrial Informatics. In press. Downloadable from http://retis.sssup.it/~marko/publi.html.

[23] G. C. Buttazzo, "Rate Monotonic vs. EDF: Judgment Day," Real-Time Systems, vol. 29, no. 1, pages 5-26, 2005.

[24] F.J. Cazorla, E. Quinones, T. Vardanega, L. Cucu, B. Triquet, G. Bernat, E. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo, and D. Maxim. “Proartis: Probabilistically analysable real-time systems”, ACM TECS, 2013.

[25] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzeti, E. Quinones, and F. J. Cazorla. “Measurement-Based Probabilistic Timing Analysis for Multi-path Programs”. In proceedings of the Euromicro Conference on Real-Time Systems (ECRTS), 2012.

[26] R.I. Davis, "Getting ones priorities right". Keynote talk at the International Conference on Real-Time and Network Systems (RTNS), 2012. Available from http://www-users.cs.york.ac.uk/~robdavis/

[27] R. I. Davis, L. Santinelli, S. Altmeyer, C. Maiza, and L. Cucu-Grosjean. “Analysis of probabilistic cache related pre-emption delays”. In proceedings of the Euromicro Conference on Real-Time Systems (ECRTS), 2013.

[28] R.I. Davis, N. Merriam, N.J. Tracey, “How Embedded Applications Using an RTOS can stay within On-chip Memory Limits”. In proceedings Work in Progress and Industrial Experience Sessions, Euromicro Conference on Real-Time Systems (ECRTS), 2000.

[29] R.I. Davis, M. Bertogna "Optimal Fixed Priority Scheduling with Deferred Pre-emption”. In proceedings IEEE Real-Time Systems Symposium (RTSS), 2012.

[30] R.I. Davis, S. Kollmann, V. Pollex, F. Slomka, "Schedulability Analysis for Controller Area Network (CAN) with FIFO Queues Priority Queues and Gateways”. Real-Time Systems, Volume 49, Issue 1, pages 73-116, Jan 2013.

[31] R.I. Davis, N. Navet “Controller Area Network (CAN) Schedulability Analysis for Messages with Arbitrary Deadlines in FIFO and Work-Conserving Queues”. In proceedings 9th Workshop on Factory Communication Systems (WFCS), pages 33-42, 2012.

[32] R.I. Davis “Burns Standard Notation” in Real-Time Systems: the past, the present, and the future, N.C. Audsley and S.K. Baruah editors, 2013.

[33] R.I. Davis, A. Zabos, A. Burns, "Efficient Exact Schedulability Tests for Fixed Priority Real-Time Systems”. IEEE Transactions on Computers, Vol. 57, No. 9, pages 1261-1276, 2008.

[34] R.I Davis, A. Burns, "Robust Priority Assignment for Fixed Priority Real-Time Systems”. IEEE Real-Time Systems Symposium (RTSS), pages 3-14, 2007.

[35] R.I. Davis, A. Burns. "Response Time Upper Bounds for Fixed Priority Real-Time Systems". In proceedings IEEE Real-Time Systems Symposium (RTSS), 2008.

[36] R.I. Davis, A. Burns, R.J. Bril, and J.J. Lukkien. “Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised”. Real-Time Systems, Volume 35, Number 3, pages 239-272, April 2007.

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

[38] R.I. Davis, T. Rothvoß, S.K. Baruah, A. Burns, “Quantifying the Sub-optimality of Uniprocessor Fixed Priority Pre-emptive Scheduling for Sporadic Tasksets with Arbitrary Deadlines”. In proceedings of Real-Time and Network Systems (RTNS), pages 23-31, October 26-27th, 2009.

[39] R.I. Davis, L. George, P. Courbin, “Quantifying the Sub-optimality of Uniprocessor Fixed Priority Non-Pre-emptive Scheduling”. In proceedings 18th International Conference on Real-Time and Network Systems (RTNS), pages 1-10, November 4-5th, 2010.

[40] R.I. Davis, A. Burns, W. Walker, “Guaranteeing Timing Constraints Under Shortest Remaining Processing Time Scheduling”. In proceedings of the Euromicro Workshop on Real-Time Systems, pages. 88-93, 1997

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

[42] P. Ekberg, Y. Wang "Bounding and Shaping the Demand of Mixed-Criticality Sporadic Tasks". In proceedings Euromicro Conference on Real-Time Systems (ECRTS), pages135-144, July 2012.

[43] L. George, J. Hermant, “A norm approach for the Partitioned EDF Scheduling of Sporadic Task Systems.” In proceedings Euromicro Conference on Real-Time Systems (ECRTS), 2009.

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

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

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

[47] K. Jeffay, D. F. Stanat, C. U. Martel, “On Non-Preemptive Scheduling of Periodic and Sporadic Tasks”.

In proceedings IEEE Real-Time Systems Symposium (RTSS), pages 129-139, 1991.

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

[49] L. Ju, S. Chakraborty, A. Roychoudhury, "Accounting for Cache-Related Preemption Delay in Dynamic Priority Schedulability Analysis". In proceedings Design, Automation and Test in Europe Conference and Exposition (DATE), pages 1623-1628, 2007.

[50] Kim, Naghibdadeh, “Prevention of task overruns in real-time non-preemptive multiprogramming systems”, In proceedings of Perf., Assoc. Comp. Mach., pages 267-276, 1980,

[51] D.A. Khan, R.I. Davis, N. Navet “Schedulability Analysis of CAN with Non-abortable Transmission Requests”. In proceedings IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2011.

[52] C. Lee, J. Hahn, Y. Seo, S. Min, H. Ha, S. Hong, C. Park, M. Lee, and C. Kim, "Analysis of Cache-related Preemption Delay in Fixed-priority Preemptive Scheduling," IEEE Transactions on Computers, vol. 47, no. 6, pages 700-713, June 1998.

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

[54] J.P. Lehoczky, “Fixed priority scheduling of periodic task sets with arbitrary deadlines”. In proceedings IEEE Real-Time Systems Symposium (RTSS), pages 201–209, 1990.

[55] J.P. Lehoczky, L. Sha, Y. Ding, “The rate monotonic scheduling algorithm: Exact characterization and average case behaviour”. In proceedings IEEE Real-Time Systems Symposium (RTSS), pages 166–171, 1989.

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

[57] J. Lopez, J. L. Diaz, J. Entrialgo, D. Garca, “Stochastic analysis of real-time systems under preemptive priority-driven scheduling”. Real-time Systems, 40(2), 2008.

[58] W. Lunniss, S. Altmeyer, C. Maiza, R.I. Davis, "Intergrating Cache Related Pre-emption Delay Analysis into EDF Scheduling," in Proceedings IEEE Conference on Real-Time and Embedded Technology and Applications (RTAS), pages 75-84, 2013,.

[59] W. Lunniss, S. Altmeyer, R.I. Davis, "Optimising Task Layout to Increase Schedulability via Reduced Cache Related Pre-emption Delays". In Proceedings of the International Conference on Real-Time and Network Systems (RTNS), pages 161-170, 2012.

[60] A.K. Mok, “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.

[61] M. Saksena and Y. Wang. “Scalable real-time system design using preemption thresholds”. In Proceedings Real-Time Systems Symposium (RTSS), 2000.

[62] L. Sha, T. Abdelzaher, K-E. Arzen, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, A.K. Mok, “Real Time Scheduling Theory: A Historical Perspective”. Real-Time Systems Journal, Vol 28, No 2/3, pages 101-155, 2004.

[63] L. Sha, R. Rajkumar, J. Lehoczky, “Priority inheritance protocols: An approach to real-time synchronisation”. IEEE Transactions on Computers 39, 9, pages 1175–1185, 1990.

[64] W. E. Smith. Various optimisers for single-stage production. N*aval research and Logistics Quarterly,* 3(1), 1956.

[65] M. Spuri, “Analysis of Deadline Scheduled Real-Time Systems”. INRIA Technical Report No. 2772, Jan. 1996.

[66] Y. Tan, V. Mooney, "Timing Analysis for Preemptive Multitasking Real-Time Systems with Caches," ACM Transactions on Embedded Computing Systems (TECS), vol. 6, no. 1, February 2007.

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

[68] S. Vestal, "Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance," In proceedings IEEE Real-Time Systems Symposium (RTSS), pages 239-243, 2007.

[69] Y. Wang and M. Saksena, “Scheduling fixed-priority tasks with pre-emption threshold”. In proceedings RTCSA, 1999.

[70] J.Whitham, R.I. Davis, N.C. Audsley, S.Altmeyer, C. Maiza "Investigation of Scratchpad Memory for Preemptive Multitasking”. In proceedings IEEE Real-Time Systems Symposium (RTSS'12), 2012.

[71] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, P. Stenstrom. “The worst-case execution-time problem overview of methods and survey of tools”. ACM Trans. Embed. Comput. Syst. 7, 3, Article 36, 53 pages, 2008.

[72] G. Yao, G. Buttazzo, M. Bertogna. "Bounding the Maximum Length of Non-Preemptive Regions Under Fixed Priority Scheduling", In Proceedings RTCSA, 2009.

[73] F. Zhang, A. Burns, "Schedulability Analysis for Real-Time Systems with EDF Scheduling," IEEE Transactions on Computers, pages 1250-1258, September, 2009

[74] F. Zhang, A. Burns, A. “Schedulability Analysis of EDF Scheduled Embedded Real-Time Systems with Resource Sharing”. ACM Transactions on Embedded Computing Systems 9, 4, Article 39, March 2011.

[75] A. Zuhily, A. Burns, “Optimal (D - J) - monotonic priority assignment”, Information Processing Letters, Vol 103, No 6, pages 247-250, 2007.