Journal of Systems Architecture 65 (2016) 64–82

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

A review of priority assignment in real-time systems

Robert I. Davis[a,b,*], Liliana Cucu-Grosjean[b], Marko Bertogna[c], Alan Burns[a]

[a] Real-Time Systems Research Group, Department of Computer Science, University of York, York, UK
[b] AOSTE Team, INRIA-Paris, Paris, France
[c] University of Modena, Italy

ARTICLE INFO

Article history:
Received 8 September 2015
Revised 5 February 2016
Accepted 13 April 2016
Available online 14 April 2016

Additional key words and phrases:
Real-time systems
Priority assignment
Fixed priority scheduling
C.3 [Real-time and embedded systems]
Performance
Design
Algorithms
Real-time scheduling
Schedulability analysis
Optimal priority assignment
Deadline monotonic
Rate monotonic
Robust priority assignment

ABSTRACT

It is over 40 years since the first seminal work on priority assignment for real-time systems using fixed priority scheduling. Since then, huge progress has been made in the field of real-time scheduling with more complex models and schedulability analysis techniques developed to better represent and analyse real systems. This tutorial style review provides an in-depth assessment of priority assignment techniques for hard real-time systems scheduled using fixed priorities. It examines the role and importance of priority in fixed priority scheduling in all of its guises, including: pre-emptive and non-pre-emptive scheduling; covering single- and multi-processor systems, and networks. A categorisation of optimal priority assignment techniques is given, along with the conditions on their applicability. We examine the extension of these techniques via sensitivity analysis to form robust priority assignment policies that can be used even when there is only partial information available about the system. The review covers priority assignment in a wide variety of settings including: mixed-criticality systems, systems with deferred pre-emption, and probabilistic real-time systems with worst-case execution times described by random variables. It concludes with a discussion of open problems in the area of priority assignment.

Preamble

Many presentations are written as a consequence of first writing a paper. This paper was written as a consequence of the first author giving the Keynote talk at the 20th International Conference on Real-Time and Network Systems (RTNS) in 2012, thus a presentation of much of the material in this paper can be found at http://rtns2012.loria.fr/#page=Invitedtalk.

1. Introduction

Hard real-time systems are characterised by the need for both functional and temporal correctness. Such systems are required not only to produce appropriate responses or outputs to their stimuli or inputs (functional correctness), but to do so within specified time constraints (temporal correctness). These time constraints are typically expressed in terms of deadlines on the elapsed time between a stimulus or input and the corresponding response or output.

Today, hard real-time systems are found in many different application areas including; aerospace, automotive, and railway systems, space and satellite systems, medical monitoring and imaging systems, industrial process control, and robotics. The majority of these systems are multitasking and use a *scheduler* within the Real-Time Operating System (RTOS) to determine which one of many tasks is given access to the processor or processors at any given time.

The vast majority of commercial Real-Time Operating Systems use a fixed priority scheduler; indeed, automotive standards such as OSEK [1] and AUTOSAR [2] mandate the use of fixed priority scheduling. With fixed priority scheduling, each task is assigned a static priority offline, then at runtime, each job of that task competes for the processor on the basis of its priority, with the highest priority job selected for execution. One of the most common questions asked regarding the scheduling of such systems is:

"How should I assign priorities?"

This is an important question, since a poor priority assignment will mean that the scheduler may run jobs in an order that is far

* Corresponding author. Tel.: 0044 1904 325573.
E-mail addresses: rob.davis@york.ac.uk (R.I. Davis), liliana.cucu@inria.fr (L. Cucu-Grosjean), marko.bertogna@unimore.it (M. Bertogna), alan.burns@york.ac.uk (A. Burns).

http://dx.doi.org/10.1016/j.sysarc.2016.04.002
1383-7621/

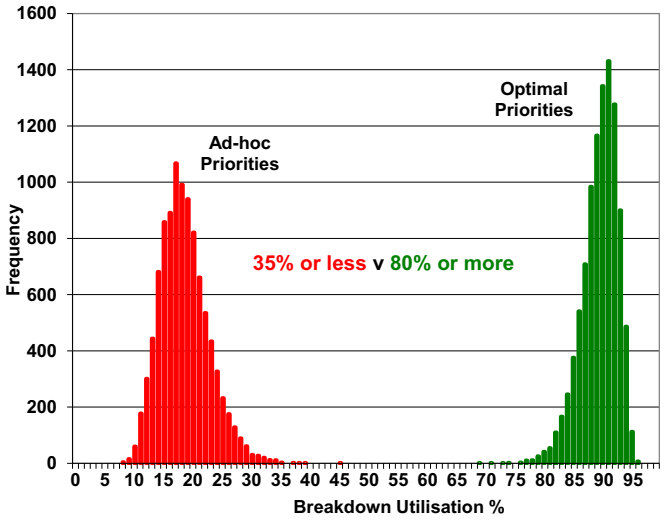


Fig. 1. Breakdown Utilisation.

from optimal[1], leading to missed deadlines, even though the overall workload or utilisation of the system is low. This can have significant commercial consequences. If a system can only utilise a small fraction of its overall processing or network capacity before deadlines start being missed, then as further functionality is added, it will become unreliable, or will need upgrading to more expensive hardware.

1.1. Why is priority assignment important?

In real-time systems that use fixed priority scheduling, appropriate priority assignment is essential to avoid overprovisioned hardware, to provide headroom for additional functionality, and to avoid reliability issues caused by intermittent failures due to deadline misses.

To illustrate this point, we use an example from the automotive industry. Controller Area Network (CAN) [27,46] is a broadcast bus that is widely used for in-vehicle networking. Communications over CAN are effectively scheduled using fixed priority non-pre-emptive scheduling, with the message identifiers (IDs) used as priorities during arbitration to determine the order in which messages are sent on the bus. In his keynote talk at ECRTS 2012 [35], Darren Buttle of ETAS remarked on the myth of CAN bus utilisation believed by many in industry:

"You cannot run CAN reliably at more than 35% utilisation[2]"

This myth comes from a general practice of assigning message IDs (i.e. priorities) in an ad-hoc way reflecting the data content of the message, ECU supplier and other legacy issues. The effect of assigning message IDs in an ad-hoc way that has no correlation with message deadlines was highlighted by Davis et al. [51]. Fig. 1 shows the frequency distribution of the breakdown utilisation [68] of 10,000 typical automotive CAN configurations with 80 messages (10 ms to 1 s periods). The breakdown utilisation is computed by scaling the bus speed until the message set is only just schedulable and then recording the overall bus utilisation (i.e. message transmission times divided by periods) at that speed. From the graph it is clear that priority assignment is important. Fig. 1 shows that assigning priorities in an optimal way leads to typical breakdown utilisations of 80% or more, whereas ad-hoc or random priority assignment leads to typical bus utilisations of 35% or less, hence the myth described by Buttle [35].

1.2. How to assign priorities?

In this paper, we provide a tutorial-style review of answers to the question, *"How to assign priorities?"*

We survey work on priority assignment through the ages. We look at simple task models where Deadline Monotonic priority assignment is optimal and see how departures from these models break this optimality. We review Audsley's algorithm for Optimal Priority Assignment (OPA), including the rules for when it can and cannot be used – as well as a catalogue of situations where it is useful. We look at how this algorithm has been extended to form Robust Priority Assignments (RPA), and how they can be used to define priority orderings when only partial information is available about a system. For systems and schedulability analyses where Audsley's algorithm cannot be directly applied, we examine what can be done to avoid checking all possible priority orderings. We also recount how the basic OPA algorithm can be modified to obtain priority assignments that minimise the number of priority levels needed, and also how it can be used to minimise the lexicographical distance or the reverse lexicographical distance from any desired priority ordering.

This review covers priority assignment for fixed priority scheduling in all of its guises, including: pre-emptive, non-pre-emptive, and deferred pre-emption scheduling; for single-processor, multi-processor, and networked systems. As well as conventional systems, we review priority assignment in mixed criticality systems, and probabilistic real-time systems where worst-case execution times are described by random variables.

At the end of the review, we set out a number of open problems in priority assignment, including priority assignment in systems with Cache Related Pre-emption Delays, and dual-priority scheduling [33].

The review ends with a summary of the key results and current challenges, and provides recommendations for those wanting to know *"How to assign priorities?"*

Note, the focus of this review is on priority assignment, we deliberately do not go into depth on the closely related topic of schedulability analysis. For more information on that topic, the interested reader is directed to reviews and surveys on single processor [12,83,53] and multiprocessor [50] scheduling.

2. System model, terminology and notation

In this paper, we consider types of systems scheduled using fixed priorities. In this section, we outline a basic task model that is capable of extension in a variety of different ways. Some of these extensions are also detailed here, whereas others are specific to particular problem domains and are discussed later. We also define the terminology used in schedulability analysis. We note that a similar system model also applies to communications on CAN with 'task' replaced by 'message' and 'execution time' replaced by 'transmission time'.

2.1. System model

The system model used in this paper focuses on the fixed priority scheduling of a set of n statically defined tasks which together make up a *task set*. Each task τ_i is identified by its index i from 1 to n. Each task is assumed to have a unique priority. The notation $hp(i)$ (and $hep(i)$) is used to denote the set of tasks with priorities higher than (higher than or equal to) i. Similarly, $lp(i)$ (and $lep(i)$) are used to denote the set of tasks with priorities lower than (lower than or equal to) i.

[1] We define what is meant by an optimal priority assignment in Section 2.

[2] Figure may vary but not significantly.

Each task τ_i is assumed to have a bounded *worst-case execution time* C_i, a minimum inter-arrival time or *period* T_i, and a relative deadline D_i. Note, we assume a discrete time model, with all of these task parameters represented by positive integers. Each task τ_i may generate a potentially unbounded sequence of invocations (or *jobs*). Each job of task τ_i has an execution time that is upper bounded by C_i, an arrival time at least T_i after the arrival of the previous job of the same task, and a *relative deadline* D_i after its arrival.

A task set is referred to as *periodic* if each job of every task τ_i arrives exactly T_i after the previous job of the same task. A task set is referred to as *sporadic* if each job of a task τ_i may arrive at any time that is at least T_i after the arrival of the previous job of the same task. Thus the sporadic task model forms a generalisation of the periodic one. In this paper, unless explicitly stated, we refer to sporadic tasks sets.

Task sets may be further classified according to the deadlines of their component tasks. If all tasks have deadlines equal to their periods ($D_i = T_i$), then we have an *implicit-deadline* task set. If instead, all tasks have deadlines that are less than or equal to their periods, then that constitutes a *constrained-deadline* task set. Finally, in an *arbitrary-deadline* task set, task deadlines may be smaller than, the same as, or larger than their periods.

The *utilisation* U_i of a task τ_i is equal to its execution time divided by its period (U_i=C_i/T_i). The total utilisation U of a task set is the sum of the utilisations of all its tasks.

This task model permits a number of simple extensions as follows.

Tasks may make mutually exclusive access to shared resources, thus task τ_i may be blocked from executing for at most the *blocking time* B_i due to lower priority tasks that access shared resources (e.g. via the Stack Resource Policy [15]) . Otherwise, tasks are assumed to be independent, and not to self-suspend.

For periodic task sets, the first arrival of a job of task τ_i is assumed to take place at an *offset* O_i from time $t = 0$. (Note offsets are assumed to be normalised so that the minimum offset of any task is zero). If $\forall i \quad O_i = 0$, then the task set is referred to as *synchronous*, since all tasks have a synchronous arrival at time $t = 0$. Otherwise it is referred to as *asynchronous*, or simply as having offsets.

There may be a delay of up to the *release jitter* J_i between the notional arrival and the release of each job of task τ_i at which point it becomes ready to execute.

The *worst-case response time* R_i [64] of a task is defined as the longest possible time from the release of a job of the task until that job completes execution. Calculation of a task's worst-case response time allows its schedulability to be trivially checked by comparison with its deadline and release jitter: $R_i \leq D_i - J_i$.

There are a number of different forms of fixed priority scheduling, depending on if and when pre-emption is permitted. With *Fixed Priority Pre-emptive Scheduling* (FPPS), when a high priority task becomes ready to execute, a lower priority job that is currently running will be suspended (pre-empted) in order to allow the higher priority job to execute. With *Fixed Priority Non-preemptive Scheduling* (FPNS) such pre-emption is not permitted, and the higher priority job has to wait until the lower priority job completes before it can access the processor. Between these two extremes, is *deferred pre-emption* (FPDS), where pre-emption may be deferred for some interval of time after a higher priority task becomes ready, either by the RTOS [16], or due to non-pre-emptable regions in the task's code (*co-operative scheduling*) [31].

With FPDS, each task is assumed to have a final non-preemptive region of length F_i in the range [1, C_i]. This model of FPDS subsumes both fully pre-emptive and fully non-pre-emptive scheduling, since if $\forall i \quad F_i = 1$, then FPDS equates to FPPS, whereas if $\forall i \quad F_i = C_i$ we have FPNS. (Note that with a discrete time model, the minimum possible length of a non-pre-emptive region is 1, since a task cannot be pre-empted during a single processor clock cycle).

With fixed priority scheduling each job of a task τ_i has the same priority given by the priority assigned to the task. This is sometimes referred to as *fixed task priority* scheduling, as distinct from *fixed job priority* scheduling where each individual job of a task can have a different priority. An example of fixed job priority scheduling is Earliest Deadline First (EDF) where job priorities correspond to absolute deadlines. In the remainder of this paper when we refer to fixed priority scheduling, we mean fixed task priority scheduling.

The *critical instant* [69] for a task τ_i refers to a scenario or pattern of job releases that result in a job of the task exhibiting the worst-case response time.

We use the term *priority level-i busy period* to mean an interval 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 prior to its end at t_2, are either executing or *ready to execute*. We note that by definition, the worst-case response time of a task at priority i must occur within a priority level-i busy period.

2.2. Schedulability analysis

Definition. *Schedulable*: A task set is said to be *schedulable* with a priority assignment Q, under some fixed priority scheduling algorithm G, if all valid sequences of jobs that may be generated by the task set can be scheduled by algorithm G using priority ordering Q without any missed deadlines.

A schedulability test for some fixed priority scheduling algorithm G is referred to as *sufficient*, if all the task sets and priority orderings that are deemed schedulable according to the test are in fact schedulable under the scheduling algorithm. Similarly, a schedulability test is referred to as *necessary*, if all the task sets and priority orderings 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*.

2.3. Priority assignment policies

The goal of a priority assignment policy is to provide a schedulable priority order whenever such an ordering exists. This leads to a definition of optimal priority assignment. We note that the *optimality* of a particular priority assignment policy is with respect to a particular task model (for example constrained-deadline, sporadic tasks), and a scheduling policy (e.g. FPPS). It is also useful to define optimality with respect to the schedulability test used, which may be exact or only sufficient. Hence, in general the optimality of a priority assignment policy is with respect to a given configuration comprising (i) the task model, (ii) the scheduling algorithm, and (iii) the schedulability test used.

Definition. : *Optimal Priority Assignment*: A priority assignment policy P is said to be *optimal* with respect to a configuration (task model M, fixed priority scheduling algorithm G, and schedulability test S), if and only if every set of tasks that is compliant with the task model and is deemed schedulable under algorithm G by test S with some priority assignment policy is also deemed schedulable under algorithm G by test S using policy P.

In other words, P is optimal if it is at least as good as any other priority assignment policy.

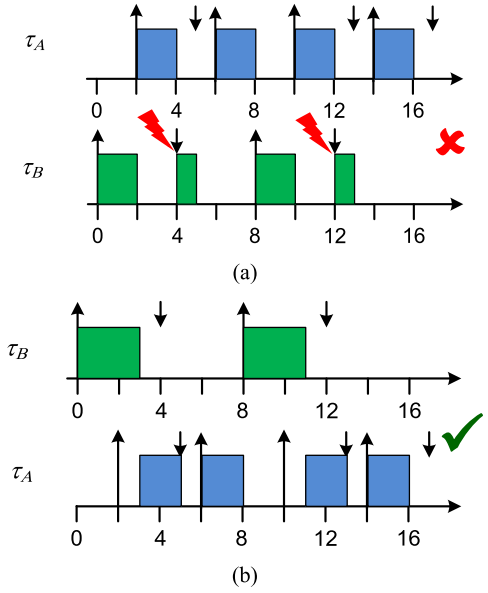


Fig. 2. Deadline Monotonic priority ordering is not optimal for tasks with offset release times.

Table 1
Task parameters.

Task	C	D	T	O
τ_A	2	3	4	2
τ_B	3	4	8	0

In the remainder of the paper, when we refer to the optimality of a priority assignment policy with respect to a particular configuration, giving only the task model and scheduling algorithm, then we are implicitly also referring to an exact test.

3. Early work on priority assignment

The first work on priority assignment considered fixed priority pre-emptive scheduling (FPPS) on a single processor, for a simple periodic task model without blocking or release jitter.

In 1967, Fineberg and Serlin [56] considered two synchronous periodic tasks with implicit-deadlines scheduled using FPPS. They showed that it is better to assign the higher priority to the task with the shorter period. In 1973, Liu and Layland [69] extended this result and showed that *Rate-Monotonic* Priority Ordering (RMPO) is optimal for synchronous periodic task sets with implicit-deadlines. (Rate-Monotonic priority assignment assigns priorities in the same order as task periods, with the task with the shortest period having the highest priority).

Liu and Layland's famous result was generalised in 1982 by Leung and Whitehead [67] who showed that *Deadline-Monotonic* Priority Ordering (DMPO) is optimal for synchronous periodic task sets with constrained-deadlines. However, minor changes to the task model (e.g. offset release times, or arbitrary deadlines) or to the scheduling algorithm (e.g. non-pre-emptive rather than pre-emptive fixed priority scheduling) are enough to break the optimality of DMPO. Leung and Whitehead [67] showed that DMPO is *not* optimal for periodic tasks with constrained deadlines and offset release times as illustrated in Fig. 2 for the set of tasks in Table 1.

With Deadline Monotonic priority ordering, task τ_A has the higher priority. In this case, jobs of task τ_B miss their deadlines (Fig. 2(a)). However, if the priority ordering is reversed, then it is easy to see that the task set is schedulable (Fig. 2(b)). (Note, Leung and Whitehead [67] showed that in order to check schedulability for periodic tasks with constrained deadlines and offsets, it is sufficient to check all deadlines in an interval of length $(2H + O^{\max})$ where H is the *hyperperiod* (Least Common Multiple of task periods) and $O^{\max}$ is the largest offset).

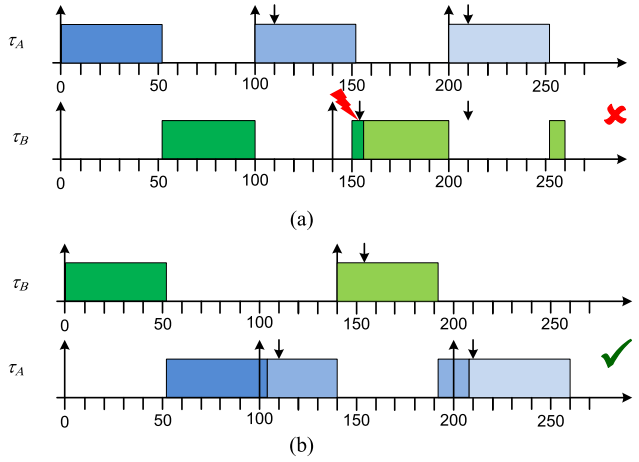


Fig. 3. Deadline Monotonic priority ordering is not optimal for tasks with arbitrary deadlines.

Table 2
Task parameters.

Task	C	D	T
τ_A	52	110	100
τ_B	52	154	140

Table 3
Task parameters.

Task	C	D	T
τ_A	4	10	10
τ_B	4	12	16
τ_C	4	13	14

Goossens and Devilliers [60] showed in 1997 that DMPO is also not optimal for so called *offset free* systems where both offsets and priorities may be freely chosen with the aim of finding a schedulable system.

In 1990, Lehoczky [66] showed that DMPO is also not optimal for synchronous periodic task sets with arbitrary deadlines as illustrated in Fig. 3 for the set of tasks in Table 2. With Deadline Monotonic priority ordering, task τ_A has the higher priority. In this case, the first job of task τ_B in the priority level-i busy period misses its deadline (Fig. 3 (a)). However, if the priority ordering is reversed, then all jobs meet their deadlines (Fig. 3 (b)). Note that in this case, the second job of task τ_A has a longer response time than the first).

In 1996, George et al. [59] showed that Deadline Monotonic priority ordering is *not* optimal for constrained-deadline periodic tasks under fixed priority non-pre-emptive scheduling, as illustrated in Fig. 4 for the set of tasks in Table 3. In this case, with the tasks in DMPO, the second job of task τ_C in the priority level-i busy period misses its deadline (Fig. 4 (a)). However, if the priority ordering of tasks τ_B and τ_C is reversed, then all jobs meet their deadlines (Fig. 4 (b)). Note that here similar to the task set with arbitrary deadlines, the second job of the lowest priority task has a longer response time than the first. Thereby emphasizing that in these cases, it is not sufficient to only check that the first job in the busy period meets its deadline.

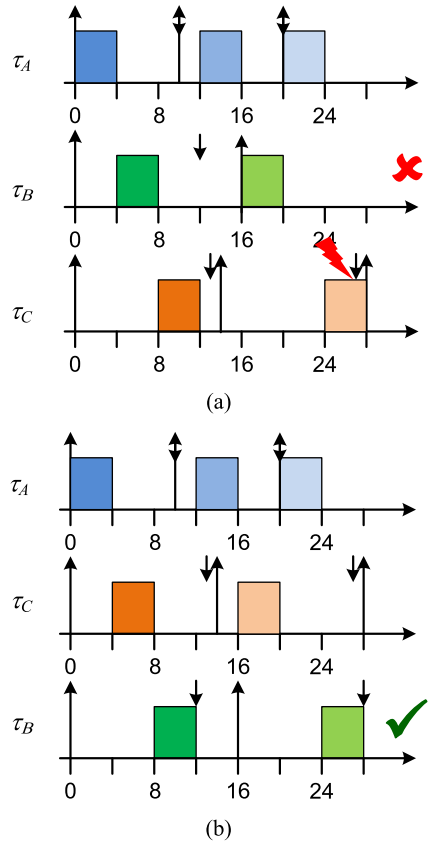


Fig. 4. Deadline Monotonic priority ordering is not optimal for fixed priority non-pre-emptive scheduling.

In 1995, Davis and Burns [43] showed that the optimal priority assignment for Aperiodic[3] jobs (with firm deadlines) arriving in a system with hard deadline sporadic or periodic tasks is to assign each aperiodic job the highest priority such that no task with an earlier next absolute deadline has a higher priority (effectively a hybrid between DMPO and Earliest Deadline First (EDF) scheduling).

4. Proving the optimality of priority assignment policies

The optimality of a priority assignment policy such as Deadline Monotonic priority ordering derives from schedulability analysis. Below, we recapitulate response time analysis for sporadic tasks with constrained deadlines under fixed priority pre-emptive scheduling. Based on this analysis, the optimality of DMPO is shown using a standard technique for proving the optimality of priority assignment policies.

The worst-case response time R_i for task τ_i corresponds to the length of the priority level-i busy period starting with synchronous release, and where all higher priority tasks are then released again as soon as possible. The length of the busy period w_i, can be calculated using the following recurrence relation [11,64], where the summation term gives the total interference over the busy period due to the set of higher priority tasks.

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

[3] Aperiodic jobs may arrive at any time and have a relative deadline that is referred to as *firm*, that is either the job must be completed by this deadline or it is of no value to the system.

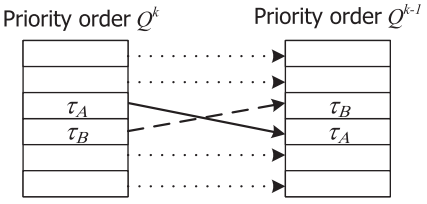


Fig. 5. Swapping the priorities of tasks at adjacent priority levels.

Iteration starts with an initial value for w_i^0, typically $w_i^0 = C_i$, and ends either when $w_i^{m+1} = w_i^m$ 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 task set utilisation is less than or equal to 1.

The standard technique for proving that a priority assignment policy is optimal is as follows:

To show that priority assignment policy P is optimal, we must prove that any task set (that complies with the task model) that is schedulable (under the scheduling algorithm considered) using some priority assignment policy Q is also schedulable using priority ordering P.

Proof is obtained by transforming priority order Q (which is known to be schedulable) into priority order P while ensuring that no tasks become unschedulable during the transformation. The proof is by induction.

Base case: Priority order Q^k is schedulable, since we set $Q^k = Q$ and Q is the schedulable priority ordering assumed in the theorem.

Inductive step: A pair of tasks that are at adjacent priorities in priority ordering Q^k, but in the opposite relative priority order under policy P are chosen and their priorities swapped to produce a new priority order Q^{k-1} (see Fig. 5). It is then demonstrated that there is no loss of schedulability, i.e. all the tasks remain schedulable under priority order Q^{k-1}.

At most $k = n(n-1)/2$ such steps are needed to complete the reordering (effectively a bubble sort) such that $Q^1 = P$, and since there is no loss of schedulability on any step, that proves the task set is also schedulable under priority ordering P. Hence there can be no task sets that are schedulable under some other priority ordering Q that are not also schedulable under the priority ordering given by policy P, which proves that P is an optimal priority ordering.

We now demonstrate the use of this technique using the exact analysis given in (1) and so provide a standard proof (derived from that given in [67]) of optimality for DMPO.

Theorem 1. *DMPO is an optimal priority assignment policy for sporadic tasks with constrained deadlines under fixed priority pre-emptive scheduling on a single processor.*

Proof. We show that any task set compliant with the model that is schedulable under some priority order Q is also schedulable under priority order P (=DMPO).

Base case: The task set is schedulable with priority order Q.

Inductive step: We select a pair of tasks that are at adjacent priorities (i and j where $j=i+1$) in priority ordering Q^k, but out of Deadline Monotonic relative priority order. Let these tasks be τ_A and τ_B, with τ_A having the higher priority in Q^k (see Fig. 5). Note that $D_A > D_B$ as the tasks are out of Deadline Monotonic relative order. Let i be the priority of task τ_A in Q^k and j be the priority of task τ_B. We need to prove that all of the tasks remain schedulable with priority order Q^{k-1}. There are four groups of tasks to consider:

$hp(i)$: tasks in this set have higher priorities than both τ_A and τ_B in both Q^k and Q^{k-1}. Since the schedulability of these tasks

Algorithm 1
Audsley's Optimal Priority Assignment Algorithm.

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

is unaffected by the relative priority ordering of τ_A and τ_B, they remain schedulable in Q^{k-1}.

Task τ_A: Let $y = R_B$ be the response time of task τ_B in priority order Q^k. Since task τ_B is schedulable in Q^k, we have $y = R_B \leq D_B < D_A \leq T_A$, hence in (1), the contribution to interference from τ_A within the response time of τ_B is exactly one job (i.e. C_A), and there is also a contribution of C_B from τ_B itself. Now consider the response time of task τ_A under priority order Q^{k-1}. This response time is $R_A = y$, as there is exactly the same contribution from tasks τ_A, τ_B and all the higher priority tasks. Since $y < D_A$, task τ_A remains schedulable.

Task τ_B: as the priority of τ_B has increased, its response time is no greater in Q^{k-1} than in Q^k, since the only change to the response time calculation for τ_B is the removal of the interference from task τ_A. Hence τ_B remains schedulable.

$lp(j)$: tasks in this set have lower priorities than both τ_A and τ_B in both Q^k and Q^{k-1}. Since the schedulability of these tasks is unaffected by the relative priority ordering of τ_A and τ_B, they remain schedulable.

All tasks therefore remain schedulable in Q^{k-1}.

At most $k = n(n-1)/2$ steps are required to transform priority ordering Q into P without any loss of schedulability □

We note that DMPO remains optimal [26] when tasks are permitted to share resources according to the Stack Resource Policy (SRP) [15] or the Priority Ceiling Protocol (PCP) [82], and that Deadline Minus Release Jitter Monotonic Priority Order is optimal for sporadic tasks with constrained deadlines and release jitter [96]. As previously noted; however, it only takes some minor changes to the task model or scheduling algorithm to undermine the optimality of DMPO. Examples of such changes include, offset release times [67], non-pre-emptive scheduling [59], arbitrary deadlines [66], and deadlines prior to completion [30].

5. Audsley's Optimality Priority Assignment (OPA) algorithm

To address the non-optimality of DMPO for tasks with offset release times, Audsley developed a more sophisticated approach to priority assignment. This approach, now commonly referred to as Audsley's Optimal Priority Assignment (OPA) Algorithm, solves the problem of priority assignment for all of the four cases cited above where DMPO is no longer optimal. It was first published in a technical report in 1991 [10] and formally published some 10 years later in [13].

The pseudo code for Audsley's algorithm, using some compatible schedulability test S is shown in Algorithm 1. For n tasks, Audsley's algorithm (Algorithm 1) makes at most $n(n+1)/2$ calls to a compatible schedulability test S. The algorithm is guaranteed to find a priority assignment that is schedulable according to test S, if such an assignment exists. The complexity of Audsley's algorithm is a significant improvement over checking all $n!$ possible priority orderings. For $n=25$, a maximum of 325 schedulability tests are required, instead of $> 10^{25}$. Note that the OPA algorithm does not specify the order in which the schedulability of unassigned tasks should be checked at each priority level.

Audsley's algorithm has been proven to be applicable in a variety of different situations, including the following:

- Periodic tasks with offset release times [10].
- Sporadic tasks with arbitrary deadlines – see Section 7 of [89]
- Sporadic task sets under non-pre-emptive scheduling – see Theorem 17 in [59].
- Tasks with mixed criticalities and an execution time budget per criticality level [90].
- Generalised Multi-frame tasks, where jobs of a task follow a fixed sequence with different worst-case execution times, deadlines and inter-arrival times between the different types of job – see Section 7.1 of [97].
- The Diagraph Real-time Task (DRT) model [85]. In this case, Stigge and Yi [86] showed that Audsley's algorithm can effectively be applied to both the problem of assigning Static Priorities (SP) to tasks, and the problem of assigning Static Job-type Priorities (SJP) to the job types (vertices) that characterise each task.
- Periodic tasks with worst-case execution times described by random variables [71].

We note that Audsley's algorithm is also applicable to the (m-k) firm task model [75] as can easily be shown by considering the three conditions for applicability discussed below.

In 2009, Davis and Burns [49] proved an important result about the applicability of Audsley's OPA algorithm. They showed that three simple Conditions are both sufficient and necessary for Audsley's algorithm to provide optimal priority assignment with respect to a given schedulability test S. This is a powerful result since it enables the OPA algorithm to be applied in a wide range of scenarios, while lowering the burden of proof of optimality to one of showing compliance with the three Conditions, something that is typically easily proved or disproved.

The three Conditions are reproduced below from [49] . They refer to properties of a task that are *independent* of its assigned priority. For example the worst-case execution time, deadline, and minimum inter-arrival time of a task are typically independent of its priority. By contrast a task's worst-case response time is typically highly dependent on its relative priority.

Condition 1: "The schedulability of a task τ_k may, according to test S, depend on any independent properties of tasks with priorities higher than k, but not on any properties of those tasks that depend on their relative priority ordering."

Condition 2: "The schedulability of a task τ_k may, according to test S, depend on any independent properties of tasks with priorities lower than k, but not on any properties of those tasks that depend on their relative priority ordering."

Condition 3: "When the priorities of any two tasks of adjacent priority are swapped, the task being assigned the higher priority cannot become unschedulable according to test S, if it was previously schedulable at the lower priority. (As a corollary, the task being assigned the lower priority cannot become schedulable according to test S, if it was previously unschedulable at the higher priority)."

Detailed proof that these conditions are sufficient and necessary for the applicability of the OPA algorithm is given in [49].

5.1. Applying Audsley's OPA algorithm to global fixed priority scheduling on a multiprocessor

Davis and Burns [49] used the above three Conditions to categorise schedulability tests for global fixed priority scheduling on identical multiprocessors (with m processors) according to their compatibility or otherwise with Audsley's algorithm.

The following schedulability tests were shown to be incompatible with OPA:

- Any exact test for global fixed priority pre-emptive scheduling [7] such as those for periodic task sets given by Cucu and Goossens [39,40].
- Response time analysis (RTA test) of Bertogna and Cirinei [20].
- Improved RTA-LC test of Guan et al. [61].
- While the following tests were shown to be compatible:
- Deadline Analysis (DA test) of Bertogna et al. [21].
- Improved DA-LC test (based on the RTA-LC test) [49].
- Response Time test of Andersson and Jonsson [7].

Below we give the schedulability equations for the DA test [21]; by simple inspection of the terms in these equations, it is clear that the three Conditions hold, since there is only a dependency on the set of higher priority tasks, but not on their relative priority order, and the interference becomes strictly smaller with increasing priority.

$$D_k \geq C_k + \left\lfloor \frac{1}{m} \sum_{i \in hp(k)} I_i^D(D_k, C_k) \right\rfloor \tag{2}$$

where:

$$I_i^D(D_k, C_k) = \min(W_i^D(D_k), D_k - C_k + 1)$$
$$W_i^D(L) = N_i^D(L)C_i + \min(C_i, L + D_i - C_i - N_i^D(L)T_i)$$
$$N_i^D(L) = \left\lfloor \frac{L + D_i - C_i}{T_i} \right\rfloor$$

By contrast, inspection of the equations given below for the RTA test [20] shows that this test is incompatible with Audsley's algorithm. This is because there is a dependency on the upper bound response times (R_i^{UB}) of higher priority tasks which in turn depends on their relative priority ordering.

$$R_k^{UB} \leftarrow C_k + \left\lfloor \frac{1}{m} \sum_{i \in hp(k)} I_i^R(R_k^{UB}, C_k) \right\rfloor \tag{3}$$

where:

$$I_i^R(R_k^{UB}, C_k) = \min(W_i^R(R_k^{UB}), R_k^{UB} - C_k + 1)$$
$$W_i^R(L) = N_i^R(L)C_i + \min(C_i, L + R_i^{UB} - C_i - N_i^R(L)T_i)$$
$$N_i^R(L) = \left\lfloor \frac{L + R_i^{UB} - C_i}{T_i} \right\rfloor$$

The incompatibility of the stronger schedulability tests for global fixed priority scheduling (the RTA test strictly dominates the DA test) raises the interesting question, which is more powerful, better priority assignment or a better schedulability test. In other words, when faced with the choice, should we use a weaker schedulability test for which we can determine an optimal priority assignment or a stronger test where priority assignment can only be accomplished by using a heuristic.

Fig. 6 shows the success ratio (percentage of schedulable task sets) for a 16 processor system, with 80 tasks reproduced from [49]. Results for the RTA test are shown as dashed lines, while those for the DA test are shown as solid lines. What is striking from the graph is that the difference between the two schedulability tests is small, but the difference between the different priority assignment heuristics (DMPO, DCMPO, DkC) and optimal priority assignment (OPA) is large. In particular, DMPO is shown to be a poor heuristic for global fixed priority scheduling, while the DkC heuristic (based on TkC [6] – see later in this section) has much

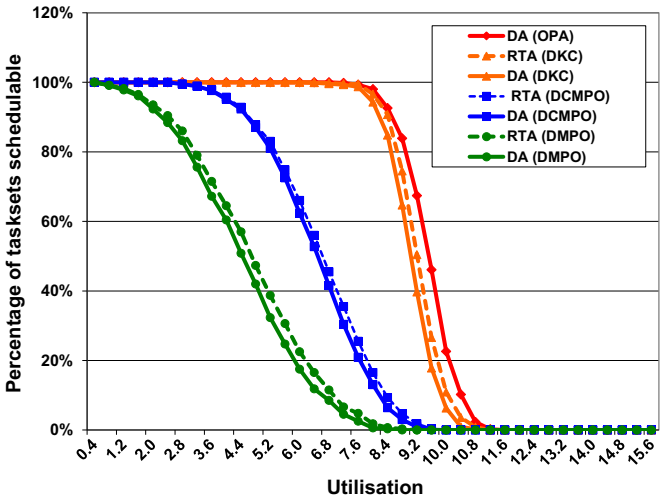


Fig. 6. The effect of priority assignment on task set schedulability for global fixed priority scheduling.

better performance. The best performance was obtained using the weaker DA schedulability test combined with optimal priority assignment, rather than the stronger RTA test and the DkC priority assignment heuristic.

The above findings raise the question, what to do if we have a schedulability test that is effective (like the RTA test), but is not compatible with Audsley's algorithm? Clearly a brute force approach, searching all $n!$ distinct priority orderings is intractable. One viable approach is to direct a backtracking search by identifying partial priority orderings that are definitely schedulable (using a weaker OPA-compatible test) and others that are definitely unschedulable (using an OPA-compatible necessary condition). This approach, introduced in [48] has subsequently been applied in the analysis of real-time flows over a wireless network [84].

A different approach to obtain improvements to the joint schedulability / priority assignment problem for global fixed priority scheduling was taken by Pathan and Jonsson [73] in 2011. Their Hybrid Priority Assignment (HPA) method takes account of the parameters of particular tasks and the intrinsic pessimism in the RTA-LC and DA-LC tests. It assigns the highest priority to a subset of k tasks with high density (execution time divided by deadline) so that they effectively occupy a processor each, and then applies the DA-LC or RTA-LC tests to the remaining tasks on (m-k) processors using Audsley's algorithm and a heuristic priority assignment policy respectively. The approach results in better schedulability by effectively trading a small increase in interference due to assuming that k heavy tasks each utilize a complete processor, for a larger reduction in interference due to a decrease in the number of tasks considered as causing *carry-in* interference from m-1 to m-k-1. This approach dominates DA-LC+OPA and RTA-LC+heuristic priority assignment.

In 2012, Chwa et al. [38] noted that the state-of-the-art schedulability tests for global fixed (task) priority scheduling appeared to outperform those for global fixed job priority scheduling (for example gEDF). They remarked that this was most likely due to ineffective approaches to assigning job priorities in the latter case. They adapted Audsley's OPA algorithm to the problem of assigning job priorities in the form of pseudo deadlines, a task-level parameter used along with the job release times to determine job-level priorities. The resulting Optimal Pseudo Deadline Assignment algorithm, and a heuristic adaptation of it, provide substantially improved schedulability compared to schedulability tests for gEDF,

and also compared to the DA-LC / OPA test for global fixed priority scheduling.

The work of Pathan and Jonsson [73] effectively combined ideas from earlier research into priority assignments aimed at combatting the so called "*Dhall Effect*" [54] with the more sophisticated schedulability tests and Audsley's algorithm applied to a subset of tasks. As an aside, we now give a brief summary of that early work.

In 1978, Dhall and Liu [54] showed that with Rate Monotonic priority order (RMPO), the utilisation bound for implicit deadline periodic tasks under global fixed priority scheduling on m processors is $1+\varepsilon$, for arbitrarily small ε. Hence RMPO and similarly DMPO can be poor priority assignments to use with global fixed scheduling on an identical multiprocessor system. In 2000, Andersson and Jonsson [6] designed the TkC priority assignment policy to avoid the Dhall effect which results in the poor performance of RMPO. TkC assigns priorities based on $T_i - kC_i$where k depends on the number of processors. Via an empirical investigation, Andersson and Jonsson [6] demonstrated the effectiveness of their TkC priority assignment policy for implicit deadline periodic tasksets. (Note the DkC heuristic used in Fig. 6 is a simple extension of TkC using $D_i - kC_i$ to determine the priority order).

Andersson et al. [8] also proposed the RM-US[ς] priority assignment algorithm. This algorithm assigns the highest priority to tasks with utilisation greater than the threshold ς and otherwise assigns priorities in RMPO. They showed that RM-US[$m/(3m-2)$] has a utilisation bound of $m^2/(3m-2)$. In 2005, Bertogna et al. [23] proved an improved bound of $(m+1)/3$ for RM-US[1/3]. Subsequently, in 2008, Andersson [9] proposed a 'slack monotonic' algorithm, called SM-US that works in the same way as RM-US but assigns priorities according to the slack $(T_i - C_i)$ of each task (DCMPO in Fig. 6 similarly uses $D_i - C_i$). SM-US has a utilisation bound of $2/(3+\sqrt{5})m \approx 0.382m$. This bound is better than the corresponding one for RM-US[1/3] when $m \geq 7$.

5.2. Minimising the number of priority levels

So far, we have only considered systems where each task has a unique priority; however, in practice, there may be good reasons for minimising the number of priority levels used. For example an RTOS may support only a limited number of priority levels (e.g. 8 or 16 in OSEK [1]), or there may be many priority levels available, but the system designer may want to minimise the number of priority levels used to reduce the overall stack usage.

Audsley's algorithm permits a simple adaptation (see Algorithm 2) described in [13] that minimises the number of priority levels required.

We note that Algorithm 2 remains an optimal priority assignment algorithm, since it finds a schedulable priority ordering whenever one exists; however, it also has the useful side effect of minimising the number of priority levels required.

5.3. Minimising lexicographical distance

Audsley's OPA algorithm can also be used to minimise the perturbation needed to obtain a schedulable priority ordering from any specified desired ordering. Such perturbations can be measured in terms of either *lexicographical distance* or *reverse lexicographical distance*. To illustrate these terms, let us assume that a set of tasks have been labelled (A, B, C) representing the desired priority ordering from highest to lowest priority. The set of all possible orderings in lexicographical (dictionary) order is given by: (A,B,C), (A,C,B), (B,A,C), (B,C,A), (C,A,B), (C,B,A). Thus the lexicographical distance between the desired ordering (A,B,C) and the ordering (B,A,C) is 2. In reverse lexicographical order, we have instead: (C,B,A), (B,C,A), (C,A,B), (A,C,B), (B,A,C), (A,B,C). This dictionary is constructed by reversing the letters in each word, sorting them in normal (lexicographical) dictionary order and then reversing the letters again. Note the reverse lexicographical distance between the desired ordering (A,B,C) and the ordering (B,A,C) is 1. (This illustrates that lexicographical distance and reverse lexicographical distance are different).

Algorithm 2
Audsley's Algorithm modified to minimize the number of priority levels required.

```
for each priority level i, lowest first {
    Z = empty set
    for each unassigned task τ {
        if τ is schedulable at priority i assuming all unassigned tasks
        have higher priorities {
            add τ to Z
        }
    }
    if no tasks are schedulable at priority i {
        return unschedulable
    }
    else {
        assign all tasks in Z to priority i
    }
    if no unassigned tasks remain {
        break (exit outer loop)
    }
}
return schedulable
```

Minimising lexicographical distance is typically the most useful, since optimising this metric is a way of placing the most important tasks at the highest priority levels while still maintaining schedulability. This provides a simple means of ensuring that should an overload occur, then the most important tasks will still meet their deadlines.

In 2008, Chu and Burns [37] showed that Audsley's algorithm minimises the reverse lexicographical distance to the desired priority ordering if the unassigned tasks are always examined in the reverse of the desired order. In other words, if the desired priority order is (A,B,C), then the task labelled C is the first to be examined at the lowest priority level, followed by task B and so on.

Minimising lexicographical distance is a more difficult problem that was initially addressed by Soto and Bernat [3] in 2006. They used a branch and bound approach to search a tree of possible priority orderings, starting by assigning tasks at the highest priority, and then checking if a branch was schedulable by assuming DMPO for all of the lower priority (unassigned) tasks in that branch. We note that this approach only works when DMPO provides an optimal priority ordering.

Below, we introduce a more general algorithm which minimises the lexicographical distance, we refer to this as the OPA-MLD algorithm.

The OPA-MLD algorithm (Algorithm 3) works as follows: For each priority level i, highest first, the algorithm tries to assign the highest importance unassigned task (i.e. the first such task in lexicographical order) to that priority level. It checks if the task itself is schedulable at priority i and if so, uses the OPA algorithm to determine if there exists a schedulable priority ordering for the other unassigned tasks at lower priority levels. If this is the case, then the trial task is assigned to priority level i, otherwise the algorithm continues with the task of next highest importance and so on until it finds a task to assign, or the system has been found to be unschedulable. Assuming that a task is assigned then the algorithm continues with the next highest priority level.

Since each task is only assigned if there exists some schedulable ordering for the unassigned tasks at lower priority levels, then it is easy to see that the algorithm is optimal (i.e. it always finds a schedulable priority ordering if such an ordering exists). Further, the algorithm constructs an ordering that minimises

Algorithm 3
Optimal Priority Assignment Minimising Lexicographical Distance (OPA-MLD).

```
for each priority level i, highest first {
    for each unassigned task τ in lexicographical order {
        if a schedulable ordering exists for the unassigned tasks by using
        the OPA algorithm on them, assuming that τ is assigned priority
        i and the other previously assigned tasks have priorities higher
        than i {

                assign τ priority i
                break (continue outer loop)

        }
    }
    if no tasks schedulable at priority i {
          return unschedulable
    }
}
return schedulable
```

lexicographical distance. This is the case because it assigns the task with the highest importance (i.e. first in the lexicographical order) whenever there exists a schedulable partial ordering for the unassigned tasks, and it does so in order, highest priority first.

The worst-case complexity of the MLD algorithm can be determined as follows. Let n be the number of tasks, p of which are currently unassigned. Consider the $(n - p+1)$th iteration of the algorithm. There are p tasks each of which is itself schedulable at priority $n - p+1$. (This is the case, since for the first iteration, every task is valid and therefore schedulable at the highest priority; further on each subsequent iteration it is known from the previous iteration that a schedulable priority order exists for those tasks that remain unassigned, which implies that each of unassigned task must be schedulable at the highest unassigned priority level). For each of the p tasks, there are $p - 1$ other unassigned tasks that need their schedulability checked using the OPA algorithm. Hence the number of single task schedulability tests required on this iteration of the algorithm is given by:

$$p(p-1)p/2 = \frac{p^3}{2} - \frac{p^2}{2} \tag{4}$$

Hence the overall complexity of the algorithm is given by:

$$\sum_{p=1}^{n} \left(\frac{p^3}{2} - \frac{p^2}{2} \right) \tag{5}$$

Using the standard formulae for sums of squares and cubes, we have:

$$\sum_{p=1}^{n} \left(\frac{p^3}{2} - \frac{p^2}{2} \right) = n^2(n+1)^2/8 + n(n+1)(2n+1)/12$$
$$= (3(n^4 - n^2) + 2(n^3 - n))/24 \tag{6}$$

The overall complexity of the MLD algorithm is therefore $O(n^4)$ single task schedulability tests, compared to $O(n^2)$ such tests required to find the reverse lexicographical ordering or simply any schedulable ordering using Audsley's algorithm. We note that this higher complexity remains tractable for reasonable sized task sets. For example, for $n = 10$ tasks, 1320 schedulability tests would be required, compared to $n! = 3628{,}800$ combinations of possible priority orderings. For $n = 100$ tasks, 1.25×10^7 schedulability tests would be required, which remains viable, compared to $n! = 10^{158}$ combinations of possible priority orderings, which certainly is not.

We note that when the class of task set being considered has a simple optimal priority assignment, for example DMPO, then that partial ordering can be used in place of the OPA algorithm in the inner loop. This reduces the complexity of each iteration to:

$$p(p-1) = p^2 - p \tag{7}$$

And hence overall complexity to $2(n^3 - n)/6$ single task schedulability tests, or $O(n^3)$: This reduction transforms the OPA-MLD algorithm into the equivalent of the algorithm given by Soto and Bernat [3].

5.4. Task importance and period transformation

Audsley's OPA algorithm focuses on achieving schedulability for all of the tasks in a system under assumptions of normal operation. In some applications; however, there are tasks that are of much higher importance than others, which require preferential treatment under overload conditions. These important tasks should not be impacted by execution time overruns in less important tasks. Appropriate run-time monitoring and budget enforcement is one way to achieve this behaviour; however, in simple systems fixed priority scheduling alone may be sufficient assuming a priority assignment that reflects task importance, with tasks of higher importance given higher priority. We have already seen that the OPA-MLD algorithm provides a means of constructing such a priority assignment when it is viable without compromising schedulability. However, if the important tasks have long execution times relative to the deadlines of other tasks then this may not be possible. In such systems, one simple technique that may be used is *period transformation* [81]. Here, important tasks with long periods are subdivided e.g. into two parts each with half the execution time and half the period. While this subdivision has the disadvantage that it requires changes to the code and increases scheduling overheads, it has the advantage that the task may then be represented as having a shorter period (and deadline) and thus becomes amenable to being given a higher priority without compromising the schedulability of other tasks.

6. Robust priority assignment

While Audsley's OPA algorithm can be applied in a broad range of cases, it has one significant drawback. It makes an arbitrary choice of which schedulable task to assign at each priority level. Such an arbitrary assignment can easily leave the system only just schedulable, and thus fragile to any minor changes in task parameters or under estimations of interference or execution budgets. This is a problem in practice, since tasks may be subject to *additional interference* in the form of execution time budget overruns, interrupts occurring at ill-defined rates, ill-defined RTOS overheads, ill-defined critical sections, and cycle stealing by peripheral devices (e.g. DMA). What is really needed is a robust priority ordering that is able to tolerate the maximum amount of such additional interference.

This problem was addressed by Davis and Burns in their work on Robust Priority Assignment [45]. They assumed a general additional interference function $E(\alpha, w, i)$, where α is a scaling factor, used to model variability in the amount of interference, w is the length of the time interval over which the interference occurs and i is the priority level affected by the interference. The function $E(\alpha, w, i)$ is required to be a monotonically non-decreasing function of its parameters. In practice, this represents little restriction, since almost all sources of interference are (i) greater in longer intervals of time than shorter ones, (ii) affect lower priority tasks if they also affect higher priority ones, and (iii) are in any case guaranteed to be monotonic in α since that is the scaling factor.

Robust Priority Assignment is defined as follows:

Definition. (from [45]): *robust priority assignment policy*: "For a given system model and additional interference function, a priority assignment policy P is referred to as *robust* if there are no systems, compliant with the system model, that are both schedulable and can tolerate additional interference characterized by a scaling factor α using another priority assignment policy Q that are not also

Algorithm 4
Robust Priority Assignment (RPA) Algorithm.

```
for each priority level i, lowest first {
    for each unassigned task τ {
        determine the largest value of α for which task τ is schedulable
        at priority i assuming that all unassigned tasks have higher
        priorities
    }
    if no tasks are schedulable at priority i {
        return unschedulable
    }
    else {
        assign the schedulable task that tolerates the max α at priority i to
        priority i
    }
}
return schedulable
```

Table 4
Task parameters.

Task	C	T	D
τ_A	125	450	450
τ_B	125	550	550
τ_C	65	600	600
τ_D	125	1000	1000
τ_E	125	2000	2000

Table 5
Computed values of α.

Priority	Task				
	τ_A	τ_B	τ_C	τ_D	τ_E
5	NS	NS	NS	120	**354**
4	NS	NS	NS	**120**	–
3	10	**110**	74	–	–
2	135	–	**199**	–	–
1	**200**	–	**–**	–	–

both schedulable and can tolerate additional interference characterized by the same or larger scaling factor using priority assignment policy *P*."

Stated otherwise, of all schedulable priority orderings, the robust priority ordering tolerates the most additional interference (i.e. largest value of α).

The Robust Priority Assignment (RPA) algorithm (see Algorithm 4) is based on Audsley's OPA algorithm and requires exactly the same three Conditions to be applicable. (Since the additional interference function is monotonically non-decreasing in its parameters, if the three Conditions hold for OPA, then they continue to do so when additional interference is considered in RPA). This means that the RPA algorithm is compatible with any schedulability test that is compatible with OPA. The RPA algorithm provides a priority ordering that is both optimal (easily seen by equivalence to Audsley's algorithm) and robust, as proven in [45].

It is instructive to compare the robust priority ordering with both DMPO and that generated by OPA on an example. The following example taken from [45] considers robust priority assignment for the tasks in Table 4 assuming fixed priority non-pre-emptive scheduling (FPNS) and the simplest possible additional interference function $E(\alpha, w, i) = \alpha$. Such an interference function might represent the unknown execution time of an interrupt handler that runs infrequently (at most once in any busy period).

Table 5 gives the values of α computed[4] by the RPA algorithm as it iterates from the lowest to the highest priority level. ('NS' indicates that a task was not schedulable at that particular priority even without any additional interference). The values highlighted in bold indicated the task that tolerated the maximum value of α at a particular priority level, and hence was assigned that priority. The robust priority ordering for this example is therefore $(\tau_A,\tau_C,\tau_B,\tau_D,\tau_E)$ which tolerates a maximum amount of additional interference of 110 time units. By comparison, DMPO $(\tau_A,\tau_B,\tau_C,\tau_D,\tau_E)$ results in values of α of (200, 175, 74, 120, 354) and hence tolerates a maximum amount of additional interference of 74 time units. As a number of priority orderings are schedulable without additional interference, the ordering chosen by the OPA algorithm depends upon the order in which the tasks are checked. If this order is $\tau_A,\tau_B,\tau_C,\tau_D,\tau_E$ then the priority ordering produced by OPA would be $(\tau_C,\tau_B,\tau_A,\tau_D,\tau_E)$ which tolerates a maximum amount of additional interference of just 10 time units. This example serves to illustrate the practical importance of not just selecting *any* schedulable priority ordering, but one that is robust.

Davis and Burns [45] proved the negative result that in general, the robust priority ordering depends on the form of the additional interference function and can therefore only be precisely determined if α is the only unknown in the function $E(\alpha, w, i)$. Nevertheless, this is often the case, and in practice, it can be instructive to use a simple additional interference function such as $E(\alpha, w, i) = \alpha$ to obtain a robust priority assignment. Further, they showed that in the case of systems where the scheduling policy (e.g. FPPS) and task parameters (e.g. constrained deadlines, resource accesses according to SRP or PCP, no offset release times), are such that DMPO is optimal, then DMPO is also the robust priority ordering irrespective of the form of the additional interference function, provided only that it is monotonically non-decreasing in its parameters.

Classifying tasks into those whose parameters are compatible with DMPO being optimal, so called *DM tasks*, and tasks whose parameters do not meet those criteria (*non DM tasks*), Davis and Burns proved the following result[5] for fixed priority pre-emptive scheduling.

Theorem 3. *(from Theorem 4 in [45]). For a system of DM and non DM tasks, where a schedulable priority ordering exists, there is a robust priority ordering P with the DM tasks in Deadline Monotonic partial order.*

Theorem 3 effectively says that we may always place DM tasks in Deadline Monotonic order and only need to determine how the non-DM tasks should be interleaved among them.

This result can be used to improve the efficiency of Audsley's algorithm and the RPA algorithm. Theorem 3 tells us that of all the DM tasks, the task with the longest deadline is the one that is able to tolerate the most additional interference at any given priority level, hence in the OPA and RPA algorithms, only one DM task need be checked at each priority level, the one with the longest deadline of all unassigned DM tasks. This reduces the number of single task schedulability tests needed from $n(n+1)/2$ to $(n(n+1) - k(k-1))/2$ when there are n tasks in total, of which k are DM tasks. For example, in a system with $n=50$ sporadic tasks, 46 of which have constrained deadlines, and $k=4$ of which have arbitrary deadlines, a maximum of 240 schedulability tests are needed instead of 1275.

Robust Priority Assignment has been extended to messages on Controller Area Network [47], showing how the RPA algorithm can be used to maximise the number of errors that could be tolerated on the network before any messages missed their deadlines, or

[4] Via binary search down to a granularity of 1 time unit.

[5] This also applies to tasks with release jitter and Deadline minus Jitter Monotonic Priority Ordering. We state the simpler form here.

to maximise the delay (bus unavailability) that could be tolerated. Schmidt [80] also used RPA as the basis for an algorithm which assigns message priorities (IDs) on CAN when a subset of the IDs are already fixed. Flaws in this work were later addressed in [98].

Prior to the work on Robust Priority Assignment, related research by Lehoczky et al. [68], Katcher et al. [65], Punnekkat et al. [74], and Regehr [76] used the *critical scaling factor* as a metric for determining task set schedulability. (The critical scaling factor was defined by Lehoczky et al. [68] as the largest scaling factor by which the worst-case execution time of every task could be increased without the task set becoming unschedulable). Regehr showed that for task sets where DMPO is the optimal priority assignment policy, it also maximises the critical scaling factor.

6.1. Priority assignment in mixed criticality systems

Theorem 3 has subsequently been used to achieve a significant simplification of the problem of priority assignment in mixed criticality systems scheduled using fixed priorities [17].

In the standard task model for mixed criticality systems, introduced by Vestal in 2007 [90], tasks have different criticality levels (e.g. HI and LO) equating to the level of assurance required for their correct and timely operation. HI-criticality tasks have different execution time bounds $C_i(LO)$ and $C_i(HI)$ for these criticality levels, representing estimates of the WCET of the task with different levels of assurance. For example a certification authority may require that highly conservative WCET estimates are used for $C_i(HI)$ for the flight-control software of a Unmanned Aerial Vehicle (UAV), whereas the system designer may use less conservative methods perhaps based on measurements to find $C_i(LO)$ for the same software ($C_i(HI) \geq C_i(LO)$).

Mixed criticality systems operate in different criticality modes: In LO-criticality mode, all tasks must meet their deadlines, assuming LO-criticality execution times for all tasks. In HI-criticality mode, all HI-criticality tasks must meet their deadlines assuming HI-criticality execution times, while LO-criticality tasks may be abandoned to ensure timely operation of the HI-criticality tasks.

The system starts in LO-criticality mode and transitions to HI-criticality mode when a HI-criticality task exceeds its LO-criticality execution budget. (Transition back to LO-criticality mode may take place when the processor becomes idle).

The analysis for Adaptive Mixed Criticality (AMC) scheduling based on fixed priorities [17] is formulated in the equations below:

$$R_i(LO) = C_i(LO) + \sum_{\forall j \in hpH(i)} \left\lceil \frac{R_i(LO)}{T_j} \right\rceil C_j(LO) + \sum_{\forall k \in hpL(i)} \left\lceil \frac{R_i(LO)}{T_k} \right\rceil C_k(LO) \quad (8)$$

where $hpL(i)$ ($hpH(i)$) is the set of LO-criticality (HI-criticality) tasks with priorities higher than that of task τ_i.

$$R_i(HI) = C_i(HI) + \sum_{\forall j \in hpH(i)} \left\lceil \frac{R_i(HI)}{T_j} \right\rceil C_j(HI) + \sum_{\forall k \in hpL(i)} \left\lceil \frac{R_i(LO)}{T_k} \right\rceil C_k(LO) \quad (9)$$

For LO-criticality tasks, the LO-criticality response time $R_i(LO)$ computed via (8) must be no greater than the task's deadline. For HI-criticality tasks, the HI-criticality response time computed via (9) must also be no greater than the deadline. Notice that in (9) the interference term for higher priority LO-criticality tasks is limited to releases within $R_i(LO)$, since that is an upper bound on the time that can be spent in LO-criticality mode since task τ_i was released (otherwise task $\tau_i$would itself cause a transition to HI-criticality mode).

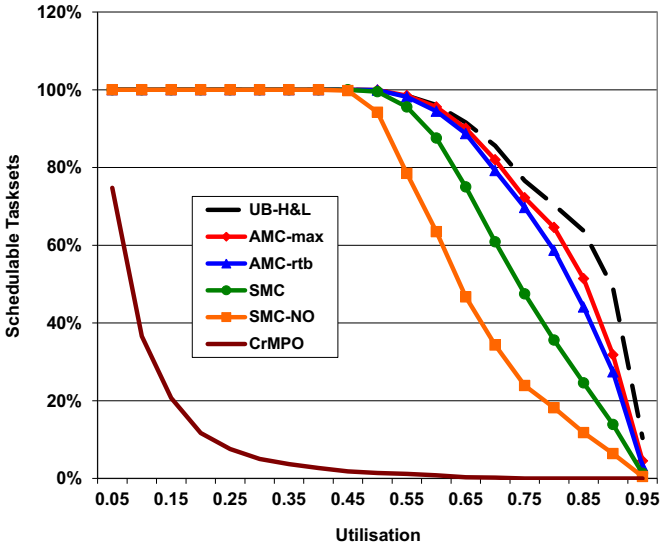


Fig. 7. Poor performance of Criticality Monotonic Priority Ordering (CrMPO).

Given the previous discussion about robust priority assignment, (8) and (9) can be interpreted in a different way. For LO-criticality tasks, the first summation term in (8) can be considered as additional interference and the LO-criticality tasks, as a set of DM tasks. Similarly for the HI-criticality tasks the second summation term in (9) can be interpreted as additional interference, and the HI-criticality tasks considered as a set of DM tasks. (Note that for a HI-criticality task, (9) is always a stricter test than (8)). It follows from Theorem 3 that a robust priority ordering exists that has the LO-criticality tasks in DM partial order, and also the HI-criticality tasks in DM partial order. Thus robust priority assignment reduces to a merge between the two sets, each sorted in DM order, as shown in [17]. This merge is accomplished by a variant of the OPA or RPA algorithms which checks only the longest deadline, unassigned HI-criticality task and the longest deadline, unassigned LO-criticality task at each priority level. Thus the maximum number of schedulability tests required is reduced from quadratic $(n(n+1)/2)$ to linear $(2n-1)$.

An alternative simple approach to scheduling mixed criticality systems is to partition the priorities, such that all HI-criticality tasks have higher priorities than all the LO-criticality tasks. This approach, referred to as Criticality Monotonic Priority Ordering (CrMPO) has the advantage that run-time policing of LO-criticality execution budgets may not be required, and there is no need to abandon LO-criticality tasks (or prevent new releases) when a HI-criticality task executes for its $C(LO)$ execution time budget without signalling completion. Fig. 7, reproduced from [17], shows the performance of CrMPO in relation to AMC-rtb which uses the analysis embodied in (8) and (9), along with a modified version of Audsley's algorithm for priority assignment.

Observe that the performance of CrMPO is relatively poor, due to the priority inversion inherent in giving short deadline LO-criticality tasks low priorities. We note that this issue can be addressed in part by Period Transformation techniques [81] that divide the periods of HI-criticality tasks so that they have shorter periods and deadlines than any LO-criticality tasks; however, this method creates additional overheads and loses its effectiveness with more criticality levels [57]. The relatively poor performance of CrMPO shows the importance of appropriate priority assignment in mixed criticality systems.

Table 6
Task parameters.

Task	C	D	T
τ_A	100	175	250
τ_B	100	300	400
τ_C	100	325	350

Note that the lines in Fig. 7 labelled SMC-NO, SMC, and AMC-max, represent other fixed priority mixed criticality scheduling schemes and analyses, while UB-H&L represents an upper bound on the performance of any such scheme that uses fixed priorities, for full details see [17].

7. Optimal fixed priority scheduling with deferred pre-emption

In the previous section on robust priority assignment, we saw how Audsley's optimal priority assignment algorithm can be augmented to also optimise an additional criterion, in that case robustness in terms of maximising the amount of additional interference that the system can tolerate before a deadline is missed. Davis and Bertogna [52] showed that Audsley's algorithm can be adapted in a similar way to optimise fixed priority scheduling with deferred pre-emption (FPDS) [31].

Recall that with FPDS, each task has a final non-pre-emptive region of length F_i. If for all tasks, this region is of the minimum possible length i.e. $F_i = 1$, then FPDS equates to fixed priority pre-emptive scheduling (FPPS), whereas if for all tasks, it is equal to the task's worst-case execution time i.e. $F_i = C_i$, then FPDS equates to fixed priority non-pre-emptive scheduling (FPNS). Thus FPDS subsumes and strictly dominates both FPPS and FPNS, since it can schedule any task set that is schedulable according to either of those policies.

The dominance of FPDS is illustrated by the example set of tasks in Table 6 (reproduced from [52]) and the schedule of their execution shown in Fig. 8. It is interesting to consider the different possible priority orderings and scheduling policies for this example. With any form of fixed priority scheduling (FPPS, FPNS, or FPDS), then the short deadline of 175 for task τ_A means that it must necessarily be assigned the highest priority, otherwise it will be unschedulable. (Assigning task τ_A a lower priority would result in a response time of at least 200 due to interference from whichever of tasks τ_B or τ_C was given the highest priority).

Considering fully non-pre-emptive scheduling (FPNS), there is clearly no schedulable priority ordering since task τ_A cannot tolerate blocking of 100 from either of tasks τ_B or τ_C. Considering fully pre-emptive scheduling (FPPS), we know that deadline monotonic priority order (DMPO) i.e. (τ_A,τ_B,τ_C) is optimal [67]; however, in this case task τ_C would miss its deadline at time 325 due to interference from the second job of task τ_A released at time 250. Similarly, if task τ_B were placed at the lowest priority, it would miss its deadline at time 300, hence there is no schedulable priority ordering for FPPS.

Considering FPDS, we might try either task τ_B or τ_C at the lowest priority. Fig. 8(a) illustrates what happens with deadline monotonic priority order (DMPO) i.e. with task τ_C at the lowest priority. In this case, the best possible schedulability for task τ_C is obtained if it has the longest possible final non-pre-emptive region, i.e. $F_C = C_C = 100$ even so, the second job of task τ_C still misses its deadline at time 675. Hence the system is unschedulable under FPDS with DMPO. Finally, we consider priority ordering (τ_A,τ_C,τ_B) and thus task τ_B at the lowest priority. In this case, with a final non-pre-emptive region of length $F_B = 51$, both jobs of task τ_B in the busy period meet their deadlines, as illustrated in Fig. 8(b). Assuming the minimum non-pre-emptive region lengths (i.e. $F_A = 1$, $F_C = 1$) for tasks τ_A and τ_C, then all three tasks are schedulable under FPDS, with worst-case response times of 150, 250, and 300 respectively. This example serves to show the strict dominance of FPDS over both FPPS and FPNS, and also the non-optimality of DMPO for fixed priority scheduling with deferred pre-emption. It also shows that to obtain the best possible performance from FPDS then it is necessary to determine an appropriate assignment of both task priorities and final non-pre-emptive region lengths.

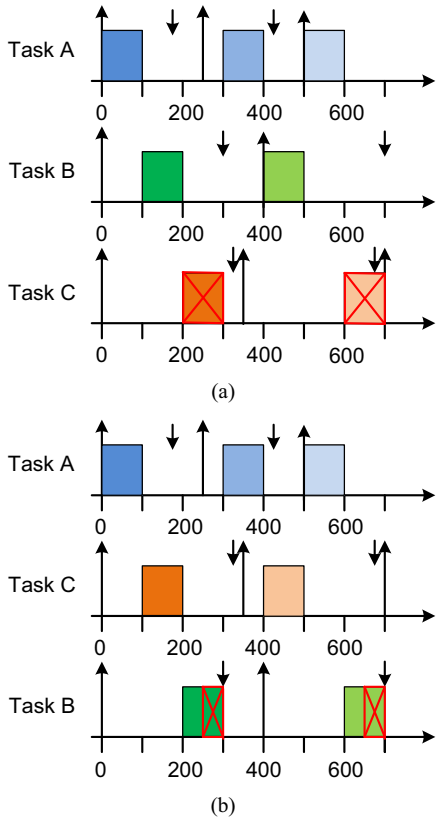


Fig. 8. Deadline Monotonic priority ordering is not optimal for fixed priority non-pre-emptive scheduling.

Algorithm 5
FNR-PA Algorithm.

```
for each priority level k, lowest first {
    for each unassigned task τ {
        determine the smallest value for the final non-pre-emptive region
        length F(k) such that task τ is schedulable at priority k, assuming
        all other unassigned tasks have higher priorities.
        Record as task Z the unassigned task with the minimum value for
        the length of its final non-pre-emptive region F(k).
    }
    if no tasks are schedulable at priority k {
        return unschedulable
    }
    else {
        assign priority k to task Z and use the value of F(k) as the length of
        its final non–pre-emptive region.
    }
}
return schedulable
```

Building upon exact schedulability analysis for FPDS derived by Bril et al. [24], Davis and Bertogna [52] modified Audsley's algorithm to assign *both* priorities and final non-pre-emptive region lengths. They proved that the *Final Non-pre-emptive Region and Priority Assignment* (*FNR-PA*) algorithm (Algorithm 5) is optimal for FPDS, stating that "it is guaranteed to find a combination of priority assignment and final non-pre-emptive region lengths that

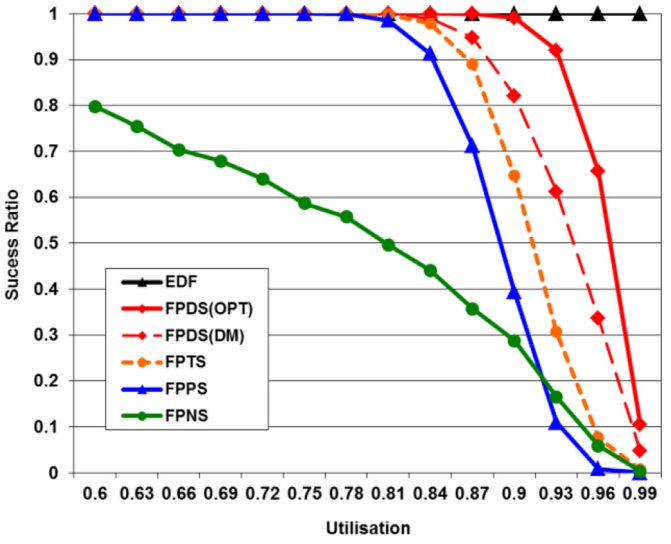


Fig. 9. Success ratio for $n = 10$, D = T.

result in a schedulable system under FPDS whenever such a schedulable combination of these parameters exists".

Fig. 9 (reproduced from [52]) illustrates the comparative performance in terms of the proportion of schedulable task sets for using the optimal FNR-PA Algorithm (red line), FPPS assuming deadline monotonic priority (blue line), and FPNS assuming an optimal priority ordering found using Audsely's algorithm (green line). Comparison is also made against Fixed priority Pre-emption Threshold Scheduling (FPTS) (dashed orange line) [91,79].

The difference between FPDS(OPT) – solid red line –and the dashed red line which shows the performance of FPDS using DMPO [22] highlights the improvement that jointly optimizing both priority assignment and final non-pre-emptive region lengths brings.

Research into fixed priority scheduling with deferred pre-emption has one of its practical applications in automotive systems. The automotive RTOS standards OSEK [1] and AUTOSAR [2] mandate fixed priority scheduling, and support co-operative scheduling of tasks made up of multiple non-pre-emptive regions. According to Buttle [35] in automotive systems there are often large numbers of separate functions (or runnables) that execute one after another within relatively few tasks (typically 50-300 functions per task). To avoid issues with access to global variables and to reduce stack usage, these functions need to be executed non-pre-emptively with re-scheduling only permitted between them. Davis and Bertogna [52] showed how the FNR-PA algorithm can be adapted to optimise task priorities and final non-pre-emptive region lengths, taking into account the constraints on when pre-emption is permitted due to the separate functions that make up each task. Thus FPDS provides an approach that can be implemented in automotive systems that use an OSEK [1] or AUTOSAR [2] compliant RTOS, improving upon the performance of FPPS and FPNS.

Other methods of limiting pre-emption include *Pre-emption Thresholds* (FPTS) [91,79] and *Non-pre-emption Groups* [44], which were implemented as *internal resources* in the OSEK [1] and AUTOSAR [2] automotive RTOS standards. Here, each task has a *base* priority at which it initially competes for the processor; however, once it starts to execute, then it assumes a *threshold* or *dispatch* priority. This limits pre-emption to those tasks that have a base priority higher than the threshold. Recent research by Bril et al. in 2012 [25] generalises the concepts of pre-emption thresholds and deferred pre-emption, providing a scheme whereby pre-emption thresholds apply between a set of functions or sub-jobs that execute non-pre-emptively within each task. For further information on limited pre-emption scheduling the reader is referred to the survey by Buttazzo et al. [34].

Table 7
Task parameters.

Task	C	D	T	DMR threshold Ψ
τ_A	$\binom{2}{0.7}\ \binom{3}{0.3}$	5	10	0.5
τ_B	$\binom{3}{0.8}\ \binom{4}{0.2}$	6	10	0.05

8. Priority assignment in probabilistic real-time systems

In the previous section on fixed priority scheduling with deferred pre-emption, we saw how Audsley's algorithm could be adapted to simultaneously optimise both priority assignment and final non-pre-emption region length. In this section we see how a similar adaptation is useful in the domain of probabilistic real-time systems.

In probabilistic real-time systems, we are interested in the probability that tasks or messages will miss their deadlines, rather than an absolute guarantee that they will never do so. These probabilities arise from random events that affect the timing behaviour of the system. These events may be external, for example errors on a Controller Area Network (CAN) bus modelled as a Poisson distribution [72,47], or internal, for example due to the behaviour of a cache that uses a random replacement policy [19]. In the latter case, the worst-case execution times of tasks may be expressed as a Probability Mass Function (PMF) (referred to as a probabilistic WCET distribution or pWCET), rather than a single value. These distributions may be found using either static [36,5], or measurement-based [41] probabilistic timing analysis. Provided that the random variables representing the pWCET of each job of a task are independent[6] [42], then these values can be combined using probabilistic response time analysis, based on the convolution operator, to obtain a distribution for the worst-case response time for each task [55].

An example of tasks with worst-case execution times expressed as independent random variables is given in Table 7.

Here, a job of task τ_A has a probability of 0.7 that it will not execute for more than two time units, and a probability of 1.0 ($= 0.7 + 0.3$) that its execution time will not exceed 3. Similarly a job of task τ_B has a probability of 0.8 that it will execute for no longer than three time units, and a probability of 1.0 ($= 0.8 + 0.2$) that its execution time will not exceed four.

In probabilistic real-time time systems, deadlines may be missed providing the probability of this occurring is suitably small, and so we need to redefine what we mean by "schedulable". Maxim et al. [71] use the Deadline Miss Ratio[7] (DMR) for this purpose, since it can be mapped to a failure rate per hour that may be specified for a task by multiplying by the number of jobs per hour. In this way, a task is deemed "schedulable", if it's DMR does not exceed the specified threshold Ψ. (As usual, a task set is schedulable if all of its tasks are schedulable).

The DMR_i of a task τ_i is computed over some time interval $[a, b]$, typically the hyperperiod or least common multiple of task periods. It is given by the sum of the probabilities of each job of task τ_i that runs in that interval missing its deadline, divided by

[6] Note independence of the pWCETs of jobs is different from the independence of their execution times as explained in [42].

[7] We note that the DMR is a failure rate as distinct from a probability.

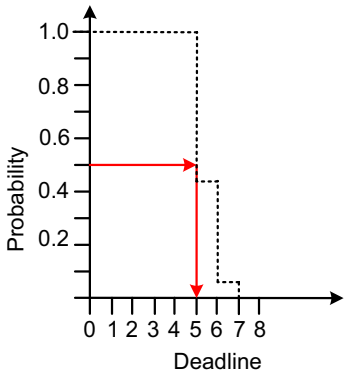


Fig. 10. Exceedance function (1-CDF).

the number of jobs:

$$DMR_i = \frac{1}{n_{[a,b]}} \sum_{j=1}^{n_{[a,b]}} P(\Re_{i,j} > D_i) \tag{10}$$

Where $P(\Re_{i,j} > D_i)$ is the probability that the response time of job j of task τ_i exceeds its deadline. Note $\Re_{i,j}$is a random variable representing the response time distribution of the job. $P(\Re_{i,j} > D_i)$ may be assessed by inspecting the Probability Mass Function of the response time and comparing it with the deadline. Fig. 10 illustrates this via a 1-CDF (Complementary Cumulative Distribution Function).

The thresholds Ψ equating to the maximum permitted Deadline Miss Ratios are given for tasks τ_A and τ_B in Table 7. Maxim et al. [71] showed that for task sets where computation times are described by independent random variables, but periods and deadlines are deterministic (i.e. single) values, and deadlines are constrained, then DMPO is not an optimal priority assignment policy for FPPS. This is illustrated by the tasks in Table 7.

With priority ordering (τ_A, τ_B), i.e. DMPO, then we have $P(\Re_A > D_A) = 0$ and $P(\Re_B > D_B) = 0.06$ (which is the probability that τ_A executes for 3 time units *and* τ_B executes for 4 time units). Note we dropped the job index since in this example there is just one job of each task in the hyperperiod. Since $P(\Re_B > D_B) > \Psi_B$ the task set does not meet its timing requirements, in effect it is unschedulable. However, if we change the priority order to (τ_B, τ_A), then we have $P(\Re_A > D_A) = 0.44 \leq \Psi_A$ and $P(\Re_B > D_B) = 0 \leq \Psi_B$ which meets the timing requirements.

Maxim et al. [71] showed that Audsley's algorithm can be used to determine an optimal priority assignment that meets constraints on the Deadline Miss Ratio of each task. We note that with a suitable definition of what is meant by schedulable, then the same three Conditions, stated in Section 5 are sufficient to determine if a schedulability test for a probabilistic real-time system is compatible with Audsley's algorithm.

Maxim et al. [71] also showed that the maximum DMR of any task can be minimised at the same time as finding an optimal priority assignment by choosing the task to assign at each priority level from the set of unassigned tasks by selecting the schedulable one with the smallest DMR. This approach used similar techniques to those employed by Davis and Burns in their work on Robust Priority assignment for messages on Controller Area Network [47]. They examined the schedulability of networks subject to errors according to a random process (Poisson distribution). In this case, the key criterion to optimise was the worst-case deadline failure probability (WCDFP) of each message.

Davis and Burns [47] adapted Audsley's algorithm to form a Probabilistic Robust Priority Assignment Algorithm (Algorithm 6), with the WCDFP computed according to analysis given by Broster et al. [28,29]. They gave an interesting example of the impact of priority assignment on the WCDFP as shown in Fig. 11 (reproduced from [47]).

Algorithm 6
Probabilistic Robust Priority Assignment (PRPA) Algorithm.

```
for each priority level m, lowest first
{
    for each unassigned message M
    {
        Compute the WCDFP of message M at priority m
    }
    if no messages are schedulable at priority m
        return unschedulable
    else
        assign the message with the smallest WCDFP at priority m to
        priority m
}
return schedulable
```

These results are for a system of 5 messages labelled A, B, C, D, E and hence 120 distinct priority assignments. The graph shows the WCDFP on a log scale against the set of 120 distinct priority orders (in lexicographical, i.e. dictionary, order) where the first priority order (A,B,C,D,E) corresponds to Deadline minus Jitter Monotonic Priority Order (DJMPO).

It is notable that the robust priority orders have a maximum WCDFP that equates to a failure rate of 1 in 28,500, whereas there are 62 priority orderings with failure rates in the range of 1 in 500 to 1 in 1000, with the remaining 54 priority orderings corresponding to failure rates of 1 in 20. This illustrates the importance of appropriate priority assignment in obtaining a robust system that is less likely to result in missed deadlines in the event of errors on the bus.

9. Problems not amenable to OPA

In the previous sections, we described Audsley's algorithm for Optimal Priority Assignment (OPA), and discussed the three conditions required for a schedulability test to be compatible with it. We also saw how Audsley's algorithm has been adapted to optimise other criteria, such as the number of priority levels, the robustness of the system to additional interference or delays, the lengths of final non-pre-emptive regions for systems using FPDS, and also the maximum probability of deadline failure in probabilistic real-time systems.

In this section, we list a number of interesting problems where Audsley's algorithm is not obviously applicable, and so it is an open problem whether optimal priority assignment can be achieved via an algorithm that is tractable. The problems themselves are not open since one could in theory try all $n!$ priority orderings; however, that is clearly not tractable even for moderate values of n.

- *FPDS*: Minimising the number of pre-emptions through maximising the length of non-pre-emptive regions. This can be done from highest priority down, rather than lowest priority up, but then requires a pre-defined priority ordering as shown by Bertogna et al. [22]. Minimising the number of pre-emptions in this way can improve schedulability by reducing overall context switch costs including Cache Related Pre-emption Delays (CRPD), thus solutions to this problem are important for single processor systems that use cache to speed up memory accesses.
- *Pre-emption thresholds*: Assignment of base priorities and pre-emption thresholds [91]. This is problematic since appropriate pre-emption threshold assignment depends on the relative priority ordering of higher priority tasks. Pre-emption threshold scheduling is an effective means of improving schedulability,

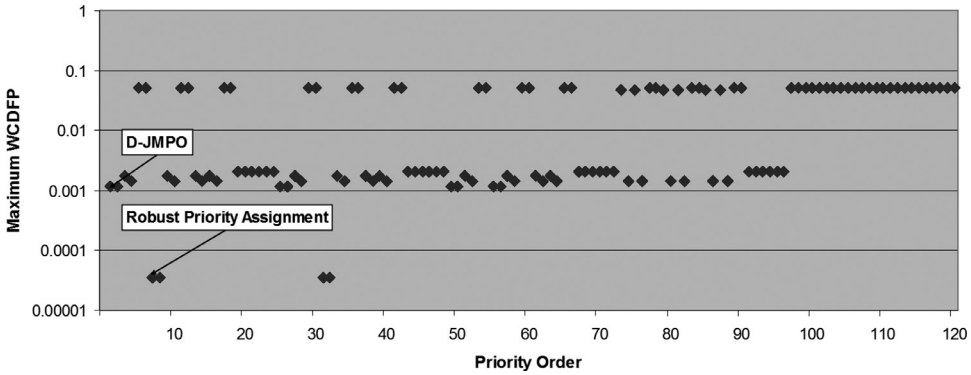


Fig. 11. WCDFP as a function of Priority Ordering.

that can reduce context switch costs including CRPD and also reduce stack usage, thus solutions to this problem are again useful for single processor systems that use cache to speed up memory accesses.

- *Probabilistic*: Minimising the total probability of deadline failure across all tasks in a probabilistic real-time system. Swapping tasks at adjacent priorities may decrease this total, even if the larger of the two probabilities of deadline failure for the individual tasks increases as shown by Maxim et al. [71]. This problem is interesting since in assessing the behaviour of a system as a whole, it is the failure rate of the ensemble of tasks implementing a particular function that is important rather than the failure rate of a single component task.
- *Network-on-Chip (NoC) wormhole communication*: Assigning priorities to network flows. Here, the response time of a network flow depends on the response times of higher priority flows as shown by Zheng and Burns [93]. Achieving optimal priority assignment for this problem would improve schedulability, enabling more real-time traffic to be supported on the network.
- *Abort-and-restart:* This task model is used in Functional Reactive Programming [14]. When a task is pre-empted by a higher priority task, then it is aborted and has to be restarted once the higher priority tasks finish executing. Here, task response times depend on the relative priority ordering of higher priority tasks as shown by Wong and Burns [92]. Solutions to this problem would improve the schedulability of systems implemented using FRP.
- *Polling Periods and Event Deadlines*: In this task model, the system is defined by event deadlines, which must be met by polling tasks which check for occurrence of the event [32]. Hence each task's period is determined by its event deadline minus its worst-case response time. Here, task response times depend on the relative priority ordering of higher priority tasks and so Audsley's algorithm is not applicable. (For the restricted case where all tasks share the same execution time, then Event Deadline Monotonic priority ordering is optimal [32]). Solutions to this priority assignment problem would improve the schedulability of systems built using this model.

The integration and analysis of overheads due to Cache Related Pre-emption Delays (CRPD) into fixed priority pre-emptive scheduling [4] also leads to an interesting and difficult to solve problem of priority assignment. This is illustrated in Fig. 12, which shows the interaction between priority assignment and CRPD.

Task τ_A has Useful Cache Blocks (UCBs) that are evicted by task τ_B (i.e. the same blocks are Evicted Cache Blocks (ECBs) of τ_B), but not vice-versa. Thus if task τ_A is given higher priority, then there is no CRPD on task τ_B as shown in Fig. 12(a); however, if we swap priorities, then when task τ_B pre-empts task τ_A, task τ_A incurs a CRPD re-loading the cache blocks that it uses that were evicted by

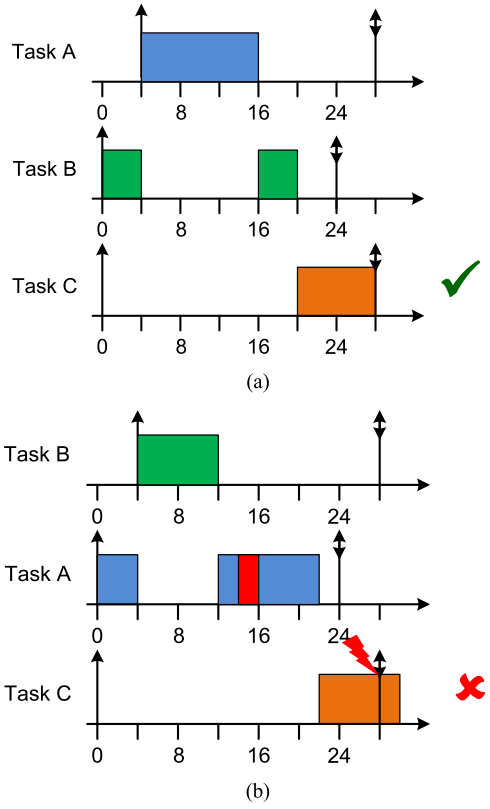


Fig. 12. Interaction between priority assignment and CRPD.

task τ_B. This has a knock-on effect on the schedulability of task τ_C (see Fig. 12(b)). This means that the schedulability of task τ_C depends on the relative priority ordering of the two higher priority tasks τ_A and τ_B, breaking Condition 1 required for Audsley's algorithm to be applicable. Thus when CRPD is integrated with schedulability analysis for FPPS as in [4], then the schedulability tests are no longer compatible with Audsley's OPA algorithm.

Solutions to this priority assignment problem would improve the schedulability of single processor systems that use cache to speed up memory accesses.

9.1. Distributed systems: allocation and assignment

All of the priority assignment policies and algorithms discussed so far rely for their operation on the existence of well-defined deadlines that apply to a single operation, for example the execution of a task or the transmission of a message. In simple

systems, directly connected to sensors and actuators, such deadlines can be defined based on the required behaviour (maximum time allowed from stimulus to response) or the designed behaviour (e.g. the periods of control algorithms) as well as requirements to avoid buffering or other I/O issues. The latter often leading to deadlines that are either implicit or constrained.

In complex, distributed real-time systems such as those found in automotive applications, the timing requirements on the system typically come from end-to-end deadlines imposed on functionality that is implemented by tasks distributed across a number of processors that communicate via messages sent over one or more networks e.g. CAN. Here, division of the end-to-end deadline into sub-deadlines on individual tasks and messages can provide a way of achieving schedulability for the larger problem [78,58]. Such a divide and conquer approach enables the use of the priority assignment policies discussed in this review for individual processors and networks; however, such subdivision can also potentially lead to sub-optimal solutions.

An alternative approach is to use holistic techniques [88], to analyse the system as a whole while taking into account propagation delays along the end-to-end flows. The problem then becomes one of determining an appropriate allocation of tasks to processors, (signals to messages[8] on CAN) and priority assignment for both tasks and messages that meet all of the time constraints. Since this problem is NP-hard, solutions proposed include the use of search and optimisation techniques such as: Branch and Bound [77], Simulated Annealing (SA) [87,18], SA and geometric programming [63], genetic algorithms [62], and Mixed Integer Linear Programming (MILP) [94,95]. These techniques are typically capable of optimising other metrics, such as different forms of extensibility or robustness [18,95], as well as schedulability.

10. Summary and conclusions

This tutorial-style survey and review examined the importance of priority assignment in systems scheduled using fixed priorities. We started with a graphic example based on Controller Area Network (CAN) showing how ignoring appropriate priority assignment techniques can reduce achievable bus utilisation from around 80% down to below 35%. This is one of the reasons for the current myth in some parts of the Automotive industry that CAN is only able to operate at around 35% utilisation without missing deadlines.

We provided a guided tour of early work on priority assignment, showing how Deadline Monotonic priority assignment is optimal for some simple systems; however, small changes to the assumptions (for example allowing offset release times, deadlines greater than periods, non-pre-emptive, or deferred pre-emption scheduling) break this optimality. In many cases, Audsley's Optimal Priority Assignment (OPA) algorithm is applicable. There are three Conditions which schedulability tests must meet in order for Audsley's algorithm to apply. These conditions greatly reduce the burden of proof required to show that a particular schedulability test is compatible with OPA.

We also described how Audsley's algorithm can be modified to minimise the number of priority levels required or to minimise the reverse lexicographical distance from any desired priority ordering. Further, we introduced a new variant of Audsley's algorithm, OPA-MLD which can be used to minimise the lexicographical distance from any desired priority ordering, enabling important tasks to be placed at high priority levels.

There is one significant drawback with Audsley's algorithm that is it only finds schedulable systems, and thus does not care if the priority assignment results in a system that is on the brink of unschedulability. To combat this problem, the Robust Priority Assignment (RPA) algorithm was introduced in [45]. RPA is also optimal, in that it is guaranteed to find a schedulable priority ordering whenever one exists; however, it also simultaneously maximises the amount of additional interference that the system can tolerate without missing a deadline, thus providing robust rather than fragile priority assignment solutions.

The concepts used in deriving the RPA algorithm have been successfully applied to priority assignment in mixed criticality systems (minimising the number of schedulability tests required), in probabilistic real-time systems (minimising the worst-case deadline failure probability and the deadline miss ratio), and also in systems using fixed priority scheduling with deferred pre-emption, (optimising schedulability via final non-pre-emptive region length assignment).

A number of interesting problem areas remain where OPA and RPA are not obviously applicable. These include examples from fixed priority scheduling with deferred pre-emption (maximising the length of final non-pre-emptive regions to reduce the amount of pre-emption), probabilistic real-time systems (minimising the overall probability of deadline failure), worm-hole routing in Network-on-Chip, the assignment of thresholds as well as priorities in fixed priority scheduling with pre-emption thresholds, and finally fixed priority pre-emptive scheduling accounting for Cache Related Pre-emption Delays.

In conclusion, appropriate priority assignment is of great importance in systems that use fixed priority scheduling. Here, effective priority assignment can ensure that a system is schedulable when otherwise deadlines would be missed, that the system is robust to changes and provides headroom for new functionality to be added without the need to upgrade to more expensive hardware. Further, it can provide enhanced robustness to errors [47] and resilience to failures [70].

Returning to the frequently asked question, "How should I assign priorities?" As a simple rule of thumb, Deadline Monotonic priority assignment i.e. assigning priorities on the basis of deadlines (the shorter the deadline, the higher the priority) or Deadline minus Jitter Monotonic priority assignment is typically effective for single processor systems and for Controller Area Network. Somewhat surprisingly it is however a poor heuristic to use for global fixed priority scheduling in multiprocessor systems.

The Robust Priority Assignment (RPA) algorithm, derived from Audsley's OPA algorithm, is highly effective in many cases and when applicable, this is the method we would recommend. It uses a form of sensitivity analysis to ensure that the priority assignments produced result in a system that is as robust as possible to any additional interference or timing delays.

In more complex distributed real-time systems, for example those prevalent in the automotive domain, where timing requirements apply to functionality that is implemented by tasks distributed across a number of processors communicating via messages sent over one or more networks, priority assignment still plays a crucial role. With these systems a divide and conquer approach may be taken at the design stage, partitioning the overall problem into a set of simpler ones by setting intermediate deadlines. Such a separation of concerns means that the priority assignment techniques discussed in this review can be applied to each sub-problem consisting of the set of tasks on one processor or the set of messages on a single network. This approach has practical advantages to do with composability, when different sub-suppliers are responsible for different components of the system (e.g. different Electronic Control Units or ECUs). However, the quality of the overall solution obtained depends on the intermediate deadlines chosen. The alternative is to take a holistic approach and use techniques such as Simulated Annealing, Genetic Algorithms or Mixed

[8] Signals are small pieces of information transferred between tasks that need to be packed into messages.

Integer Linear Programming to allocate tasks and assign priorities with the aim of optimising schedulability as well as extensibility or robustness to change.

Finally, we note that when designing and implementing hard real-time systems that require guarantees of timing correctness, it is essential that the implemented system behaviour precisely matches the system model assumed by the schedulability analysis. Otherwise such analysis can give no valid guarantees about the timing correctness of the actual system. In some application areas, for example automotive, standards such as those for CAN [27], and the OSEK [1] and AUTOSAR [2] real-time operating systems aid in building predictable real-time systems. They do so by mandating functionality with which it is *possible* to implement analysable systems; however, such an outcome is far from certain, rather the system needs to be carefully designed and engineered to comply with an appropriate, analysable system model so that its timing behaviour can be guaranteed. Choice of a corresponding, robust and optimal priority assignment policy then flows from the system model chosen.

While the last two decades have seen significant progress in priority assignment techniques, many interesting and important problems remain. We hope that this review will encourage other researchers to tackle some of these problems. As a challenge, we point to a 20+ year old conjecture and open problem in priority assignment regarding fixed priority pre-emptive systems where each task has two priorities and switches between them at a fixed time (the promotion time) after it is released. The conjecture states that the utilisation bound for an implicit deadline periodic task system with an appropriate priority and promotion time assignment is 100%, the same as EDF. Currently this remains a conjecture, neither proved nor disproved. Full details are given in [33].

Acknowledgements

This work was partially funded by the UK EPSRC MCC project (EP/K011626/1), the French BGLE Departs and LEOC Capacites projects, the EU FP7 grants P-SOCRATES (611016) and PROXIMA (611085), and the Inria International Chair program. EPSRC Research Data Management: No new primary data was created during this study

References

[1] OSEK/VDK operating system specification, version 2.2.3OSEK/VDK http://portal.osek-vdx.org/files/pdf/specs/os223.pdf 2007.
[2] AUTOSAR specification of operating system v4.10.AUTOSAR http://www.autosar.org/, AUTOSAR, 2010.
[3] A. Aguilar-Soto, G. Bernat, Bi-criteria fixed-priority scheduling in hard real–time systems: Deadline and importance, in: Proceedings Real-Time and Network Systems (RTNS), 2006.
[4] 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 Syst. 48 (5) (2012) 499–526.
[5] S. Altmeyer, L. Cucu-Grosjean, R.I. Davis, "Static probabilistic timing analysis for real-time systems using random replacement caches", Real-Time Syst. 51 (1) (2015) 77–123.
[6] B. Andersson, J. Jonsson, "Fixed-priority preemptive multiprocessor scheduling: to partition or not to partition", In Proceedings Real-Time Computing Systems and Applications (RTCSA), 2000
[7] B. Andersson, J. Jonsson, Some insights on fixed-priority pre-emptive non-partitioned multiprocessor scheduling, in: Proceedings Real-Time Systems Symposium (RTSS) – Work-in-Progress Session, 2000.
[8] B. Andersson, S. Baruah, J. Jonsson, Static-priority scheduling on multiprocessors, in: Proceedings Real-Time Systems Symposium (RTSS), 2001, pp. 193–202.
[9] B. Andersson, Global static-priority preemptive multiprocessor scheduling with utilization bound 38%, proceedings International Conference on Principles of Distributed Systems, 2008.
[10] 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.
[11] N.C. Audsley, A. Burns, M. Richardson, A.J. Wellings, Applying new Scheduling Theory to Static Priority Pre-emptive Scheduling, Softw. Eng. J. 8 (5) (1993) 284–292.
[12] N.C. Audsley, A. Burns, R.I. Davis, K.W. Tindell, A.J. Wellings, Fixed priority scheduling an historical perspective, Real-Time Syst. 8 (3) (1995) 173–198.
[13] N.C. Audsley, On priority assignment in fixed priority scheduling, Inf. Process. Lett. 79 (1) (2001) 39–44.
[14] F. Balarin, Priority assignment for embedded reactive real-time systems, in: Proceedings of Languages, Compilers, and Tools for Embedded Systems (LCTES), 1998, pp. 146–155.
[15] T.P. Baker, Stack-based scheduling of real-time processes, Real-Time Syst. J. 1 (3) (1991) 67–100.
[16] S. Baruah, The limited-preemption uniprocessor scheduling of sporadic task systems, in: Proceedings Euromicro Conference on Real-Time Systems (ECRTS), 2005, pp. 137–144.
[17] S.K. Baruah, A. Burns, R.I. Davis, Response time analysis for mixed criticality systems, in: Proceedings Real-Time Systems Symposium (RTSS), 2011, pp. 34–43.
[18] I. Bate, P. Emberson, Incorporating scenarios and heuristics to improve flexibility in real-time embedded systems, in: Proceedings Real-Time and embedded Technology and Applications Symposium (RTAS), 2006, pp. 221–230.
[19] L.A. Belady, A study of replacement algorithms for a virtual-storage computer, IBM Syst. J. 5 (2) (1966) 78–101.
[20] M. Bertogna, M. Cirinei, Response time analysis for global scheduled symmetric multiprocessor platforms, in: proceedings Real-Time Systems Symposium (RTSS), 2007, pp. 149–158.
[21] M. Bertogna, M. Cirinei, G. Lipari, Schedulability analysis of global scheduling algorithms on multiprocessor platforms, IEEE Trans. parallel Distributed Syst. 20 (4) (2009) 553–566.
[22] M. Bertogna, G. Buttazzo, G. Yao, Improving feasibility of fixed priority tasks using non-preemptive regions, in: Proceedings Real-Time Systems Symposium (RTSS), 2011.
[23] M. Bertogna, M. Cirinei, G. Lipari, New schedulability tests for real-time task sets scheduled by Deadline Monotonic on multiprocessors, in: Proceedings of the International Conference on Principles of Distributed Systems (OPODIS 2005), Pisa, Italy, 2005.
[24] R. Bril, J. Lukkien, W. Verhaegh, Worst-case response time analysis of real–time tasks under fixed-priority scheduling with deferred preemption, Real–Time Syst. 42 (1-3) (2009) 63–119.
[25] 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), 2012, pp. 209–220.
[26] K. Bletsas, N. Audsley, Optimal priority assignment in the presence of blocking, Inf. Process. Lett. 99 (3) (2006) 83–86.
[27] Bosch. 1991. "CAN Specification version 2.0". Robert Bosch GmbH, Postfach 30 02 40, D-70442 Stuttgart.
[28] I. Broster, A. Burns, G. Rodríguez-Navas, Probabilistic analysis of CAN with faults, in: Proceedings Real-Time Systems Symposium (RTSS), 2002, pp. 269–278.
[29] I. Broster, A. Burns, G. Rodriguez-Navas, Timing analysis of real-time communication under electromagnetic interference, Real-Time Syst. 30 (1-2) (2005) 55–81.
[30] A. Burns, K. Tindell, A.J. Wellings, Fixed priority scheduling with deadlines prior to completion, in: Proceedings Euromicro Workshop on Real-Time Systems, 1994, pp. 138–142.
[31] A. Burns, Preemptive priority based scheduling: An appropriate engineering approach, in: S. Son (Ed.), Advances in Real-Time Systems, Springer, 1994, pp. 225–248.
[32] A. Burns, R.I. Davis, Choosing task periods to minimise system utilisation in time triggered systems, Inf. Process. Lett. Vol. 58 (1996) 223–229 Elsevier.
[33] A. Burns, Dual priority scheduling: is the processor utilisation bound 100%, in: Proceedings Real-Time Scheduling Open Problems Symposium (RTSOPS), 2010.
[34] G.C. Buttazzo, M. Bertogna, G. Yao. "Limited preemptive scheduling for real-time systems: a survey". IEEE Trans. Ind. Inform. In press. Downloadable from http://retis.sssup.it/~marko/publi.html.
[35] D. Buttle, Real-time in the prime time, Keynote talk at Euromicro Conference on Real-Time Systems (ECRTS), 2012 Presentation: http://ecrts.eit.uni-kl.de/index.php?id=69.
[36] F. 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, D. Maxim, Proartis: Probabilistically analysable real-time systems, Trans. Embedded Comput. Syst. (2013).
[37] Y. Chu, A. Burns, Flexible hard real-time scheduling for deliberative AI systems, Real-Time Syst. 40 (3) (2008) 241–263.
[38] H.S. Chwa, H. Back, S. Chen, J. Lee, A. Easwaran, I. Shin, I. Lee, Extending task-level to job-level fixed priority assignment and schedulability analysis using pseudo-deadlines, in: Proceedings Real-Time Systems Symposium (RTSS), 2012, pp. 51–62.
[39] L. Cucu, J. Goossens, Feasibility intervals for fixed-priority real-time scheduling on uniform multiprocessors, in: Proceedings Emerging Technologies and Factory Automation, (ETFA), 2006.
[40] L. Cucu, J. Goossens, Feasibility intervals for multiprocessor fixed-priority scheduling of arbitrary deadline periodic systems, in: Proceedings Design Automation and Test in Europe (DATE), 2007, pp. 1635–1640.
[41] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quiones, F.J. Cazorla, Measurement-based probabilistic timing analysis for multi-path programs, in: Proceedings Euromicro Conference on Real-Time Systems (ECRTS), 2012, pp. 91–101.

[42] L. Cucu-Grosjean, Independance - a misunderstood property of and for probabilistic real-time systems, Alan Burns 60th Anniversary workshop, 2013.

[43] R.I. Davis, A. Burns, Optimal priority assignment for aperiodic tasks with firm deadlines in fixed priority pre-emptive systems, Inf. Process. Lett. 53 (5) (1995).

[44] 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.

[45] R.I. Davis, A. Burns, "Robust priority assignment for fixed priority real–time systems", in: Proceedings Real-Time Systems Symposium (RTSS), 2007, pp. 3–14.

[46] R.I. Davis, A. Burns, R.J. Bril, J.J. Lukkien, Controller Area Network (CAN) schedulability analysis: refuted, revisited and revised, Real-Time Syst. 35 (3) (2007) 239–272.

[47] R.I. Davis, A. Burns, Robust priority assignment for messages on Controller Area Network (CAN), Real-Time Syst. 41 (2) (2009) 152–180.

[48] R.I. Davis, A. Burns, Department of Computer Science Technical Report, YCS-2009-451, University of York, 2010.

[49] R.I. Davis, A. Burns, Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems", Real-Time Syst. 47 (1) (2011) 1–40.

[50] R.I. Davis, A. Burns, A survey of hard real-time scheduling for multiprocessor systems, ACM Comput. Surv. 43 (4) (2011) 44 Article 35.

[51] 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 Syst. 49 (1) (2013) 73–116.

[52] R.I. Davis, M. Bertogna, Optimal fixed priority scheduling with deferred pre-emption, in: Proceedings Real-Time Systems Symposium (RTSS), 2012, pp. 39–50.

[53] R.I. Davis, A review of fixed priority and EDF scheduling for hard real-time uniprocessor systems, in: ACM SIGBED Review - Special Issue on the 3rd Embedded Operating Systems Workshop (Ewili 2013), 11, 2014, pp. 8–19, doi:10.1145/2597457.2597458.

[54] S.K. Dhall, C.L. Liu, On a Real-Time Scheduling Problem, Oper. Res. 26 (1) (1978) 127–140.

[55] J. Diaz, D. Garcia, K. Kim, C.-G. Lee, L. Lo Bello, J. Lopez, S.-L. Min, O. Mirabella, Stochastic analysis of periodic real-time systems, in: Proceedings Real-Time Systems Symposium (RTSS), 2002, pp. 289–300.

[56] M.S. Fineberg, O. Serlin, Multiprogramming for hybrid computation, in: Proceedings AFIPS Fall Joint Computing Conference, 1967, pp. 1–13.

[57] T. Fleming, A. Burns, Extending mixed criticality scheduling, in: Proceedings 1st International Workshop on Mixed Criticality Systems (WMC), 2013, pp. 7–12.

[58] J.J.G. Garcia, M.G. Harbour, Optimized priority assignment for tasks and messages in distributed real-time systems, in: Proceedings Workshop on Parallel and Distributed Real-Time Systems, 1995.

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

[60] J. Goossens, R. Devillers, The non-optimality of the monotonic priority assignments for hard real-time offset free systems, Real-Time Syst. 13 (2) (1997) 107–126.

[61] N. Guan, M. Stigge, W. Yi, G. Yu, New response time bounds for fixed priority multiprocessor scheduling, in: Proceedings of the Real-Time Systems Symposium, 2009, pp. 388–397.

[62] A. Hamann, M. Jersak, K. Richter, R. Ernst, Design space exploration and system optimization with SymTA/S - symbolic timing analysis for systems, in: Proceedings Real-Time Systems Symposium (RTSS), 2004.

[63] X. He, W. Gu, Y. Ahum, Task allocation and optimization of distributed embedded systems with simulated annealing and geometric programming, Computer J (2009).

[64] M. Joseph, P.K. Pandya, Finding response times in a real-time system, Comput. J. 29 (5) (1986) 390–395.

[65] D.I. Katcher, H. Arakawa, J.K. Strosnider, Engineering and analysis of fixed priority schedulers, IEEE Trans. Softw. Eng. 19 (9) (1993) 920–934.

[66] J. Lehoczky, Fixed priority scheduling of periodic task sets with arbitrary deadlines, in: Proceedings Real-Time Systems Symposium (RTSS), 1990, pp. 201–209.

[67] J.Y.-T. Leung, J. Whitehead, On the complexity of fixed-priority scheduling of periodic real-time tasks, Perform. Eval. 2 (4) (1982) 237–250.

[68] J.P. Lehoczky, L. Sha, Y. Ding, The rate monotonic scheduling algorithm: exact characterization and average case behaviour, in: Proceedings Real-Time Systems Symposium (RTSS), 1989, pp. 166–171.

[69] C.L. Liu, J.W. Layland, Scheduling algorithms for multiprogramming in a hard-real-time environment, J. ACM 20 (1) (1973) 46–61.

[70] G. de, A Lima, A. Burns, An optimal fixed-priority assignment algorithm for supporting fault-tolerant hard real-time systems, IEEE Trans. Comput. 52 (10) (2003) 1332–1346.

[71] D. Maxim, O. Buffet, L. Santinelli, L. Cucu-Grosjean, R.I. Davis, Optimal priority assignment algorithms for probabilistic real-time systems, in: Proceedings Real-Time and Network Systems (RTNS), 2011, pp. 129–138.

[72] N. Navet, Y-Q. Song, F. Simonot, Worst-case deadline failure probability in real-time applications distributed over controller area network, J. Syst. Archit. 46 (1) (2000) 607–617.

[73] R.M. Pathan, J. Jonsson, Improved schedulability tests for global fixed-priority scheduling, in: Proceedings Euromicro Conference on Real-Time Systems (ECRTS), 2011, pp. 136–147.

[74] S. Punnekkat, R. Davis, A. Burns, Sensitivity analysis of real-time task sets, in: Proceedings of the Asian Computing Science Conference, Nepal, 1997, pp. 72–82.

[75] P. Ramanathan, Overload management in real-time control applications using (m, k)-firm guarantee, in: IEEE Transactions on Parallel and Distributed Systems, vol.10, 1999, pp. 549–559.

[76] J. Regehr, Scheduling tasks with mixed pre-emption relations for robustness to timing faults, in: Proceedings Real-Time Systems Symposium (RTSS), 2002, pp. 315–326.

[77] M. Richard, P. Richard, F. Cottet, Task and message priority assignment in automotive systems, in: Proceedings 4th FeT IFAC conference on fieldbus systems and their applications, 2001, pp. 105–112.

[78] M. Saksena, S. Hong, An engineering approach to decomposing end-to-end delays on a distributed real-time system, in: Proceedings of the International Workshop on Parallel and Distributed Real-Time Systems, 1996, pp. 244–251.

[79] M. Saksena, Y. Wang, Scalable real-time system design using preemption thresholds, in: Proceedings Real-Time Systems Symposium (RTSS), 2000.

[80] K.W. Schmidt, Robust priority assignments for extending existing controller area network applications, in: Industrial Informatics, IEEE Transactions on, 10, 2014, pp. 578–585, doi:10.1109/TII.2013.2266636.

[81] L. Sha, J.P. Lehoczky, R. Rajkumar, Solutions for some practical problems in prioritizing preemptive scheduling, in: Proceedings Real-Time Sytems Symposium (RTSS), 1986.

[82] L. Sha, R. Rajkumar, J. Lehoczky, Priority inheritance protocols: an approach to real-time synchronization, IEEE Trans. Comput. 39 (9) (1990).

[83] L. Sha, T. Abdelzaher, K.-E. Arzen, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, A. Mok, Real-time scheduling theory: a historical perspective, Real-Time Syst. 28 (2-3) (2004) 101–155.

[84] A. Saifullah, Y. Xu, C. Lu, Y. Chen, Priority assignment for real-time flows in wirelessHART networks, in: Proceedings Euromicro Conference on Real-Time Systems (ECRTS), 2011.

[85] M. Stigge, W. Yi, Combinatorial abstraction refinement for feasibility analysis, in: Proceedings Real-Time Systems Symposium (RTSS), 2013, pp. 340–349.

[86] M. Stigge, W. Yi, Combinatorial abstraction refinement for feasibility analysis of static priorities, Real-Time Systems 51 (6) (2015) 1383–1573.

[87] K.W. Tindell, A. Burns, A.J. Wellings, Allocating hard real-time tasks: an NP-hard problem made easy, Real-Time Syst. 4 (2) (1992) 145–165.

[88] K.W. Tindell, J. Clark, Holistic schedulability analysis for distributed hard real–time systems, Microprocess. Microprogram. 40 (2-3) (1994) 117–134.

[89] K.W. Tindell, A. Burns, A.J. Wellings, An extendible approach for analyzing fixed priority hard real-time tasks, Real-Time Syst. 6 (2) (1994) 133–151.

[90] S. Vestal, Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance, in: Proceedings of Real-Time Systems Symposium (RTSS), 2007, pp. 239–243.

[91] Y. Wang, M. Saksena, Scheduling fixed-priority tasks with pre-emption threshold, in: Proceedings Real-Time Computing Systems and Applications (RTCSA), 1999.

[92] H.C. Wong, A. Burns, Schedulability analysis for the abort-and-restart (AR) model, in: Proceedings International Conference on Real-Time Networks and Systems (RTNS), 2014.

[93] S. Zheng, A. Burns, Priority assignment for real-time wormhole communication in on-chip networks, in: Proceedings Real-Time Systems Symposium (RTSS), 2008, pp. 421–430.

[94] Q. Zhu, H. Zeng, W. Zheng, M. Di Natale, A. Sangiovanni-Vincentelli, Optimization of task allocation and priority assignment in hard real-time distributed systems, ACM Trans. Embed. Comput. Syst. 11 (4) (2013) Article 85.

[95] Q. Zhu, Y. Yang, E. Scholte, M. Di Natale, A. Sangiovanni-Vincentelli, Optimizing extensibility in hard real-time distributed systems, in: Proceedings Real-Time and Embedded Technology and Applications Symposium (RTAS), 2009.

[96] A. Zuhily, A. Burns, Optimality of (D-J)-monotonic priority assignment, Inf. Process. Lett. 103 (6) (2007).

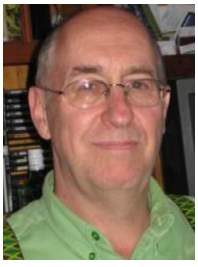
[97] A. Zuhily, A. Burns, Exact scheduling analysis of non-accumulatively monotonic multiframe tasks, Real-Time Syst. 43 (2) (2009) 119–146.

[98] R.I. Davis, A. Burns, V. Pollex, F. Slomka, On Priority Assignment for Controller Area Network when some Message Identifiers are Fixed, in: Proceedings 23rd International Conference on Real-Time Networks and Systems (RTNS 2015), 2015, pp. 279–288.

Robert I. Davis is a Senior Research Fellow in the Real-Time Systems Research Group at the University of York, UK, and an INRIA International Chair with the AOSTE team at INRIA, Paris, France. Robert received his DPhil in Computer Science from the University of York in 1995. Since then he has founded three start-up companies, all of which have succeeded in transferring real-time systems research into commercial products. Robert's research interests include the following aspects of real-time systems: scheduling algorithms and analysis for single processor, multiprocessor and networked systems; analysis of cache related pre-emption delays, mixed criticality systems, and probabilistic hard real-time systems.

Liliana Cucu-Grosjean is an INRIA researcher with the AOSTE team at Inria Paris, France. She has a degree in Mathematics, an MSc in Physics, and a PhD in Computer Science awarded in 2004 by the University of Paris Sud. Since then she has been active in three major European real-time and embedded systems research groups, in Portugal, Belgium, and France. Liliana has been involved in, and led, numerous national and European projects focused on real-time scheduling, timing analysis and the use of probabilistic and statistical methods.

Marko Bertogna is Associate Professor at the University of Modena, Italy. He previously was Assistant Professor at the Scuola Superiore Sant'Anna of Pisa, Italy, where he also received (cum laude) a Ph.D. in Computer Engineering.
He has authored over 60 papers in international conferences and journals in the field of real-time and multiprocessor systems, receiving six Best Paper Awards and one Best Dissertation Award. He served in the Program Committees of the major international conferences on real-time systems. He is Senior Member of the IEEE.



Alan Burns co-leads the Real-Time Systems Research Group at the University of York. His research interests cover a number of aspects of real-time systems including the assessment of languages for use in the real-time domain, distributed operating systems, the formal specification of scheduling algorithms and implementation strategies, and the design of dependable user interfaces to real-time applications. Professor Burns has authored/co-authored 500 papers/reports and books. Most of these are in the real-time area. His teaching activities include courses in Operating Systems and Real-time Systems. In 2009 Professor Burns was elected a Fellow of the Royal Academy of Engineering. In 2012 he was elected a Fellow of the IEEE.